ExcellAnts

New Mexico
Supercomputing Challenge
Final Report
April 4, 2012

Team 66
Los Alamos High School

Team Members:
Peter Ahrens
Dustin Tauxe

Teacher:
Lee Goodwin

Mentors:
James Ahrens
Christine Ahrens

Contents

1 Executive Summary

The aim of this project is to create an efficient parallel implementation of Ant Colony Optimization **(ACO)** applied to Traveling Salesman Problem **(TSP)**. It should also be portable and easy to understand or modify. 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. ACOs have applications in problems including vehicle routing, networking, communications, and scheduling. **Data Parallelism** is a style of parallelism that usually consists of running the same fine grained operation for each piece of data in a very long vector.[6] The very large size of data that must be processed in a parallel ACO makes a data parallel implementation attractive. Due to the small size and large number of computations that must be performed at the same time, data parallelism particularly lends itself to computation on the Graphics Processing Unit **(GPU)**. GPUs must do many tasks via "threads" in parallel to display pixels on the screen. They are convenient to use as general purpose processors for hardware acceleration of programs, as they are readily available on most computers. We used **Thrust**, a data-parallel C++ template library modeled off of the C++ Standard Library's[13] vector operations and implemented in a data parallel fashion, to implement our data-parallel ACO. Our implementation proved to be very effective. We achieved a speedup of about 100 (it varies with problem size) over a serial implementation. Our program has a computational complexity of O($nlog(n)$) while the serial implementation has a complexity of O(n^3). Excellants has made original contributions. Firstly, our implementation is portable to targets other than GPU. Also, we describe a tree algorithm that is important. Our code is also available and open source, both in this report and online. Additionally, our code is written in an easy to understand and modify C++ template library called Thrust. These advantages are important to anyone looking to use our work in a practical application or to extend it in the research world.

2 Problem Statement

Ants can forage for food quite efficiently. When an ant finds food, it leaves a pheromone trail back to the ant hill, which compels other ants to follow the same path. However, as the wind blows and the sun shines on this trail, the pheromones start to evaporate. Only the most traveled trails can continue to exist. Thus shorter, more popular paths are generated. Ant Colony Optimization **(ACO)** is a technique inspired by this ant foraging behavior, and can be used to generate good solutions to combinatorial optimization problems very quickly.[8]

Although Ant Colony Optimization seems more suited to foraging, it has proved itself a powerful metaheuristic that can be applied to problems ranging from routing to machine learning. However, ACOs are most 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 TSP represents the dilemma of one unlucky salesman who has several cities to visit, but limited gas money and time in which he may do so. Our salesman would like to travel the shortest tour possible. Unfortunately, the TSP is a very difficult problem. A brute force approach to a TSP of **n** cities would have a computational complexity of

$$\frac{(n-1)!}{2} \tag{1}$$

Solving a 200-city TSP using brute force would take approximately $2.062x10^{360}$ years on Computer A [1], yet an ACO can get to within 2% of the optimum in 200 seconds. For the purposes of this project, the Travelling Salesman Problem will be used as a standard problem to solve, but is important to note that the TSP is not the focus of this project. Many excellent TSP solvers already exist. The focus is on ACO, which can be applied to many problems ranging from networking to protein folding, and TSP will be used as an example problem for ACO to tackle.

In some cases, the problem may change during the time an ACO is generating a solution. Let us use, as an example, the case of a truck driver delivering packages. He must deliver several hundred packages a day, and finding an optimal tour could take a while. To make matters worse, unanticipated packages may arrive in the early morning. A fast ACO would save him time (less time spent waiting for his solution to be generated). It would also save money (if the optimization runs faster, using the time he has more efficiently, he would get a better tour and thus have to spend less money on gas). Thus, there are two advantages to having a faster ACO. Our driver's day is not over, however. Suppose there was an eleven-car pileup on a major road. The solution his program had generated was operating under the assumption that this road would be a convenient route, but now the driver needs a new tour. He could wait for the long simulation to run again, but if he had an ACO that could quickly generate a new solution based on previous calculations, he could get home to his family much faster.

This TSP with cities that change as the problem is being solved is called the Dynamic Travelling Salesman Problem **(DTSP)**. The DTSP can be thought of in two ways. It could be that the problem changes after a solution has been generated, and the ACO simply resumes working with the previously calculated pheromones, or it could be thought that the problem changes as the ACO is solving it, and it must cope with the changes. The former situation being more suited toward a more stable real world application like our truck driver, and the latter being suited to something more volatile like a network routing problem. These two perspectives may arise out of different situations, but they are fundamentally identical in solution, in that the simulation must simply alter the data it has calculated before the change to fit the new conditions.

Clearly, an ACO applied to a DTSP would have to be fast, and could thus benefit from a parallel implementation. Implementing ACO in parallel is difficult, however, due to the random memory access patterns and the coordination of large parallel tasks.[8] Even though an ACO will not actually be applied to DTSP in this paper, it provides an excellent reason for an ACO to be sped up, and any advances in ACO techniques for a TSP could easily be applied to DTSP.

Due to the large amount of data that must be processed in an ACO and the relative simplicity of the computations that must be performed, a parallel implementation of ACO would be desirable. Also, because the TSP is simply a sample problem, the code of such an implementation would have to be simple enough to be modified for other problems. The aim of this project is to create an efficient parallel implementation of ACO on the GPU applied to TSP. It should also be portable and easily understood or modified.

3 Background

In order to understand the methods used to create an efficient parallel implementation of ACO, one must first understand the traditional implementation of an ACO applied to TSP. The mechanism of action for an ACO can be described as follows.

All ACOs have the same approximate structure. To initialize, they calculate all the distances between cities, make pheromone and probability matricies (a way to store the values of all the pheromones on all the trails), and create ants. Once the data is initialized, then the program enters the main loop in which the ants construct solutions and then lay pheromone based on the quality of these solutions.

In a TSP, 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:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} \text{ , } ifj \in N_i^k \tag{2}$$

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, evaporation occurs on the routes first. The new amount of pheromone on a route τ'_{ij} is given by:

$$\tau'_{ij} = (1-\rho)\tau_{ij} \tag{3}$$

where ρ is a parameter (from 0-1). Then, the ants deposit pheromone on their tours. Usually, the base unit of pheromone an ant lays down, $\Delta\tau_{ij}$, is given by:

$$\Delta\tau_{ij} = \frac{1}{C_{ij}} \tag{4}$$

where C_{ij} is the ant's tour length. After the ants deposit pheromone in some configuration, Pheromone update occurs in many ways, so the above equations are to help the reader understand the basic ways the pheromone update works.

3.1 Previous Work

For last year's Supercomputing Challenge, we created the most common implementations of ACO in Python. These implementations ran in parallel on the CPU using Python's multiprocessing module. We had great success with the performance enhancements that came from the parallelization of the algorithms. However, last year's program was not optimized for speed.[5] Designing last year's code inspired us to build a much faster, more optimized, more efficient version this year. All code from last year had to be scrapped as we were using a faster language and more powerful tools, including an entirely new approach to ACO parallelization.

4 Methods

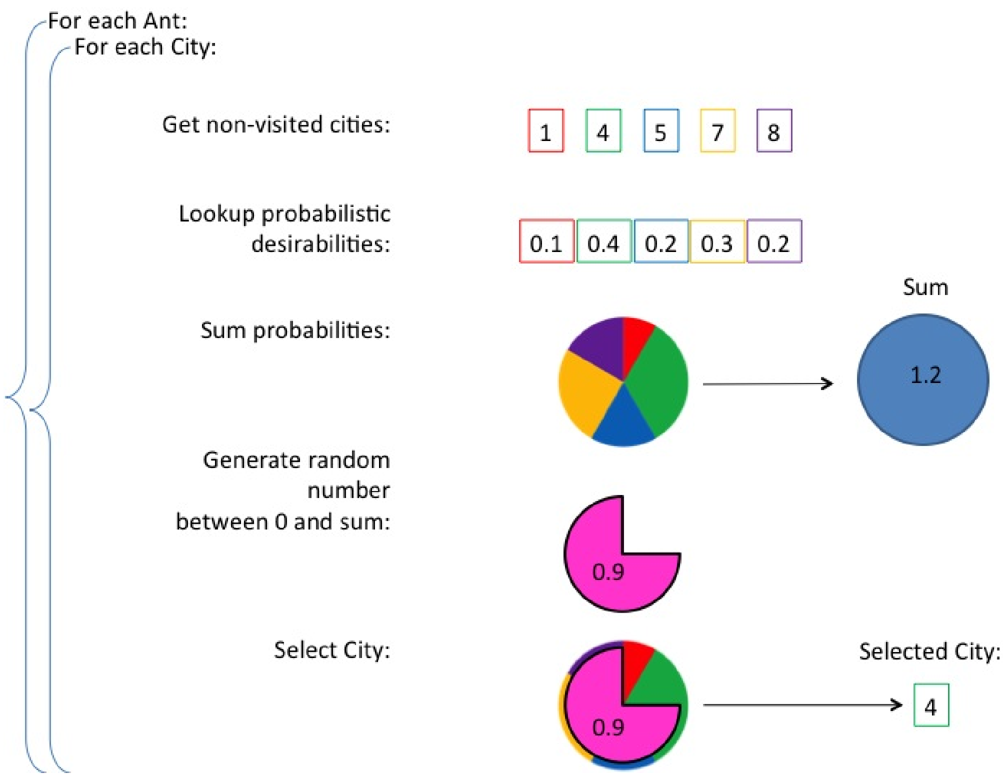
4.1 Tour Construction

Tour construction is the step in an ACO when all ants must construct paths that visit every city only once. Because tour construction takes up most of the time in an ACO[8], this is the aspect on which we focused most. Many implementations were tried and tested to find a suitable parallel tour construction method.

4.1.1 Serial Implementation

The probability of an ant at city i going into city j is described in Equation 2. In a traditional tsp, this probabilistic selection is accomplished through method analogous to a roulette wheel. The various probabilities that an ant may visit are gathered into a list. A random number is generated between 0 and the sum of these probabilities. The ant then iterates over each probability and selects its next city to visit. (One can imagine the roulette ball starting at the top and travels counter clockwise and ends up landing in a pie piece corresponding to a particular city) This process is repeated until an entire tour is created. See Figure 1. This will be very computationally costly as it is done for every ant for every city.

