Parallel EigenAnt algorithm based on MPI by ACO approach

Najmeh Damghani1*, Vahid Sattari Naeini2

1 Department of Information Technology, Payame Noor University, Bam, Iran
2 Department of Computer Engineering, Shahid Bahonar University of Kerman, Kerman, Iran

Abstract
Parallelization is among the best approaches that can be used to enhance the performance in heuristic algorithms. The EigenAnt algorithm model is an Ant Colony Optimization (ACO) that leads to the selection of the shortest paths. In this research, we propose a Parallel EigenAnt Algorithm (PEAA) using Ant Colony Optimization approach. The algorithm is implemented by Message Passing Interface (MPI) in a parallel form. It is possible to send or receive messages to a group of processes in collective communications. Parallel EigenAnt algorithm improves the speed to find a better solution by ants. Experimental results showed that the proposed parallel EigenAnt algorithm yields a significant improvement by finding the shortest path from a source to a destination. The results also showed that the suggested algorithm improves the execution time and efficiency. The performance of the proposed method improved the speedup, efficiency and quality of solution compared to Sequential EigenAnt algorithm.

Keywords: Ant Colony Optimization, Parallel Computing, EigenAnt Algorithm, Message Passing Interface.

1 Introduction
Ant Colony Optimization (ACO) is a bio-inspired optimization algorithm [1]. ACO has been used for the solving of numerous optimization problems. First, ACO algorithm called ant system that was proposed by Dorigo et al. and Ant Colony System (ACO) [2].

Production and hosting by ISPACS GmbH.

* Corresponding Author. Email address: najmeh00@gmail.com
Since then, a number of ACO algorithms have been developed including Max–Min AS (MMAS) [3]. Some strategies for parallelization have been reported recently. Jie et al. [4] proposed parallel ant colony optimization algorithm based on Message Passing Interface, which has the characteristics of coarse-granularity interacting multi ant colonies, partially asynchronous parallel implementation and also a new information exchange strategy. Relevant research of PACO has been focused on the use of MPI which is executed on clusters [5,6]. Many parallel algorithms have been proposed for the artificial Ant Colony algorithm. Parallel implementation of ACO [7] can obviously reduce the computational time for the large scale combinatorial optimization problem, and much of the previous researches in this field focuses on parallel implementation using MPI which is executed on clusters. Effective parallelization strategies for the Ant Colony Optimization metaheuristic on Graphics Processing Units (GPUs) presented in [8]. Kucukkoc et al. [9] proposed a Mixed-model parallel two-sided assembly line system that can be utilized to produce large-sized items in an inter-mixed sequence. Parallel ant colony optimization based metaheuristic proposed in [10] for solving the maximum-weight clique problem. Parallelization of algorithms has proven to be very powerful method in the case of population based algorithms like genetic algorithms, ant colony optimization and artificial bee colony algorithm. Asadzadeh [11] suggested a parallel artificial bee colony algorithm for the job shop scheduling problem by a dynamic migration strategy. In the method, artificial bee colony algorithm consists of several colonies that is located on different hosts of the network and is executed in various colonies in parallel manner. Damghani et al. [12] proposed Parallel implementation of Bees Algorithm based on Message Passing Interface. This parallel approach increases the Speedup. EigenAnt algorithm [13] employs Jayadeva et al. al’s EigenAnt algorithm to find the shortest path between a source and a destination, based on selective pheromone removal that occurs only on the path that is actually chosen for each trip. Ezzat et al. [14] proposed an Ant Colony Optimization Approaches for the Sequential Ordering Problem. In the suggested method, they present two approaches aimed at improving the performance of the ant colony optimization. EigenAnt is an ACO algorithm introduced by Jayadeva et al. [15]. It has been applied only to the problem of finding shortest paths, an easy problem that was meant just to provide context for studying of the convergence properties in the algorithm. Xiong et al. [16] employed an Improved Parallel Ant Colony Optimization Based on Message Passing Interface to improve the parallel efficiency. The proposed Parallel implementation in ACO can obviously reduce the computational time. The reminder of this paper is organized as follow. In Section 2, we describe the artificial ant colony algorithm. In Section 3, we describe the EigenAnt algorithm. Our proposed parallel EigenAnt algorithm and execution time in parallel method would be explained in section 4 and 5. Our conclusion is given in Section 6.

2 Artificial Ant Colony Algorithm

Ant algorithm operates in the way that the ants regularly create the candidate responses and they may store pheromones values. The created candidate responses start from a blank response by each ant. Then, they add the ant one by one to the response until the completion of a complete solution. After completing of the solution, the parts of the response, as a part of the found response, will be used further by other ants in the iteration stage. After initializing the pheromone ways and other parameters, a main loop is repeated until the preparing of termination condition (which can be a certain number of solutions or limitations on CPU time). In the main loop, the ants initially create possible solutions. Finally, the created solutions may be improved by applying a local search and finally, pheromone traces are updated. The pseudo-code of the ACO algorithm is given below:
2.1. Initialization Process
The first stage mainly involves the formation of the pheromone paths and initialization.

2.2. Construction Process
In the iteration stage, each ant produces solution for the problem according to the status transfer law. When all ants produce a solution, a nationwide law will be presented during updating of the pheromones in two phases: One is the evaporation phase, which is a part of pheromone, and the other one is strengthening and resonance that each ant leaves some pheromones on itself that corresponds to the value of the found solution. The process is repeated until the achieving of a stop condition.

3 EigenAnt Algorithm
EigenAnt uses a colony consisting of a single ant that is charged by finding the shortest path from a source to a destination. During every iteration $t$, it probabilistically chooses a path $i$ to follow according to the following equation (3.1):

$$\text{Pr(select } i) = \frac{\tau_i(t)}{\sum_j \tau_j(t)}$$

Where $\tau_i(t)$ is the amount of pheromone on the path $i$ at iteration $t$. After an ant selects a path $i$, the pheromone trail on that path is updated according to the following equation:

$$\tau_i(t+1) \leftarrow (1-a)\tau_i(t) + b d_i \sum_j \tau_j(t)$$

Where $a$ is the evaporation rate, $b$ is a weighting parameter, and $d_i = 1/Ct$, where $Ct$ is the cost of the solution found in iteration $t$. Jayadeva et al. [15] suggest default parameter settings of $a = \frac{1}{2}$ and $b = 1$, and report that the performance of the algorithm is not very sensitive to the choice of parameter values. They have also shown that the shortest path is the system’s unique stable equilibrium state. Even if the pheromone trails are initialized to favor longer paths, or if path lengths change dynamically during execution, the system will still tend towards the shortest path.

4 Parallel EigenAnt Algorithm
The use of parallel algorithms increases the program speed and optimal use of multi-processor computers power. In this article, MPI (message passing interface) was used for parallelization of EigenAnt algorithm. MPI is intended as a standard implementation in the "message passing" model of parallel computing. A parallel computation consists of a number of processes, each working on some local data. Each process has purely local variables, and there is no mechanism for any process to directly access the memory of another.
Sharing of data between processes takes place by message passing, that is, explicit sending and receiving data between processes. One of the methods is Master-Slave in which a process acts as a Master and others as the slaves. Master process is responsible for sending and receiving messages to the slave, messages harmonization and doing managerial tasks. Slaves are usually responsible for running the main part of algorithm and perform the same operations. In MPI, any available process in a communicator is determined by a unique number called rank. The rank, which is associated to each process, is an integer more than or equal to zero (≥0), as the rank of master process is zero in the method. In this algorithm first, master process produces $SN$ random solutions with $D$ parameter, where $D$ is the number of optimization parameters. Any solutions vector $x_i \ (i = 1, 2, ..., SN)$ is $D$-dimensional. The $SN$ is equal to the Pheromone paths. Then $SN$ solutions are divided among $P$ processes so that each processor takes $SN_P$ solution according to Equation (4.1):

$$SN_P = \frac{SN}{P}$$

Distribution of probabilities for choosing of a random path is obtained below:

$$p_j^i, j=1 \ldots n$$

(4.4)

The pseudo-code in the parallel EigenAnt algorithm is given below:

```plaintext
Begin
  - Initialize the path all $K$ ants and the pheromone concentration by the master process;
  - Initialize path $i$ with pheromone concentration and probability of choosing the $i$th path;
  - Initialize trip counter vector and total ant trip counter;
  Get the processor rank
    If rank = 0 (processor $P_0$), Choose the pheromone removal parameter $\alpha$ and the pheromone deposition parameter $\beta$;
    - Broadcast the (4.3) solution and Send the local-best-solution to master;
  While ($\sum_j J_j \leq J_{\text{max}}$) do
    - Choose a path randomly in accordance with the distribution of probabilities (4.4);
    - For all path receives the best solution from master processor;
    - Update pheromone one only on path $i$ according to (3.2);
    end if
    - Update path choice probabilities according to (4.5);
    - Update total ant trip counter: $t \leftarrow t + 1$
  end while
  - For all paths, replace (3.2) by the best solution found at the current iteration over all paths;
End.
```

Figure 2: Parallel EigenAnt Algorithm (PEAA)

The Update path choice probabilities are obtained using (4.5):

$$p_i^{t+1} = C_i^{t+1} \sum_{j=1}^n C_j^{t+1}, i = 1 \ldots n$$

(4.5)
MPI provides communication between the processes. The speedup factor is a measure to captures the relative benefit of solving a computational problem in parallel. The speedup factor of a parallel computation utilizing \( p \) processors is derived as the following ratio:

\[
S_p = \frac{T_S}{T_p}
\]  
\( (4.6) \)

Where \( T_s \) is the execution time taken to perform the computation on one processor, and \( T_p \) is the execution time needed to perform the same computation using \( p \) processors. The efficiency factor of a parallel computation is obtained below:

\[
\text{efficiency} = \frac{S_p}{p}
\]  
\( (4.7) \)

### 5 execution time in parallel method

In this article, parallel EigenAnt Algorithm that is an improved ACO algorithm has been evaluated by the MPI interface. The implementation of the parallel algorithm approach reduces the running time. As the processors numbers increased, the processing time would be reduced. It shows the improvement of algorithm to the EigenAnt sequential algorithm. To obtain better results, we compared the parallel with two interfaces of MPI and OpenMT that MPI showed better results. We implemented the parallel EigenAnt Algorithm in C++ language by the use of MPI, then compared the obtained results with sequential EigenAnt Algorithm. All experiments were carried out on a computer with an Intel Core 2 Duo P7450 2.13 GHz CPU and 4 GB's of RAM. For parallel EigenAnt algorithms we set \( m = 1 \), \( \alpha = 0.5 \) and \( \beta = 1 \). We set the pheromone trail on an edge \((i; j)\) to be equal to zero if it is infeasible (i.e.: it is a precedence constraint), a small constant if the edge length is zero, and to its length otherwise.

Speedup in MPI and OpenMP buydifferent threads numbers is shown i

<table>
<thead>
<tr>
<th>Number of Threads</th>
<th>Time of Sequential Execution(ms)</th>
<th>Time of MPI Execution(ms)</th>
<th>Time of OpenMP Execution(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>12148</td>
<td>8456</td>
<td>8734</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>8440</td>
<td>8728</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>8437</td>
<td>8716</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>8421</td>
<td>8701</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>8405</td>
<td>8685</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>4380</td>
<td>8662</td>
</tr>
</tbody>
</table>

Table 1: Results of experiment with different thread numbers, 200 iterations.

The results of the best time in the execution of algorithm for two interfaces are shown in Table 2. MPI interface shows the time improvement in the execution of parallel algorithm.
Table 2: The best value of different with processor numbers

<table>
<thead>
<tr>
<th>Interface Type</th>
<th>Algorithm</th>
<th>Best value (times)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI</td>
<td>EigenAnt</td>
<td>12148</td>
</tr>
<tr>
<td></td>
<td>PEigenAnt</td>
<td>8380</td>
</tr>
<tr>
<td>OpenMP</td>
<td>EigenAnt</td>
<td>12148</td>
</tr>
<tr>
<td></td>
<td>PEigenAnt</td>
<td>8662</td>
</tr>
</tbody>
</table>

Speedup of serial EigenAnt algorithm and parallel EigenAnt algorithm with different number of processors is shown in Table 2. Simulation results show that the speedup on the volume of data increases linearly as the number of processors is increased.

Table 3: Results of experiment with different thread numbers, 200 iterations.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Execution Time (sec)</th>
<th>Speedup</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23115</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>15219</td>
<td>1.75</td>
<td>0.87</td>
</tr>
<tr>
<td>3</td>
<td>7512</td>
<td>3.49</td>
<td>1.16</td>
</tr>
<tr>
<td>4</td>
<td>597</td>
<td>5.08</td>
<td>1.27</td>
</tr>
</tbody>
</table>

We compared Speedup and Efficiency for the solution quality of parallel EigenAnt Algorithm with MPI and OpenMP is shown in Figure 3 and Figure 4.

Figure 3: Comparison of Speedup with MPI and OpenMP

Figure 4: Comparison of Efficiency with MPI and OpenMP
5 Conclusion

In this paper, we propose a parallel EigenAnt optimization algorithm by ACO approach. Our aim was to increase the execution time in the algorithm. To improve the speedup and efficiency in the algorithm, we use the parallelization in the MPI interface. According to the algorithm, an ant is easily able to find the shortest path from a source to a destination and this reduces the running time of the algorithm based on the amount of pheromone. A comparison of the performance between MPI and OpenMP is made by different threads in Table I. The result of implementation has shown that it was possible to design an efficient parallel EigenAnt Optimization algorithm in MPI. This method improves the speed and quality of solution compared to the sequential algorithm. Benchmark instances are used to investigate the performance of the proposed approach.

References


https://doi.org/10.1109/IPDPSW.2016.111

https://doi.org/10.1016/j.cie.2016.06.025

https://doi.org/10.1109/IranianCIS.2014.6802525

https://doi.org/10.1007/s11721-013-0076-9

https://doi.org/10.1109/SIS.2014.7011806

https://doi.org/10.1007/s11721-013-0076-9

https://doi.org/10.1007/978-3-642-13495-1_31