A MapReduce based Ant Colony Optimization Approach to Combinatorial Optimization Problems

Bihan Wu[1], Gang Wu[2], Mengdong Yang[1]
1, School of Computer Science and Engineering, Southeast University, Nanjing 210096, China
2, College of Information Science and Engineering, Northeastern University, Shenyang 110004, China
Email: wubh@seu.edu.cn, wugang@ise.neu.edu.cn, mdyang@seu.edu.cn

***Abstract*—Ant Colony Optimization (ACO) is a kind of meta-heuristics algorithm, which simulates the social behavior of ants and could be a good alternative to existing algorithms for NP hard combinatorial optimization problems, like the 0-1 knapsack problem and the Traveling Salesman Problem (TSP). Although ACO can get solutions that are quite near to the optimal solution, it still has its own problems. Premature bogs the system down in a locally optimal solution rather than the global optimal one. To get better solutions, it requires a larger number of ants and iterations which consume more time. Parallelization is an effective way to solve large-scale ant colony optimization algorithms over large dataset. We propose a MapReduce based ACO approach. We show how ACO algorithms can be modeled into the MapReduce framework. We describe the algorithm design and implementation of ACO on Hadoop.**

Keywords-Ant Colony; MapReduce; Meta-heuristics; 0-1 Knapsack Problem; Traveling Salesman Problem

I. INTRODUCTION

A. *Background*

Ant Colony Optimization (ACO) algorithm, first introduced by Dorigo [1], is a bionic meta-heuristic algorithm stemming from the finding food process of natural ant colony. Many research results on the ACO algorithm show that it has the advantages of distributed calculation, robustness, etc. [4]. Meanwhile, the ACO has its shortcomings: 1) may be bogged down in local optimums 2) usually needs a sufficient colony size 3) needs a sufficient number of iterations. These shortcomings become more severe as the scale of the target problems grow. For the first shortcoming, we could take on some improvements to avoid stagnation. For the second and third, parallelization can be a good solution.

Parallelization is an effective way to perform large-scale computing. As for parallel models, different ones have diverse abstraction layers. OpenMP [8] implements parallel computing programming with threads. MPI provides the capability of parallelism in terms of processes. There are also approaches with both OpenMP and MPI. OpenMP requires computers with high capacity in both computing and storage because threading is a shared-storage approach. MPI stores data in an explicit distributed manner and is well scalable. But developers that write MPI programs need to know detailed knowledge of system architecture to take care of the underlying communication details. This makes the model complex and unreliable. MapReduce is a powerful abstraction proposed by Google for making scalable and fault tolerant applications. It enables users to easily develop large-scale distributed applications. Whereas, researches on applying MapReduce based ACO to combinatorial optimization problems are still at the very preliminary stage. Existing solutions have strong problem specific characteristics.

B. *Contributions*

In this paper, our contributions include:

a) ACO algorithm improvements. We proposed several improvements to traditional ACO algorithms involving heiristic factor, selection probability determination and pheromone update strategy.

b) Parallel ACO implementations with MapReduce. We implemented two parallel versions of ACO algorithms. The first approach illustrates a baseline of MapReduce-based ACO algorithm, and the second approach has different designs depending on the correlation of input items of taget problems. We set the 0-1 knapsack problem and the TSP as example, showing how to partition the input with the MapReduce framework and perform efficient parallel ACO.

We hope that this work could arouse interest of researchers who want to improve the performance of ACO by fine-tuning the algorithm parameters and/or parallelizing the algorithm with MapReduce framework.

II. RELATED WORK

A. *ACO Improvements*

As mentioned before, ACO has the premature and stagnation problems [4], which lead up to stagnation in the local solution and a premature solution after a certain number of iterations. Tsutsui introduced the inner mutation and the outer mutation strategies together with a metric called concentrate degree to solve the premature problem [6]. The concentration degree is used to decide whether it's necessary to perform mutation. Inner mutation mutates a certain column in the pheromone matrix based on the concentration degree of the column while outer mutation mutates the whole pheromone matrix. Mutation is performed via adding a random matrix to the existing pheromone matrix. In [3], the probability of selection is amplified in the beginning of the algorithm to speed up the expansion of search space and in the end to boost the convergence. [3] also improved the pheromone update strategy for the TSP by dynamically weakening or strengthening historical paths. In [7], a unit consisting of two ants: a cunning ant and a donator ant, is used. An in-unit optimization strategy is introduced as well.

B. Parallelization of ACO

Most existing parallelized ACO works lie in implementing ACO with traditional parallel programming models. M. Manfrin et al. implemented a parallel ACO to the TSP in a multi-machine environment with MPI [10]. Basically, there are two information exchange strategies in parallel ACO: synchronous and asynchronous. In the synchronous strategy, all execution units run synchronously, and the pheromone is updated after all units complete one iteration. In the asynchronous model, each execution unit is run independently and the wait-free pheromone update is performed. Craus and Rudeanu also implemented a MPI-based parallel ACO algorithm with a one-master-multi-slave architecture [11]. In [11], checkpoint is used to update pheromone asynchronously. The master identifies the sequential order of pheromone update committed by slaves with logical clock. Tsutsui and Fujimoto implemented a parallel version of cAS [7] in a multi-core environment with Java threading [13]. [10], [11] and [12] all illustrate that the asynchronous model has a higher performance than the synchronous one.

Compared with traditional models like MPI, parallelization with MapReduce has less attention in the academia. Huang and Lin implemented a genetic algorithm with MapReduce to the job scheduling problem and has a reasonable output [9]. The approach in [9] requires multiple MapReduce iterations, which drop the performance of the algorithm. The same problem also exists in [5], which implements parallel ACO with MapReduce to the TSP. Experimental result in [5] illustrates that the one-iteration approach has a far better performance than the multi-iteration approach.

III. Improvements

We added improvements to the existing ACO variants [2][3][6] to get better solutions. Our improvements include:

a) An improved solution generation strategy. It is based on roulette wheel selection and traversal;

b) A new pheromone update strategy. We proposed a new formula for pheromone update.

c) Two parallel approaches for our improved ACO algorithm. They will be described in detail in section IV.

To illustrate the application of our improvements, we summarized the standalone version of the ACO algorithm for 0-1 knapsack as follows:

Step 1. Initialization

Iteration counter $nc \leftarrow 0$. For every item i, set an initial pheromone value $\tau_i(0)=\tau_{\max}$, where $\tau_{\max} = 1$.

Step 2. Crawling.

Calculate the selection probability P_i for item i in $rest_k$, where $rest_k$ is the unselected items after k-1 selections.

$$P_i = \begin{cases} \dfrac{\tau_i^\alpha(t)\eta^\beta(i)}{\sum_{r \in rest_k} \tau_r^\alpha(t)\eta^\beta(r)}, i \in rest_k \\ 0, otherwise \end{cases} \tag{1}$$

In Equation (1), α and β are used to control the weights of the item pheromone τ_i and the heuristic factor η in generating the selection probability P_i. In our experiments, α and β are both set to 1. For item i, its heuristic factor $\eta(i) = v_i/w_i$, where v_i/w_i is the value density of items. If the $rest_k$ is not empty, an ant select an item randomly from it based on the selection probability P. When the next item would overload the bag, it is just discarded from $rest_k$, then repeat the selection process.

Step 3. Pheromone Update

Update the pheromone for all items with the following rule:

$$\tau_i(t+1)=(1-\rho)\tau_i(t)+\Delta\tau^{ib} \tag{2}$$

where $\Delta\tau^{ib}$ is the pheromone increment laid down on each item, ρ ($0<\rho<1$) is pheromone evaporation rate. $\Delta\tau^{ib}$ has the following definition:

$$\Delta\tau^{ib} = \begin{cases} T \bullet e^{-\frac{w\_iteration \bullet V\_total}{v\_iteration \bullet W\_total}}, i \in item\_set(ib) \\ 0, otherwise \end{cases} \tag{3}$$

T is the parameter for restricting the maximal value of $\Delta\tau$, and is set to 0.5 in our experiments. *w_iteration* and *v_iteration* are the weight sum and the value sum of iteration-best solution. $\Delta\tau^{ib}$ increases as v/w increases, and has a range of $(0,T]$. The weight and value of all items, *W_total* and *V_total*, are used to neutralize the impact of different item sets.

Step 4. Iteration and Output

Repeat Step 2 and Step 3. Increment nc by 1 each time a Step 2-3 iteration completes. Terminate the iteration when nc reaches the maximal number of iterations.

The TSP has a similar principle to the 0-1 knapsack problem. Differences include:

a) The pheromone is a two-dimensional matrix;

b) The heuristic factor $\eta^\beta(i,j)$ is set to be $1/d_{ij}$, and j is selected from $rest_k$, which are all the unreached cities connected with city i directly.

c) In step 3, pheromone update, pheromone increment $\Delta\tau$ is changed as follow:

$$\Delta\tau^{ib}(i,j) = \begin{cases} M \bullet \dfrac{L_{gb}}{L_{ib}}, if\,(i,j) \in route(ib) \\ 0, otherwise \end{cases} \tag{4}$$

where M is a parameter to adjust the average value of $\Delta\tau$. L_{gb} and L_{ib} stand for the length of the route visited by the global-best ant and iteration-best ant.

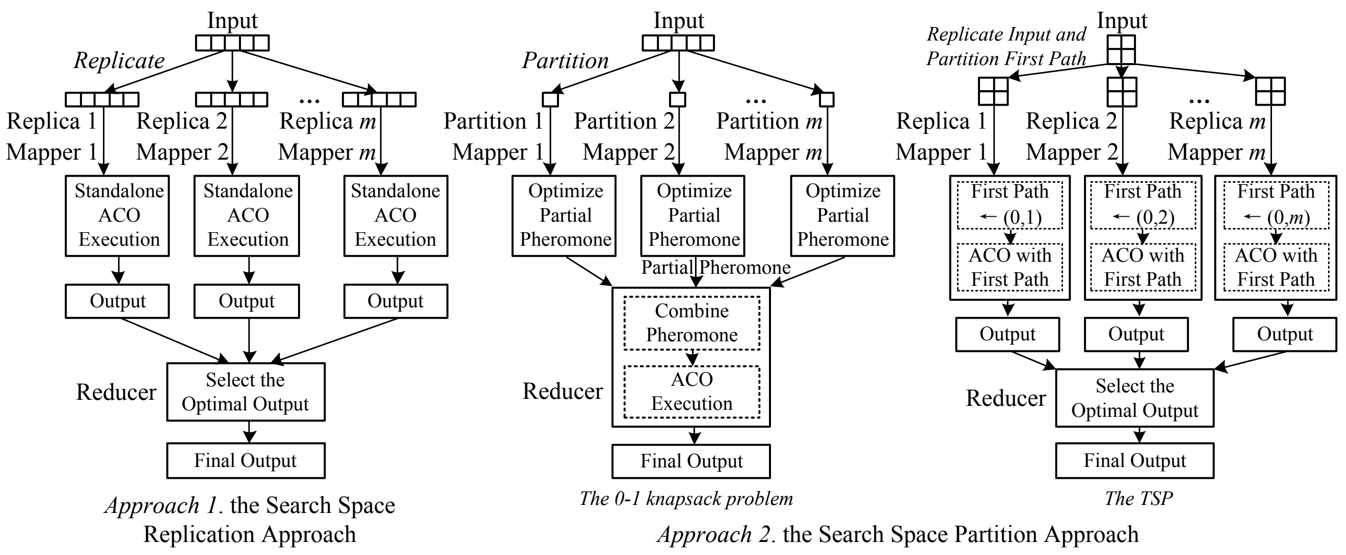


Figure 1. Parallel ACO Approaches with MapReduce

IV. Parallelization

MapReduce, which is usually designed for data-intensive problem, provides a two-phase (map and reduce) divide-and-conquer processing strategy. The principle of MapReduce framework illustrates that in the map phase no data exchange between map tasks can be performed. Reduce is the only way to aggregate outputs from different map tasks. This restriction ensures the simplicity and reliability of the model. However, at the same time, it also requires developers to design the partition and aggregation of the problem carefully.

Our MapReduce algorithm design aims at solving the problem with existing approaches and make sufficient use of the simplicity and scalability of the MapReduce model. In this paper, two approaches are introduced, as is shown in Figure 1.

A. Approach 1. The Search Space Replication Approach

In this approach, the input is replicated m copies, where m is the number of map tasks. Each map task executes the ACO algorithm independently while emitting its output with a constant integer as the key and its ACO result as the output. Since the outputs all have the same keys, they will be sent to the only reduce task. In the reduce task, the optimal value is selected and recorded as the output. In Approach 1, the input is not partitioned, nor does any information exchange happen between execution units.

B. Approach 2. The Search Space Partition Approach

In this approach, the solution space is partitioned to map tasks. Since the partition is carefully designed, each split has a reasonable locality that can be solved independently.

For 0-1 knapsack problem, both items and the pheromone are evenly split and sent to map tasks. Ants crawl for temporary solutions of the partial input based on Equation (1), calculate the ratio of total value and the weight (v/w) of each temporary solution. The pheromone matrix is updated according to the best ant that owns the maximal ratio value. In the reduce phase, the pheromone emitted from map tasks are connected, and ACO algorithm is further run on the whole items. With the optimized pheromone, only a small number of ants and iterations are needed.

As to the TSP, it is unreasonable to divide the input to different map tasks because there may be connection between any two cities. Considering that TSP has a search space of tree structure, we assign different branches of the tree to map tasks. All map tasks have a copy of adjacent matrix, and are assigned with an even number of path prefixes so that the search space is divided into even pieces and solved in a parallel way. Then the reduce task compares all input pairs and outputs the optimal path as the final output of the algorithm.

V. Experiments

A. Environment and Datasets

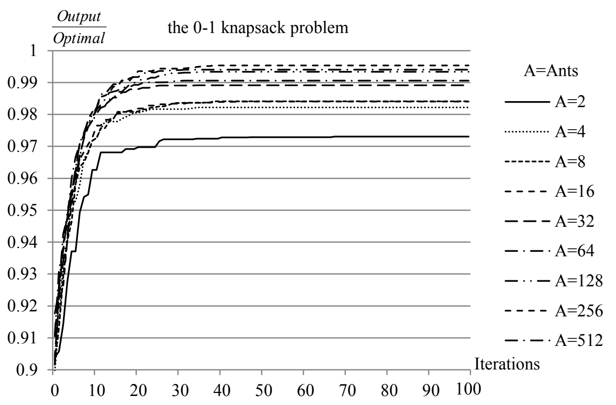
a) Environment. We implemented our algorithm with Java 1.6 update 17. Each computer in the cluster has an Intel Q8400 quad-core CPU and 4GB memory, and the operating system Ubuntu 10.04. The MapReduce cluster is configured with Apache Hadoop 0.21.0 and consists of 16 nodes.

b) Dataset for the 0-1 knapsack problem. We generated a dataset of 1,000 items. The item weight W and value V are independent and uniformly distributed within integer interval [1..100]. The bag capacity is 25,000. The maximal possible value of containable items is 40,342 according to the dynamic programming algorithm.

c) Dataset for the TSP. We use TSPLIB, the dataset `lin105`, whose optimal path length is 27,874.

B. Determination of Iteration Number and Ant Number

Figure 2 shows the output trace of our improved standalone ACO with different ant numbers. The curves stand for the global-best solutions after certain iterations with different ant numbers. Here we use the output/optimal ratio. We can see that for the 0-1 knapsack problem and the TSP the output gradually



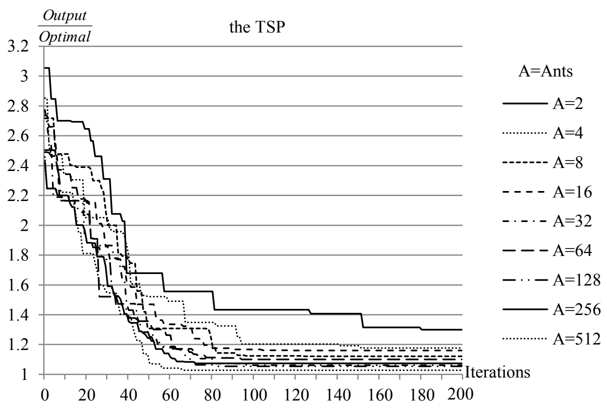
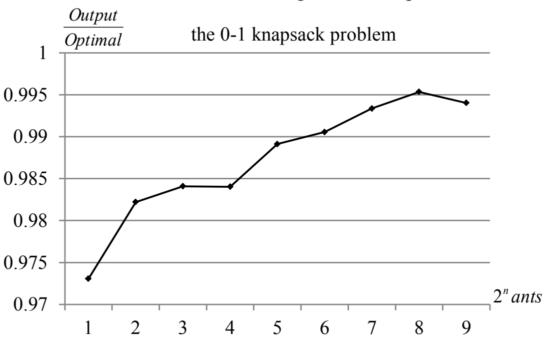


Figure 2. Output Trace of ACO Algorithm as Iteration Goes



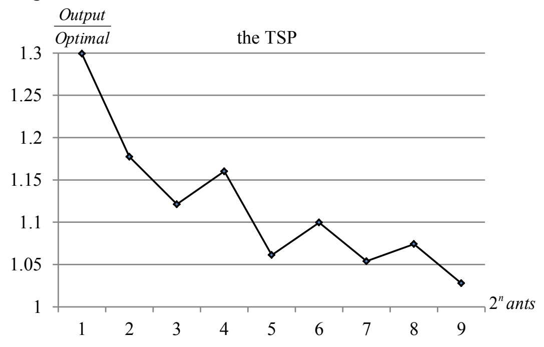


Figure 3. Algorithm Output under Different Ant Numbers

TABLE I. OUTPUT COMPARISON OF UNIMPROVED AND IMPROVED ACO

	Optimal	Unimproved		Improved Standalone		MapReduce Approach 1		MapReduce Approach 2	
		Solution	*Ratio*	*Solution*	*Ratio*	*Solution*	*Ratio*	*Solution*	*Ratio*
The 0-1 knapsack problem	40,342	30,890	0.766	40,154	0.995	39,698	0.984	40,142	0.995
The TSP	27,874	50,132	1.799	28,976	1.040	30,287	1.086	29,013	1.041

converges after 100 and 200 iterations regardless of the ant number. Therefore we fix the iteration number to 100 and 200 accordingly.

Figure 3 shows the final output with the iteration numbers determined above under different ant numbers. For the 0-1 knapsack problem, the growth trend stops when the number of ants reaches 256 (2^8). For the TSP, the output/optimal ratio tends to be stable after the number of ants reaches 512 (2^9).

C. Determination of ρ

The optimal value of ρ depends on the iteration number in our experiments. ρ is the attenuation factor defines weights of history experience and the impact of latest best solutions. We tested the output under different values of ρ in standalone environment. Totally, 101 ρ values are evaluated: from 0 to 1 with interval 0.01. The algorithm output under different ρ values is shown in Figure 4. For the 0-1 knapsack problem the optimal ρ value is 0.3 and for the TSP the optimal ρ value is 0.1.

D. Solution

We tested the output with three different configurations:

a) Standalone. The 0-1 knapsack problem is run with 256 ants, 100 iterations and the ρ value of 0.3. The TSP is run with 512 ants, 200 iterations and the ρ value of 0.1.

b) Approach 1. In approach 1, a MapReduce job with 16 map tasks and 1 reduce task is initiated. Every map task runs a 16-ant ACO for the 0-1 knapsack problem or a 32-ant ACO for the TSP. Iteration numbers are 100 and 200 respectively.

c) Approach 2. In approach 2, a MapReduce job with 16 map tasks and 1 reduce task is initiated. Each map task runs with the same numbers of ants and iterations as in the standalone mode.

Table I shows the solutions for both problems in different environments. The output is close to the optimal solution, indicating that the improvements are effective.

In approach 2, as the problem is partitioned according to map tasks, the ants in different map tasks optimize solutions separately and work out the problem as a whole through cooperation. It conforms with the idea of MapReduce framework i.e. partition and integration.

E. Performance

A parallel approach is valuable only when it effectively speed up the execution. Figure 5 shows the time expense of the MapReduce-based ACO algorithms. Since both axes are displayed logarithmically, the linear trend shows that the two approaches achieve nearly linear speed up in parallel run.

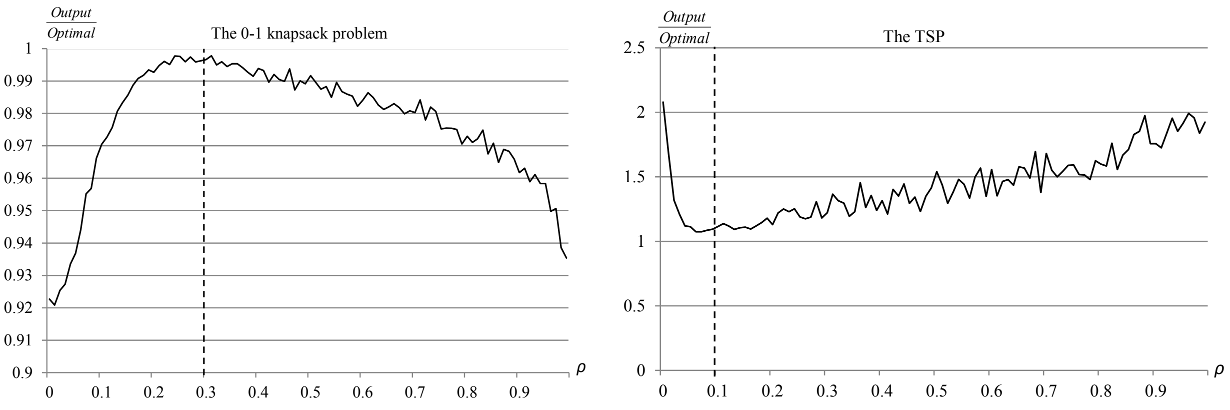


Figure 5. Algorithm Output under Different ρ Values

VI. Conclusion and Future Work

In this paper, we present some improvements for ACO algorithm and two parallel approaches on Map Reduce framework. Extensive experiments are conducted to illustrate the effectiveness of our approaches. The future research directions include: 1) development of more efficient problem partition strategy; 2) development of MapReduce-based parallelization of other computing-intensive algorithms, such as the genetic algorithm, simulated annealing, etc.

Acknowledgment

The work in this paper is supported by the National Natural Science Foundation of China under grants 60903010, 61025007 and 60933001, the National Basic Research Program of China under grant 2011CB302206, the Natural Science Foundation of Jiangsu Province under grant BK2009268, and the Key Laboratory of Advanced Information Science and Network Technology of Beijing under grant XDXX1011. Supported by the Fundamental Research Funds for the Central Universities No.N110404013.

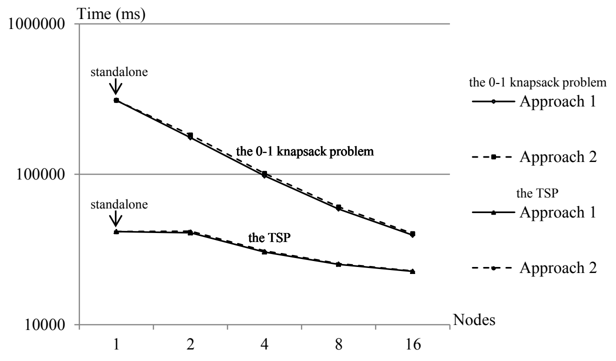


Figure 4. Scalability of Parallel ACO Algorithm on MapReduce

References

[1] Marco Dorigo, Ottimizzazione, Apprendimento Automatico, ed Algoritmi Basati su Metafora Naturale. *PhD thesis*, Politecnico di Milano,(1992)

[2] T. Stützle and H. H. Hoos. Max-min ant system. *Future Gener. Comput. Syst.*, 16:889–914, June 2000.

[3] X. Song, B. Li, and H. Yang. Improved ant colony algorithm and its applications in tsp. *In Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications - Volume 02*, ISDA '06, pages 1145–1148, Washington, DC, USA, 2006. IEEE Computer Society.

[4] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B*, 26(1):29–41, 1996.

[5] G. Cesari. Divi de and conquer strategies for parallel tsp heuristics. *Comput. Oper. Res.*, 23:681–694, July 1996.

[6] Peiyi Zhao, Peixin Zhao, Xin Zhang. A New Any Colony Optimization for the Knapsack Problem. In *the 7th International Conference on Computer-Aided Industrial Design and Conceptual Design, 2006*. CAIDCD '06, pages 1-3, Hangzhou, China.

[7] S. