A Parallel Implementation of Ant Colony Optimization for TSP based on MapReduce Framework

Anuraj Mohan
Assistant Professor
Dept of CSE,
NSS College of Engineering,
Palakkad, Kerala, India

Remya G
M.E Scholar
Maharaja Institute of Technology,
Coimbatore,
TamilNadu, India

ABSTRACT
The Travelling Salesman Problem (TSP) is one of the most widely and deeply studied problems in optimization. It belongs to the category of NP Complete problems in which no polynomial time solution is possible unless P=NP. Various researches are done on finding efficient heuristics to get provably optimal and near to optimal results to TSP. Having the push towards grid and cloud computing, it will become more necessary to adopt existing algorithms to distributed computing frameworks like MapReduce. The aim is to parallelize the ant colony optimization algorithm for solving TSP over the Apache Hadoop MapReduce framework. The paper also compares the results of the parallel implementation with the performance of the serial version of the ACO algorithm.

Keywords
MapReduce, Ant Colony Optimization, Parallel Computing

1. INTRODUCTION
In its most basic form, the Travelling Salesman Problem (TSP) is defined as finding a minimum cost Hamiltonian cycle in a graph. We are reduced to the Euclidian Travelling Salesman Problem when the edge weights on the graph form a Euclidian metric. The problem was first formulated as a mathematical problem in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally complex, a large number of heuristics and exact methods are known, so that some instances with tens of thousands of cities can be solved. The TSP has several applications even in its purest formulation such as planning, logistics, and the manufacture of microchips. Slightly modified, it can be considered as a sub-problem in many challenging areas, such as DNA sequencing.

In the theory of computational complexity, the TSP can be considered as a decision problem where given a length L, the task is to decide whether any tour is shorter than L, which belongs to the class of NP-complete problems. Thus, it is likely that the worst case execution time for any algorithm for the TSP increases exponentially with the number of cities. Generally, for a TSP solver, one either tries to obtain a provably optimal solution or one tries to get a solution as close to the optimum as possible without actually proving that the solution is close to the optimum. While the former goal has an advantage in that it gives a guarantee of the quality of the solution, it is generally very slow and infeasible to apply to instances of a large size. Thus we opt for the second goal. The system uses the Ant Colony Optimization algorithm that simulates the way ants find the shortest route to a food source. There already exist sequential versions of this algorithm. The aim is to parallelize this algorithm over Hadoop MapReduce and thereby improve its performance. Ant colony optimization algorithms have been applied to many combinatorial optimization problems, ranging from quadratic assignment to protein folding or routing vehicles and a lot of derived methods have been adapted to dynamic problems in real variables, stochastic problems, multi-targets and parallel implementations.

MapReduce is a distributed computing framework developed at Google for the analytics of large heterogeneous data that have been divided over many computers. Its programming model allows the user to neglect about many of the issues associated with distributed computing: splitting up and assigning the input to various systems, scheduling and running computation tasks on the available nodes and coordinating the necessary communication between tasks. Map Reduce deals with its input in terms of key-value pairs, which are generated from an input file by user-configurable rules. Map Reduce uses a very simple programming abstraction, which in its most general form requires its user to write only two functions - map and reduce. Hadoop is an open source framework which uses the MapReduce programming model for writing and running distributed applications that process large amounts of data.

2. RELATED WORK
Ant colony optimization has been formalized into a met heuristic for combinatorial optimization problems by Dorigo and co-workers [1], [2]. Various adaptations includes an algorithm based on the basis of the ant evolution rules [3], dynamic control of solution construction and mergence of local search ([4], [5], [6]), max-min ant system [7] and a strategy to partition artificial ants into two groups: scout ants and common ants [8] are studied to improve the quality of the final solution and lead to speedup of the algorithm .The possibility of parallelizing the Traveling Salesman Problem[9] over the MapReduce architecture is also studied. A MapReduce Max-Min Ant System [10] implementation based on the MapReduce parallel programming model is also proposed.

3. PROPOSED SYSTEM
The proposed system is based on parallelizing the sequential version of Ant Colony Optimization algorithm over a cluster to solve TSP. This is done based upon the MapReduce framework. The design of the map reduce job basically involves how the map and reduce functions are implemented and how the Ant Colony Optimization algorithm has been modified so as to be able to run in a MapReduce framework. As far as the map and reduce are concerned, the entire job is divided into one initial map reduce phase, several intermediate map reduce phases and a final map reduce phase. Each phase produces a result which is expected to be closer to the optimum value. The number of stages is user controlled.

In the sequential implementation of the Ant Colony Optimization algorithm each artificial ant is placed in randomly chosen node. The graph initially contains the initial pheromone levels. The ants initially traversing the graph select the path they travel on the basis of the attractiveness (reciprocal of the distance of the edge) of an edge. As they traverse through the graph they deposit the pheromones, thereby making different pheromone levels at different edges of the graph. Now the consequent ants choose their path by considering the amount of the pheromones deposited and the attractiveness of each path. Also the pheromone deposits in an edge evaporate or decrease when this edge is not used. In this way the pheromone deposit goes on increasing in the path through which most ants travel and becomes the path for the travelling salesman with a distance close to the optimal solution.

The system modifies the existing sequential version of the ACO algorithm to fit into the MapReduce framework of Hadoop. The basic idea is to give copies of the TSP instances to multiple mappers which will work in parallel to produce the results. In Hadoop, all the mappers must work on independent data and so the sharing of pheromone updates is not possible between the mappers. So, in order to implement the algorithm in Hadoop, the concept of chaining MapReduce jobs is used. Chaining means executing multiple MapReduce jobs one after the other. In this method, there will be multiple stages of MapReduce. In the first stage, the mappers will work on the input data using the initial pheromone levels. Once the mappers have finished their work, they will pass on the calculated routes to the reducer which will then update the pheromone values. The results are then passed on to the next stage. Therefore, the mappers of the intermediate stages will get to work on the updated pheromone values. The number of intermediate stages required can be chosen by the user. Once all the stages are complete, the final stage will give the best result obtained among all stages. In order to further improve the performance of the parallel version, multiple ants inside a single mapper is implemented. These ants will work just like in the sequential version as they will have access to the pheromone updates produced by each other instantaneously.

4. STAGES OF IMPLEMENTATION

4.1 Initial stage

The basic algorithm for the initial stage of the MapReduce implementation is as follows:
In the first stage, the input to the MapReduce job is the TSPLIB input file that contains the Euclidian 2D coordinates of a TSP problem. Once the job has started, the Job Client returns the input splits generated from the input file. Each map task will be given exactly n lines of the input. Our input file contains m lines, where each line holds the single TSP instance. The functioning of the mapper is as follows.

- i) Each mapper obtains the key-value pair whose value is one instance of the TSP problem.
- ii) The mapper then parses the text item and retrieves the coordinates. It then constructs the graph.
- iii) Then the pheromone array is initialized and ‘p’ ants (value p is specified by the user) are generated.
- iv) Each ant uses the classic ACO algorithm to find routes in the graph. Within a mapper, the updates to the pheromone array are available to all ants instantaneously.
- v) After all the ants have finished their work, the mapper initializes the TspInput object with all the paths generated by the ants.
- vi) Finally it produces a key-value pair. All mappers use the same key, so that all key-value pairs go to the same reducer.

The reducer takes the key-value pairs generated by all the mappers and iterates through them. During this process, the reducer performs updates to the global pheromone array which will be passed on to the next stage of the MapReduce. Also, the reducer calculates the best route of each stage. Once the reducer finishes its work, it writes the key-value pairs into a binary file. This format stores the key-value pairs so that subsequent stages in the chained MapReduce jobs can process them faster. Like the original file, the output file from the first stage also contains replicated data. Also, it is important to note that the output of the first stage contains the updated pheromone table along with the original TSP input.
The map and reduce methods of the intermediate and final stages are almost similar to these. The only difference is in the types of the key-value pairs.

4.2 Intermediate Stage

The output obtained from the reducer of the initial stage is fed to the intermediate stages. It uses a modified mapper and reducer as they are different from the initial map reduces functions as they work on input in the binary form.

4.3 Final Stage

The input obtained from the previous intermediate stages is in binary format and the output should be in text format as they have to be understood by the user. The final reducer, as the previous reducers compares the solutions of different maps in the stage and finds the best possible one and writes these to a file for the user to access.

5. SYSTEM PARAMETERS

5.1 Input Parameters

There are many input parameters that the user has control over while using the algorithm:

5.1.1 Number of mappers per stage

This parameter determines the number of mappers that will be executed in each stage of the MapReduce program. This value determines the amount of parallelism in each stage as the mappers are executed in parallel.

5.1.2 Number of ants per mapper

This value controls how many ants will function inside each mapper across all stages. Higher value means that each stage will have more pheromone updates to pass on to the next stage of the job. Higher values will also increase the amount of computation to be performed inside each mapper.

5.1.3 Number of stages

This value indicates the number of stages in the MapReduce chain. Each subsequent stage after the initial stage will have updated pheromone values for their ants to work with. This should make the computations of the ants more accurate as the number of stages increases.

5.2 Output

The two main outputs from our system are:

1. Time taken for the algorithm to execute
2. The shortest path computed by the algorithm

Since we are using a heuristic algorithm (Ant Colony Optimization), we will not be getting exact optimal solutions. However the algorithm gives approximate solutions that are quite close to the optimal solution. Usually the errors are between 2-5%.

6. COMPARISON OF PROPOSED SYSTEM AND CURRENT SYSTEM

The final system, which uses the Best of Breed parallelized algorithm, is compared with the original sequential implementation to check the difference in results and thereby conclude the optimization achieved by using the new algorithm. The test is divided into two parts, namely load comparison and quality comparison.

6.1 Load Comparison

Load comparison involves testing the system against the original algorithm for the same amount of computational load. The computational load in this system is the number of ants that we are executing in the system. Various tests were conducted to evaluate the ability of the parallelized system to handle large loads when compared to the sequential algorithm. Memory requirements were more in the sequential system when compared to the parallelized system as the memory footprint of this algorithm was fairly small compared to the requirements of the sequential system as the number of ants running in each stage was lesser than that in the sequential system.

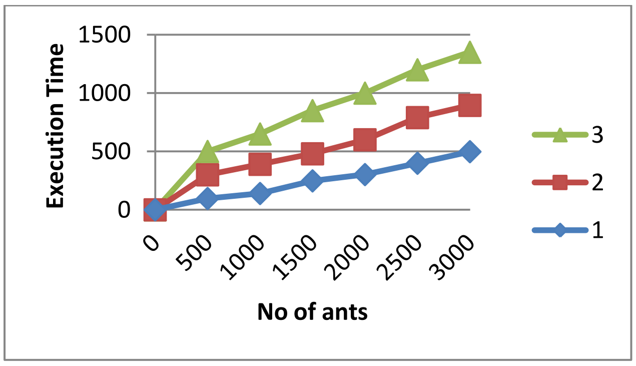

Figure 6.1 – Comparison of jobs with 1, 2 and 3 stages chained

The figure 6.1 depicts the running time of the algorithm for different number of ants represented on the X-axis and three different lines representing 1, 2 and 3 map stages. On increasing the number of ants the running time is not affected to the extent that increasing the number of stages would cost. The overhead on the system is large when the algorithm is parallelized as the network congestion greatly raises the time required for the computation to complete. Moreover, Hadoop is not designed to handle large number of inter-node communication, but rather for large amounts of single transactions. It must be noted that this system is efficient on a larger number of cities as the computational efficiency masks the overhead caused by the underlying Hadoop system.

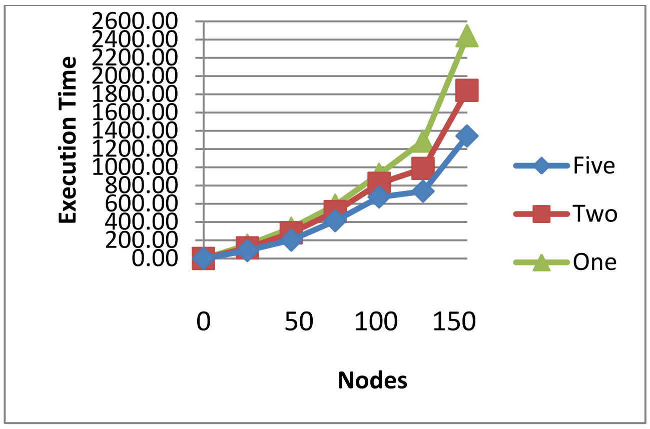

Figure 6.2–Variation in execution time against different inputs for one, two and five system cluster

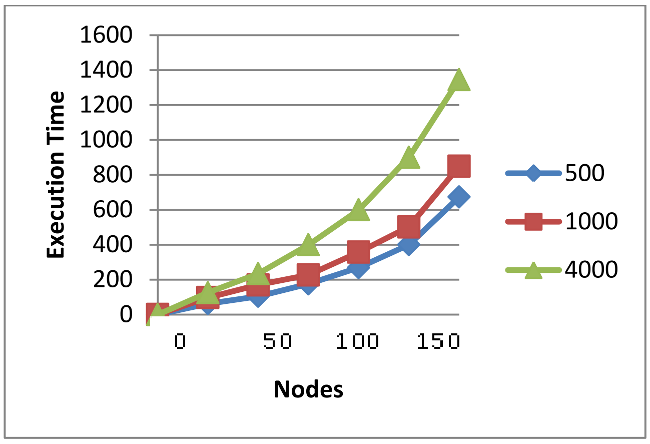

Figure 6.3 –Variation of execution time against different number of inputs for 500, 1000 and 4000 ants

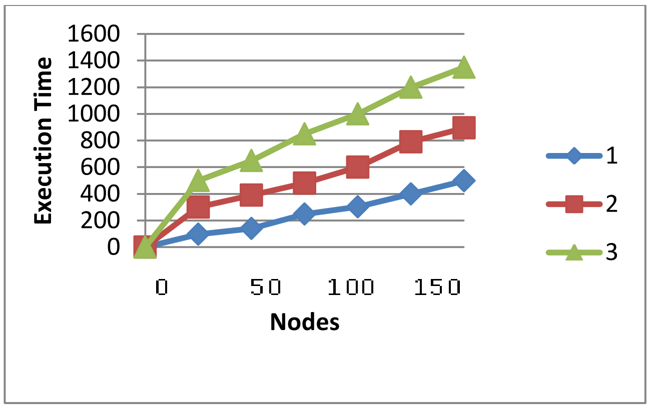

Figure 6.4–Variation of execution time against different number of inputs for 1, 2 and 3 MapReduce stages.

Figure 6.4 depicts the behavior of the algorithm when the number of MapReduce stages is increased. From the graph it can be observed that as the number of stages increases the running time of the algorithm also increases. It can be inferred from the previous results that for accurate result and optimum performance of the Ant Colony Optimization algorithm it is necessary to weigh the factors of time and error to carefully choose the number of stages of MapReduce, the number of ants (agents) per mapper and the number of nodes in the cluster which can produce the best result among all and with the most optimum performance. However, as the system is parallelized, we are able to run a larger amount of ants per system which makes the system more efficient when handling much larger loads than the sequential system. The sequential system is not able to handle as many ants as the parallelized system as it is a single processor with limited computational power. When the parallelized system is implemented on a large cluster, the computational power is greatly magnified and the system is able to perform computation much better.

6.2 Quality Comparison

The tested system is shown in Figure 6.5. The algorithm consistently generated results that were close to the optimal result. The advantage of this system over the sequential system is that we can control the quality to a larger extent than the sequential system. While in the sequential system, once the system begins execution, there is no means to control it until we get the final result.

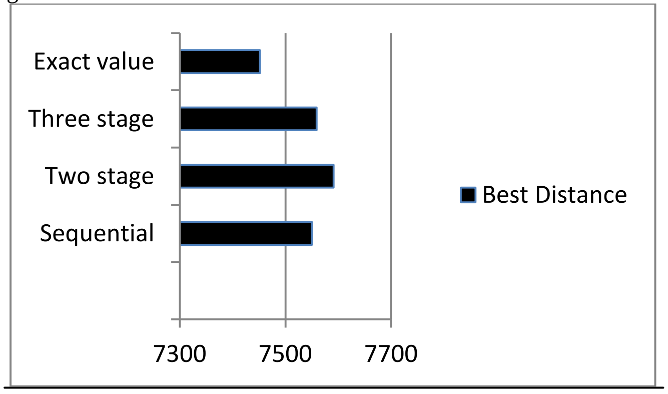

Figure 6.5 -Best Distance

The above graph depicts the fact that even though the sequential version of the Ant Colony Optimization algorithm produces a more accurate value with a lesser error, the algorithm when run of a multi-node cluster produces close to sequential values and even if the accuracy is not as good as the sequential version, the result is produced with a better running time.

7. CONCLUSION AND FUTURE SCOPE

The aim was to parallelize ACO over TSP problem and found by conducting several experiments with different parameters that, we can arrive at optimum values for each of the parameters used i.e. number of mappers per stage, the number of ants per mapper and the number of stages in the chained job. More studies can be done in order to find a better way to implement the pheromone updates. Because it was unable to share pheromone updates between mappers in Hadoop, in the current implementation all the pheromone updates are done by the single reducer in each stage. So, the reducer has too much work to do as the number of ants increases. This behavior can be improved by studying the possibility of having multiple reducers for pheromone updates and reducing the duplication of work. Finally, research can be done on more heuristic algorithms for other NP complete problems and try to parallelize them also.

8. REFERENCES

[1] M. Dorigo and G. Di Caro, “The Ant Colony Optimization meta-heuristic,” in New Ideas in Optimization, D. Corne et al., Eds., McGraw Hill, London, UK, pp. 11–32, 1999.

[2] M. Dorigo, G. Di Caro, and L.M. Gambardella, “Ant algorithms for discrete optimization,” Artificial Life, vol. 5, no. 2, pp. 137–172, 1999. W. Tsai and F. Tsai, “A New Approach for Solving Large Traveling Salesman Problem Using Evolutionary Ant Rules,” IJCNN 2002,IEEE.

[3] W. Tsai and F. Tsai, “A New Approach for Solving Large Traveling Salesman Problem Using Evolutionary Ant Rules,” IJCNN 2002,IEEE.

[4] H. Md. Rais, Z. A. Othman, and A. R. Hamdan, “Improved dynamic ant colony system (DACS) on symmetric Traveling Salesman Problem(TSP) ,” International Conference on Intelligence and Advanced Systems, IEEE, 2007.

[5] J. Han and Y. Tian, “An improved ant colony optimization algorithm based on dynamic control of solution construction and mergence of local search solutions,” Fourth International Conference on Natural Computation, IEEE, 2008.

[6] M. Colpan, “Solving geometric tsp with ants,” the pennsylvania state university, 2005

[7] T. Stutzle and H. H. Hoos. “MAX-MIN ant system and local search for the traveling salesman problem,” in IEEE Int’l Conf on Evolutionary Computation. Indianapolis: IEEE Press, 1997.309~314.

[8] R. Gan, Q. Guo, H. Chang, and Y. Yi, “Improved ant colony optimization algorithm for the traveling salesman problems,” Journal of Systems Engineering and Electronics, April 2010

[9] Siddhartha Jain Matthew Mallozzi “Parallel Heuristics for TSP on MapReduce” Brown University – CSCI 2950-u – Fall 2010

[10] Qing Tan, Qing He, and Zhongzhi Shi “Parallel Max-Min Ant System Using MapReduce” ICSI 2012, Part I, LNCS 7331, pp. 182–189, 2012.

[11] Sanjeev Arora. “Polynomial-time Approximation Schemes for Euclidean TSP and other Geometric Problems”. Journal of the ACM 45(5), 753–782, 1998.

[12] .http://hadoop.apache.org/.docs/r1.2.1/mapred_tutorial.htm l

[13] TSPLIB: A Library of Sample Instances for the TSP http://comopt.ifi.uni-heidelberg.de/software/TSPLIB/