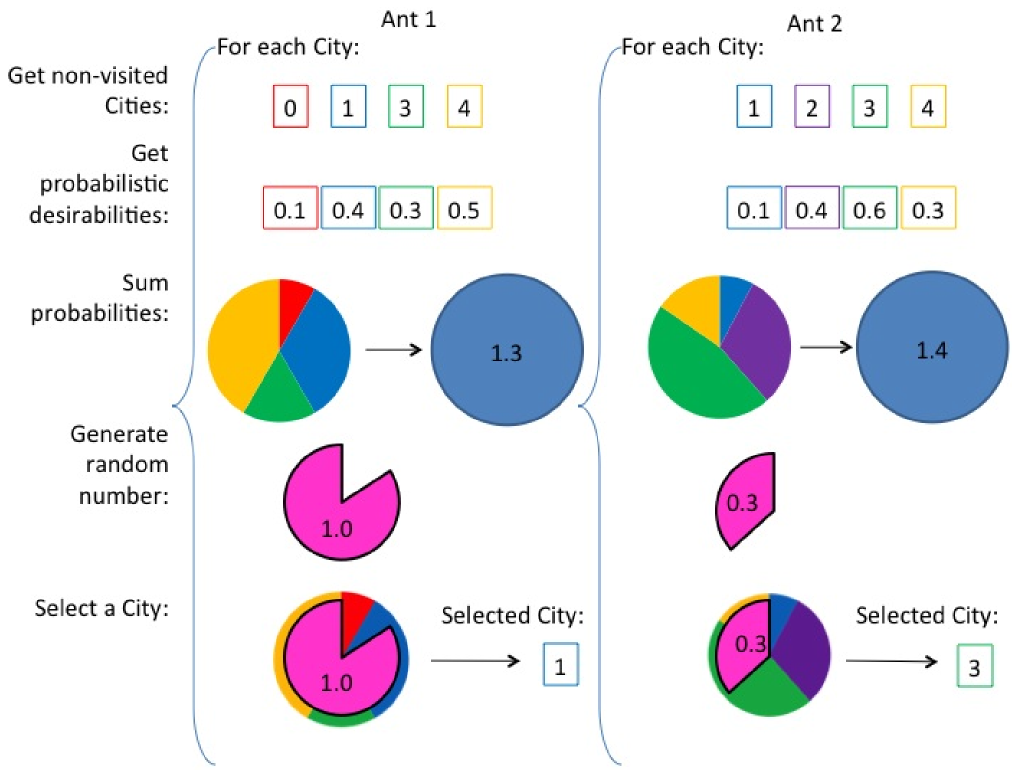
Figure 1: A typical serial implementation.



4.1.2 Task Parallel Implementation

Task Parallelism is the typical style of parallelism in which the parallelism is focused on doing different tasks on different pieces of data. Task parallelism usually consists of running processes to compute the results of more coarse grained tasks. In hopes of speeding up the traditional ACO, it would be tempting to simply run the ants in parallel. It does not scale efficiently, as each ant must still look at all of the next cities at every step of its tour sequentially (O(n^2), where n is the number of cities. It also creates excessive overhead. All of the ants have random access patterns, the tasks may take different times to complete, and the size and number of tasks is also neither suited to a GPU nor CPU. Each processor must make an entire tour for one ant. The overhead significantly reduces the power of such an approach.[8] See Figure 2.

Figure 2: A typical task parallel implementation.



4.1.3 Turning to Data Parallelism

Data Parallelism is a style of parallelism focused on performing the same tasks on the same pieces of data. It usually consists of running the same fine-grained operation for each piece of data in a very long vector. [6] The very large size of data that must be processed in parallel makes a data parallel implementation attractive.[6] Due to the small size and large number of computations that must be performed at the same time, data parallelism particularly lends itself to computation on the Graphics Processing Unit **(GPU)**. GPUs must do many tasks via "threads" in parallel to display pixels on the screen. They are convenient to use as general purpose processors for hardware acceleration of programs, as they are readily available on most computers. We chose to use Thrust [11], a data-parallel C++ template library modeled off of the C++ Standard Library's[13] vector operations and implemented in a data parallel fashion described by Guy Blelloch[6] in his thesis, which has sometimes been described as a data-parallel bible. **Thrust** is very easy to use and modify for data-parallel operations. The most common Thrust target is CUDA, but it can also target OpenMP, OpenCL, or Thread Building Blocks. Thrust essentially translates the data parallel operations to primitive functions in CUDA, OpenMP, etc. It also provides host and device vector types, that store data on either the host or computation device. [11] Using Thrust eliminated the need for us to create a very specific and optimized data parallel functions and allowed us to create a more general ACO within the time allotted for this project. Some of these Thrust functions are used extensively in our code and assume a very important role in what we do. These functions are described by example in Table 1.

Table 1: Common primitive data-parallel functions supplied by Thrust.

Inputs		
A	0 5 1 8 7 3	
B	2 2 2 2 2 2	
C	1 2 5	
D	0 1 1 2 3 3 4 5	
Function	**Output**	**Description**
gather(C in A)	5 1 3	Index A by indicies in C
permutation_ iterator(C in A)	5 1 3	Gather with kernel fusion
inclusive_ scan(A)	0 5 6 14 21 24	Cumulative sum of A
transform(A and B with +)	2 7 3 10 9 5	Add A to B
reduce(A with +)	24	Sum A
sort(A)	0 1 3 5 7 8	Sort A
upper_ bound(C in D)	3 4 7	Find last index of D where C could be inserted without violating ordering

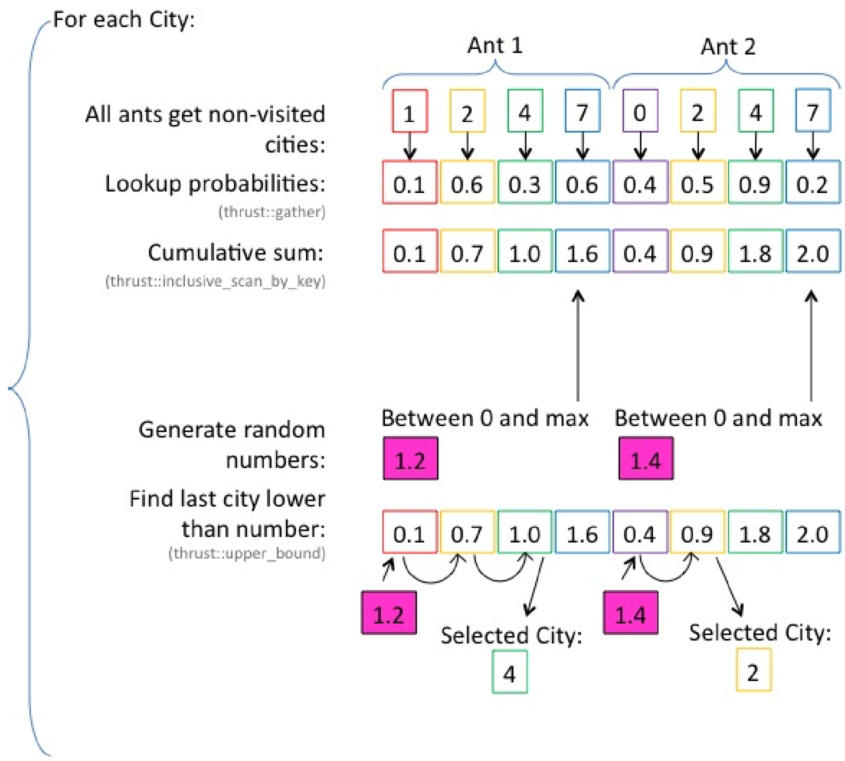
If a programmer can use Thrust functions as frequently and correctly as possible, their code will be both modular, portable, and efficient. For example, when performing multiple memory bound operations on a vector of data, it is better to use kernel fusion, or condense each of the operations to be performed into the same kernel, or chunk of code that will be executed on the device. Another example is the usage of Thrust functions such as the zip_ iterator to create virtual arrays that can be processed without having to actually move or reorder large amounts of data.

4.1.4 Data Parallel Implementations

The first data parallel method we tested was analogous to stacking the previously described roulette wheels and selecting cities for every ant at the same time. Each ant gathered data for all the cities it was going to visit. This is very quick on a GPU because all of the lookups can be performed in parallel and there are many processors with which to do this. Then, a prefix sum(cumulative sum) was performed on a list of all the probabilities. This has the effect of evaluating each piece of the roulette wheel previously described in the traditional ACO at the same time. Then, the random numbers are generated, and the list of probabilities is iterated over in parallel by every ant.

This implementation suffers, however, due to the number of operations that must be performed in series. At every step of the tour construction, the cities the ant visited had to be updated, the probability gathered, the probabilities prefix summed, random number bounds selected, random numbers generated, and searches performed. While this implementation was fairly straightforward to code and understand it needed to be improved upon. See Figure 3.

Figure 3: Our initial data parallel implementation.



A new method had to be implemented that was completely different from all the others. More of the operations had to be grouped and performed at the same time. To accomplish this a tree-based algorithm was implemented. All the probabilities are gathered as previously described for each ant. Then, each probability is assigned to a thread, along with the city it is associated with, and a random number. At each step in the tree, two cities are reduced to one. The random number is used to probabilistically select a city based on the probabilities given. Then, the probabilities are summed and the chosen city is given to the next level of the tree. The unused random number is given to the next level of the tree. This mathematically can select from a large list of cities a single city randomly with a bias toward the probabilities in the same way that the previous algorithms have described. The reason the probabilities are summed has to do with multiplication of probabilities. The probability of going from city i to city j should be equivalent to the desirability metric of that city divided by the sum of the probabilities of all the other cities that ant can visit. The probability of an example city 1 being selected out of five cities in the tree algorithm is shown below in Equation 5. The probability of city 1 being selected is equivalent to the probability of city 1 being selected at every level of the tree.

$$p^k_{5\ 1} = \frac{p_1}{p_1 + p_2} * \frac{p_1 + p_2}{p_1 + p_2 + p_3 + p_4} = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta} \tag{5}$$

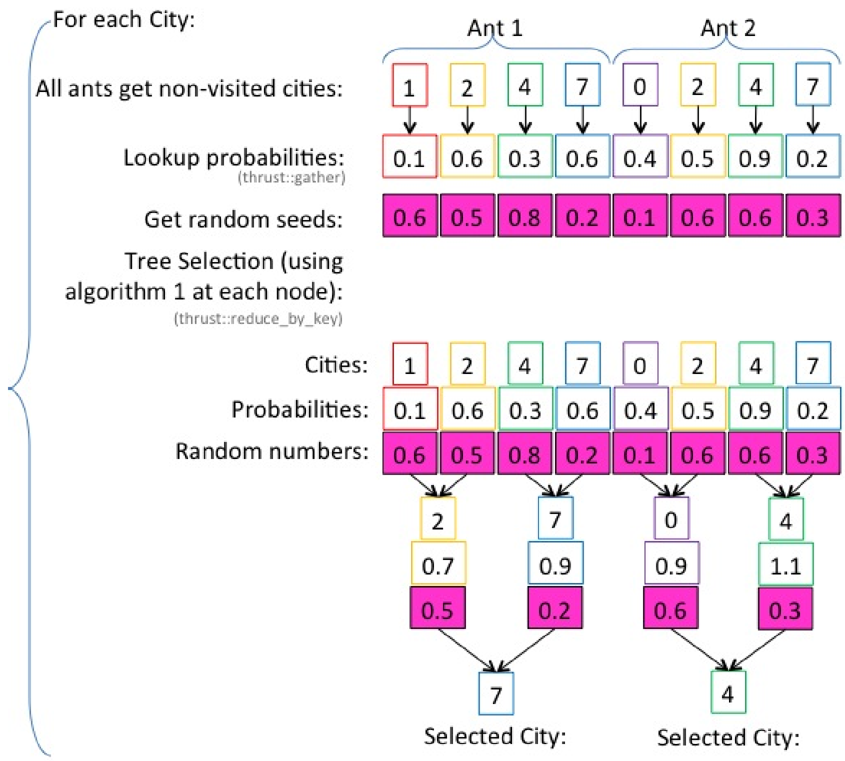
As one can see, the probabilities simplify to Equation 2 as desired. This implementation performed very well and can also scale efficiently. The reduction can perform city selection for all of the ants and all of the cities in parallel for every level of the tree, making the reduction O($log(n)$). The reduction must be performed for every city, so the whole tour construction step is O($nlog(n)$). The implementation is described in the diagram below, and pseudocode is given for the decision function at each node in the tree. See Algorithm 1, Figure 4.

Algorithm 1 The algorithm used to reduce two cities in the tree selection method.

```
Inputs: (City A, Probability A, Random A),
        (City B, Probability B, Random B)

if (Random A * (Probability A + Probability B) < Probability A)
{
      return (City A,
            (Probability A + Probability B), // Probabilities added
            Random B) // Unused random returned
} else {
      return (City B,
            (Probability A + Probability B), // Probabilities added
            Random B) // Unused random returned
}
```

Figure 4: Our best data parallel implementation.



4.2 Pheromone Update

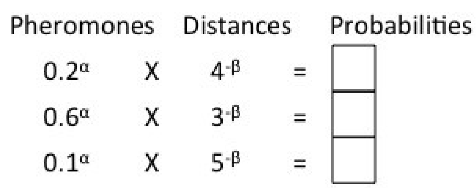
Even though the tour construction is the most time consuming, it was simple enough to implement pheromone updates in parallel, and important to do as it must not become a bottleneck. It is also advantageous to do this on a GPU as data transfer between the GPU and the CPU is expensive in terms of computation time.

The pheromone update is straightforward to implement but can be different for every ACO variant. As a general rule, it makes sense to add the previously calculated amount of pheromone to each route in the necessary tours. For our project we settled on a Rank Based Ant System as it is simple to implement. In a Rank Based Ant System, ants deposit pheromone according to their rank. Their rank is assigned based on the quality of their tour. Most of the time the number of ranks is usually around six so only six ants will deposit pheromone. Thankfully, Thrust has a very well implemented and efficient sort, so this sort was used to assign the six ranks.

4.3 Probability Calculation

Calculating probabilties is also simple to do in parallel. Each probability for a particular route is calculated by a separate thread, using the distance of that route and the pheromone on that route. Probability calculations for each route are easily parallelized. Each probability is calculated in parallel. The distance and pheromone for that particular route can be looked up by a separate thread, making this a very simple and quick implementation. thrust::transform() was used to calculate Equation 2 ($probability = pheromone^{\alpha} * 1/distance^{\beta}$) for all of the cities in parallel. See Figure 5

Figure 5: Probability calculation with thrust::transform().



5 Results

It was initially decided that Marco Dorigo's code would be used as a benchmark test against which to compare our code.[8] This, however, led to issues because his code did not allow for large numbers of ants (i.e. greater than 100). For this reason a new ACO was selected. We settled on libaco[10] because it is a very standard ACO, and can accommodate the large number of ants that we require. It is as accurate as Dorigo's, but may be slower. We cannot measure because Dorigo's code does not allow for large numbers of ants. The exact speed may not be as important of a metric as how the algorithm scales. This is because the two algorithms are being run on different devices. For this reason, both experiments measuring speedup and experiments validating theoretical scaling have been run. There is, however, a limit to scaling, as the number of threads and amount of local memory on a GPU may reach an upper limit. Another metric used to assess the quality of both algorithms is a comparison of the quality of the tour after a certain amount of time. This is a similar metric to a simple speedup calculation, but is looking at a fixed time limit instead of a fixed quality (same number of mathematically identical iterations) as shown above. These two metrics address both sides of the time/quality trade-off described in detail in the problem statement. All statistical calculations were done with R, a statistical language.[9]

5.1 Validation

5.1.1 Experiment 1

The first challenge was to validate our ACO. Even though the two implementations were created to produce equal results, we felt that evidence of this should be given. To do this, we compare the tour lengths after 20 iterations from 20 trials of both implementations on Computer A[1] on dj38.tsp from National TSP [3]. This is a simple test problem of the largest population centers in Djbouti. A two sample t-test of these 20 trials yields a p-value of .638. Thus, assuming the programs produce identical results, the probability that the discrepancies in the results we obtained were due to chance is 63.8%. This is likely enough to assume that the two implementations produce identical results. Knowing that the implementation is valid, tests can be run to determine scaling capability.

5.2 Scaling and Speedup

An ACO could scale with respect to a few metrics. For clarity, the number of cities will be referred to by the parameter n. The number of ants will be referred to by the parameter m.

5.2.1 Experiment 2

The first test of scaling capability was done with respect to the number of cities. The number of ants was held at a constant 128. The city sets for this test were the first n cities of usa13509.tsp[12], a set of cities in the United States. This and all following scaling tests were performed on Computer B[2]. The results are shown for both implementations in Table 3. Note that a maximum speedup of 100 was achieved.

Table 2: The time to complete one iteration with respect to the number of cities.

Number of Cities	libaco Time (s)	ExcellAnts Time (s)	Speedup
32	0.0675	0.0138	4.89
64	0.2881	0.0254	11.34
128	1.2141	0.0630	19.27
256	5.2334	0.1344	39.35
512	22.3871	0.3707	60.39
1024	95.1977	0.9484	100.38

The expected scaling of a serial implementation with respect to cities should be O(n^2), as each city must be examined at every city in the tour. With this assumption, a fitted linear regression was performed on the data. The correlation constant was 1.000, and the residuals were scattered. r^2 was 0.9998, meaning that 99.98% of the variation in the data is explained by the theoretical scaling. This means that we can safely say the above model is correct.

The expected scaling of our data-parallel implementation with respect to cities should be O($nlog(n)$), as all the cities are reduced (in log(n) time) at every city in the tour. With this assumption, a fitted linear regression was performed on the data. The correlation constant was 0.9973, and the residuals were scattered. r^2 was 0.9947, meaning that 99.47% of the variation in the data is explained by the theoretical scaling. This means that we can safely say the above model is correct.

5.2.2 Experiment 3

The second test of scaling capability used different numbers of ants with the same number of cities. The city set for this problem was held at the constant first 128 cities of usa13509.tsp[12]. The results are shown for both implementations in Table 3. Note that we achieved a maximum speedup of 81.

Table 3: The time to complete one iteration with respect to the number of ants.

Number of Ants	libaco Time (s)	ExcellAnts Time (s)	Speedup
32	0.3032	0.0484	6.26
64	0.6077	0.0532	11.42
128	1.2124	0.0628	19.31
256	2.4238	0.0642	37.75
512	4.842	0.0912	53.09
1024	9.6713	0.1184	81.68

The expected scaling of a serial implementation with respect to ants should be O(m), as the tour construction is performed for every ant. With this assumption, a fitted linear regression was performed on the data. The correlation constant was 1.000. The residuals were patterned, but they were too small to acknowledge. r^2 was 0.9999, meaning that 99.99% of the variation in the data is explained by the theoretical scaling. This means that we can safely say the above model is correct.

The expected scaling of data-parallel implementation with respect to ants should be O($log(m)$), as all the ants' possible cities are reduced (in log(n) time) for a constant amount of cities. With this assumption, a fitted linear regression was performed on the data. The correlation constant was 0.9310. r^2 was 0.8665, meaning that 86.65% of the variation in the data is explained by the theoretical scaling. Although this is a relatively low r^2 value, the residuals are very scattered, and show absolutely no clear pattern. This means that we can safely say the above model is correct.

5.2.3 Experiment 4

The third test of scaling capability considered both ants and cities. The number of ants was set to the recommended amount, the number of cities[8]. The city sets for this test were the first n cities of usa13509.tsp[12]. The results are shown for both implementations graphically in figures 6 and 7. Note that the vertical axes have different values. Note that a maximum speedup of 340 was achieved.

Table 4: The time to complete one iteration with respect to the number of cities and ants.

Number of Cities and Ants (n = m)	libaco Time (s)	ExcellAnts Time (s)	Speedup
32	0.0169	0.0099	1.71
64	0.1429	0.0281	5.09
128	1.2123	0.0631	19.21
256	10.4507	0.1861	56.16
512	89.6375	0.6784	132.13
1024	756.8478	2.2204	340.86

Figure 6: The time for libaco to complete one iteration with respect to the number of cities and ants.

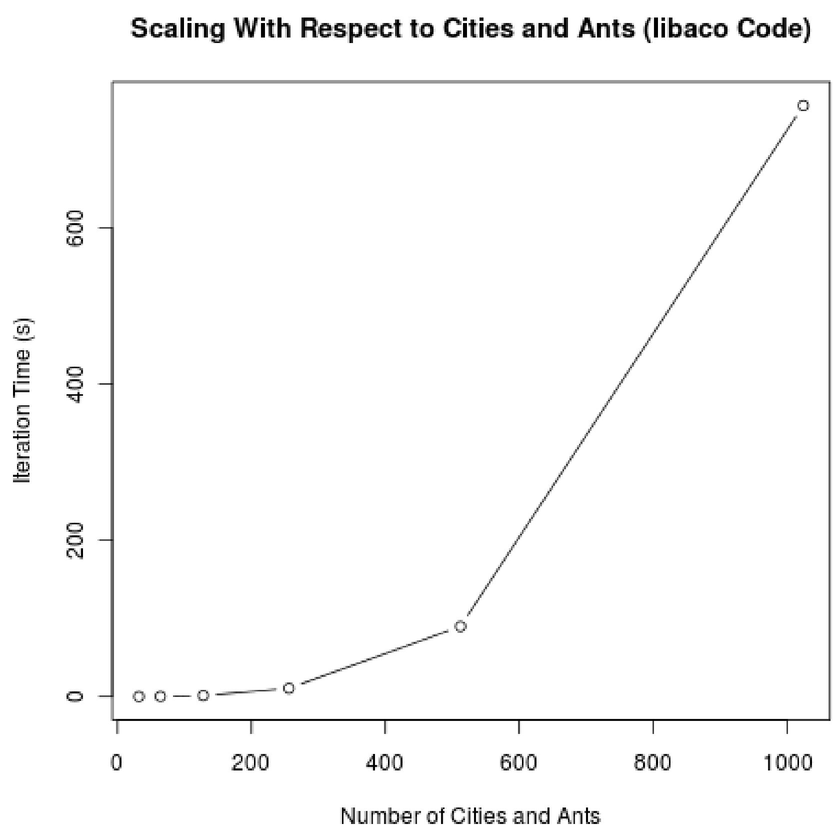
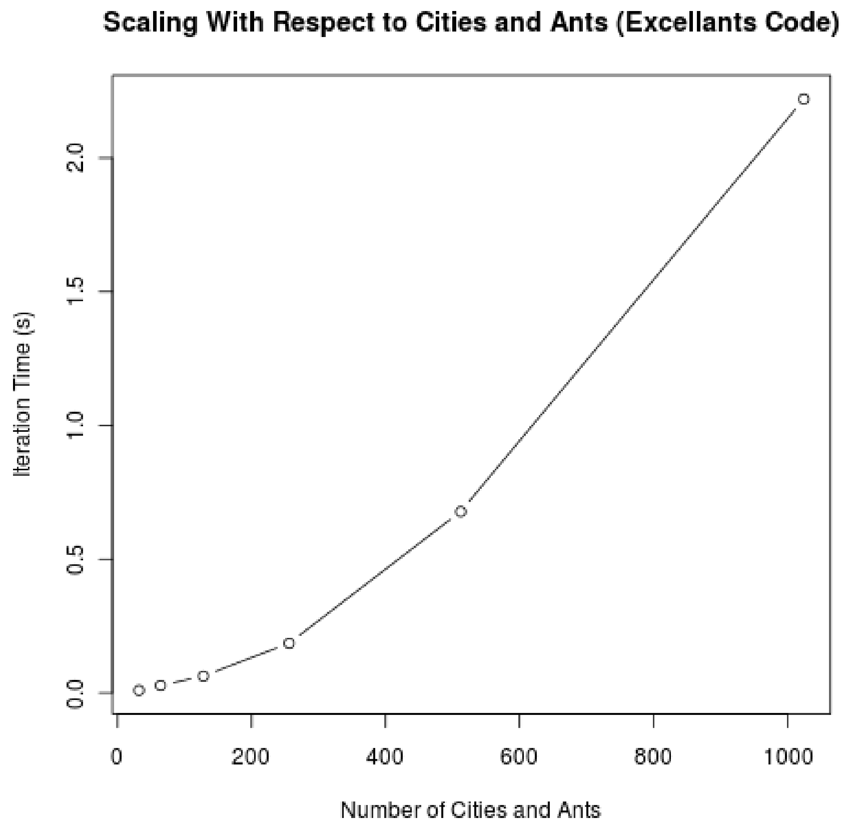


Figure 7: The time for ExcellAnts to complete one iteration with respect to the number of cities and ants.



The expected scaling of a serial implementation with respect to cities and ants should be O(n^2m) or O(n^3) ($n = m$), as each city must be examined at every city in each ant's tour. With this assumption, a fitted linear regression was performed on the data. The correlation constant was 1.000, and the residuals were scattered. r^2 was 0.9999, meaning that 99.99% of the variation in the data is explained by the theoretical scaling. This means that we can safely say the above model is correct. The expected scaling of our data-parallel implementation with respect to cities and ants should be O($nlog(n)$) as all the cities for all the ants are reduced (in log(n) time) for every city in the tour. This is the same scaling equation that we saw in Experiment 2 for the serial code. The reason for this is that although the cities must be evaluated for every ant, they are all reduced at the same time, producing O($nlog(nm)$), or O($nlog(n^2)$) ($n = m$), which reduces to O($nlog(n)$). With this assumption, a fitted linear regression was performed on the data. The correlation constant was 0.9982, and the residuals were scattered. r^2 was 0.9765, meaning that 97.65% of the variation in the data is explained by the theoretical scaling. This means that we can safely say the above model is correct.

5.3 Quality

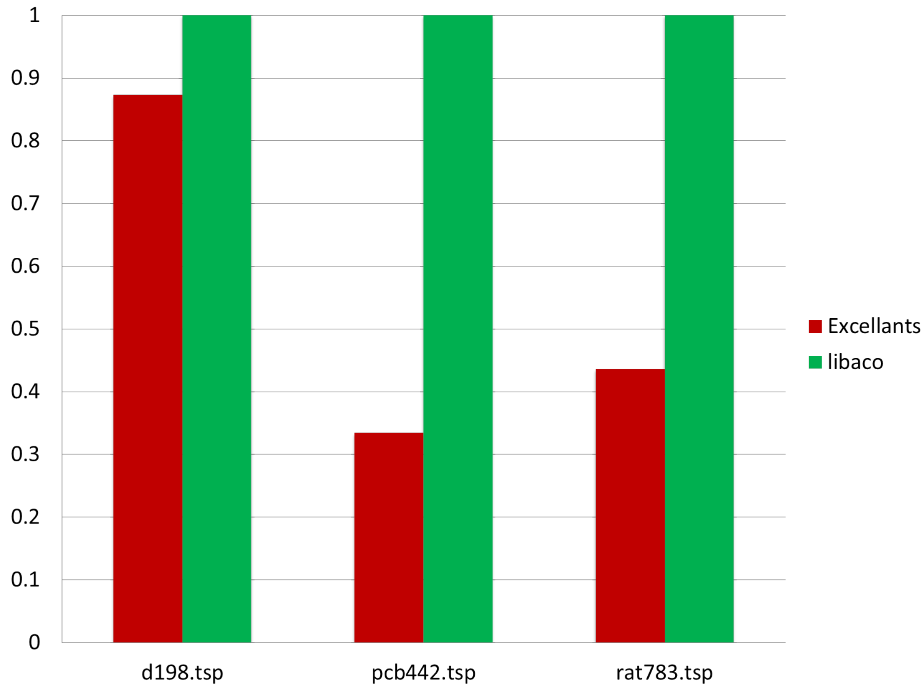
5.3.1 Experiment 5

One could think of performance as getting a faster solution or getting a better solution. In this experiment, the two implementations were compared by their end tour quality. The following tests were run on Computer A.[1] Both libaco and ExcellAnts tour qualities were measured at the end of 100 seconds on different datasets from TSPLIB. [12] The results are presented in Table 5 and the proportional improvement can be visually compared in Figure 8. The Excellants quality was up to three times better.

Table 5: The best tour lengths generated by each ACO in 100 seconds.

TSP	d198.tsp	pcb442.tsp	rat783.tsp
ExcellAnts	16536	64727.6	3165.93
libaco	18941	193819	7263.16
libaco/ExcellAnts	0.87	0.33	0.44

Figure 8: The proportional difference in best tour lengths generated by each ACO in 100 seconds.



6 Conclusions

A data-parallel ACO on the GPU that is easy to program and is portable to multiple targets has its advantages. First and foremost, it's faster. The parallel code outperformed the serial code in all cases.(see Tables 3, 2, 4) We achieved a speedup over the libaco code of about 100 with 512 ants and 512 cities. The speed can also lead to better quality. In some cases, ExcelLAnts achieved three times the quality of a serial implementation. Also, it scales much more efficiently. If one ran their simulations at the recommended number of ants, libaco would scale at O(n^3) while ExcellAnts would scale at O($nlog(n)$). This project successfully created a working data-parallel implementation of ACO that significantly outperformed the serial version, with vast improvements in scaling capability. Even a simple task-parallel approach would only reach a scaling of O(n^2) with respect to cities and ants, as all cities must still be evaluated at each city in the tour. Our program was also portable to different targets. We were able to run our program with OpenMP. Our code also made use of an off the shelf data parallel library, for ease of programming.

7 Significant Original Achievement

ExcellAnts has made some important contributions. Firstly, we wrote an efficient, high-quality data-parallel implementation of ACO (via Thrust [11]) that is portable to multiple targets (CUDA, OpenCL, OpenMP, or Thread Building Blocks) without having to change the code. Secondly, the methods used to construct the tree-based selection process are described in detail. This is important to anyone attempting their own implementation using our methods. Thirdly, our code is also available and open source, both in this report and online.[4] Fourth, the time to build this efficient, portable implementation was less than what it would have taken to create a hand-tuned implementation on each target, since Thrust provides a library that is easy to use. These advantages are important to anyone looking to use our work in a practical application or to extend it in the research world.

8 Related Work

It came to our attention as this paper was being written that a paper on a hand-tuned GPU implementation of ACO was available online in January 2012.[7] However, all the original ideas presented here were conceived independently from this work. We did not read it until we had done our implementation and started writing our paper. We have had several original achievements, and actually some improvements over this work. The first one of these is the tree-based selection process. This sped up our code considerably and also allows it to scale to larger data sets more efficiently. Ours is described in detail, their's is not. The other methods initially attempted were also of our own design. Our code is also written in Thrust, while theirs is written in CUDA and highly optimized to a GPU, making it difficult to understand and non-portable. While their code is not currently available, The code for ExcellAnts is available in an open source format online.[4]

9 Future Work

This report does not mark the end of our project. We aim to continue our work on our Google Project page.[4] We have several plans which we expect to apply to the code in the future. One of the most prominent of these is that we plan to use our ACO on a dynamic traveling salesman problem (DTSP) and optimize it for use with DSTPs. Another plan is to overlap CPU and GPU computation by running suitable operations on both processors in order to maximize the use of the computational power of the system on which it is running. Yet another goal for us to work toward in the future is to add more implementations of ACO to give the user more flexibility.

