BrilliAnts

New Mexico
Supercomputing Challenge
Final Report
April 6, 2011

Team 56
Los Alamos High School

Team Members

Peter Ahrens
Dustin Tauxe
Stephanie Djidjev

Teacher

Lee Goodwin

Project Mentors

Christine Ahrens

Table of Contents

1.0 Executive Summary

This project explores which Ant Colony Optimizations (ACOs) work best for the Dynamic Traveling Salesman Problem (DTSP). ACOs are algorithms based on ant foraging behavior. The TSP is a problem in which cities in an undirected graph must be connected by the shortest tour possible. A tour is a path that visits each city once and only once. A DTSP is a Traveling Salesman Problem instance in which cities may be added or removed as the optimization is running. This has applications in problems including vehicle routing, networking, communications, and scheduling. Python was used to implement five ACO algorithms on TSP: Ant System, Ant Colony System, Min-Max Ant System, Rank-Based Ant System, and Elitist Ant System. We later modified them for use in DTSP. We created a test harness, complete with a scripting language. We then modified the algorithms to run in parallel. Experiments testing deletion and addition of 10 or 50 cities at various points in the simulation were run. To evaluate the performance of the algorithms, we found the average percentage deviation from the optimum over a time period of 0-100 seconds. Using this data, we determined Min-Max Ant System to be the most efficient, reliable, and adaptable algorithm to changes in the longer term. We found Rank-Based Ant System to be the fastest-converging algorithm after changes, i.e. it produced good solutions very quickly after the change.

2.0 Problem Statement

Ants are ingenious creatures. Using only the **pheromones** (attractive chemicals produced by the ants) that they lay down, ants can optimize the length of their foraging trails within short periods of time. The ants leave pheromone on the paths that they take, and the pheromone evaporates over time. Longer paths then have less pheromone accumulation than shorter ones, causing more ants to be more attracted to the shorter paths. Applying this behavior to real world problems in computer simulations is called **Ant Colony Optimization** (ACO). Although Ant Colony Optimization seems more suited to foraging, it has proven itself a powerful metaheuristic that can be applied to problems ranging from routing to machine learning.

ACOs are conceptually suited to and commonly applied to the **Traveling Salesman Problem** (TSP). This is a very well-documented combinatorial optimization problem. In **Symmetric TSP** (referred to as TSP in this paper), n nodes in an undirected graph must be connected in the shortest tour possible. A **tour** is a path that visits each node once and only once. Each node is defined as a **city**, and a path connecting two cities is called a **route**. The number of possible tours in a data set with n cities is given by:

$$\frac{(n-1)!}{2} \quad \text{(eq. 1)}$$

Solving a 200-city TSP using brute force would take approximately 2.062×10^{360} years on an ASUS G53JW with an Intel Core i7 1.73GHz running Ubuntu Linux 10.10, yet an ACO can get to within 2% of the optimum in 200 seconds.

In most applications of ACO, the problem does not change in the same time period in which the optimization is being run. In many cases, the ACO is rerun on a changed problem. This is true for applications such as logistics and planning, where things like a delivery cancellation can be handled in a matter of hours. For very time-sensitive applications involving networks and communications, however, changes must be handled in very short time periods. Having an implementation of ACO that can cope with this is advantageous in a situation where

time is of the essence, because in situations where time is not a factor, the optimization can simply be run again after the change is implemented. A **Dynamically Changing TSP**, or DTSP, is a Traveling Salesman Problem instance in which cities may be added or removed as the optimization is running. An algorithm that works well on the DTSP is advantageous because previously computed data does not have to be recalculated.

Our project compares the behaviors (performance and quality) of the most common implementations of ACO on Dynamically Changing TSPs. This helps to select a suitable ACO for different DTSPs based on performance and/or quality criteria.

Previous research in applying ACOs to DTSP shows that running the optimization again from scratch on the changed set of cities will eventually yield a better solution in the long term. Also, research has been conducted on applying several pheromone modification strategies after the change to yield better results. As of yet, different implementations of ACO have not been compared in their fitness with DTSP. [9][10][11]

3.0 Description of the Method Used to Solve the Problem

We implemented five different ACOs to static TSPs, and later modified them for use in DTSP. We created a test harness, complete with scripting language. The ACOs were modified to run in parallel, and a graphic display was coded to show real time behavior of the algorithm. Experiments were run using the test harness and data was analyzed using plotting tools. What follows is a background on the ACO algorithms and an explanation of all steps involved in running these experiments.

3.1 Ant Colony Optimization Background

All Ant Colony Optimizations have the same approximate structure. To initialize, they calculate all the distances between all the cities, make a pheromone and probability matrix (a way to store the values of all the pheromones on all the trails), and create ants. They then move into their first iteration. An iteration of an ant colony optimization applied to TSP consists of tour construction by all ants and pheromones update.

Ants start their tour construction at a random city. They then use probabilistic rules to decide where to move next until they have visited all the cities. Two factors influence these decisions. The first factor, τ_{ij} is the pheromone on a route from city i to city j. The second factor, η_{ij} is the inverse of the distance. The probability (p_{ij}) that ant k at city i will move to city j is given by the equation:

$$Pij = \frac{\left(\eta_{ij}\right)^{\alpha}\left(\tau_{ij}\right)^{\beta}}{\sum_{l \in N_i^k}\left(\eta_{il}\right)^{\alpha}\left(\tau_{il}\right)^{\beta}}, \quad if\ j \in N_i^k$$ (eq. 2) [1]

where N_i^k is a collection of all the cities the ant has not yet visited and α and β are parameters. Pheromone update is achieved in many different ways for different algorithms. In all cases, the base unit of pheromone an ant lays down, $\Delta\tau_{ij}$, is given by:

$$\Delta\tau_{ij} = \frac{1}{C_{ij}}$$ (eq. 3) [1]

where C_{ij} is the ant's tour length. After the ants deposit pheromone in some configuration, evaporation occurs on all routes. The new amount of pheromone on a route τ_{ij}' is given by the equation:

$$\tau_{ij}' = \tau_{ij}\left(1-\rho\right)$$ (eq. 4) [1]

where ρ is a parameter (from 0-1). Pheromone update occurs in many ways, so the above equations are to help the reader understand the basic ways the pheromone update works.

We chose five different implementations of ACO algorithms to compare: Ant System, Ant Colony System, Min-Max Ant System, Rank-Based Ant System, and Elitist Ant System. These are the most well researched implementations. They differ mainly in the way they evaporate and deposit pheromone.

Table 1

Algorithm:	Pheromone Deposited by:	Evaporation Occurs on:	Special Considerations:
Ant System	All ants equally	All routes very quickly	The original implementation proposed by Marco Dorigo, does not scale (in terms of quality of solution) to a large number of cities.
Elitist Ant System	All ants, and the iteration best ant deposits a very large amount	All routes very quickly	This algorithm was proposed by Marco Dorigo to make AS scale to larger datasets
Rank-Based Ant System	The amount deposited decreases according to an assigned rank of the ant	All routes	Behaves similarly to EAS, but is slightly better
Min-Max Ant System	The iteration or global best ant	All routes very slowly	Maximum and minimum pheromone levels are imposed to prevent stagnation. This is one of the most researched implementations
Ant Colony System	The global best ant	The global best ant's tour	Each time an ant uses a route, that route's pheromone decreases. During tour construction, ants use the pseudorandom proportional action choice rule, increasing the probability of choosing the most probable (eq. 2) route.

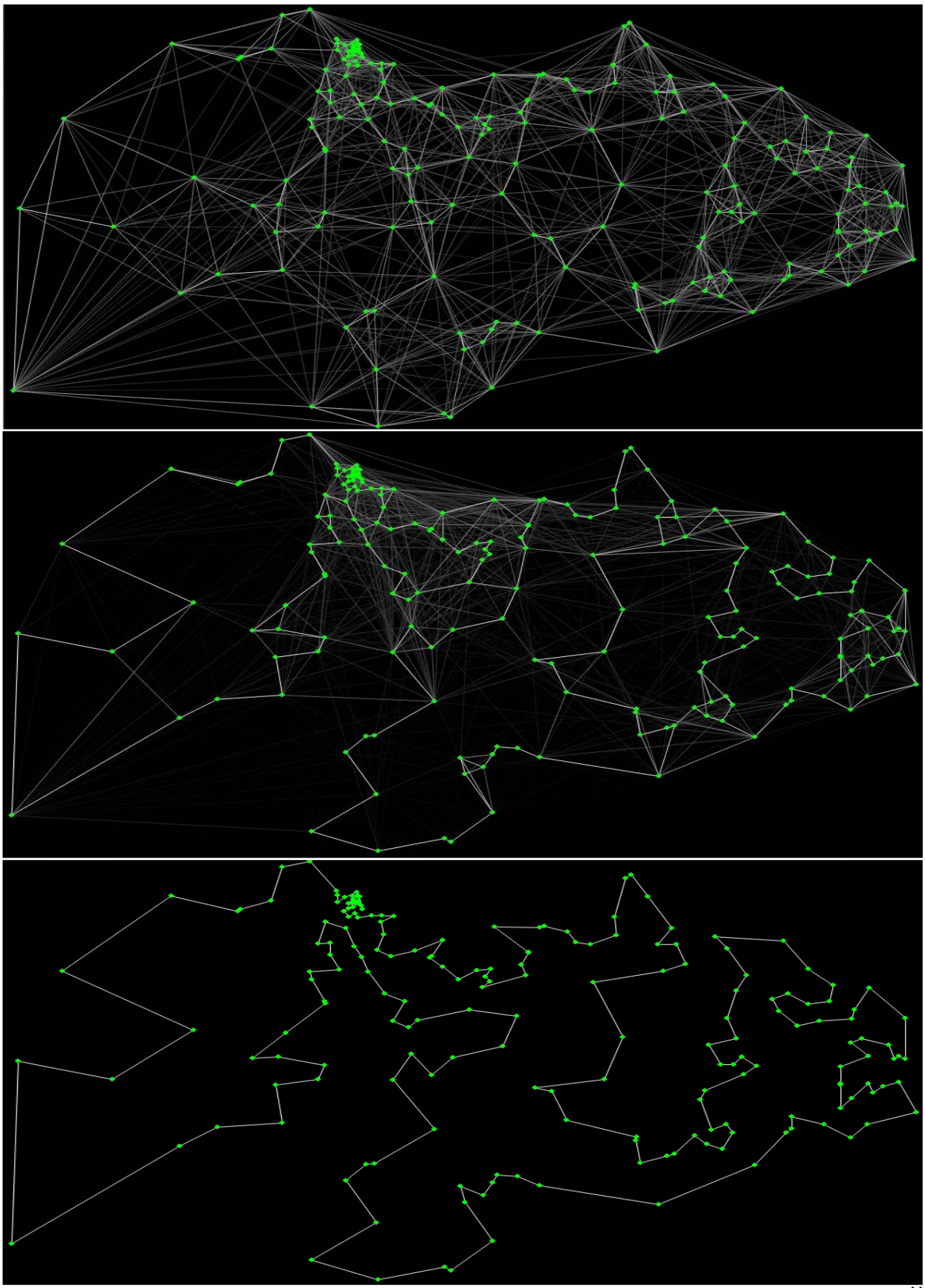
Ant Colony Optimization is similar to many metaheuristics in that it has **exploratory** and **exploitative** phases. When the optimizations initialize, they are very exploratory, and the

pheromone trails are unbiased, meaning that ants are not overly influenced to take one route over another due to the pheromone on that route. Algorithms such as Min-Max Ant System are very exploratory, and thus take a while to converge. As pheromones along routes start to evaporate, the trails do become biased, and the algorithm starts to improve heavily on its best tour using the data it found while exploring, exploiting data in the pheromone trails. This stage is necessary, and is where the most improvement happens, but it can lead to an unwanted situation, called **stagnation**. This is when one tour is so heavily emphasized in the pheromone trails that all the ants are forced to take it, making it more emphasized and wasting computation time.

In order to better understand the state of our simulations, we created the MagnifyingGlass visualizer program that allows the user to view the state of the simulation live. MagnifyingGlass was created in PyGame, an add-on module that allows for 2-D displays in Python. The program is capable of creating a visual representation of the problem. Although the actual data exists in the program as lists of numbers, MagnifyingGlass represents cities as points connected by lines that represent the routes. This is much easier to look at and be able to understand the simulation. Besides showing the nodes and the current tour, the MagnifyingGlass will also display a text readout giving the user information about the problem, such as the type of implementation currently running, the best-so-far tour, and the current time elapsed for the simulation. In addition, the program is capable of displaying the levels of pheromones along different routes, as expressed by the change in color of the lines among those routes. It can also highlight routes that have remained static in the simulation for a given number of iterations.

In Picture 1, an example of Rank-Based Ant System over time is shown with MagnifyingGlass. The dataset is qa194.tsp [14] (see section 7.7). The shade of the routes represents the intensity of the pheromone (white is most intense). The upper frame represents time 0, in which many pheromone trails are available for an ant to follow, and the graph is fairly unbiased. This is the beginning of the simulation. As the simulation progresses, in the middle frame, the attraction of ants to certain routes becomes very strong. On the bottom frame, the algorithm is in its last stages, one can see the emphasis of pheromone attraction on a single tour. This shows a stagnation situation, where ants are drawn to a single tour, and the algorithm is caught in a local minima.

(Picture 1)



To quantify stagnation, a metric called the lambda branching factor is used. The λ-branching factor for a city i is given as all the incident arcs to i that satisfy the inequality shown below:

$$\tau_{ij} \geq \tau^{i}_{min} + \lambda\left(\tau^{i}_{max} - \tau^{i}_{min}\right) \text{ (eq. 5) [1]}$$

where τ^{i}_{max} and τ^{i}_{min} are the maximum and minimum pheromone values on trails incident to i and λ is a parameter, traditionally set to 0.05. The average λ-branching factor is the average of the λ-branching factor for all nodes. This metric can be used to measure the exploration the ants are doing.

ACO is an agent-based algorithm. Each individual ant can be though of as an agent, which communicates with the other agents (ants) via pheromones that it leaves on the edges of the path. These algorithms simulate the interactions of these autonomous agents and the collective behavior of all ants converges towards a near-optimal tour. As well as the cooperation shown between agents in ACO, there is also an individual decision-making formula that further defines this agent-based model. There is a probability in an ACO applied to TSP that an ant can choose the less attractive path over the more attractive path, which promotes exploration for the optimal tour. In the process of laying out pheromones on paths, the agents modify their environment to come up with the near-optimal solution. In summary, ACO is an agent-based algorithm because ants exhibit emergent behaviors, make individual decisions, and modify their environment.

3.2 Implementation

All the algorithms described above are available on online in C and described by Dorigo in in [1]. We found it difficult to modify these C algorithms to do DTSP. This was because dynamic memory allocation is required for the addition or deletion of cities. In the C code, all the memory was already statically defined. Therefore, we re-implemented the algorithms in Python.

To verify these algorithms we compared our solutions to the optimal solution of the d198.tsp dataset, which is published online at TSPLIB [13]. The solutions were at a maximum of 2 % difference from published results in deviation from the optimum.

After the implementations were coded and verified on static TSPs, they were modified to dynamic TSP's. As dynamic systems are relatively unexplored, there were assumptions made. Two functions were defined, one to delete cities, the other to add cities. When deleting a city, all the pheromone data for routes going to and from that city were deleted. When adding a city, pheromone levels were initialized to the average pheromone level before addition.

A test harness, named AntFarm, was created for the experiment in Python. It includes timing mechanisms via the clock() function in the time module of Python. All timings were only the CPU time used for the ant colony optimization (tour construction, pheromone updates, and probability computation), not the test harness or data initialization. AntFarm was modified to take a script for input, allowing processing tasks to be distributed among team members' computers.

The AntFarm test scripting language (see section 7.5) allows the user to specify the TSP problem to solve and the ACOs to use on the problem. The user can specify if MagnifyingGlass should be turned on or off, set the refresh rate for it, and specify what to display (pheromones, tour and/or stagnation). Parameters such as rho (ρ) or beta (β) can be specified with initial value, increment and end value. The user can specify how long to run the ACOs on the problem and what metrics to list in the output (eg. global best tour length or lambda branching factor). AntFarm can output just results or continuous output. Example output is in section 7.6. The user can specify any number of changes (additions and deletions) to make to the TSP, the time intervals for the changes and specify the changed cities by name or from a file.

The algorithms were then implemented in parallel. Since their most computationally intensive section was tour construction, all the ants constructed their tours in parallel. This was executed using Python's built-in multiprocessing library. Speedup is defined as follows it was found that in all implementations other than Ant Colony System, the speedup experienced was approximately 2.0 according to the speedup formula (time on single processor / time on multiple processors), on the ASUS computer. In Ant Colony System, however, there was a speedup of

0.57. This is due to ACS only using 10 ants. The extra processing power was not justified by the overhead needed to compute in parallel. The ants need to see the entire pheromone matrix to construct a tour, creating a data intensive computation, and running the ants in parallel seemed to only be justified by an excess of roughly 50 ants that need to construct tours. Having all algorithms run at their optimum speed helped save time running the lengthy tests needed to generate the results.

MagnifyingGlass, the graphic display used to create Picture 1, was created in PyGame, an add-on module that allows for 2-D displays in Python. To create such a display the updates in real time requires a separate process for the display to run in. We implemented this using the multiprocessing built-in module in Python, and communicated between the test harness and the display using a pipe (Python's version of a socket for interprocess communication).

Three experiments were conducted after coding, and the results were analyzed using OpenOffice.org Calc and QtiPlot.

4.0 Results

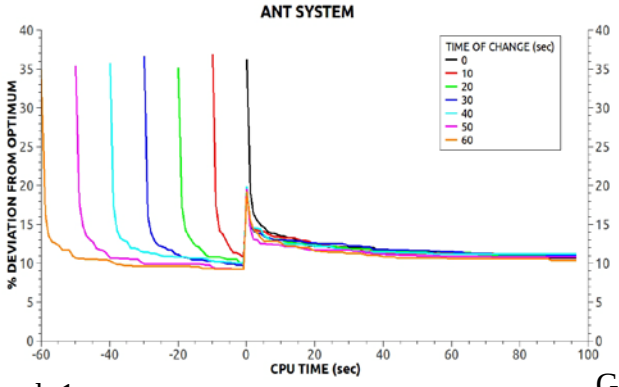
We ran three experiments. All experiments were run on the qa194 dataset (describing all 194 major population centers in Qatar). This dataset was suitable because the cities had a reasonably even spacing and there were not so many of them that it would bog down the experiments. A change was implemented at the beginning of the simulation, taking away or adding a certain number of cities. The cities were reintroduced or deleted later in the simulation, making the % deviation from optimum possible to calculate, as the optimum was known for the original dataset. For all experiments, ten trials were run and averaged. Experiments 1 and 2 were run on a homemade machine with an Intel Core2 Quad Q8400 2.66GHz processor and 5GB RAM running Windows 7. Experiment 3 was run on an ASUS G53JW with an Intel Core i7 1.73GHz processor and 10GB RAM running Ubuntu Linux 10.10.

4.1 Experiment 1

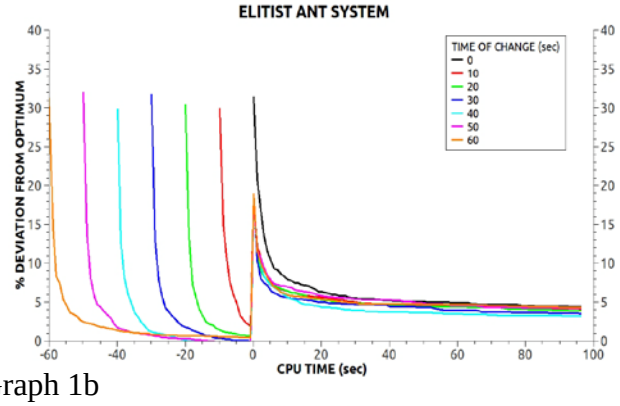
In Experiment 1, 10 cities were taken away when the optimization started and reintroduced later. For the discussion of these results, a preconditioned pheromone matrix is defined as the pheromone matrix that was created by optimizing for the original TSP, before the change. The results for seven possible reintroduction times of ten cities are shown below

(Graph1). The X-axis represents CPU time. The Y-axis represents % deviation from the optimum. For the purposes of comparison, time zero represents the time the cities were reintroduced. Looking at the curves to the right of zero seconds, one will notice that different algorithms react to changes in different ways. The charts are all drawn to the same scale, and solving the same problem, so one may compare how the algorithms behave. Look at the curves to the left of zero seconds.

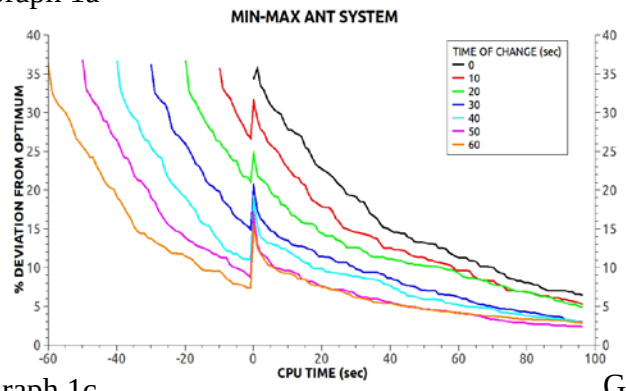
These curves show preconditioning of the pheromone matrix. These are not completely accurate, as the TSP that they are solving is 10 cities different than the one on the right of zero seconds.



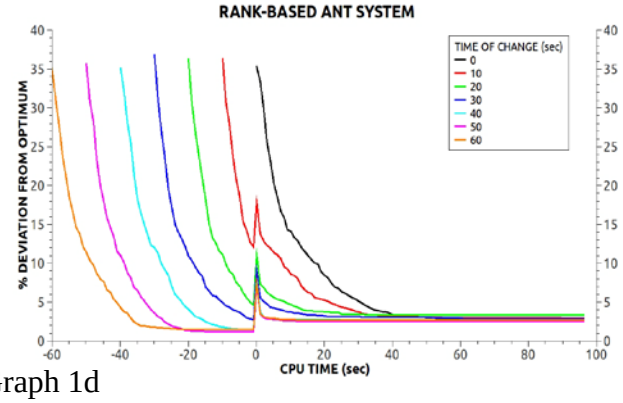
Graph 1a



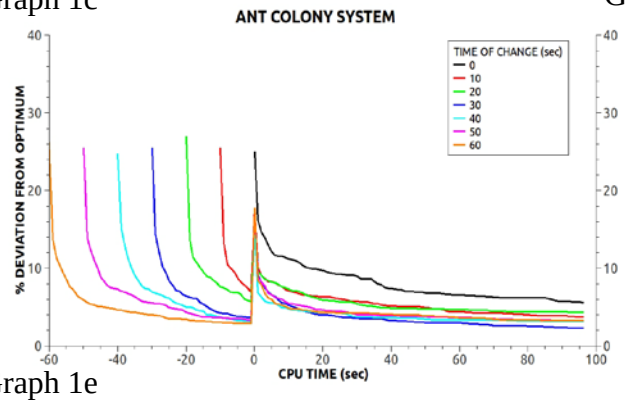
Graph 1b



Graph 1c



Graph 1d



Graph 1e

In Graphs 1a-e, despite the inaccuracy of the curves on the left, they show the preconditioning for the pheromone matrices. If the change is small, like it is above, the data shows that it can be beneficial in the short and long term to use a preconditioned pheromone matrix. This is a contradiction with previous research showing that a rerun of the system is always beneficial in the long term. This is not always true. Depending on the time of the introduction of the change, it could be beneficial to the simulation. The data above also shows that there is a limit to the benefit of a preconditioned matrix. In cases where a change was implemented around 60 seconds, the pheromone data gathered was too exploitative, and either showed equal or less quality than the 50 second cases. This suggests an optimum time to introduce changes exists, and in ant colony optimizations that deal with dynamic problem sets, it may be beneficial to save preconditioned pheromone matrix before heavy exploitation.

It is difficult to directly compare the algorithms because they have such different behaviors. Chart 1 (below) is proposed to show the differences between them. The data for Chart 1 and Chart 2 was gathered from Experiment 1. Since dynamic problem sets are so time-sensitive, the differences between the algorithms must be compared at all data points. Chart 1 shows the average values in the interval 0-100 seconds after the change, for an addition of ten cities introduced at various intervals. If quick convergence is not a concern, the algorithms may be compared by their quality at a fixed time. The percentage deviation from the optimum for each algorithm is shown below in Chart 2.

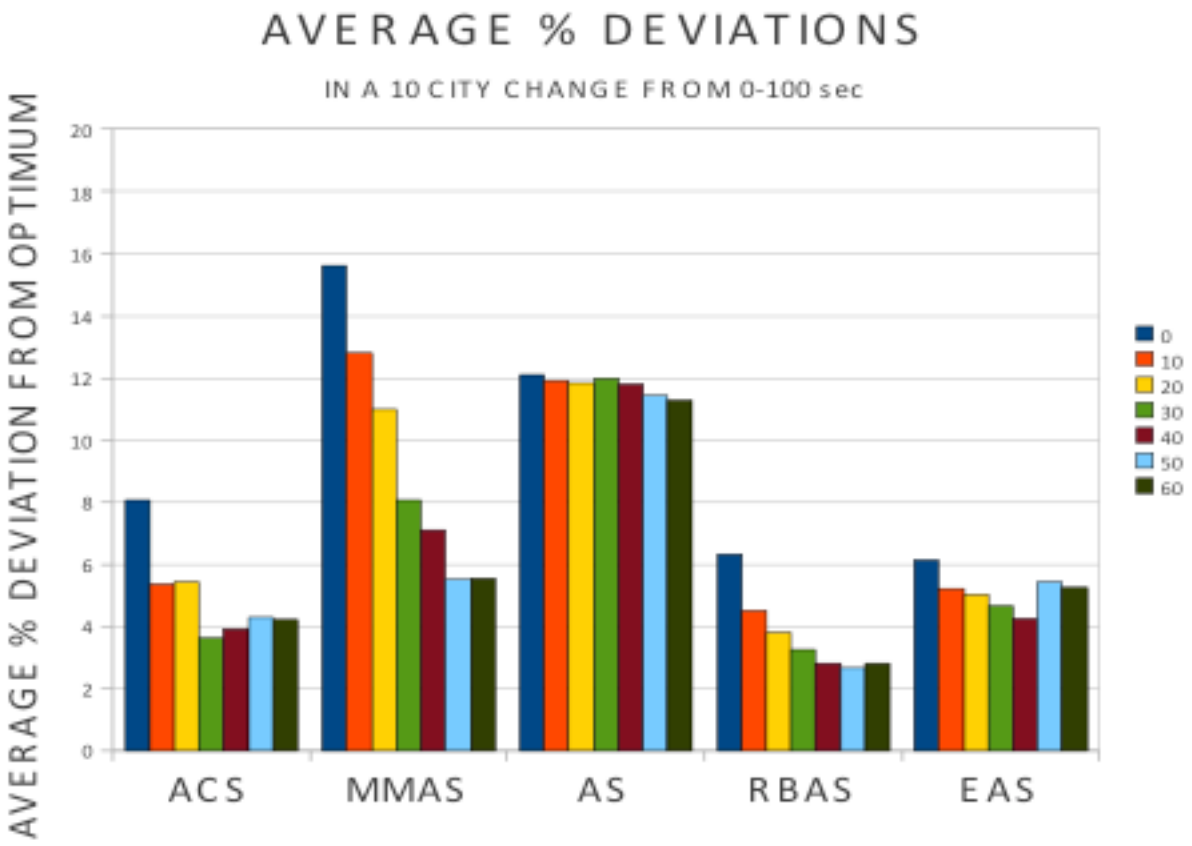


Chart 1

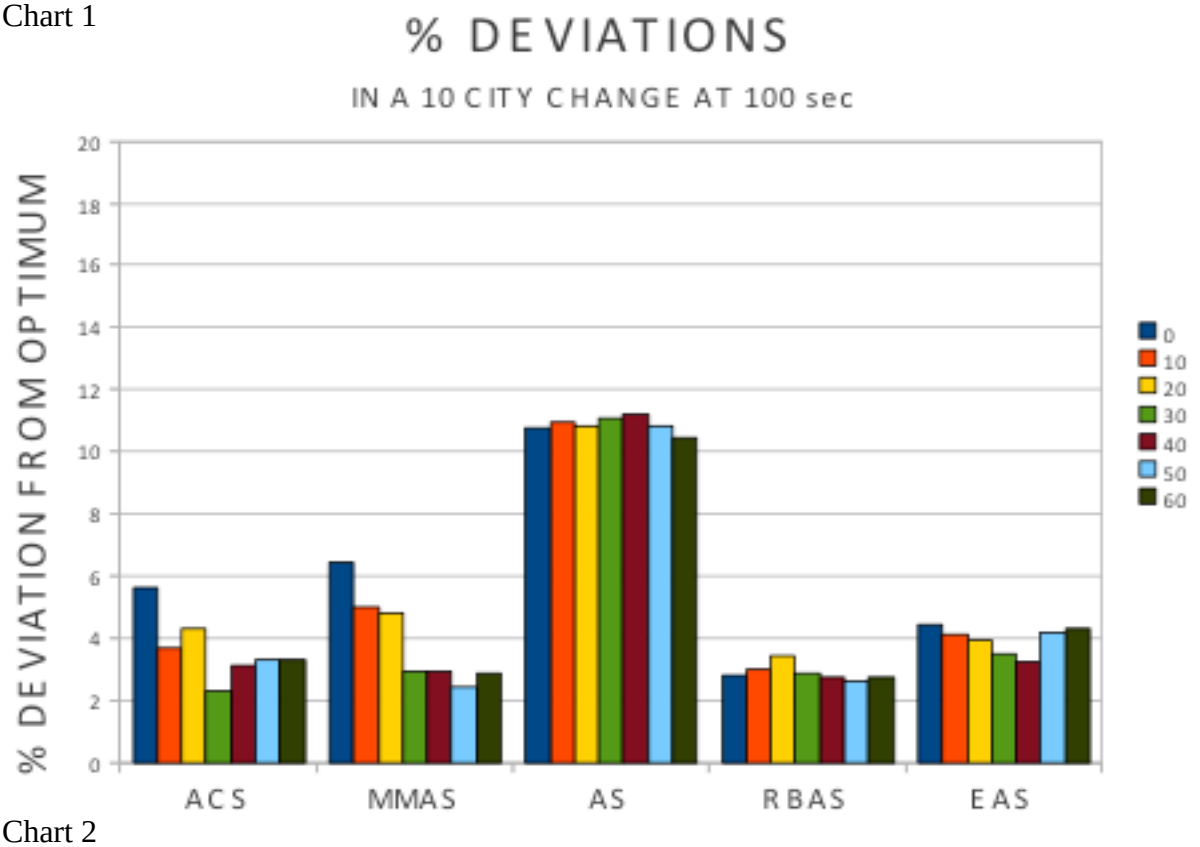
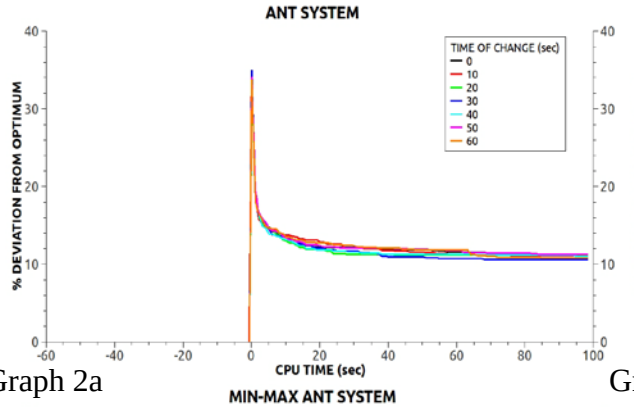


Chart 2

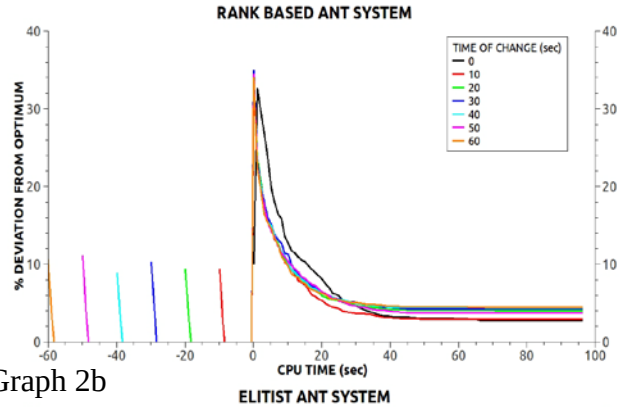
Chart 1 illustrates the differences between the algorithm's short-term effects. The Rank-Based Ant system had the best short-term results, as it adapts very quickly. The Ant Colony System had similar results to the Elitist Ant System. The Min Max ant system takes a while to converge, and thus did not have very impressive short-term results. What can be gathered from Chart 2 is that although Rank-Based Ant System produced good results very quickly, the Min-Max Ant System produced the best results of all algorithms with a preconditioned pheromone matrix made after 50 seconds with a small change (10 cities added).

4.2 Experiment 2

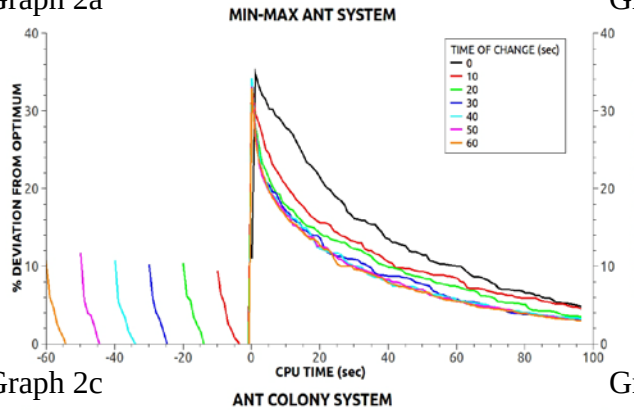
Since these algorithms cannot simply be compared at one change magnitude, an experiment similar to Experiment 1 was conducted. 50 cities were removed from the dataset and reintroduced at 7 time intervals.



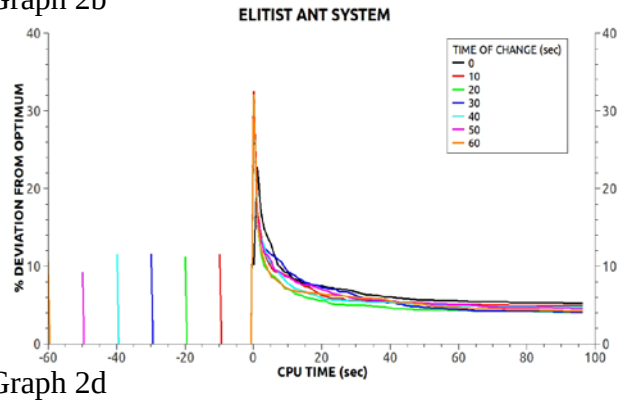
Graph 2a



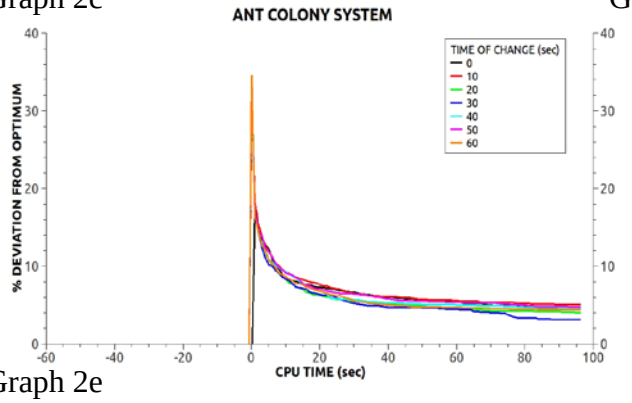
Graph 2b



Graph 2c



Graph 2d



Graph 2e

Graphs 2a-e show that algorithms running on a preconditioned pheromone matrix with less data in it than in the case where 10 cities are removed take longer to converge. In the short term, running the optimizations with a preconditioned pheromone matrix is still more effective than resetting the pheromone matrix.

To show the scalability of change magnitudes of the algorithms, similar charts to Charts 1 and 2 are shown below with data from Experiment 2.

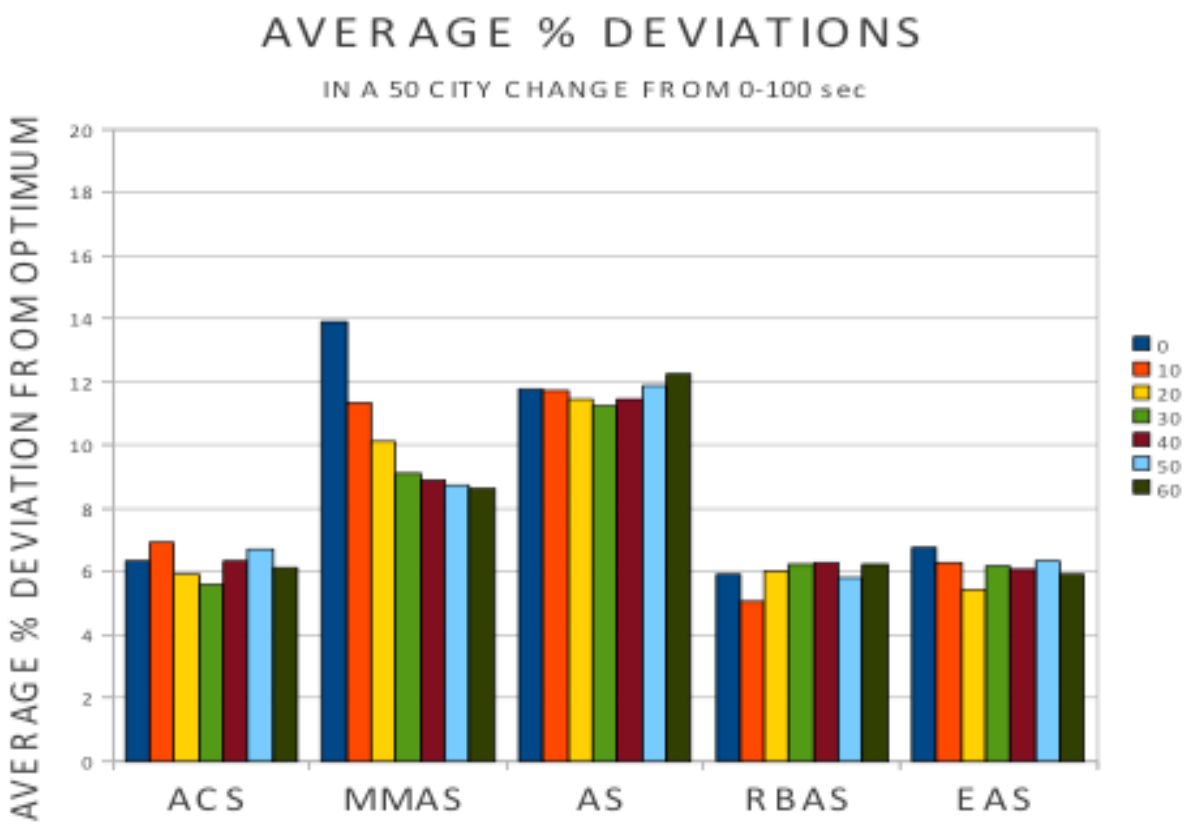


Chart 3

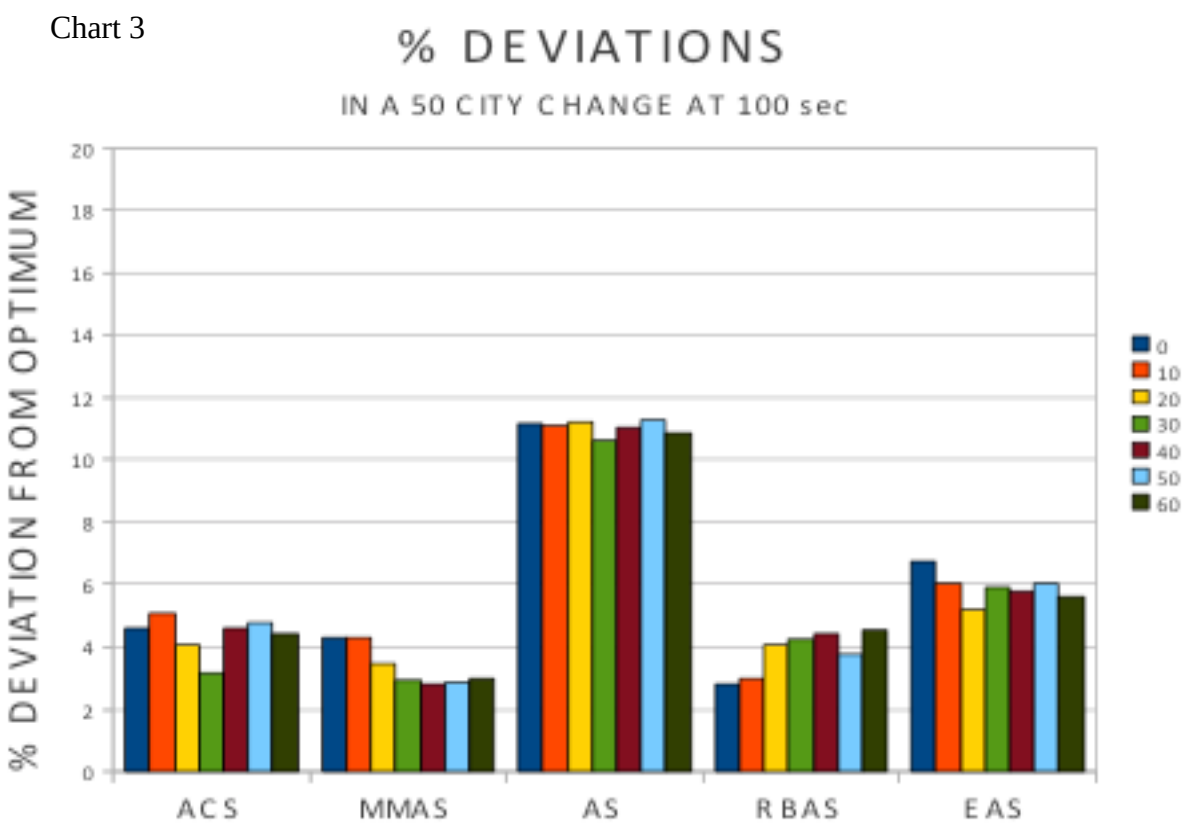
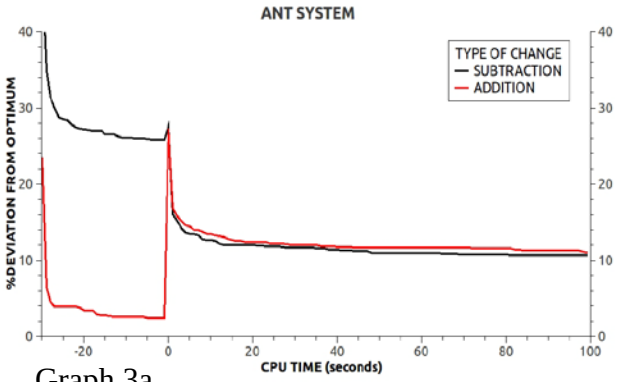


Chart 4

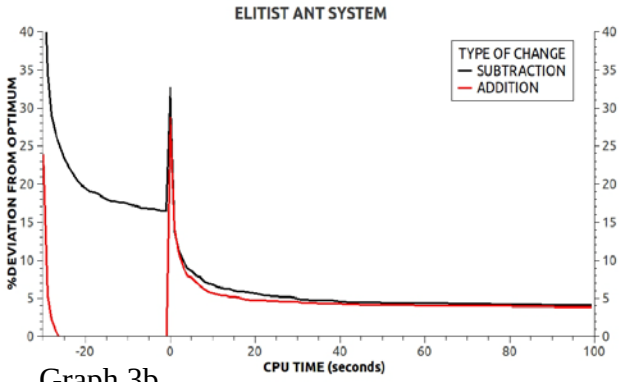
According to the Charts 3 and 4, the Min-Max method produced the best result at 100 seconds with a pheromone matrix preconditioned for 40 seconds, but this result was almost equal to that of Rank-Based Ant system. Unlike Rank-Based Ant System, however, Min-Max showed positive responses to preconditioning at both the 10 and 50 city changes. Rank-Based Ant System showed a negative response to preconditioning in a 50-city change. This means that Min-Max is the most scalable (in terms of change magnitude) algorithm, and Rank-Based is the fastest converging. Ant Colony System produced results between the two, and exhibited an interesting phenomenon. The results show that Ant Colony System had an optimal preconditioning time of thirty seconds in both cases, although the overall algorithm was not as reliable as Min Max Ant System.

4.3 Experiment 3

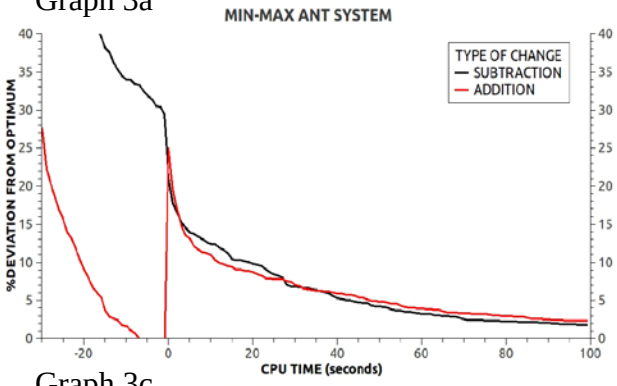
Addition is not the only type of change that can occur. Subtractions are a different kind of change. To test the behavior of subtractions relative to additions, an altered dataset of qa194 was created by shifting all the points in the dataset upward by 100 units. 30 of these points were added at the beginning of the simulation, and subtracted after 30 seconds of CPU time. This is compared with subtracting 30 and adding them back in after 30 seconds of CPU time. Again, the curves to the left of 0 seconds are inaccurate.



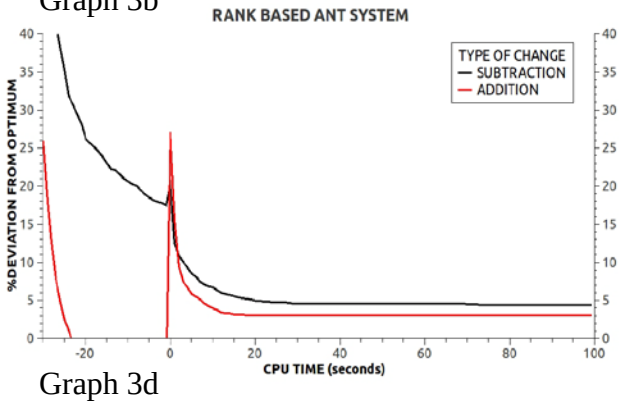
Graph 3a



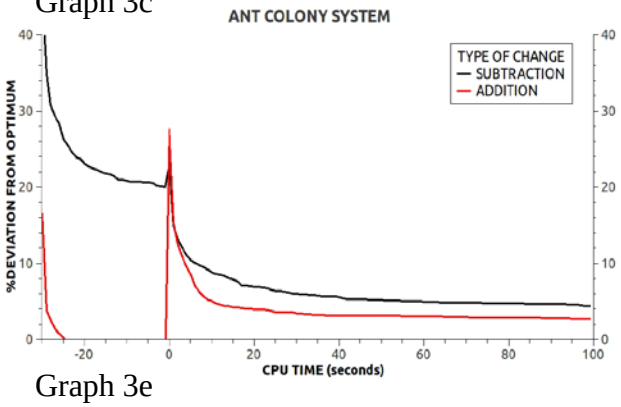
Graph 3b



Graph 3c



Graph 3d



Graph 3e

As can be seen above, the algorithms responded differently to additions and subtractions. The faster converging algorithms were more biased towards additions, likely because when cities are added, a complete tour is already in the matrix, and when cities are subtracted, the fragmented tour must be recreated in a short period of time, before stagnation occurs. Min-Max Ant System, Elitist Ant System, and Ant System were the least affected by differences between subtraction and addition.

5.0 Conclusions

The Min-Max Ant System was the most adaptable and reliable algorithm for DTSP, but all the algorithms have unique characteristics that may make them useful. Min-Max Ant System was the most adaptable and reliable algorithm because it produced the best quality solutions at 100 seconds in both magnitudes of change (10 and 50 city additions), and behaved in much the same way when subjected to an addition or subtraction. These traits could be very useful when selecting an algorithm to apply to DTSP. Min-Max Ant System probably performed the way that it did due to its exploratory nature (i.e. taking the time to explore a dataset before converging). The minimum and maximum pheromone values (Table 1) that combat early stagnation may have also helped it to utilize the preconditioned pheromone matrix.

The Rank-Based Ant System produced good solutions very quickly. This fast convergence is a very important advantage in DTSP applications, which are usually very time sensitive. The Rank-Based Ant System had trouble using stored pheromone data in 50-city changes, however, and behaved differently for additions and subtractions. This shows that this algorithm is not very adaptable.

Another interesting conclusion that can be drawn from our data is that there is an optimum time to introduce a change. In other words, there exists for ACOs an optimum level of preconditioning for a pheromone matrix for a change of some magnitude. This means that algorithms applied to DTSP may want to save states of their pheromone trails to use if a change is introduced.

6.0 Significant Original Achievement

The most significant original achievement that was made by Team 56 was the comparison of the five most studied ACOs applied to a dynamically changing TSP. We did not find this in the ACO literature and we believe it is important to academia and to industry. These comparisons may be of interest to those wishing to study DTSP, as it will help them to choose algorithms to use. Another contribution was the idea that there is an optimum preconditioning level on pheromone matrices.

7.0 Work Products

7.1 Experiment 1 Test Script

```
SYSTEMS
Ant_Colony_System
Min_Max_Ant_System
Ant_System
Rank_Based_Ant_System
Elitist_Ant_System
TEST_PARAMETERS
TRIALS 10
DISPLAY OFF PHEROMONES=ON TOUR=ON
REFRESH_RATE 1
METRICS
glob_best_ant_tour_length
iter_best_ant_tour_length
lambda_branching_factor
WRITE_FILE /home/peter/Desktop/comparative_qa194_vartime_-+_results10 WRITE
CONTINUOUS
PROBLEM
PROBLEM_FILE /home/peter/Desktop/qa194.tsp
CHANGE 0 -+[10,(0;10;60),i{TIME_LIMIT!100}]
```

7.2 Experiment 2 Test Script

```
SYSTEMS
Ant_Colony_System
Min_Max_Ant_System
Ant_System
Rank_Based_Ant_System
Elitist_Ant_System
TEST_PARAMETERS
TRIALS 10
DISPLAY OFF PHEROMONES=ON TOUR=ON
REFRESH_RATE 1
METRICS
glob_best_ant_tour_length
iter_best_ant_tour_length
lambda_branching_factor
WRITE_FILE /home/peter/Desktop/comparative_qa194_vartime_-+_results50 WRITE
CONTINUOUS
PROBLEM
PROBLEM_FILE /home/peter/Desktop/qa194.tsp
CHANGE 0 -+[50,(0;10;60),i{TIME_LIMIT!100}]
```

7.3 Experiment 3 Test Scripts

ADDITION:

SYSTEMS
Ant_Colony_System
Min_Max_Ant_System
Ant_System
Rank_Based_Ant_System
Elitist_Ant_System
TEST_PARAMETERS
TRIALS 10
DISPLAY OFF
REFRESH_RATE 1
METRICS
glob_best_ant_tour_length
iter_best_ant_tour_length
lambda_branching_factor
WRITE_FILE /home/peter/Desktop/qa194_-++-_-+_results30 WRITE CONTINUOUS
PROBLEM
PROBLEM_FILE /home/peter/Desktop/qa194.tsp
CHANGE 0 -+[30,30,i{TIME_LIMIT!100}]
SUBTRACTION:
SYSTEMS
Ant_Colony_System
Min_Max_Ant_System
Ant_System
Rank_Based_Ant_System
Elitist_Ant_System
TEST_PARAMETERS
TRIALS 10
DISPLAY OFF
REFRESH_RATE 1
METRICS
glob_best_ant_tour_length
iter_best_ant_tour_length
lambda_branching_factor
WRITE_FILE /home/peter/Desktop/qa194_-++-_+-_results30 WRITE CONTINUOUS
PROBLEM
PROBLEM_FILE /home/peter/Desktop/qa194.tsp
CHANGE 0 +-[/home/peter/Desktop/qa194prime.tsp,30,30,i{TIME_LIMIT!100}]

7.4 Code

```
##############################################
#####                                   #####
#####       ANT COLONY OPTIMIZATION     #####
#####               v 7-8               #####
#####                                   #####
#####  VARIOUS AUTHORS DESCRIBED BELOW   #####
#####                                   #####
##############################################


from random import randint,sample
from random import uniform as randfloat
from time import time,clock,asctime
from itertools import izip
from math import log10 as log
import multiprocessing
import sys
import pygame

##############################################
#####                                   #####
#####          THE OPTIMIZATIONS        #####
#####                                   #####
#####         AUTHOR: PETER AHRENS      #####
#####                                   #####
#####        METHODS DESCRIBED IN:      #####
#####       Ant Colony Optimization     #####
#####                  BY               #####
#####  Marco Dorigo AND Thomas Stutzle  #####
#####                2004               #####
#####                                   #####
##############################################

class Ant_Optimization():# a class with all of the attributes for other algorithms. Defines slots
to run faster
  __slots__ = ("problem", "city_coords", "city_names", "number_cities", "distances",
"neighbors_on", "number_neighbors", "nearest_neighbors", "heuristics",
"pheromones","probabilities", "ants", "beta", "rho", "Q", "number_ants", "initial_pheromone",
"glob_best_ant", "glob_best_ant_tour_length", "iter_best_ant_tour_length", "iter_best_ant", "e",
"w", "a", "iterglob", "global_best_correction", "zeta", "lambda_branching_factor",
"pheromone_limit_stagnation","probability_entropy_stagnation")
  def __init__(self):
    self.problem = ""
    self.city_coords = []
    self.city_names = []
    self.number_cities = 0
    self.distances = []
    self.neighbors_on = True
    self.number_neighbors = 15
    self.nearest_neighbors = []
    self.heuristics = []
    self.pheromones = []
    self.probabilities = []
    self.ants = []

    #self.alpha = 1, is always 1
    self.beta = 0
    self.rho = 0
    self.Q = 0
    self.number_ants = 0
    self.initial_pheromone = 0

    self.glob_best_ant = None
    self.iter_best_ant = None

class Ant_System(Ant_Optimization):
  def __init__(self):
    Ant_Optimization.__init__(self)

  def initialize_parameters(self):
    #self.alpha = 1, is always 1
    self.beta = 3.0
    self.rho = 0.5
    self.number_ants = self.number_cities
    self.compute_initial_pheromone_level()
```

```
  def compute_initial_pheromone_level(self):
    self.initial_pheromone = self.number_ants/nearest_neighbor_tour_length(self)

  def update_pheromone_trails(self):
    evaporate(self)
    for a in self.ants:
      deposit_pheromone(a,1,self)

class Elitist_Ant_System(Ant_Optimization):
  def __init__(self):
    Ant_Optimization.__init__(self)
    self.e = 0

  def initialize_parameters(self):
    #self.alpha = 1, is always 1
    self.beta = 3.0
    self.rho = 0.5
    self.number_ants = self.number_cities
    self.e = self.number_ants
    self.compute_initial_pheromone_level()

  def compute_initial_pheromone_level(self):
    self.initial_pheromone = (self.e + self.number_ants)/(self.rho *
nearest_neighbor_tour_length(self))

  def update_pheromone_trails(self):
    evaporate(self)
    for a in self.ants:
      deposit_pheromone(a,1,self)
    deposit_pheromone(self.glob_best_ant,self.e,self)

class Rank_Based_Ant_System(Ant_Optimization):
  def __init__(self):
    Ant_Optimization.__init__(self)
    self.w = 0

  def initialize_parameters(self):
    #self.alpha = 1, is always 1
    self.beta = 3.0
    self.rho = 0.1
    self.w = 6
    self.number_ants = self.number_cities
    self.compute_initial_pheromone_level()

  def compute_initial_pheromone_level(self):
    self.initial_pheromone = 0.5*(self.w)*(self.w+1)/(self.rho *
nearest_neighbor_tour_length(self))

  def update_pheromone_trails(self):
    evaporate(self)
    w = self.w
    for r,a in enumerate(self.ants):
      deposit_pheromone(a,max([0,(w-(r + 1))]),self)
    deposit_pheromone(self.glob_best_ant,w,self)

class Min_Max_Ant_System(Ant_Optimization):
  def __init__(self):
    Ant_Optimization.__init__(self)
    self.a = 0
    self.iterglob = 0
    self.global_best_correction = 0
    self.glob_best_ant = None

  def initialize_parameters(self):
    #self.alpha = 1, is always 1
    self.beta = 3.0
    self.rho = 0.02
    rho_dec = (1.0-self.rho)
    avg = (self.number_cities)/2.0
    self.a = (1 - rho_dec)/((avg-1.0) * rho_dec)
    if self.a > 1:
      self.a = 1.0
    self.global_best_correction = 0.90
    self.iterglob = (200.0 / (max(self.number_cities,200.0)))*self.global_best_correction
    self.number_ants = self.number_cities
    self.compute_initial_pheromone_level()

  def compute_initial_pheromone_level(self):
    self.pheromone_max = 1.0/(self.rho * nearest_neighbor_tour_length(self))
    self.pheromone_min = self.pheromone_max * self.a
```

```
    self.initial_pheromone = self.pheromone_max

  def update_pheromone_trails(self):
    evaporate(self)
    r = randfloat(0,1)
    if r < self.iterglob:
      deposit_pheromone(self.iter_best_ant,1,self)
    else:
      deposit_pheromone(self.glob_best_ant,1,self)
    self.update_pheromone_limits()

  def update_pheromone_limits(self):
    number_cities = self.number_cities
    self.pheromone_max = 1.0/(self.rho * self.glob_best_ant[1])
    self.pheromone_min = self.pheromone_max * self.a
    pheromone_max = self.pheromone_max
    pheromone_min = self.pheromone_min
    pheromones = self.pheromones
    for i in xrange(0,number_cities):
      for j in xrange(i, number_cities):
        pheromone = pheromones[i][j]
        if pheromone > pheromone_max:
          pheromones[i][j] = pheromone_max
          pheromones[j][i] = pheromone_max
        if pheromone < pheromone_min:
          pheromones[i][j] = pheromone_min
          pheromones[j][i] = pheromone_min

class Ant_Colony_System(Ant_Optimization):
  def __init__(self):
    Ant_Optimization.__init__(self)
    self.zeta = 0

  def initialize_parameters(self):
    #self.alpha = 1, is always 1
    self.beta = 3.0
    self.rho = 0.1
    self.Q = 0.9
    self.zeta = 0.1
    self.number_ants = 10
    self.compute_initial_pheromone_level()

  def compute_initial_pheromone_level(self):
    self.initial_pheromone =1.0/(self.number_cities*nearest_neighbor_tour_length(self))

  def update_pheromone_trails(self):
    evaporate_on_tour(self.glob_best_ant,self.rho, self)
    deposit_pheromone(self.glob_best_ant,self.rho, self)
    pheromone_to_add = self.initial_pheromone * self.zeta
    for a in self.ants:
      evaporate_on_tour(a,self.zeta, self)
      add_pheromone(a,pheromone_to_add, self)
    #reinitialization test

class Stinky_Ant_System(Ant_Optimization):
  def __init__(self):
    Ant_Optimization.__init__(self)
    self.w = 0
    self.v = 0
    self.stink = 0

  def initialize_parameters(self):
    #self.alpha = 1, is always 1
    self.beta = 3.0
    self.rho = 0.1
    self.w = 6
    self.v = 4
    self.stink = 0.3
    self.number_ants = self.number_cities
    self.compute_initial_pheromone_level()

  def compute_initial_pheromone_level(self):
    self.initial_pheromone = 0.5*(self.w)*(self.w+1)/(self.rho *
nearest_neighbor_tour_length(self))

  def update_pheromone_trails(self):
    evaporate(self)
    w = self.w
    v = self.v
    s = self.stink
    for r,a in enumerate(self.ants):
```

```
      deposit_pheromone(a,max([0,(w-(r + 1))]),self)
    for r,a in enumerate(reversed(self.ants)):
      evaporate_on_tour(a,max([0,s*(v-(r))/v]),self)
    deposit_pheromone(self.glob_best_ant,w,self)

##############################################
#####                                  #####
#####           DAEMON METHODS         #####
#####                                  #####
#####         AUTHOR: PETER AHRENS     #####
#####                                  #####
#####        METHODS DESCRIBED IN:     #####
#####       Ant Colony Optimization    #####
#####                  BY              #####
#####  Marco Dorigo AND Thomas Stutzle #####
#####                2004              #####
#####                                  #####
##############################################

def initialize_data(system,TSPLIB): #Initializes data for an algorithm.
  parse(TSPLIB,system)
  initialize_data_structures(system)
  compute_distances(system)
  compute_nearest_neighbors(system)
  compute_heuristics(system)
  system.initialize_parameters()
  initialize_pheromones(system)
  compute_probabilities(system)
  initialize_ants(system)
  initialize_bests(system)

def parse(TSPLIB,system): #Parses a TSPLIB file for the algorithm. Original method by PETER
AHRENS.
  city_coords = []
  city_names = []
  problem = ""
  number_cities = 0
  f = open(TSPLIB)
  coord_mode = False
  for line in f:
    content = line.split()
    if len(content) >= 1:
      if content[0] == "NAME:":
        problem = content[1]
    if (len(content) == 1):
      if content[0] == "EOF":
        coord_mode = False
    if (coord_mode and (len(content) == 3)):
      city_names.append(content[0])
      x = float(content[1])
      y = float(content[2])
      city_coords = city_coords + [(x,y)]
    if (len(content) == 1):
      if content[0] == "NODE_COORD_SECTION":
        coord_mode = True
  f.close()
  system.number_cities = len(city_coords)
  system.city_coords = city_coords
  system.city_names = city_names
  system.problem = problem

def make_matrix(size,fill): #Returns a square matrix of size size. Original method by PETER
AHRENS.
  m = []
  for x in range(0, size):
    m.append([fill]*size)
  return m

def initialize_data_structures(system):
  number_cities = system.number_cities
  system.distances = make_matrix(number_cities,0)
  system.heuristics = make_matrix(number_cities,0)
  system.probabilities = make_matrix(number_cities,0)

def compute_distances(system): #Computes distances from a list of points. Stores symmetrically.
  distances = system.distances
  city_coords = system.city_coords
  number_cities = system.number_cities
  for i, coords_I in enumerate(city_coords):
    for j in xrange(i+1,number_cities):
      coords_J = city_coords[j]
```

```
      dx = coords_I[0] - coords_J[0]
      dy = coords_I[1] - coords_J[1]
      distance = ((dx**2.0)+(dy**2.0))**0.5
      distances[i][j] = distance
      distances[j][i] = distance

def compute_nearest_neighbors(system): #Creates a nearest neighbor list for every city. Speeds up
program.
  nearest_neighbors = []
  number_neighbors = system.number_neighbors
  distances = system.distances
  number_cities = system.number_cities
  if system.neighbors_on:
    nearest_neighbors = []
    for i in xrange(0,number_cities):
      neighbors = range(0,number_cities)
      neighbors.remove(i)
      neighbors.sort(key = lambda j:distances[i][j])
      nearest_neighbors.append(neighbors[0:number_neighbors])
  system.nearest_neighbors = nearest_neighbors

def compute_heuristics(system): #Computes heuristics. Stores symmetrically.
  heuristics = system.heuristics
  distances = system.distances
  number_cities = system.number_cities
  for i in xrange(0,number_cities):
    for j in xrange(i+1,number_cities):
      distance = distances[i][j]
      if distance == 0:
        distance = 0.000000000000001
      heuristic = 1.0/distance
      heuristics[i][j] = heuristic
      heuristics[j][i] = heuristic

def initialize_pheromones(system): #Creates and initializes the pheromone matrix to a certain
value.
  system.pheromones = make_matrix(system.number_cities,system.initial_pheromone)

def compute_probabilities(system): #Computes probabilities from the heristics and pheromones.
Stores symmetrically.
  probabilities = system.probabilities
  heuristics = system.heuristics
  pheromones = system.pheromones
  beta = system.beta
  number_cities = system.number_cities
  for i in xrange(0,number_cities):
    for j in xrange(i, number_cities):
      probabilities[j][i] = probabilities[i][j] = (pheromones[i][j])*(heuristics[i][j]**beta)

def initialize_ants(system): #Creates all the ants for the system. (ant = [tour, tour_length])
  ants = []
  for i in xrange(0,system.number_ants):
    ants.append(([],0))
  system.ants = ants

def initialize_bests(system): #Initializes the best ants.
  system.glob_best_ant = ([],999999999999999999)
  system.iter_best_ant = ([],999999999999999999)

def evaporate(system): #Evaporates globally on all routes.
  pheromones = system.pheromones
  number_cities = system.number_cities
  rho = system.rho
  for i in xrange(0,number_cities):
    for j in xrange(i, number_cities):
      pheromone = pheromones[i][j]*(1.0-rho)
      pheromones[i][j] = pheromone
      pheromones[j][i] = pheromone

def reinitialize_pheromone_trails(system,amount): #Reinitializes the pheromone trails to some
value.
  pheromones = system.pheromones
  number_cities = system.number_cities
  for i in xrange(0,number_cities):
    for j in xrange(i, number_cities):
      pheromones[i][j] = pheromones[j][i] = amount

def update_bests(system): #Updates the best so far ants, etc.
  system.ants.sort(key = lambda a:a[1])
  if system.ants[0][1] < system.glob_best_ant[1]:
    system.glob_best_ant = (list(system.ants[0][0]),float(system.ants[0][1]))
```

```
  system.iter_best_ant = (list(system.ants[0][0]),float(system.ants[0][1]))

def local_search(): #This is where a local search method would go.
  pass

##############################################
#####                                    #####
#####            CHANGE METHODS          #####
#####                                    #####
#####         AUTHOR: PETER AHRENS       #####
#####                                    #####
##############################################

def add_city(system,xcoord,ycoord,name): #Adds a node to all applicable matricies. Saves
pheromone data. Stores symmetrically.
  add_node(system.pheromones,average_pheromone(system))
  system.number_cities += 1
  system.city_coords.append((xcoord,ycoord))
  system.city_names.append(name)
  add_node(system.distances,0)
  compute_distances(system)
  add_node(system.heuristics,0)
  compute_heuristics(system)
  compute_nearest_neighbors(system)
  add_node(system.probabilities,0)
  compute_probabilities(system)

def add_node(matrix,value): #Adds a node to a matrix. Stores symmetrically.
  for i in xrange(len(matrix)):
    matrix[i].append(value)
  matrix.append([value for j in xrange(len(matrix))] + [0])

def remove_node(matrix,j): #Removes a node from a matrix. Stores symmetrically.
  for i in xrange(len(matrix)):
    del(matrix[i][j])
  del matrix[j]

def remove_city(system,city_name): #Removes a node from all applicable matricies. Saves pheromone
data. Stores symmetrically.
  system.number_cities -= 1
  city_index = system.city_names.index(city_name)
  system.city_names.remove(city_name)
  del system.city_coords[city_index]
  remove_node(system.distances,city_index)
  compute_nearest_neighbors(system)
  remove_node(system.heuristics,city_index)
  remove_node(system.pheromones,city_index)
  remove_node(system.probabilities,city_index)

##############################################
#####                                    #####
#####              ANT METHODS           #####
#####                                    #####
#####         AUTHOR: PETER AHRENS       #####
#####                                    #####
#####        METHODS DESCRIBED IN:       #####
#####       Ant Colony Optimization      #####
#####                  BY                #####
#####  Marco Dorigo AND Thomas Stutzle   #####
#####                2004                #####
#####                                    #####
##############################################

def construct_tour(system): #Constructs a tour. Returns an ant. (ant = [tour, tour_length])
  probabilities = system.probabilities
  nearest_neighbors = system.nearest_neighbors
  distances = system.distances
  number_cities = system.number_cities
  neighbors_on = system.neighbors_on
  Q = system.Q
  #Construct the tour.
  visited = [False for i in xrange(0,number_cities)]
  tour = []
  tour.append(randint(0,number_cities-1))
  visited[tour[0]] = True
  if neighbors_on:
    for k in xrange(0,number_cities-1):
      q = randfloat(0,1)
      i = tour[k]
      selection_probabilities = []
      sum_probabilities = 0
```

```
      for j in nearest_neighbors[i]:
        if (not visited[j]):
          probability = probabilities[i][j]
          selection_probabilities.append(probability)
          sum_probabilities = sum_probabilities + probability

        else:
          selection_probabilities.append(0)
      if sum_probabilities == 0:
        best_probability = 0
        for candidate_city, probability in enumerate(probabilities[i]):
          if not visited[candidate_city]:
            if probability > best_probability:
              best_probability = probability
              j = candidate_city
        tour.append(j)
        visited[j] = True
      elif q <= Q:
        best_probability = 0
        for neighbor_index, probability in enumerate(selection_probabilities):
          if not visited[nearest_neighbors[i][neighbor_index]]:
            if probability > best_probability:
              best_probability = probability
              j = nearest_neighbors[i][neighbor_index]
        tour.append(j)
        visited[j] = True
      else:
        r = randfloat(0,sum_probabilities)
        roulette = 0
        for neighbor_index, probability in enumerate(selection_probabilities):
          roulette = roulette + probability
          if roulette > r:
            j = nearest_neighbors[i][neighbor_index]
            tour.append(j)
            visited[j] = True
            break
  else:
    for k in range(0,number_cities-1):
      i = tour[k]
      selection_probabilities = []
      sum_probabilities = 0
      for j, probability in enumerate(probabilities[i]):
        if (not visited[j]):
          selection_probabilities.append(probability)
          sum_probabilities = sum_probabilities + probability
        else:
          selection_probabilities.append(0)
      if q <= Q:
        best_probability = 0
        for candidate_city, probability in enumerate(probabilities[i]):
          if not visited[candidate_city]:
            if probability > best_probability:
              best_probability = probability
              j = candidate_city
        tour.append(j)
        visited[j] = True
      else:
        r = randfloat(0,sum_probabilities)
        roulette = 0
        for j, probability in enumerate(selection_probabilities):
          roulette = roulette + probability
          if roulette > r:
            tour.append(j)
            visited[j] = True
            break
  tour.append(tour[0])
  #Compute the tour length.
  tour_length = 0
  i = tour[0]
  for j in tour[1:]:
    tour_length = tour_length + distances[i][j]
    i = j
  return (tour,tour_length)

def
parallel_construct_tour((probabilities,nearest_neighbors,distances,number_cities,neighbors_on,Q))
: #Is the same as construct tour, but takes inputs directly from the calling process. This
appeared to be faster than using shared memory, because so many lookups are performed on the
matricies.
  #probabilities = system.probabilities
  #nearest_neighbors = system.nearest_neighbors
```

```
#distances = system.distances
#number_cities = system.number_cities
#neighbors_on = system.neighbors_on
#Q = system.Q
#Construct the tour.
visited = [False for i in xrange(0,number_cities)]
tour = []
tour.append(randint(0,number_cities-1))
visited[tour[0]] = True
if neighbors_on:
  for k in xrange(0,number_cities-1):
    q = randfloat(0,1)
    i = tour[k]
    selection_probabilities = []
    sum_probabilities = 0
    for j in nearest_neighbors[i]:
      if (not visited[j]):
        probability = probabilities[i][j]
        selection_probabilities.append(probability)
        sum_probabilities = sum_probabilities + probability
      else:
        selection_probabilities.append(0)
    if sum_probabilities == 0:
      best_probability = 0
      for candidate_city, probability in enumerate(probabilities[i]):
        if not visited[candidate_city]:
          if probability > best_probability:
            best_probability = probability
            j = candidate_city
      tour.append(j)
      visited[j] = True
    elif q <= Q:
      best_probability = 0
      for neighbor_index, probability in enumerate(selection_probabilities):
        if not visited[nearest_neighbors[i][neighbor_index]]:
          if probability > best_probability:
            best_probability = probability
            j = nearest_neighbors[i][neighbor_index]
      tour.append(j)
      visited[j] = True
    else:
      r = randfloat(0,sum_probabilities)
      roulette = 0
      for neighbor_index, probability in enumerate(selection_probabilities):
        roulette = roulette + probability
        if roulette > r:
          j = nearest_neighbors[i][neighbor_index]
          tour.append(j)
          visited[j] = True
          break
else:
  for k in range(0,number_cities-1):
    i = tour[k]
    selection_probabilities = []
    sum_probabilities = 0
    for j, probability in enumerate(probabilities[i]):
      if (not visited[j]):
        selection_probabilities.append(probability)
        sum_probabilities = sum_probabilities + probability
      else:
        selection_probabilities.append(0)
    if q <= Q:
      best_probability = 0
      for candidate_city, probability in enumerate(probabilities[i]):
        if not visited[candidate_city]:
          if probability > best_probability:
            best_probability = probability
            j = candidate_city
      tour.append(j)
      visited[j] = True
    else:
      r = randfloat(0,sum_probabilities)
      roulette = 0
      for j, probability in enumerate(selection_probabilities):
        roulette = roulette + probability
        if roulette > r:
          tour.append(j)
          visited[j] = True
          break
tour.append(tour[0])
#Compute the tour length.
```

```
  tour_length = 0
  i = tour[0]
  for j in tour[1:]:
    tour_length = tour_length + distances[i][j]
    i = j
  return (tour,tour_length)

def compute_tour_length(tour,system): #Returns the tour length of a tour.
  distances = system.distances
  tour_length = 0
  i = tour[0]
  for j in tour[1:]:
    tour_length = tour_length + distances[i][j]
    i = j
  return tour_length

def deposit_pheromone(ant,amount,system): #Deposits pheromone on an ant's tour
  amount = float(amount)
  pheromones = system.pheromones
  pheromone_to_deposit = amount/ant[1]
  i = ant[0][0]
  for j in ant[0][1:]:
    pheromones[i][j] = pheromones[j][i] = pheromones[i][j] + (pheromone_to_deposit)
    i = j

def evaporate_on_tour(ant,amount,system): #Evaporates pheromone along an ant's tour.
  pheromones = system.pheromones
  i = ant[0][0]
  for j in ant[0][1:]:
    pheromones[i][j] = pheromones[j][i] = pheromones[i][j] * (1.0-amount)
    i = j

def add_pheromone(ant,amount,system): #Adds a given amount of pheromone on an ant's tour.
  pheromones = system.pheromones
  i = ant[0][0]
  for j in ant[0][1:]:
    pheromones[i][j] = pheromones[j][i] = pheromones[i][j] + amount
    i = j

###############################################
#####                                     #####
#####              STATISTICS             #####
#####                                     #####
#####         AUTHOR: PETER AHRENS        #####
#####                                     #####
#####        METHODS DESCRIBED IN:        #####
#####       Ant Colony Optimization       #####
#####                 BY                  #####
#####  Marco Dorigo AND Thomas Stutzle    #####
#####                2004                 #####
#####                                     #####
###############################################

def lambda_branching_factor(system,L): #Returns the lambda branching factor of an algorithm with
lambda = L.
  pheromones = system.pheromones
  number_cities = system.number_cities
  branches = 0
  for i in xrange(0,number_cities):
    max_pheromone = max(pheromones[i])
    min_pheromone = min(pheromones[i])
    branch = min_pheromone + (L * (max_pheromone - min_pheromone))
    for j in xrange(0, number_cities):
      pheromone = pheromones[i][j]
      if pheromone >= branch:
        branches += 1
  branching_factor = float(branches)/number_cities
  return branching_factor

def pheromone_limit_stagnation(system): #Returns the pheromone limit stagnation of an algorithm.
  pheromones = system.pheromones
  number_cities = system.number_cities
  max_pheromone = max([max(x) for x in pheromones])
  min_pheromone = min([min(x) for x in pheromones])
  total_stagnation = 0
  for i in xrange(0,number_cities):
    for j in xrange(i+1,number_cities):
      pheromone = pheromones[i][j]
      total_stagnation += min(max_pheromone - pheromone,pheromone - min_pheromone)
  stagnation = 2*total_stagnation/(number_cities*(number_cities - 1))
  return stagnation
```

```
def probability_entropy_stagnation(system): #Returns the probability entropy stagnation of an
algorithm.
  probabilities = system.probabilities
  number_cities = system.number_cities
  total_entropy = 0
  for i in xrange(0,number_cities):
    total_probability = sum(probabilities[i])
    for j in xrange(i+1,number_cities):
      probability = probabilities[i][j]/total_probability
      try:
        total_entropy -= probability*log(probability)
      except ValueError:
        pass
  stagnation = total_entropy/number_cities
  return stagnation

def nearest_neighbor_tour_length(system): #Returns the nearest-neighbor tour length for an
algorithm.
  number_cities = system.number_cities
  distances = system.distances
  not_visited = range(0,number_cities)
  start_city = not_visited.pop(randint(0,number_cities-1))
  i = start_city
  tour_length = 0
  k = 0
  while (k < number_cities-1):
    not_visited.sort(key = lambda j:distances[i][j])
    j = not_visited[0]
    tour_length = tour_length + distances[i][j]
    del not_visited[0]
    i = j
    k = k + 1
  tour_length = tour_length + distances[i][start_city]
  return tour_length

def average_pheromone(system): #Returns the average pheromone level for an anlgorithm.
  pheromones = system.pheromones
  number_cities = system.number_cities
  total_pheromone = 0
  for i in xrange(0,number_cities):
    for j in xrange(i, number_cities):
      total_pheromone += pheromones[i][j]
  average_pheromone = total_pheromone / (number_cities * (number_cities - 1) / 2.0)
  return average_pheromone

##############################################
#####                                    #####
#####               AntFarm              #####
#####            TEST HARNESS            #####
#####                                    #####
#####         AUTHOR: PETER AHRENS       #####
#####                                    #####
##############################################

def input_float(title): #Returns a float from the user through the command line.
  print title
  print "x EXIT"
  while True:
    num = raw_input(": ")
    if num == "x":
      sys.exit(0)
    try:
      num = float(num)
      break
    except ValueError:
      pass
  return num

def input_int(title): #Returns an int from the user through the command line.
  print title
  print "x EXIT"
  while True:
    num = raw_input(": ")
    if num == "x":
      sys.exit(0)
    try:
      num = int(num)
      break
    except ValueError:
      pass
```

```
  return num

def menu_select(title,items): #Returns a selection from the user through the command line.
  print title
  for n, item in enumerate(items):
    print "%s %s" % (str(n),item)
  print "x EXIT"
  while True:
    selection = raw_input(": ")
    if selection == "x":
      sys.exit(0)
    if selection.isdigit():
      selection = int(selection)
      if selection <= (len(items) - 1):
        break
  return (items[selection])

def file_select(title):  #Returns a file from the user through the command line.
  print title
  print "x EXIT"
  while True:
    selection = raw_input(": ")
    if selection == "x":
      sys.exit(0)
    try:
      f = open(selection)
      f.close()
      break
    except IOError:
      pass
  return selection

class stopwatch(): #A stopwatch to time the simulations.
  def __init__(self):
    self.start_cpu = 0
    self.start_real = 0
    self.residual_cpu = 0
    self.residual_real = 0
    self.on = False
  def start(self): #Starts the stopwatch.
    self.start_cpu = clock()
    self.start_real = time()
    self.on = True
  def stop(self): #Stops the stopwatch.
    if self.on:
      self.residual_cpu = self.residual_cpu + clock() - self.start_cpu
      self.residual_real = self.residual_real + time() - self.start_real
      self.on = False
  def reset(self): #Resets the times to 0. Stops the stopwatch.
    self.residual_cpu = 0
    self.residual_real = 0
    self.on = False
  def cpu_time(self): #Returns the CPU time on the stopwatch.
    if self.on:
      return (self.residual_cpu + clock() - self.start_cpu)
    else:
      return (self.residual_cpu)
  def real_time(self): #Returns the real time on the stopwatch.
    if self.on:
      return (self.residual_real + time() - self.start_real)
    else:
      return (self.residual_real)

class stop_criteria(): #An object to store stoping criteria data and evaluate when to stop.
  def __init__(self):
    self.stopping = False
    self.time_limit_stop_criteria = False
    self.time_limit = 0
    self.repeated_result_stop_criteria = False
    self.repeated_result_limit = 0
    self.previous_time = 0
    self.repeated_results = 0
    self.previous_result = 0

  def update_stop_criteria(self,system,time): #Checks whether to stop.
    if self.repeated_result_stop_criteria:
      glob_best = system.glob_best_ant[1]
      if self.previous_result == glob_best and self.previous_time != 0:
        self.repeated_results += (time - self.previous_time)
      else:
        self.repeated_results = 0
```

```
      self.previous_result = glob_best
      if self.repeated_results >= self.repeated_result_limit:
        self.stopping = True
    if self.time_limit_stop_criteria:
      if time > self.time_limit:
        self.stopping = True
    self.previous_time = time

  def reset(self): #Sets stopping to False. Resets the counters, but not the stored information.
    self.stopping = False
    self.repeated_results = 0
    self.previous_result = 0
    self.previous_time = 0
    self.time_limit_stop_criteria = False
    self.repeated_result_stop_criteria = False

  def repeated_result_stop_criteria_setup(self,repeated_result_limit): #Sets up repeated result
limit test values.
    self.repeated_result_limit = repeated_result_limit

  def time_limit_stop_criteria_setup(self,time_limit): #Sets up time limit test values.
    self.time_limit = time_limit

def expand(parameter,convert): #Returns an expanded (start,increment,stop).
  parameter = list(parameter)
  if convert == "f":
    parameter = [float(x) for x in parameter]
  if convert == "i":
    parameter = [int(x) for x in parameter]
  if len(parameter) == 1:
    expansion = parameter
  else:
    expansion = []
    i = parameter[0]
    while i <= parameter[2]:
      expansion.append(i)
      i = i + parameter[1]
  return expansion

def combine(parameters): #Returns all the combinations of values given a list of lists.
  combination = [[]]
  for x in parameters:
    d = []
    for a in combination:
      c = [a+[b] for b in x]
      d = d + c
    combination = d
  return combination

class test(): #The test harness.
  def __init__(self):
    self.input_p = None
    self.parameters = []
    self.parameter_names = []
    self.test_time = 0
    self.problem_file = ""
    self.change_times = []
    self.changes = []
    self.change_sequence_instances = []
    self.change_index = 0
    self.trials = 0
    self.metrics = []
    self.systems = []
    self.write_file = ""
    self.write_mode = ""
    self.writing = False
    self.write_result = False
    self.outputstring = ""
    self.display = False
    self.display_tour = True
    self.display_pheromones = False
    self.display_stagnation = False
    self.refresh_rate = 0
    self.system = None
    self.s = stop_criteria()
    self.initial_time_limit_stop_criteria = False
    self.initial_time_limit = 0
    self.initial_repeated_result_stop_criteria = False
    self.initial_repeated_result_limit = 0

  def setup(self): #Gathers and processes data neccesary to run.
```

```
    mode = menu_select("INPUT DATA FROM:",("USER","FILE"))
    test_script = "USER_INPUT"
    if mode == "USER":
      self.user_input()
    else:
      test_script = file_select("FILE TO READ:")
      self.parse_test_script(test_script)
    self.configure_change_sequence()
    for a, parameter in enumerate(self.parameters):
      self.parameters[a] = expand(parameter,"f")
    self.parameters = combine(self.parameters)
    self.writing = (self.write_mode == "APPEND" or self.write_mode == "WRITE")
    if self.writing:
      self.format_output(test_script)

  def get_pipe(self,input_p): #Gets the pipe (socket) to communicate with the display.
    self.input_p = input_p

  def user_input(self): #Reads in user input.
    self.parameters = []
    self.parameter_names = []

    self.test_time = 0
    self.problem_file = ""

    self.change_times = []
    self.changes = []
    self.trials = 0
    self.metrics = []
    self.systems = []
    self.write_file = ""
    self.write_mode = ""
    self.writing = False
    self.write_result = False
    self.outputstring = ""
    self.display = False
    self.display_tour = True
    self.display_pheromones = False
    self.display_stagnation = False
    self.refresh_rate = 0
    self.initial_time_limit_stop_criteria = False
    self.initial_time_limit = 0
    self.initial_repeated_result_stop_criteria = False
    self.initial_repeated_result_limit = 0
    self.problem_file = file_select("FILE TO OPTIMIZE:")
    while True:
      system = menu_select("ALGORITHMS TO USE:", ("Ant_System", "Elitist_Ant_System",
"Rank_Based_Ant_System", "Min_Max_Ant_System", "Ant_Colony_System", "Stinky_Ant_System", "That's
all"))
      if system == "DONE":
        break
      else:
        self.systems.append(system)
    self.refresh_rate = input_float("REFRESH RATE (seconds):")
    while True:
      stop_criteria = menu_select("STOP CRITERIA:", ("TIME_LIMIT", "REPEATED_RESULT_LIMIT",
"DONE"))
      if stop_criteria == "DONE":
        break
      else:
        if stop_criteria == "TIME_LIMIT":
          self.initial_time_limit = input_int("TIME_LIMIT (seconds):")
          self.s.time_limit_stop_criteria_setup(self.initial_time_limit)
          self.initial_time_limit_stop_criteria = True
        elif stop_criteria == "REPEATED_RESULT_LIMIT":
          self.initial_repeated_result_limit = input_int("REPEATED_RESULT_LIMIT (seconds):")
          self.s.repeated_result_stop_criteria_setup(self.initial_repeated_result_limit)
          self.initial_repeated_result_stop_criteria = True
    self.trials = input_int("TRIALS:")
    while True:
      metric = menu_select("WHAT TO MONITOR:", ("glob_best_ant_tour_length",
"iter_best_ant_tour_length", "lamda_branching_factor", "DONE"))
      if metric == "DONE":
        break
      else:
        self.metrics.append(metric)
    self.write_mode = menu_select("WRITE TO FILE?", ("WRITE", "APPEND", "OFF"))
    if self.write_mode == "WRITE" or self.write_mode == "APPEND":
      self.write_file = file_select("FILE TO WRITE TO:")
    self.display = ("ON" == menu_select("DISPLAY:", ("ON", "OFF")))
    if self.display:
```

```
    self.display_tour = ("ON" == menu_select("DISPLAY TOUR:", ("ON", "OFF")))
    self.display_pheromones = ("ON" == menu_select("DISPLAY PHEROMONES:", ("ON", "OFF")))
    self.display_stagnation = ("ON" == menu_select("DISPLAY STAGNATION:", ("ON", "OFF")))
def parse_tsplib(self,tsplib): #Reads in lists of cities from a TSPLIB format file.
  city_coords = []
  city_names = []
  f = open(tsplib)
  coord_mode = False
  for line in f:
    content = line.split()
    if len(content) >= 1:
      if content[0] == "NAME:":
        problem = content[1]
    if (len(content) == 1):
      if content[0] == "EOF":
        coord_mode = False
    if (coord_mode and (len(content) == 3)):
      city_names.append(content[0])
      x = float(content[1])
      y = float(content[2])
      city_coords = city_coords + [(x,y)]
    if (len(content) == 1):
      if content[0] == "NODE_COORD_SECTION":
        coord_mode = True
  f.close()
  return city_coords,city_names

def parse_test_script(self,test_script): #Reads in a test script.
  self.parameters = []
  self.parameter_names = []
  self.test_time = 0
  self.problem_file = ""
  self.change_times = []
  self.changes = []
  self.trials = 0
  self.metrics = []
  self.systems = []
  self.write_file = ""
  self.write_mode = ""
  self.writing = False
  self.write_result = False
  self.outputstring = ""
  self.display = False
  self.display_tour = True
  self.display_pheromones = False
  self.refresh_rate = 0
  self.initial_time_limit_stop_criteria = False
  self.initial_time_limit = 0
  self.initial_repeated_result_stop_criteria = False
  self.initial_repeated_result_limit = 0
  f = open(test_script)
  mode = None
  for line in f:
    content = line.split()
    if content == []:
      pass
    elif content[0] == "TEST_PARAMETERS":
      mode = "test_parameters"
    elif content[0] == "METRICS":
      mode = "metrics"
    elif content[0] == "SYSTEMS":

      mode = "systems"
    elif content[0] == "PROBLEM":
      mode = "problem"
    elif content[0] == "WRITE_FILE":
      self.write_file = content[1]
      self.write_mode = content[2]
      self.write_result = (content[3] != "CONTINUOUS")
    elif (mode == "test_parameters"):
      if content[0] == "TRIALS":
        self.trials = int(content[1])
      elif content[0] == "DISPLAY":
        self.display = (content[1] == "ON")
        if len(content) >= 3:
          for preference in content[2:]:
            [element,is_on] = preference.split("=")
            is_on = (is_on == "ON")
            if element == "TOUR":
              self.display_tour = is_on
            if element == "PHEROMONES":
```

```
              self.display_pheromones = is_on
        elif content[0] == "PARAMETER":
          parameter = content[2].lstrip("(").rstrip(")").split(";")
          minimum = float(parameter[0])
          increment = float(parameter[1])
          maximum = float(parameter[2])
          self.parameters.append((minimum,increment,maximum))
          self.parameter_names.append(content[1])
        elif content[0] == "REFRESH_RATE":
          self.refresh_rate = float(content[1])
        elif content[0] == "STOP_CRITERIA":
          if content[1] == "TIME_LIMIT":
            self.initial_time_limit = int(content[2])
            self.s.time_limit_stop_criteria_setup(self.initial_time_limit)
            self.initial_time_limit_stop_criteria = True
          elif content[1] == "REPEATED_RESULT_LIMIT":
            self.initial_repeated_result_limit = int(content[2])
            self.s.repeated_result_stop_criteria_setup(self.initial_repeated_result_limit)
            self.initial_repeated_result_stop_criteria = True
      elif (mode == "metrics"):
        self.metrics = self.metrics + [content[0]]
      elif (mode == "systems"):
        self.systems.append(content[0])
      elif (mode == "problem"):
        if content[0] == "PROBLEM_FILE":
          self.problem_file = content[1]
        elif content[0] == "CHANGE":
          self.changes.append(list(content[1:]))
    f.close()

  def configure_change_sequence(self): #Converts the changes specified in the test script into
instances of sequences of changes that are read into the change maker.
    #Create all instances of +- or -+ cities commands.
    change_sequence_instances = [list(self.changes)]
    change_names = []
    change_display = []
    for i,change in enumerate(self.changes):
      for command in change[1:]:
        if command[0:2] == "+-" or command[0:2] == "-+":
          commands_to_add = []
          old_command = command
          mode = command[0:2]
          command = command[2:].lstrip("[").rstrip("]").split(",")
          if mode == "+-":
            #Create +- commands.
            (coords,names) = self.parse_tsplib(command[0])
            random_city_indicies = sample(xrange(0,len(names)),len(names)-1)
            if command[1][0] == "(":
              numbers_of_cities_to_add =
expand(command[1].lstrip("(").rstrip(")").split(";"),"i")
            else:
              numbers_of_cities_to_add = [int(command[1])]
            for number_cities_to_add in numbers_of_cities_to_add:
              city_indicies = random_city_indicies[0:number_cities_to_add]
              additions = ["+[%s,%s,%s]" % (coords[x][0],coords[x][1],names[x]) for x in
city_indicies]
              subtractions = ["-[%s]" % (names[x]) for x in city_indicies]
              commands_to_add.append((additions,subtractions))
            restore_time = command[2]
            restore_commands = [x.replace("!",",").replace("{","[").replace("}","]") for x in
command[3:]]
          else:
            #Create -+ commands.
            (coords,names) = self.parse_tsplib(self.problem_file)
            random_city_indicies = sample(xrange(0,len(names)),len(names)-1)
            if command[0][0] == "(":
              numbers_of_cities_to_subtract =
expand(command[0].lstrip("(").rstrip(")").split(";"),"i")
            else:
              numbers_of_cities_to_subtract = [int(command[0])]
            commands_to_add = []
            for number_cities_to_subtract in numbers_of_cities_to_subtract:
              city_indicies = random_city_indicies[0:number_cities_to_subtract]
              subtractions = ["-[%s]" % (names[x]) for x in city_indicies]
              additions = ["+[%s,%s,%s]" % (coords[x][0], coords[x][1], names[x]) for x in
city_indicies]
              commands_to_add.append((subtractions,additions))
            restore_time = command[1]
            restore_commands = [x.replace("!",",").replace("{","[").replace("}","]") for x in
command[2:]]
          #Create an instance of the change for each command that needs to be added.
```

```
          new_change_sequence_instances = []
          for change_sequence_instance in change_sequence_instances:
            change_sequence_instance[i].remove(old_command)
            for command_to_add in commands_to_add:
              new_change = list(change_sequence_instance[i])
              new_change.extend(command_to_add[0])
              new_change_sequence_instance = list(change_sequence_instance)
              new_change_sequence_instance[i] = new_change
              extra_change = [restore_time] + command_to_add[1] + restore_commands
              new_change_sequence_instance.append(extra_change)
              new_change_sequence_instances.append(new_change_sequence_instance)
          change_sequence_instances = new_change_sequence_instances
      #Reformat each instance before expansion.
      for i, change_sequence_instance in enumerate(change_sequence_instances):
        new_change_sequence_instance = []
        for change in change_sequence_instance:
          new_change = [change[0]]
          for command in change[1:]:
            mode = command[0]
            command = list(mode)+command[1:].lstrip("[").rstrip("]").split(",")
            new_change.extend(command)
          new_change_sequence_instance.append(new_change)
        change_sequence_instances[i] = new_change_sequence_instance
      #Expand and combine each instance.
      new_change_sequence_instances = []
      for change_sequence_instance in change_sequence_instances:
        new_change_sequence_instance = []
        for change in change_sequence_instance:
          new_change = []
          for i,element in enumerate(change):
            if element[0] == "(":
              element = element.lstrip("(").rstrip(")").split(";")
              element = expand(element,"i")
            else:
              if i == 0:
                element = [int(element)]
              else:
                element = [element]
            new_change.append(element)
          new_change_sequence_instance.append(combine(new_change))
        new_change_sequence_instances.extend(combine(new_change_sequence_instance))
      change_sequence_instances = new_change_sequence_instances
      #Sort and reformat change sequences.
      for change_sequence_instance in change_sequence_instances:
        change_sequence_instance.sort(key = lambda x:x[0])
      new_change_sequence_instances = []
      for change_sequence_instance in change_sequence_instances:
        change_times = []
        changes = []
        for change in change_sequence_instance:
          change_times.append(change[0])
          changes.append(change[1:])
        new_change_sequence_instance = [change_times,changes]
        new_change_sequence_instances.append(new_change_sequence_instance)
      change_sequence_instances = new_change_sequence_instances
      self.change_sequence_instances = change_sequence_instances

    def format_output(self,test_script): #Sets up the output file with a heading. Creates the
  output string to put values in.
      if self.write_mode == "WRITE":
        f = open(self.write_file,"wt")
      else:
        f = open(self.write_file,"at")
      f.write("TEST_SCRIPT: " + test_script + "\n")
      f.write("PROBLEM_FILE: " + self.problem_file + "\n")
      f.write("DATE: " + asctime() + "\n")
      f.write("\n")
      titlestring = "SYSTEM,TRIAL,ITERATION,CHANGE_TIMES,CHANGE_SIZES,REAL_TIME,CPU_TIME"
      self.output_string = "%s,%s,%s,%s,%s,%s,%s"
      for parameter in self.parameter_names:
        titlestring = titlestring + "," + parameter
        self.output_string = self.output_string + ",%s"
      for metric in self.metrics:
        titlestring = titlestring + "," + metric
        self.output_string = self.output_string + ",%s"
      self.output_string = self.output_string + "\n"
      f.write(titlestring + "\n")
      f.close()

    def output(self, f, system_string, trial, iteration, change_sequence_instance, real_time,
  cpu_time, parameter_instance): #Manages the output of data to either the terminal or a file.
```

```
    system = self.system
    monitor = []
    #Only calculate neccesary statistics.
    for metric in self.metrics:
      if metric == "glob_best_ant_tour_length":
        system.glob_best_ant_tour_length = system.glob_best_ant[1]
      if metric == "iter_best_ant_tour_length":
        system.iter_best_ant_tour_length = system.iter_best_ant[1]
      if metric == "lambda_branching_factor":
        self.system.lambda_branching_factor = lambda_branching_factor(system,0.05)
      if metric == "pheromone_limit_stagnation":
        self.system.pheromone_limit_stagnation = pheromone_limit_stagnation(system)
      if metric == "probability_entropy_stagnation":
        self.system.probability_entropy_stagnation = probability_entropy_stagnation(system)
    if self.writing and ((not self.write_result) or self.s.stopping):
      change_sizes = []
      try:
        for change in change_sequence_instance[1]:
          change_size = 0
          for command in change:
            if command == "+" or command == "-":
              change_size += 1
          change_sizes.append(change_size)
      except IndexError:
        pass
      change_sizes = str(change_sizes).lstrip("[").rstrip("]").replace(",","")
      change_times = str(change_sequence_instance[0])
      change_times = change_times.lstrip("[").rstrip("]").replace(",","")
      output =
(system_string,str(trial),str(iteration),change_times,change_sizes,str(real_time),str(cpu_time))
      output = output + tuple([str(getattr(system,parameter,"*" + str(instance[i]))) for
i,parameter in enumerate(self.parameter_names)])
      output_metrics = []
      for metric in self.metrics:
        output_metrics.append(str(getattr(system,metric,"N/A")))
      output = output + tuple(output_metrics)
      f.write(self.output_string % output)
    if not self.writing:
      for metric in self.metrics:
        monitor.append(str(getattr(system,metric,"N/A")))
    print system_string,trial,iteration,cpu_time,system.glob_best_ant_tour_length,monitor

  def make_changes(self,cpu_time,change_sequence_instance): #Implements changes.
    system = self.system
    if len(change_sequence_instance[0]) != 0 and self.change_index != -1:
      if change_sequence_instance[0][self.change_index] < cpu_time:
        print "CHANGE"
        changes = change_sequence_instance[1][self.change_index]
        i = 0
        while i <= len(changes)-1:
          change_type = changes[i]
          if change_type == "+":
            change = changes[i+1:i+4]
            i += 4
            (x,y,name) = change
            x = float(x)
            y = float(y)
            add_city(system,x,y,name)
            system.glob_best_ant = [[],999999999999999999]
          elif change_type == "-":
            change = changes[i+1]
            i += 2
            city_name = change
            remove_city(system,city_name)
            system.glob_best_ant = [[],999999999999999999]
          elif change_type == "i":
            change = changes[i+1:i+3]
            i += 3
            command = change
            if command[0] == "REPEATED_RESULT_LIMIT":
              self.s.repeated_result_stop_criteria_setup(int(command[1]))
              self.s.repeated_result_stop_criteria = True
            elif command[0] == "TIME_LIMIT":
              self.s.time_limit_stop_criteria_setup(int(command[1])+cpu_time)
              self.s.time_limit_stop_criteria = True
          #Alert the display of the changes.
          if self.display: self.input_p.send(["Cities",system.city_coords])
        if self.change_index + 1 >= len(change_sequence_instance[0]):
          self.change_index = -1
        else:
          self.change_index += 1
```

```
  def run(self,p): #Runs the algorithms. Manages the display.
    c = stopwatch()
    s = self.s
    for system_string in self.systems:
      if system_string == "Ant_System":
        self.system = Ant_System()
      if system_string == "Elitist_Ant_System":
        self.system = Elitist_Ant_System()
      if system_string == "Rank_Based_Ant_System":
        self.system = Rank_Based_Ant_System()
      if system_string == "Min_Max_Ant_System":
        self.system = Min_Max_Ant_System()
      if system_string == "Ant_Colony_System":
        self.system = Ant_Colony_System()
      if system_string == "Stinky_Ant_System":
        self.system = Stinky_Ant_System()
      system = self.system
      for parameter_instance in self.parameters:
        for change_sequence_instance in self.change_sequence_instances:
          for trial in xrange(self.trials):
            #Initialize the ACO.
            initialize_data(system,self.problem_file)
            #Setup key values.
            iteration = 0
            update_time = 0
            number_ants = system.number_ants
            chunksize = int(system.number_ants/multiprocessing.cpu_count()) + 1
            for parameter,value in izip(self.parameter_names,parameter_instance):
              if hasattr(self.system,parameter):
                setattr(system,parameter,float(value))
            #Setup outputs.
            if self.writing:
              f = open(self.write_file,"at")
            else:
              f = ""
            if self.display:
              input_p.send(["Cities",system.city_coords])
            self.change_index = 0
            #Setup stop criteria.
            s.reset()
            if self.initial_time_limit_stop_criteria:
              s.time_limit_stop_criteria_setup(self.initial_time_limit)
              s.time_limit_stop_criteria = True
            if self.initial_repeated_result_stop_criteria:
              s.repeated_result_stop_criteria_setup(self.initial_repeated_result_limit)
              s.repeated_result_stop_criteria = True
            #Start the stopwatch.
            c.reset()
            c.start()
            print "NEW RUN"
            while not s.stopping:
              cpu_time = c.cpu_time()
              real_time = c.real_time()
              #Detect and respond to change requests.
              self.make_changes(cpu_time,change_sequence_instance)
              c.start()
              #Run an iteration of the system.
              if number_ants > 50: #Running in parallel is only profitable if there are roughly
50 ants.
                system.ants = p.map(parallel_construct_tour,((system.probabilities,
system.nearest_neighbors, system.distances, system.number_cities, system.neighbors_on, system.Q)
for foo in xrange(0,number_ants)), chunksize)
              else:
                system.ants = [construct_tour(system) for foo in system.ants]
              local_search()
              update_bests(system)
              system.update_pheromone_trails()
              compute_probabilities(system)
              #Update the system.
              c.stop()
              if update_time < cpu_time:
                s.update_stop_criteria(system,cpu_time)
                self.output(f, system_string, trial, iteration, change_sequence_instance,
real_time, cpu_time, parameter_instance)
                #Send data to the display.
                if self.display:
                  if self.display_tour: input_p.send(["Tour", system.iter_best_ant[0]])
                  input_p.send(["System", system_string])
                  input_p.send(["Best", system.glob_best_ant[1]])
                  input_p.send(["CpuT", cpu_time])
```

```
                input_p.send(["RealT", real_time])
                if self.display_pheromones: input_p.send(["Pheromones", system.pheromones])
              update_time += self.refresh_rate
            iteration += 1
    if self.writing:
      f.close()

##############################################
#####                                   #####
#####            MagnifyingGlass        #####
#####             GRAPH DISPLAY         #####
#####                                   #####
#####         AUTHOR: DUSTIN TAUXE      #####
#####                                   #####
##############################################

class Graph(multiprocessing.Process):      # Class for making a graph of the ACO route
  def __init__(self,pipe):
    multiprocessing.Process.__init__(self)
    self.pipe = pipe
    self.Xsize = 1000           # Window X size, in pixels; defualt value
    self.Ysize = 1000           # Window Y size, in pixels; defualt value

    self.bgcolor = (0, 0, 0)        # Background color
    self.ctcolor = (0,255,0)        # City color
    self.trcolor = (255, 255, 255)      # Tour color
    self.txcolor = (0, 0, 0)        # Text color
    self.tbcolor = (200, 200, 200)      # Text Background color
    self.stcolor = (255, 0, 0)        # Stagnating Route color
    self.wbcolor1 = (0, 0, 0)       # Pheromone web color1
    self.wbcolor2 = (255,255, 255)        # Pheromone web color2

    self.txtSize = 24           # Value for the size of the readout text displayed
    self.system = "ACO-something"     # Name of the ACO implementation, this is default
    self.bestsofar = 1234567890       # Default value for the best solution so far
    self.cpu_time = 1234567890        # Default value for the CPU time
    self.real_time = 1234567890       # Default value for real time
                        # Defualt values should not display, they should be replaced

    self.matrix = None            # Route matrix

    self.refreshRate = 50         # Graph refresh rate (in milliseconds)

    self.cities = []            # List of all cities in this problem set
    self.name = "DISPLAY"           # Name for this process
    self.tour = None            # List by city index of tour

    self.lastTour = None          # Previous tour city list
    self.stagData = []            # Variables useful in displaying stagnating routes
    self.stagHelp = []            # ...
    self.stagWait = 100           # How long a route is static before 'stagnating'

    self.Xmin = 0             # Bounds for grid
    self.Xmax = 0             # ...
    self.Ymin = 0             # ...
    self.Ymax = 0             # ...

    self.WinPixBuffer = 10          # Pixel buffer btwn side of window, and farthest point.
Purely asthetic.

    self.AdjustWindow = True        # Allow adjustment of window size to fit problem
    self.displayText = True         # Display text overlay
    self.displayTour = True         # Display tour
    self.displayWeb = True          # Display pheromone web
    self.displayCity = True         # Display cities
    self.displayStag = True       # Display stagnating routes

    pygame.init()             # Initialize the pygame module

  def tuples_to_lists(self,l):
    return [[x,y] for (x,y) in l]

  def run(self):                # Display function
    output_p, input_p = self.pipe     # Initialize pipe, for data transfer between processes
    input_p.close()             # Do not output. (labeled input on this side of pipe)
    data = None

    self.cities = self.tuples_to_lists(output_p.recv()[1])
    self.findBounds()           # Do initial operations for graphing
    self.translatePoints()          # ...
```

```
    Window = pygame.display.set_mode((self.Xsize,self.Ysize)) # Frame
    Canvas = pygame.Surface(Window.get_size())          # For drawing on, fills frame
    Clock = pygame.time.Clock()                 # Used to time refreshes
    pygame.display.set_caption("ACO Output")

    # This is the main display loop
    while True:
      if output_p.poll():
        try:
          data = output_p.recv()
        except EOFError:
          break
      if data == False:
        break
      if data != None:                  # Search pipe for data, which is inserted as tuples where
        if data[0] == "Cities":             # the first is the ID and the second is the data.
          self.cities = self.tuples_to_lists(data[1]) # ...
          self.translatePoints()            # ...
        if data[0] == "Tour":             # ...
          self.tour = data[1]             # ...
        if data[0] == "System":             # ...
          self.system = data[1]           # ...
        if data[0] == "Best":             # ...
          self.bestsofar = data[1]          # ...
        if data[0] == "CpuT":             # ...
          self.cpu_time = data[1]           # ...
        if data[0] == "RealT":              # ...
          self.real_time = data[1]          # ...
        if data[0] == "Pheromones":           # ...
          self.matrix = data[1]           # ...

      for event in pygame.event.get():        # This exits prgram on QUIT event (Clicking the
close button)
        if event.type == pygame.QUIT: break     # Doesn't actually work with multiple processes
      Clock.tick(self.refreshRate)          # Wait until at least refreshRate msec after last
tick
                              # Otherwise it would refresh constantly, use up CPU power
      Canvas.fill(self.bgcolor)           # Color background

                              # Draw stuff on Canvas here:

      #$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
      if self.displayWeb and self.matrix != None:   # Displays the pheromone web
        # Find the high and low pheromones along the route
        hiP = 0
        loP = self.matrix[0][1]
        index = 0
        for city1 in self.matrix:
          count = 0
          for city2 in city1:
            if count < index:
              if city2 > hiP:
                hiP = city2
              if city2 < loP:
                loP = city2
              count = count + 1
            else:
              break
          index = index + 1
        self.hiP = hiP
        self.loP = loP
        # Create the colors for the routes in the web
        webColors = []
        for i in xrange(0,len(self.matrix)):
          for j in xrange(i+1, len(self.matrix)):
            try:
              coord1 = self.cities[i]
              coord2 = self.cities[j]
            except IndexError:
              pass
            color = self.getColor(self.matrix[i][j])
            webColors.append((color,coord1,coord2))
        webColors.sort(key = lambda x:x[0]) # This sorts the lines so that a darker color does
not appear over a
                          # lighter one, for asthetics.
        # Draw the web
        for city in webColors:
          pygame.draw.aaline(Canvas, city[0], city[1], city[2], 3)

      #$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
      if self.displayCity:              # Displays Cities
```

```
        # Draw the cities
        for city in self.cities:
          pygame.draw.circle(Canvas, self.ctcolor, (city[0], city[1]), 3)

      #$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
      if self.displayTour and self.tour != None:    # Dispalys the tour
        # Create list of points visited in tour
        ptlist = []
        for point in self.tour:
          try:
            ptlist.append(self.cities[point])
          except IndexError:
            pass
        # Draw the lines
        pygame.draw.aalines(Canvas, self.trcolor, True, ptlist, 3)

      #$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
      if self.displayStag and self.lastTour != None:  # Displays stagnation
        # Create lists of routes:
        tour = self.toRoutes(self.tour)
        lastTour = self.toRoutes(self.lastTour)

        # If any routes in tour are not already in stagHelp, add them to it
        inThere = False
        for route1 in lastTour:
          inThere = False
          for route2 in self.stagHelp:
            if route1 == route2:
              inThere = True
          if not inThere:
            self.stagHelp.append(route1)

        # Make sure stagData is as long as stagHelp
        count = 0
        for route in self.stagHelp:
          try:
            self.stagData[count] = self.stagData[count]
          except IndexError:
            self.stagData.append(0)
          count = count + 1

        # If a route is in both the current and previous tours, add one to its identifier in
stagData
        # If a route is not in both, set the identifier to 0
        for route1 in lastTour:
          equ = False
          for route2 in tour:
            if route1 == route2:
              equ = True
              index = 0
              for route3 in self.stagHelp:
                if route3 == route1:
                  self.stagData[index] = self.stagData[index] + 1
                index = index + 1
          if not equ:
            index = 0
            for route3 in self.stagHelp:
              if route3 == route1:
                self.stagData[index] = 0
              index = index + 1

        # Draw each route that has been the same for stagWait iterations
        count = 0
        for point in self.stagData:
          if point >= self.stagWait:
            route = self.stagHelp[count]
            pygame.draw.aaline(Canvas, self.stcolor, self.cities[route[0]],
self.cities[route[1]], 3)
          count = count + 1

      #$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
      if self.displayText:              # Displays the text readout
        ctime = "%.3f" % self.cpu_time        # Format numbers to display nicely
        rtime = "%.3f" % self.real_time       # ...
        best = "%.3f" % self.bestsofar        # ...
        # Create string for readout:
        strout = "Testfile: " + self.name + ".tsp    System: " + str(self.system) + "    Best so
far: " + best + "    CPU time: " + ctime + "    Real time: " + rtime
        pygame.font.init()              # Initialize pygame's font module
        Text = pygame.font.Font(None, self.txtSize) # Make the string into text for the font
module
```

```
        txtSurface = Text.render(strout, True, self.txcolor, self.tbcolor)  # Create surface to
display text on
        textRect = txtSurface.get_rect()
        Canvas.blit(txtSurface, textRect) # Draw text surface onto canvas

      #$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

      self.lastTour = self.tour # Make the lastTour var be the tour that was just run through

      Window.blit(Canvas, (0,0))  # Draw canvas in buffer
      pygame.display.flip()   # Display buffer

  def findBounds(self): # Finds X/Y min/max

    Xmin = self.cities[0][0]
    Ymin = self.cities[0][1]
    Xmax = 0
    Ymax = 0

    for city in self.cities:
      if city[0] < Xmin:
        Xmin = city[0]
      if city[0] > Xmax:
        Xmax = city[0]
      if city[1] < Ymin:
        Ymin = city[1]
      if city[1] > Ymax:
        Ymax = city[1]

    self.Xmin = Xmin
    self.Xmax = Xmax
    self.Ymin = Ymin
    self.Ymax = Ymax

  def translatePoints(self): # Translates points in cities to cooresponding pixels

    Xlen = self.Xmax - self.Xmin
    Ylen = self.Ymax - self.Ymin

    Xratio = (self.Xsize - 2*self.WinPixBuffer)/Xlen
    Yratio = (self.Ysize - (self.WinPixBuffer + self.txtSize))/Ylen
    ConversionRatio = 0

    if Xratio <= Yratio:
      ConversionRatio = Xratio
    else:
      ConversionRatio = Yratio

    for city in self.cities:
      city[0] = int((city[0] - self.Xmin)*ConversionRatio)
      city[1] = int((city[1] - self.Ymin)*ConversionRatio)
      # For some reason, the Y points appears flipped, so here's some code to fix it
      city[1] = int((self.Ymax - self.Ymin)*ConversionRatio - city[1])

      city[0] = city[0] + self.WinPixBuffer
      city[1] = city[1] + (self.WinPixBuffer + self.txtSize)

    if self.AdjustWindow:
      self.Xsize = int(Xlen*ConversionRatio + 2*self.WinPixBuffer)
      self.Ysize = int(Ylen*ConversionRatio + self.txtSize + 2*self.WinPixBuffer)
      self.AdjustWindow = False

  def getColor(self, route): # Return a color based on the pheromone at that route. Used for the
web.
    if route == 0:
      return (255, 255, 255)
    else:
      red1 = self.wbcolor1[0]
      green1 = self.wbcolor1[1]
      blue1 = self.wbcolor1[2]
      red2 = self.wbcolor2[0]
      green2 = self.wbcolor2[1]
      blue2 = self.wbcolor2[2]

      hiP = self.hiP
      loP = self.loP


      ratio = ((route - loP) / (hiP - loP))
      ratio = ratio**0.30
      newred = red1 + (red2-red1)*ratio
```

```
      newgreen = green1 + (green2-green1)*ratio
      newblue = blue1 + (blue2-blue1)*ratio
      newColor = (newred, newgreen, newblue)
      return newColor

  def toRoutes(self, ptlist): # Given a set of points, will return a list of routes that those
points make
    routes = []
    index = 1
    while index < len(ptlist):
      routes.append([ptlist[index-1], ptlist[index]])
      index = index + 1
    return routes

##############################################
#####                                    #####
#####          MAIN RUN SEQUENCE         #####
#####                                    #####
#####         AUTHOR: PETER AHRENS       #####
#####                                    #####
##############################################

if __name__ == "__main__":
  t = test()
  t.setup()
  p = multiprocessing.Pool()
  if t.display:
    (output_p,input_p) = multiprocessing.Pipe()
    d = Graph((output_p,input_p))
    d.daemon = True
    d.start()
    t.get_pipe(input_p)
    output_p.close()
    t.run(p)
    input_p.close()
    d.join()
  else:
    t.run(p)
  p.close()
  p.join()
```

7.5 AntFarm Test Harness Scripting Language

Note int, float, string and eoln (end of line) are primitives.

testscript:

system testparameters metrics writefile problem

system:

SYSTEMS eoln

testparameters:

TEST_PARAMETERS eoln

TEST_PARAMETERS eoln testparams

testparams:

testparam

testparam testparams

testparam: one of

paramname display refresh stop

paramname:

```
        param range

param:  one of
        rho alpha beta Q zeta

range:
        (init;increment;end)

init: one of
        int float

increment: one of
        int float

end: one of
        int float

display:
        DISPLAY displaypheromone displayparams
        DISPLAY displaypheromone

displaypheromone:
        bool

bool: one of
        ON OFF

displayparams:
        displayparam
        displayparam displayparams

displayparam:
        PHEROMONES = bool
        TOUR = bool
        STAGNATION = bool

refresh:
        REFRESH_RATE int

stop:
        STOP_CRITERIA TIME_LIMIT int

metrics:
        metric
        metric metrics
```

metric: one of
    global_best_ant_tour_length
    iter_best_ant_tour_length
    lambda_branching_factor
    pheromone_limit_stagnation
    pheromone_entropy_stagnation

writefile:
    WRITE_FILE filename

filename:
    string

problem:
    PROBLEM eoln probfile change
    PROBLEM eoln probfile

probfile:
    PROBLEM_FILE tspfile eoln

tspfile:
    *.tsp

change:
    CHANGE time changespecs eoln

changespecs:
    changespec
    changespec changespecs
changespec:
    add
    remove
    ispec
    remove_add
    add_remove

add:
    +[x_coordinate,y_coordinate,cityname]

x_coordinate:
    int

y_coordinate:
    int

remove:
	-[cityname]

ispec:
	i[TIME_LIMIT,int]
	i[REPEATED_RESULT_LIMIT,int]

add_remove:
	+-[tspfile,numcities,remtime,remcmds]

remove_add:
	-+[numcities,remtime,remcmds]

numcities: one of
	int range

remtime: one of
	int range

remcmds:
	remadd
	remremove
	remispec

remadd:
	+{x_coordinate!y_coordinate!cityname}

remremove:
	-{cityname}

remispec:
	i{TIME_LIMIT!int}
	i{REPEATED_RESULT_LIMIT!int}

7.6 Example Output

The following is an example of the output generated. We have over 100000 lines of raw data, so it is not feasible to include all the data here. An example of the data has been provided.

```
TEST_SCRIPT: qa194_dynamic_test_-+50.txt
PROBLEM_FILE: C:\SCC\qa194.tsp
DATE: Sun Apr 03 22:54:28 2011
SYSTEM,TRIAL,ITERATION,CHANGE_TIMES,CHANGE_SIZE,REAL_TIME,CPU_TIME,glob_best_ant_tour_length,iter
_best_ant_tour_length,lambda_branching_factor
Ant_Colony_System,0,0,0 0,50,0.000999927520752,0.000570029224088,9286.27822528,9286.27822528,2.0
Ant_Colony_System,0,32,0
0,50,1.02200007439,1.02271314692,10781.0118035,11008.5843418,45.1597938144
```

```
Ant_Colony_System,0,63,0
0,50,2.01399993896,2.01398512697,10463.5494514,10595.3945984,43.2989690722
Ant_Colony_System,0,95,0
0,50,3.02200007439,3.02127904103,10436.4142821,11220.6033252,44.0979381443
Ant_Colony_System,0,126,0
0,50,4.02999997139,4.02967611347,10423.1364301,10469.1769737,45.2371134021
Ant_Colony_System,0,155,0
0,50,5.00100016594,5.00127881809,10381.1993196,10845.177727,42.0360824742
Ant_Colony_System,0,185,0
0,50,6.0030002594,6.00353702557,9908.18252338,9908.18252338,44.1134020619
Ant_Colony_System,0,215,0
0,50,7.02700042725,7.02665184514,9870.20902638,9916.29958917,44.2216494845
Ant_Colony_System,0,244,0
0,50,8.01200079918,8.01131602182,9723.07497956,9723.07497956,44.2371134021
Ant_Colony_System,0,275,0
0,50,9.00400066376,9.00287167116,9715.62814074,9715.62814074,46.1030927835
Ant_Colony_System,0,307,0
0,50,10.0260007381,10.0249390668,9715.62814074,9734.59451079,46.9226804124
```

7.7 qa194.tsp

Taken from the National TSP site [14].

```
NAME : qa194
COMMENT : 194 locations in Qatar
COMMENT : Derived from National Imagery and Mapping Agency data
COMMENT : Optimal is 9352
TYPE : TSP
DIMENSION : 194
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 24748.3333 50840.0000
2 24758.8889 51211.9444
3 24827.2222 51394.7222
4 24904.4444 51175.0000
5 24996.1111 51548.8889
6 25010.0000 51039.4444
7 25030.8333 51275.2778
8 25067.7778 51077.5000
9 25100.0000 51516.6667
10 25103.3333 51521.6667
11 25121.9444 51218.3333
12 25150.8333 51537.7778
13 25158.3333 51163.6111
14 25162.2222 51220.8333
15 25167.7778 51606.9444
16 25168.8889 51086.3889
17 25173.8889 51269.4444
18 25210.8333 51394.1667
19 25211.3889 51619.1667
20 25214.1667 50807.2222
21 25214.4444 51378.8889
22 25223.3333 51451.6667
23 25224.1667 51174.4444
24 25233.3333 51333.3333
25 25234.1667 51203.0556
26 25235.5556 51330.0000
27 25235.5556 51495.5556
28 25242.7778 51428.8889
29 25243.0556 51452.5000
30 25252.5000 51559.1667
31 25253.8889 51535.2778
32 25253.8889 51549.7222
33 25256.9444 51398.8889
34 25263.6111 51516.3889
35 25265.8333 51545.2778
36 25266.6667 50969.1667
37 25266.6667 51483.3333
38 25270.5556 51532.7778
39 25270.8333 51505.8333
40 25270.8333 51523.0556
41 25275.8333 51533.6111
42 25277.2222 51547.7778
43 25278.3333 51525.5556
44 25278.3333 51541.3889
45 25279.1667 51445.5556
46 25281.1111 51535.0000
```

47 25281.3889 51512.5000
48 25283.3333 51533.3333
49 25283.6111 51546.6667
50 25284.7222 51555.2778
51 25286.1111 51504.1667
52 25286.1111 51534.1667
53 25286.6667 51533.3333
54 25287.5000 51537.7778
55 25288.0556 51546.6667
56 25290.8333 51528.3333
57 25291.9444 51424.4444
58 25292.5000 51520.8333
59 25298.6111 51001.6667
60 25300.8333 51394.4444
61 25306.9444 51507.7778
62 25311.9444 51003.0556
63 25313.8889 50883.3333
64 25315.2778 51438.6111
65 25316.6667 50766.6667
66 25320.5556 51495.5556
67 25322.5000 51507.7778
68 25325.2778 51470.0000
69 25326.6667 51350.2778
70 25337.5000 51425.0000
71 25339.1667 51173.3333
72 25340.5556 51293.6111
73 25341.9444 51507.5000
74 25358.8889 51333.6111
75 25363.6111 51281.1111
76 25368.6111 51226.3889
77 25374.4444 51436.6667
78 25377.7778 51294.7222
79 25396.9444 51422.5000
80 25400.0000 51183.3333
81 25400.0000 51425.0000
82 25404.7222 51073.0556
83 25416.9444 51403.8889
84 25416.9444 51457.7778
85 25419.4444 50793.6111
86 25429.7222 50785.8333
87 25433.3333 51220.0000
88 25440.8333 51378.0556
89 25444.4444 50958.3333
90 25451.3889 50925.0000
91 25459.1667 51316.6667
92 25469.7222 51397.5000
93 25478.0556 51362.5000
94 25480.5556 50938.8889
95 25483.3333 51383.3333
96 25490.5556 51373.6111
97 25492.2222 51400.2778
98 25495.0000 50846.6667
99 25495.0000 50965.2778
100 25497.5000 51485.2778
101 25500.8333 50980.5556
102 25510.5556 51242.2222
103 25531.9444 51304.4444
104 25533.3333 50977.2222
105 25538.8889 51408.3333
106 25545.8333 51387.5000
107 25549.7222 51431.9444
108 25550.0000 51433.3333
109 25560.2778 51158.6111
110 25566.9444 51484.7222
111 25567.5000 50958.8889
112 25574.7222 51486.3889
113 25585.5556 51151.3889
114 25609.4444 51092.2222
115 25610.2778 51475.2778
116 25622.5000 51454.4444
117 25645.8333 51450.0000
118 25650.0000 51372.2222
119 25666.9444 51174.4444
120 25683.8889 51505.8333
121 25686.3889 51468.8889
122 25696.1111 51260.8333
123 25700.8333 51584.7222
124 25708.3333 51591.6667
125 25716.6667 51050.0000
126 25717.5000 51057.7778
127 25723.0556 51004.1667

```
128 25734.7222 51547.5000
129 25751.1111 51449.1667
130 25751.9444 50920.8333
131 25758.3333 51395.8333
132 25765.2778 51019.7222
133 25772.2222 51483.3333
134 25775.8333 51023.0556
135 25779.1667 51449.7222
136 25793.3333 51409.4444
137 25808.3333 51060.5556
138 25816.6667 51133.3333
139 25823.6111 51152.5000
140 25826.6667 51043.8889
141 25829.7222 51245.2778
142 25833.3333 51072.2222
143 25839.1667 51465.2778
144 25847.7778 51205.8333
145 25850.0000 51033.3333
146 25856.6667 51083.3333
147 25857.5000 51298.8889
148 25857.5000 51441.3889
149 25866.6667 51066.6667
150 25867.7778 51205.5556
151 25871.9444 51354.7222
152 25872.5000 51258.3333
153 25880.8333 51221.3889
154 25883.0556 51185.2778
155 25888.0556 51386.3889
156 25900.0000 51000.0000
157 25904.1667 51201.6667
158 25928.3333 51337.5000
159 25937.5000 51313.3333
160 25944.7222 51456.3889
161 25950.0000 51066.6667
162 25951.6667 51349.7222
163 25957.7778 51075.2778
164 25958.3333 51099.4444
165 25966.6667 51283.3333
166 25983.3333 51400.0000
167 25983.6111 51328.0556
168 26000.2778 51294.4444
169 26008.6111 51083.6111
170 26016.6667 51333.3333
171 26021.6667 51366.9444
172 26033.3333 51116.6667
173 26033.3333 51166.6667
174 26033.6111 51163.8889
175 26033.6111 51200.2778
176 26048.8889 51056.9444
177 26050.0000 51250.0000
178 26050.2778 51297.5000
179 26050.5556 51135.8333
180 26055.0000 51316.1111
181 26067.2222 51258.6111
182 26074.7222 51083.6111
183 26076.6667 51166.9444
184 26077.2222 51222.2222
185 26078.0556 51361.6667
186 26083.6111 51147.2222
187 26099.7222 51161.1111
188 26108.0556 51244.7222
189 26116.6667 51216.6667
190 26123.6111 51169.1667
191 26123.6111 51222.7778
192 26133.3333 51216.6667
193 26133.3333 51300.0000
194 26150.2778 51108.0556
EOF
```

8.0 Bibliography

[1] Dorigo, Marco, and Thomas Stutzle. *Ant Colony Optimization*. Hong Kong, China: Massachusetts Institute of Technology, 2004. Print.

[2] Benzatti, Danilo *Emergent Intelligence.* Ants Colony and Multi-Agents 5 Nov. 2010. <http://ai-depot.com/CollectiveIntelligence/Ant-Colony.html>

[3] Cook, Damon. *Evolved and Timed Ants: Optimizing the Parameters of a Time-Based Ant System Approach to the Traveling Salesman Problem Using a Genetic Algorithm.* Las Cruces, NM: New Mexico State University: Computer Science Department, 2000. Print.

[4] Cung, Van-Dat, Simone Martins, Celso Ribiero, and Catherine Roucairol. "Strategies for the Parallel Implementation of Metaheuristics." (2001): Print.

[5] Dorigo, Marco et al. Ant *Colony Optimization 2004*: Lecture Notes-Computer Science 3172. 2004. Print.

[6] Kennedy, James, and Russell Eberhart. *Swarm Intelligence*. San Francisco, CA: Academic Press, 2001. Print.

[7] Manfrin, Max, Mauro Birattari, Thomas Stutzle, and Marco Dorigo. "Parallel Ant Colony Optimization for the Traveling Salesman Problem." *IRIDIA - Technical report series*. (2006): Print.

[8] Mitchell, Melanie. *Complexity: A Guided Tour*. San Francisco, CA: Oxford University Press, Inc, 2009. Print.

[9] Joost, Eyckelhoflehof, and Snoek, Marko. "Ant systems for dynamic TSP." *Lecture Notes in Computer Science M Dorigo et. al.*. 2463. Berlin: Springer-Verlag, 2002. Print.

[10] M. Guntsch and M. Middendorf. Pheromone modification strategies for ant algorithms applied to dynamic TSP. In *Applications of Evolutionary Computing: Proceedings* of EvoWorkshops 2001. Springer Verlag, 2001.

[11] M. Guntsch, M. Middendorf, and H. Shmeck. An ant colony approach to dynamic TSP. In Lee spector et. al., editor, Abstract Proceedings of ANTS'2000, pages 59-62, 2000.

[12] - ,

Brussels, Belgium, May 2009.

[13] TSPLIB http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[14] National TSPs http://www.tsp.gatech.edu/world/countries.html

9.0 Acknowledgements

First and foremost, our team would like to thank the people of the New Mexico Supercomputing

Challenge. The opportunity given by the Supercomputing Challenge to work with some of the best in the field of computer science was a real privilege. Working on this project has given everybody on this team perception on the application and the usefulness of algorithmic study in computer programming.

Secondly, Team 56 would like to thank Mr. Goodwin and Mr. Smith, our sponsor teachers from Los Alamos High and La Cueva High, respectively, who have encouraged us throughout our project and who have supported our team with their advice and knowledge. They have provided for our team the optimism that was necessary for the move forward in our work.

Next, we would all like to thank to our mentors, Christine and James Ahrens for their expertise in data visualization and computing.

We would also like to give thanks to all of the Supercomputing judges. They have taken time out of their lives to read through our ideas in the proposal and interim reports, evaluate our project, and give us feedback and suggestions on how to improve our project. We could not have made it this far without the judges' support and we are greatly appreciative of them.

And last, but not least, we would like to thank all of our families, our moms, dads, sisters, and brothers for their continuing support for us in the Supercomputing Challenge.