10 Work Products

10.1 Code

```
/**************************************
 * Setup.cpp                          *
 * Peter Ahrens                       *
 * Sets up the ACO                    *
 **************************************/

#include "RankBasedAntSystem.h"
#include "TSPReader.h"
#include "Comm.h"
#include "Writer.h"
#include <iostream>
#include <unistd.h>
#include <string>
#include <cctype>
#include <ctime>
using namespace std;

//Setup: The main control loop to the whole program.
int main(int argc, char* argv[]){
  //declare variables
  cout << "|SETUP|\n";
  string antHillType = "RBAS";
  int m = -1;
  int ranks = 6;
  int maxTime = 0;
```

```
  int maxIter = 0;
  int maxReps = 0;
  int reps = 0;
  bool stopping = false;
  bool graphics = false;
  char* filen;
  Writer O; //writes output to stdout and an optional file
  //Read initially neccesary command-line arguments.
  for(int i = 0; i < argc;i++){
    if (string(argv[i]) == "-ras"){
      antHillType = "RBAS";
      if(i + 1 < argc){
  if(string(argv[i])[0] != '-'){
    ranks = atoi(argv[i+1]);
  }
      }
    }
    if (string(argv[i]) == "-tsp"){
      filen = argv[i+1];
    }
    if (string(argv[i]) == "-gui"){
      graphics = true;
    }
    if (string(argv[i]) == "-m"){
     m = atoi(argv[i+1]);
    }
    if (string(argv[i]) == "-out"){
      if(!O.setFile(argv[i+1])){
  cout << "Unable to open output file\n";
      }
    }
    if (string(argv[i]) == "-maxTime"){
      maxTime = atoi(argv[i+1]);
    }
    if (string(argv[i]) == "-maxIter"){
      maxIter = atoi(argv[i+1]);
    }
    if (string(argv[i]) == "-maxReps"){
      maxReps = atoi(argv[i+1]);
    }
  }
  cout << ">" << flush; //-----Checkpoint 1
  if(graphics){
    //If graphics are running, create pipes and processes.
    int parentPipe[] = {-1,-1};  // parent -> child
```

```
    int childPipe[] = {-1,-1};  // child -> parent
    if ( pipe(parentPipe) < 0  ||  pipe(childPipe) < 0 )
      {
  std::cout << "Failed to create pipe";
  return 1;
      }
#define PARENT_READ   childPipe[0]
#define CHILD_WRITE   childPipe[1]
#define CHILD_READ    parentPipe[0]
#define PARENT_WRITE  parentPipe[1]
    pid_t pID = fork();
    if(pID == 0){//child
      close(PARENT_WRITE);
      close(PARENT_READ);
      //Graphics run in a separate process from the computations so neither is slowed down.
      Comm C(CHILD_READ,CHILD_WRITE);
      cout << ">" << flush; //----Checkpoint 3
      if(!C.send(string("Test"))){
  cout << "\n Interprocess Comm Failed";
  return 1;
      }
    }else if(pID < 0){//fail
      cout << "Failed to fork";
      return 1;
    }else{//parent
      close(CHILD_READ);
      close(CHILD_WRITE);
      //Computations run in a separate process from the graphics so neither is slowed down.
      Comm C(PARENT_READ,PARENT_WRITE);
      cout << ">" << flush;//----Checkpoint 2
      TSPReader t;
      t.read(filen);
      cout << ">" << flush;//----Checkpoint 4
      if(m == -1){
  m = t.getNumNodes();
      }
      RankBasedAntSystem antHill(t.getDistances(),t.getNumNodes(),m);
      cout << ">" << flush;//----Checkpoint 5
      //If any parameters need to be changed, they are modified from their defaults here.
      for(int i = 0; i < argc;i++){
  if (string(argv[i]) == "-b"){
```

```
    antHill.setBeta(atof(argv[i+1]));
  }
  if (string(argv[i]) == "-r"){
    antHill.setRho(atof(argv[i+1]));
  }
      }
      cout << ">" << flush;//----Checkpoint 6
      antHill.initialize();
      if(C.recieve() == "Test"){ //A check to find out if the comm is working.
  cout << ">\n" << flush;//----Checkpoint 7
      }else{
  cout << "\n Interprocess Comm Failed";
  return 1;
      }
      O.writeHeader(antHill.getBeta(),antHill.getRho(),antHill.getNumAnts(),antHillType,t.getName());
      clock_t t1, t2, t3;
      t1 = t2 = t3 = clock();
      //Main control sequence.
      for(int i = 0; !stopping; i++){
  t2 = t3;
  antHill.forage();
  t3 = clock();
  O.write(i, antHill.getIterBestDist(), antHill.getGlobBestDist(), (double)(t3 - t1) / CLOCKS_PER_SEC, (double)(t3 - t2) / CLOCKS_PER_SEC);
  if(maxTime != 0){
    if((int)((double)(t3 - t1) / CLOCKS_PER_SEC)> maxTime){
      stopping = true;
    }
  }
  if(maxIter != 0){
    if(i >= maxIter){
      stopping = true;
    }
  }
  if(maxReps != 0){
    if(antHill.getReps() >= maxReps){
      stopping = true;
    }
  }
      }
    }
  }else{
```

```
    cout << ">" << flush;//-----Checkpoint 2
    TSPReader t;
    t.read(filen);
    cout << ">" << flush;//-----Checkpoint 3
    if(m == -1){
      m = t.getNumNodes();
    }
    RankBasedAntSystem antHill(t.getDistances(),t.getNumNodes(),m);
    //If any parameters need to be changed, they are modified from their defaults here.
    cout << ">" << flush;//-----Checkpoint 4
    for(int i = 0; i < argc;i++){
      if (string(argv[i]) == "-b"){
  antHill.setBeta(atof(argv[i+1]));
      }
      if (string(argv[i]) == "-r"){
  antHill.setRho(atof(argv[i+1]));
      }
    }
    cout << ">" << flush;//-----Checkpoint 5
    antHill.initialize();
    cout << ">>\n" << flush;//-----Checkpoint 6/7
    O.writeHeader(antHill.getBeta(),antHill.getRho(),antHill.getNumAnts(),antHillType,t.getName());
    clock_t t1, t2, t3;
    t1 = t2 = t3 = clock();
    //Main control sequence.
    for(int i = 0; !stopping; i++){
      t2 = t3;
      antHill.forage();
      t3 = clock();
      O.write(i, antHill.getIterBestDist(), antHill.getGlobBestDist(), (double)(t3 - t1) / CLOCKS_PER_SEC, (double)(t3 - t2) / CLOCKS_PER_SEC);
      if(maxTime != 0){
  if((double)(t3 - t1) / CLOCKS_PER_SEC > maxTime){
    stopping = true;
  }
      }
      if(maxIter != 0){
  if(i >= maxIter){
    stopping = true;
  }
      }
```

```
        if(maxReps != 0){
  if(antHill.getReps() >= maxReps){
    stopping = true;
  }
        }
      }
    }
}

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
//
//    Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
//    Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
//    Neither the name of Excellants nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


/***************************************
 * RankBasedAntSystem.h                *
```

```cpp
 * Peter Ahrens                        *
 * Performs specific RBAS procedures    *
 *************************************/

#ifndef RANKBASEDANTSYSTEM_H
#define RANKBASEDANTSYSTEM_H
#include "Colony.h"

//RankBasedAntSystem: Provides the neccesary extensions to Colony to create a Rank-Based Ant System
class RankBasedAntSystem : Colony{
 public:
  RankBasedAntSystem(thrust::host_vector<float> newDistances, int newNumCities, int newNumAnts); // Allocates memory and sets defaults.
  void initialize(); // Runs the Colony initialize, then creates additional maps and keys.
  void computeParameters(); // Simply computes neccesary parameters.
  void forage(); // Runs Colony forage.
  void setRho(float newRho);
  void setBeta(float newBeta);
  void setW(int newW);
  int getW();
  double getRho();
  double getBeta();
  int getNumAnts();
  double getIterBestDist();
  double getGlobBestDist();
  int getReps();
 private:
  void computeInitialPheromone(); // Computes the initial pheromone level with the formula described by Marco Dorigo.
  void updatePheromones(); // Evaporates, then the ants lay pheromone at levels corresponding to their rank, judged by the distances of their tours.
  int w;
  thrust::device_vector<float> RBASWeight;
  thrust::device_vector<int> RBASMap;
};

#endif

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
```

```
//
//Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
//
//    Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
//    Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
//    Neither the name of Excellants nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


/**************************************
 * RankBasedAntSystem.cu              *
 * Peter Ahrens                       *
 * Performs specific RBAS procedures  *
 **************************************/

#include "RankBasedAntSystem.h"

//constructor: Allocates memory and sets defaults.
RankBasedAntSystem::RankBasedAntSystem(thrust::host_vector<float> newDistances, int newNumCities, int newNumAnts)
  : Colony(newDistances, newNumCities, newNumAnts)
{
```

```
  w = 6;//default
  RBASWeight = thrust::device_vector<float>(numAnts);
  RBASMap = thrust::device_vector<int>(numAnts*numCities);
}

//initialize: Runs the Colony initialize, then creates additional maps and keys.
void RankBasedAntSystem::initialize()
{
  Colony::initialize();
  thrust::fill(RBASWeight.begin(),RBASWeight.end(),0);
  thrust::copy_n(thrust::make_reverse_iterator(thrust::make_counting_iterator(w)),
      w,
      RBASWeight.begin());
  thrust::fill(ACInt.begin(),
          ACInt.end(),
          0);
  ACInt[numCities] = 1;
  thrust::inclusive_scan(ACInt.begin(),
        ACInt.end(),
        ACInt.begin());
  thrust::exclusive_scan_by_key(ACInt.begin(),
         ACInt.end(),
         thrust::make_constant_iterator(1,0),
         RBASMap.begin());
}

//computeParameters: Simply computes neccesary parameters.
void RankBasedAntSystem::computeParameters()
{
  if (numCities < w){
    w = numCities;
  }
  computeInitialPheromone();
}

//computeInitialPheromone: Computes the initial pheromone level with the formula described by Marco Dorigo.
void RankBasedAntSystem::computeInitialPheromone()
{
  initialPheromone = 0.5*w*(w-1)/(rho * Colony::greedyDistance());
}

```

```
//updataPheromones: Evaporates, then the ants lay pheromone at levels corresponding to their rank, judged by the distances of their tours.
void RankBasedAntSystem::updatePheromones()
{
  //evaporate
  thrust::transform(pheromones.begin(),
        pheromones.end(),
        thrust::make_constant_iterator(1.0f-rho),
        pheromones.begin(),
        thrust::multiplies<float>());
  //determine ant pheromone levels
  thrust::stable_sort_by_key(thrust::make_permutation_iterator(antDistances.begin(),ACKey.begin()),
           thrust::make_permutation_iterator(antDistances.end(),ACKey.end()),
           antTours.begin());
  thrust::transform(RBASWeight.begin(),
        RBASWeight.end(),
        antDistances.begin(),
        AFloat.begin(),
        thrust::divides<float>());
  //AFloat[w-1] = w/globBestDist;
  AFloat[w-1] = w/iterBestDist;//for a simple rankbased, without global pheromone
  ACInt.assign(antTours.begin(),antTours.end());
  //thrust::scatter(globBestTour.begin(),globBestTour.end(),thrust::make_counting_iterator(numCities*(w-1)),ACInt.begin());
  thrust::scatter(iterBestTour.begin(),
      iterBestTour.end(),
      thrust::make_counting_iterator(numCities*(w-1)),
      ACInt.begin()); //for a simple rankbased, without global pheromone
  thrust::transform(ACInt.begin(),
        ACInt.end(),
        thrust::make_permutation_iterator(ACInt.begin(),distMap.begin()),
        ACInt2.begin(),
        saxpy_functor(numCities));
  //lay Pheromone
  for(int i = 0; i < numCities*w; i += numCities){
    thrust::transform(thrust::make_permutation_iterator(pheromones.begin(),ACInt2.begin() + i),
```

```
            thrust::make_permutation_iterator(pheromones.end(),
                ACInt2.begin() + i + numCities),
            thrust::make_permutation_iterator(AFloat.begin(),ACKey
                .begin() + i),
            thrust::make_permutation_iterator(pheromones.begin(),
                ACInt2.begin() + i),thrust::plus<float>());
  }
}

//forage: Runs Colony forage.
void RankBasedAntSystem::forage()
{
  Colony::forage();
}

void RankBasedAntSystem::setRho(float newRho)
{
  Colony::setRho(newRho);
}

void RankBasedAntSystem::setBeta(float newBeta)
{
  Colony::setBeta(newBeta);
}

void RankBasedAntSystem::setW(int newW)
{
  w = newW;
}

int RankBasedAntSystem::getW()
{
  return w;
}

double RankBasedAntSystem::getRho()
{
  return Colony::getRho();
}

double RankBasedAntSystem::getBeta()
{
  return Colony::getBeta();
}

```

```
int RankBasedAntSystem::getNumAnts()
{
  return Colony::getNumAnts();
}

double RankBasedAntSystem::getIterBestDist()
{
  return Colony::getIterBestDist();
}

double RankBasedAntSystem::getGlobBestDist()
{
  return Colony::getGlobBestDist();
}

int RankBasedAntSystem::getReps()
{
  return Colony::getReps();
}

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or
    without modification, are permitted provided that the
    following conditions are met:
//
//    Redistributions of source code must retain the above
    copyright notice, this list of conditions and the following
    disclaimer.
//    Redistributions in binary form must reproduce the above
    copyright notice, this list of conditions and the following
    disclaimer in the documentation and/or other materials
    provided with the distribution.
//    Neither the name of Excellants nor the names of its
    contributors may be used to endorse or promote products
    derived from this software without specific prior written
    permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
    CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
    INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
    MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
    CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
```

```
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE
, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


/***************************************
 * Colony.h                            *
 * Peter Ahrens                        *
 * Main ACO procedures                 *
 ***************************************/

#ifndef COLONY_H
#define COLONY_H
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/copy.h>
#include <thrust/fill.h>
#include <thrust/binary_search.h>
#include <thrust/sequence.h>
#include <thrust/adjacent_difference.h>
#include <thrust/random.h>
#include <thrust/functional.h>
#include <thrust/gather.h>
#include <thrust/iterator/constant_iterator.h>
#include <thrust/iterator/discard_iterator.h>
#include <thrust/scan.h>
#include <thrust/sort.h>
#include <thrust/remove.h>
#include <iostream>
#include <sys/time.h>
#include <math.h>
//saxpy_functor: Performs the operation s = a * x + y, where a is a constant.
struct saxpy_functor
{
  const float a;
  saxpy_functor(float _a) : a(_a) {}
  __host__ __device__
  float operator()(const float &x, const float &y) const
  {
    return a * x + y;
```

```
  }
};

//treeSelect: The main function used to reduce two cities that an ant might visit.
struct treeSelect : public thrust::binary_function<thrust::tuple<int,float,unsigned int>,thrust::tuple<int,float,unsigned int>,thrust::tuple<int,float,unsigned int> >
{
  treeSelect() {}
  __host__ __device__
    thrust::tuple<int,float,unsigned int> operator()(const thrust::tuple<int,float,unsigned int> tup1,const thrust::tuple<int,float,unsigned int> tup2) const
  {
    const float prob = tup1.get<1>() + tup2.get<1>();
    if(tup1.get<2>() * prob / 4294967296 < tup1.get<1>()){
      return thrust::make_tuple(tup1.get<0>(),prob,tup2.get<2>());
    } else{
      return thrust::make_tuple(tup2.get<0>(),prob,tup2.get<2>());
    }
  }
};

//prob_functor: Performs the probabilistic desireability calculation s = (pheromone level)^alpha * (1/distance)^beta
struct prob_functor: public thrust::binary_function<float,float,float>
{
  const float beta;
  prob_functor ( float _beta ) : beta ( _beta ) {}
  __host__ __device__
    float operator () ( const float & pher , const float & dist ) const
  {
    return pher * pow(1 / dist, beta) ;
  }
};

//randStep: Performs one step of a linear congruential generator
struct randStep : public thrust::unary_function<unsigned int,unsigned int>
{
```

```
  __host__ __device__
    unsigned int operator()(const unsigned int x) const
  {
  //numerical recipies LCG values
    return ((x * 1664525) + 1013904223) % 4294967296;
  }
};

//unaryMultiplies: Multiplies all the values of an array by a
    value.
struct unaryMultiplies : public thrust::unary_function<int, int>
{
  const int y;
  unaryMultiplies (int _y ) : y ( _y ) {}
  __host__ __device__
    int operator()(const int x) const
  {
    return x * y;
  }
};

//unaryPlus: Adds a value to all the elements of an array.
struct unaryPlus : public thrust::unary_function<int, int>
{
  const int y;
  unaryPlus ( int _y ) : y ( _y ) {}
  __host__ __device__
    int operator()(const int x) const
  {
    return x + y;
  }
};

//isX: Checks to see if a elements of an array are equal to a
    given constant.
struct isX
{
  const int y;
isX( int _y ) : y ( _y ) {}
  __host__ __device__
  bool operator()(const int x) const
  {
    return x == y;
  }
};
```

```

//Colony: The main ACO functions and data.
class Colony
{
 public:
  Colony(thrust::host_vector<float> newDistances, int
      newNumCities, int newNumAnts);
  void initialize(); // Initializes data, creates maps and keys,
       performs standard ACO initialization steps etc.
  void forage(); // Main ACO loop. Performs the solution
      constructruction step, then updates distances, pheromones,
      probabilities.
  void computeAntDistances(); // Computes the distances of each
      ant's tour, then updates records.
  void computeProbabilities(); // Computes the probabilities
      from the distances and pheromones.
  void setRho(float newRho);
  void setBeta(float newBeta);
  double getRho();
  double getBeta();
  int getNumAnts();
  double getIterBestDist();
  double getGlobBestDist();
  int getReps();
  virtual void computeParameters() = 0; //Implemented
      differently in each ACO.
 protected:
  float greedyDistance(); // Returns the value of a simple
      greedy solution starting at city 0.
  virtual void computeInitialPheromone() = 0; //Implemented
      differently in each ACO.
  virtual void updatePheromones() = 0; //Implemented differently
       in each ACO.
  //world vars
  int numCities;
  int reps;
  //float alpha = 1, alpha is always 1
  float beta;
  float rho;
  float initialPheromone;
  thrust::device_vector<float> pheromones;
  thrust::device_vector<float> distances;
  thrust::device_vector<float> probabilities;
  //ant vars
  int numAnts;
```

```
  float iterBestDist;
  float globBestDist;
  thrust::device_vector<int> iterBestTour;
  thrust::device_vector<int> globBestTour;
  thrust::device_vector<float> antVisits;
  thrust::device_vector<int> toVisit;
  thrust::device_vector<int> antTours;
  thrust::device_vector<float> antDistances;
  thrust::device_vector<float> currentProbabilities;
  thrust::device_vector<int> currentNeighbors;
  //maps and keys
  thrust::device_vector<int> ACMapF;
  thrust::device_vector<int> ACMapL;
  thrust::device_vector<int> tourMap;
  thrust::device_vector<int> distMap;
  thrust::device_vector<int> ACKey;
  thrust::device_vector<int> ARepeatCMap;
  thrust::device_vector<int> ANMapF;
  thrust::device_vector<int> ANMapL;
  thrust::device_vector<int> ANKey;
  thrust::device_vector<int> CCKey;
  thrust::device_vector<int> ARepeatNMap;
  //scratch variables
  thrust::device_vector<float> AFloat;
  thrust::device_vector<int> AInt;
  thrust::device_vector<int> NInt;
  thrust::device_vector<int> ACInt;
  thrust::device_vector<int> ACInt2;
  thrust::device_vector<int> ACInt3;
  thrust::device_vector<int> ANInt;
  thrust::device_vector<float> ACFloat;
  thrust::device_vector<float> CCFloat;
  thrust::device_vector<unsigned int> AUnsignedInt;
  //random numbers
  thrust::device_vector<unsigned int> ARandom;
  thrust::device_vector<unsigned int> ACRandom;
};
#endif

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or
   without modification, are permitted provided that the
   following conditions are met:
```

```
//
//     Redistributions of source code must retain the above
    copyright notice, this list of conditions and the following
    disclaimer.
//     Redistributions in binary form must reproduce the above
    copyright notice, this list of conditions and the following
    disclaimer in the documentation and/or other materials
    provided with the distribution.
//     Neither the name of Excellants nor the names of its
    contributors may be used to endorse or promote products
    derived from this software without specific prior written
    permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
    CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
    INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
    MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
    CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
    SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
    NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
    HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
    OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE
    , EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


/***************************************
 * Colony.cu                           *
 * Peter Ahrens                        *
 * Main ACO procedures                 *
 ***************************************/

#include "Colony.h"

//Constructor: Sets defaults and allocates memory.
Colony::Colony(thrust::host_vector<float> newDistances, int
    newNumCities, int newNumAnts)
{
  //defaults
  beta = 2;
  rho = 0.1;
  //world vars
  reps = 0;
```

```
  distances.assign(newDistances.begin(),newDistances.end());
  numCities = newNumCities;
  probabilities = thrust::device_vector<float>(numCities*numCities);
  pheromones = thrust::device_vector<float>(numCities*numCities);
  //ant vars
  numAnts = newNumAnts;
  antDistances = thrust::device_vector<float>(numAnts);
  antVisits = thrust::device_vector<float>(numCities*numAnts);
  toVisit = thrust::device_vector<int>(numCities*numAnts);
  antTours = thrust::device_vector<int>(numCities*numAnts);
  iterBestDist = std::numeric_limits<float>::max() - 1;
  globBestDist = std::numeric_limits<float>::max();
  iterBestTour = thrust::device_vector<int>(numCities);
  globBestTour = thrust::device_vector<int>(numCities);
  //maps and keys
  ACMapF = thrust::device_vector<int>(numAnts);
  ACMapL = thrust::device_vector<int>(numAnts);
  tourMap = thrust::device_vector<int>(numAnts);
  distMap = thrust::device_vector<int>(numCities*numAnts);
  ACKey = thrust::device_vector<int>(numAnts*numCities);
  ARepeatCMap = thrust::device_vector<int>(numAnts*numCities);
  CCKey = thrust::device_vector<int>(numCities*numCities);
  //scratch variables
  AFloat = thrust::device_vector<float>(numAnts);
  AInt = thrust::device_vector<int>(numAnts);
  ACInt = thrust::device_vector<int>(numAnts*numCities);
  ACInt2 = thrust::device_vector<int>(numAnts*numCities);
  ACInt3 = thrust::device_vector<int>(numAnts*numCities);
  ACFloat = thrust::device_vector<float>(numAnts*numCities);
  CCFloat = thrust::device_vector<float>(numCities*numCities);
  AUnsignedInt = thrust::device_vector<unsigned int>(numAnts);
  //Random numbers
  ARandom = thrust::device_vector<unsigned int>(numAnts);
  ACRandom = thrust::device_vector<unsigned int>(numAnts*numCities);
}

//initialize: Initializes data, creates maps and keys, performs standard ACO initialization steps etc.
void Colony::initialize()
{
  //seed the random numbers
  thrust::transform(thrust::make_counting_iterator(0),
```

```
        thrust::make_counting_iterator(numAnts*numCities),
        thrust::make_counting_iterator(time(NULL)),ACRandom.begin(),thrust::multiplies<int>());
thrust::transform(thrust::make_counting_iterator(0),
        thrust::make_counting_iterator(numAnts),
        thrust::make_counting_iterator(time(NULL)),
        ARandom.begin(),
        thrust::multiplies<int>());
//constant seeds
//thrust::transform(thrust::make_counting_iterator(0),thrust::make_counting_iterator(numAnts*numCities),thrust::make_constant_iterator(1),ACRandom.begin(),thrust::multiplies<int>());
//thrust::transform(thrust::make_counting_iterator(0),thrust::make_counting_iterator(numAnts),thrust::make_constant_iterator(1),ARandom.begin(),thrust::multiplies<int>());
thrust::transform(ACRandom.begin(),
        ACRandom.end(),
        ACRandom.begin(),
        randStep());
thrust::transform(ARandom.begin(),
        ARandom.end(),
        ARandom.begin(),
        randStep());
//create maps and keys
//CCKey
thrust::sequence(ACInt.begin(),
       ACInt.begin() + numCities,
       0,
       numCities);
thrust::scatter(thrust::make_constant_iterator(1,0),
      thrust::make_constant_iterator(1,numCities),
      ACInt.begin(),
      CCKey.begin());
thrust::inclusive_scan(CCKey.begin(),
       CCKey.end(),
       CCKey.begin());
//ACMapF
thrust::sequence(ACMapF.begin(),
       ACMapF.end(),
       0,
       numCities);
//ACMapL
thrust::transform(ACMapF.begin(),
```

```
        ACMapF.end(),
        thrust::make_constant_iterator(numCities-1),
        ACMapL.begin(),
        thrust::plus<int>());
//ACKey
thrust::scatter(thrust::make_constant_iterator(1,0),
    thrust::make_constant_iterator(1,numAnts),
    ACMapF.begin(),
    ACKey.begin());
thrust::inclusive_scan(ACKey.begin(),
     ACKey.end(),
     ACKey.begin());
thrust::transform(ACKey.begin(),
        ACKey.end(),
        thrust::make_constant_iterator(-1),
        ACKey.begin(),
        thrust::plus<int>());
//distMap
thrust::fill(distMap.begin(),
         distMap.end(),
         1);
thrust::inclusive_scan_by_key(ACKey.begin(),
        ACKey.end(),
        distMap.begin(),
        distMap.begin());
thrust::scatter(thrust::make_constant_iterator(0,0),
    thrust::make_constant_iterator(0,numAnts),
    ACMapL.begin(),
    distMap.begin());
thrust::transform(ACKey.begin(),
        ACKey.end(),
        distMap.begin(),
        distMap.begin(),
        saxpy_functor(numCities));
//ARepeatCMap
thrust::exclusive_scan_by_key(ACKey.begin(),
        ACKey.end(),
        thrust::make_constant_iterator(1),
        ARepeatCMap.begin());
//ACO Initialize
computeParameters();
thrust::fill(pheromones.begin(),
         pheromones.end(),
         initialPheromone);
computeProbabilities();
```

```
}

//forage: Main ACO loop. Performs the solution constructruction
    step, then updates distances, pheromones, probabilities.
void Colony::forage()
{
  //initialize variables and select start cities
  toVisit.assign(ARepeatCMap.begin(),ARepeatCMap.end());
  ACInt2.assign(ACKey.begin(),ACKey.end());
  thrust::fill(antVisits.begin(),
         antVisits.end(),
         0);
  thrust::sequence(tourMap.begin(),
        tourMap.end(),
        0,
        numCities);
  thrust::transform(ARandom.begin(),
         ARandom.end(),
         ARandom.begin(),
         randStep());
  thrust::transform(ARandom.begin(),
         ARandom.end(),
         thrust::make_constant_iterator(numCities),
         thrust::make_permutation_iterator(antTours.begin(),
            tourMap.begin()),
         thrust::modulus<unsigned int>());
  thrust::transform(ACMapF.begin(),
         ACMapF.end(),
         thrust::make_permutation_iterator(antTours.begin(),
            tourMap.begin()),
         AInt.begin(),
         thrust::plus<int>());
  for(int x = 1; x < numCities; x++)
    {
      //update antVisits
      thrust::scatter(thrust::make_constant_iterator(x,0),
          thrust::make_constant_iterator(x,numAnts),
          AInt.begin(),
          antVisits.begin());
      thrust::remove_if(thrust::make_zip_iterator(thrust::
          make_tuple(toVisit.begin(),
                     ACInt2.begin())),
      thrust::make_zip_iterator(thrust::make_tuple(toVisit.begin
          () + ((numCities-x + 1) * numAnts),
```

```
                        ACInt2.begin()+ ((numCities-x + 1) *
                            numAnts))),
        antVisits.begin(),isX(x));
        //get probabilities
        thrust::transform(thrust::make_permutation_iterator(thrust
            ::make_permutation_iterator(antTours.begin(),tourMap.
            begin()),ACInt2.begin()),
        thrust::make_permutation_iterator(thrust::
            make_permutation_iterator(antTours.end(),tourMap.end())
            ,ACInt2.begin()+ ((numCities-x) * numAnts)),
        toVisit.begin(),
        ACInt.begin(),
        saxpy_functor(numCities));
        //update tour map
        thrust::transform(tourMap.begin(),
        tourMap.end(),
        tourMap.begin(),
        unaryPlus(1));
        //update random numbers
        thrust::transform(ACRandom.begin(),
        ACRandom.begin() + ((numCities-x + 1) * numAnts),
        ACRandom.begin(),
        randStep());
        //select cities
        thrust::reduce_by_key(ACInt2.begin(),
            ACInt2.begin()+ ((numCities-x) * numAnts),
            thrust::make_zip_iterator(thrust::make_tuple(thrust::
                make_counting_iterator(0),
                      thrust::make_permutation_iterator(
                          probabilities.begin(),ACInt.begin()),
                      ACRandom.begin())),
            thrust::make_discard_iterator(),
            thrust::make_zip_iterator(thrust::make_tuple(AInt.
                begin(),
                      AFloat.begin(),
                      AUnsignedInt.begin())),
            thrust::equal_to<int>(),
            treeSelect());
        thrust::gather(AInt.begin(),
           AInt.end(),
           toVisit.begin(),
           thrust::make_permutation_iterator(antTours.begin(),
               tourMap.begin()));
    }
  computeAntDistances();
```

```
  updatePheromones();
  computeProbabilities();
}

//computeAntDistances: Computes the distances of each ant's tour
    , then updates records.
void Colony::computeAntDistances()
{
  //compute distances
  thrust::transform(antTours.begin(),
        antTours.end(),
        thrust::make_permutation_iterator(antTours.begin(),
            distMap.begin()),
        ACInt.begin(),
        saxpy_functor(numCities));
  thrust::gather(ACInt.begin(),
      ACInt.end(),
      distances.begin(),
      ACFloat.begin());
  thrust::reduce_by_key(ACKey.begin(),
       ACKey.end(),
       ACFloat.begin(),
       thrust::make_discard_iterator(),
       antDistances.begin());
  //update bests
  int i = thrust::min_element(antDistances.begin(),
            antDistances.end()) - antDistances.begin();
  thrust::gather(thrust::make_counting_iterator(i*numCities),
      thrust::make_counting_iterator((i+1)*numCities),
      antTours.begin(),
      iterBestTour.begin());
  iterBestDist = antDistances[i];
  if(iterBestDist < globBestDist){
    reps = 0;
    globBestDist = iterBestDist;
    globBestTour.assign(iterBestTour.begin(),iterBestTour.end())
        ;
  }else{
    reps++;
  }
}

//greedyDistance: Returns the value of a simple greedy solution
    starting at city 0.
float Colony::greedyDistance()
```

```
{
  float distance;
  int i = 0;
  int init = i;
  thrust::device_vector<int> visits(numCities);
  thrust::fill(visits.begin(),
         visits.end(),
         1);
  thrust::device_vector<float> Cfloat(numCities);
  int j;
  for(int x = 1; x < numCities; x++){
    visits[i] = 0;
    thrust::transform(visits.begin(),
          visits.end(),
          thrust::make_permutation_iterator(distances.begin(),
              thrust::make_counting_iterator(i*numCities)),
          Cfloat.begin(),
          thrust::divides<float>());
    j = thrust::max_element(Cfloat.begin(),
          Cfloat.end()) - Cfloat.begin();
    distance += distances[numCities * i + j];
    i = j;
  }
  distance += distances[numCities * i + init];
  return distance;
}

//computeProbabilities: Computes the probabilities from the
    distances and pheromones.
void Colony::computeProbabilities()
{
  thrust::transform(pheromones.begin(),
        pheromones.end(),
        distances.begin(),
        probabilities.begin(),
        prob_functor(beta));
}

void Colony::setBeta(float newBeta)
{
  beta = newBeta;
}

void Colony::setRho(float newRho)
{
```

```
  rho = newRho;
}

double Colony::getBeta()
{
  return beta;
}

double Colony::getRho()
{
  return rho;
}

int Colony::getNumAnts()
{
  return numAnts;
}

double Colony::getIterBestDist()
{
  return iterBestDist;
}

double Colony::getGlobBestDist()
{
  return globBestDist;
}

int Colony::getReps()
{
  return reps;
}

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
//
//    Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
//    Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following
```

```
    disclaimer in the documentation and/or other materials provided with the distribution.
//     Neither the name of Excellants nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



/***************************************
 * TSPReader.h                          *
 * Dustin Tauxe and Peter Ahrens        *
 * Reads .tsp files                     *
 ***************************************/

#ifndef TSPREADER_H
#define TSPREADER_H
#include <iostream> // Used for command line I/O
#include <fstream>  // Used for file Input
#include <string>
#include <math.h>
#include <cctype>
#include <float.h> // Used to find maximum float
#include <thrust/host_vector.h>
using namespace std;

//TSPReader: Used to read .tsp files.
class TSPReader
{
  string name;        // TSP name - max length 16
  int numCities;      // Number of cities (dimension)
```

```
  float* Xcoords;    // X coords
  float* Ycoords;    // Y coords
  string* cityNames;
  thrust::host_vector<float> distances; // Distances between cities

 public:
  //Constructors/Destructors
  TSPReader(){}
  ~TSPReader();


  bool read(char* filen); // Reads a given tsp file and extracts data.
  string getName();
  float* getXcoords();
  float* getYcoords();
  int getNumNodes();
  thrust::host_vector<float> getDistances();
 private:
  void calculateDistances(); // Calculates distances on the CPU.
};

#endif

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
//
//    Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
//    Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
//    Neither the name of Excellants nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
//
```

```
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


/**************************************
 * TSPReader.cu                        *
 * Dustin Tauxe and Peter Ahrens       *
 * Reads .tsp files                    *
 **************************************/

#include "TSPReader.h"

//destructor
TSPReader::~TSPReader()
{
  delete[] cityNames;
  delete[] Xcoords;
  delete[] Ycoords;
}

//read: Reads a given tsp file and extracts data.
bool TSPReader::read(char* filen)
{
  ifstream infile(filen, ios_base::in);
  if(!infile){
    cout << "\n" << "Unable to open file: " << filen << "\n";
    return false;
  }
  string line;
  string tag;
  string value;
  while(infile.good()){
    getline(infile,line);
    if(line.length() > 1){
```

```
      if(!isprint(line[line.length()-1])) line.erase(line.length()-1,1);
      if(line.find(":") != string::npos){
  tag = line.substr(0,line.find(":"));
  value = line.substr(line.find(":") + 1, line.length() - line.find(":") - 1);
      }else{
  tag = line;
  value = "";
      }
      while(tag.find(" ") != string::npos){
  tag.replace(tag.find(" "),1,"");
      }
      while(value.find(" ") != string::npos){
  value.replace(value.find(" "),1,"");
      }
      if(tag == "NAME"){
  name = value;
      }else if(tag == "TYPE"){
  if(value != "TSP" && value != "STSP"){
    cout << "\n" << "Invalid problem type: " << value << "\n";
    return false;
  }
      }else if(tag == "DIMENSION"){
  numCities = atoi(value.c_str());
      }else if(tag == "EDGE_WEIGHT_TYPE"){
  if(value != "EUC_2D"){
    cout << "\n" << "Invalid edge weight type: " << value << "\n";
    return false;
  }
      }else if(tag == "NODE_COORD_SECTION"){
  //Set coord arrays to appropriate lengths
  cityNames = new string [numCities];
  Xcoords = new float [numCities];
  Ycoords = new float [numCities];
  for(int i = 0; infile.good() && i < numCities; i++){
    getline(infile,line);
    if(!isprint(line[line.length()-1])) line.erase(line.length()-1,1);
    if(line == "EOF"){
      return false;
    }
    cityNames[i] = line.substr(0,line.find(" "));
```

```
    Xcoords[i] = atof(line.substr(line.find(" ") + 1, line.
        find_last_of(" ") - line.find(" ") - 1).c_str());
    Ycoords[i] = atof(line.substr(line.find_last_of(" ") + 1,
        line.length() - line.find_last_of(" ") - 1).c_str());
  }
      }

    }
  }
  calculateDistances();
  return true;
}

//calculateDistances: Calculates distances on the CPU.
void TSPReader::calculateDistances(){
  distances = thrust::host_vector<float> (numCities*numCities);
  float k;
  for(int i = 0; i < numCities; i++){
    for(int j = 0; j < numCities; j++){
      k = sqrt(pow(Xcoords[i]-Xcoords[j],2)+pow(Ycoords[i]-
          Ycoords[j],2));
      if(i == j)
  k = std::numeric_limits<float>::max();
      distances[i * numCities + j] = k;
    }
  }
}

string TSPReader::getName()
{
  return name;
}

float* TSPReader::getXcoords()
{
  return Xcoords;
}

float* TSPReader::getYcoords()
{
  return Ycoords;
}

int TSPReader::getNumNodes()
{
```

```
  return numCities;
}

thrust::host_vector<float> TSPReader::getDistances()
{
  return distances;
}

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or
    without modification, are permitted provided that the
    following conditions are met:
//
//    Redistributions of source code must retain the above
    copyright notice, this list of conditions and the following
    disclaimer.
//    Redistributions in binary form must reproduce the above
    copyright notice, this list of conditions and the following
    disclaimer in the documentation and/or other materials
    provided with the distribution.
//    Neither the name of Excellants nor the names of its
    contributors may be used to endorse or promote products
    derived from this software without specific prior written
    permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
    CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
    INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
    MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
    CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
    SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
    NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
    HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
    CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
    OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE
    , EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


/***************************************
 * Comm.h                              *
 * Peter Ahrens                        *
```

```
 * Communicates over a pipe              *
 ****************************************/

#ifndef COMM_H
#define COMM_H
#include <iostream>
#include <sstream>
#include <iomanip>
#include <unistd.h>
#include <stdlib.h>
#include <string>
#include <cctype>
using namespace std;

//Comm: A class used to simplify the data transfer over a pipe.
class Comm
{
 public:
  Comm(int read, int write); //constructor: Takes as arguments the neccesary pipes to work with.
  ~Comm();
  string recieve(); //recieve: Looks for data on the pipe. If there is some, it is returned. If not, an empty string is returned.
  bool send(string message); //send: Puts the given data on the pipe
 private:
  int tagLength;
  int readPipe;
  int writePipe;
  string intToString(int t, int padding); //intToString: Converts an int to a specified size string with 0s as padding.
};
#endif

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
//
//     Redistributions of source code must retain the above copyright notice, this list of conditions and the following
```

```
disclaimer.
//     Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
//     Neither the name of Excellants nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


/***************************************
 * Comm.cpp                             *
 * Peter Ahrens                         *
 * Communicates over a pipe             *
 ***************************************/

#include "Comm.h"

//constructor: Takes as arguments the neccesary pipes to work with.
Comm::Comm(int newReadPipe,int newWritePipe)
{
  readPipe = newReadPipe;
  writePipe = newWritePipe;
  tagLength = 8;
}

Comm::~Comm() //Destructor.
{}

```

```cpp
//recieve: Looks for data on the pipe. If there is some, it is returned. If not, an empty string is returned.
string Comm::recieve()
{
  char tag[tagLength];
  int rv = read(readPipe,tag,tagLength);
  if(rv < 0){
    cout << "Read Error 1";
    exit(1);
  }
  if(rv == 0){
    return string("");
  }
  string inputTag = tag;
  int toRead = atoi(inputTag.substr(0,tagLength).c_str());
  char charIn[toRead];
  if(read(readPipe,charIn,toRead) < 0){
    cout << "Read Error 2";
    exit(1);
  }
  string output = charIn;
  output = output.substr(0,toRead);
  return output;
}

//send: Puts the given data on the pipe
bool Comm::send(string message)
{
  message = intToString(message.length(),tagLength) + message;
  return(write(writePipe,message.data(),message.length()) > 0);
}

//intToString: Converts an int to a specified size string with 0s as padding.
string Comm::intToString(int t, int padding)
{
  std::ostringstream oss;
  oss << setfill('0') << setw(padding) << t;
  return oss.str();
}

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
//
```

```
//Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
//
//     Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
//     Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
//     Neither the name of Excellants nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


/****************************************
 * Writer.h                             *
 * Dustin Tauxe and Peter Ahrens        *
 * Writes output to file and stdout     *
 ****************************************/

#ifndef WRITER_H
#define WRITER_H
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
#include <time.h>
using namespace std;
```

```

class Writer
{
 public:
  Writer(); // Sets defaults.
  Writer(char* filen); // Sets defaults and opens the given file.
  ~Writer();
  bool setFile(char* filen); // Tries to open given file. If it does, it is changed to writing mode.
  void writeHeader(float beta, float rho, int numAnts,string ACO, string TSPName); // Writes a header to the file and (if in writing mode) to the file.
  void write(int iter, double iterBest, double globBest, double time, double iterTime); // Writes a standard line of output to stdout and (if in writing mode) to the file.
 private:
  char* fileName;
  ofstream f; // this is the file
  char* temp; // scratch
  bool writing;
};

#endif

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
//
//    Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
//    Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
//    Neither the name of Excellants nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
//
```

```cpp
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



/**************************************
 * Writer.cpp                         *
 * Dustin Tauxe and Peter Ahrens      *
 * Writes output to file and stdout   *
 **************************************/

#include "Writer.h"

//Constructor: Sets defaults.
Writer::Writer()
{
  writing = false;
}

//Constructor: Sets defaults and opens the given file.
Writer::Writer(char* filen)
{
  setFile(filen);
}

Writer::~Writer()//Destructor.
{
  delete[] fileName;
}

//setFile: Tries to open given file. If it does, it is changed to writing mode.
bool Writer::setFile(char* filen)
{
```

```
  fileName = filen;
  writing = true;
  f.open(filen,ios_base::app);
  if (!f){
    writing = false;
  }
  return writing;
}

//writeHeader: Writes a header to the file and (if in writing mode) to the file.
void Writer::writeHeader(float beta, float rho, int numAnts, string ACO, string TSPName)
{
  time_t rawtime;
  time ( &rawtime );
  if(writing){
    f << "\n" << "Date: " << ctime (&rawtime) <<
      "TSP: " << TSPName << "\n" <<
      "ACO: " << ACO << "\n" <<
      "numAnts: " << numAnts << "\n" <<
      "Alpha: 1 " << "Beta: " << beta << " Rho: " << rho << "\n" <<
      "Iteration, Iteration_Best, Global_Best, Time, Iteration_Time\n" << flush;
  }
  cout << "\n" << "Date: " << ctime (&rawtime) <<
    "TSP: " << TSPName << "\n" <<
    "ACO: " << ACO << "\n" <<
    "numAnts: " << numAnts << "\n" <<
    "Alpha: 1 " << "Beta: " << beta << " Rho: " << rho << "\n" <<
    std::left << setw(10) << "Iteration"<< setw(10) << "Iter_Best" << setw(10) << "Glob_Best" << setw(10) << "Time" << setw(10) << "Iter_Time" << "\n";
}

//write: Writes a standard line of output to stdout and (if in writing mode) to the file.
void Writer::write(int iter, double iterBest, double globBest, double time, double iterTime)
{
  if(writing){
    f << iter << "," << iterBest << "," << globBest << "," << time << "," << iterTime << "\n" << flush;
```

```
  }
  cout << std::left << setw(10) << iter << setw(10) << iterBest
     << setw(10) << globBest << setw(10) << time << setw(10) <<
     iterTime << "\n";
}

//Copyright (c) 2012, Peter Ahrens
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or
   without modification, are permitted provided that the
   following conditions are met:
//
//    Redistributions of source code must retain the above
   copyright notice, this list of conditions and the following
   disclaimer.
//    Redistributions in binary form must reproduce the above
   copyright notice, this list of conditions and the following
   disclaimer in the documentation and/or other materials
   provided with the distribution.
//    Neither the name of Excellants nor the names of its
   contributors may be used to endorse or promote products
   derived from this software without specific prior written
   permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
   CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
   INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
   MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
   DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
   CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
   NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
   LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
   HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
   CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
   OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE
   , EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

11 Acknowledgements

Thanks so much to our mentors and to Mr. Goodwin for supporting and guiding us along the past couple years of computing. We have come a long way.

References

[1] Peter's Computer: Asus G53J, Intel i7-74OQM 1.73GHz, 8GB Memory, Nvidia GeForce GTX 460M ; VRAM: 1.5GB, running Ubuntu Linux 11.10.

[2] Dustin's Computer: Asus P5QL pro, Intel Core2Quad Q8400 2.66GHz, 8GB Memory, Nvidia GTX 560 Ti Overclocked to 850 MHz ; Vram: 1GB, running Ubuntu Linux 11.10.

[3] *National TSP*, `http://www.tsp.gatech.edu/world/countries.html`.

[4] Peter Ahrens, *Excellants*, `code.google.com/p/excellants`.

[5] Peter Ahrens, Dustin Tauxe, and Stephanie Djidjev, *Brilliants*, `http://challenge.nm.org/archive/10-11/finalreports/56.pdf`.

[6] Guy Blelloch, *Vector Models for Parallel Computing*, Ph.D. thesis, The Massachusetts Institute of Technology, 1988.

[7] Jose M. Cecilia, Jose M. Garcia, Andy Nisbet, Martyn Amos, and Manuel Ujaldon, *Enhancing Data Parallelism for Ant Colony Optimisation on GPUs*, Journal of Parallel and Distributed Computing (2012).

[8] Marco Dorigo and Thomas Stutzle, *Ant Colony Optimization*, The MIT Press, Cambridge, Massachusetts, 2004.

[9] Robert Gentleman and Ross Ihaka, *The R Project For Statistical Computing*, `http://www.r-project.org/`.

[10] Thomas Hammerl, *Ant Colony Optimization for Tree and Hypertree Decompositions*, Vienna University of Technology (2009).

[11] Jared Hoberock and Nathan Bell, *Thrust: A Parallel Template Library*, `http://www.meganewtons.com/`, 2012, Version 1.6.0.

[12] Gerhard Reinelt, *TSPLIB*, `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

[13] Bjarne Stroustrup, *The C++ Programming Language*, Addison-Wesley, Boston, Massachusetts, 1997.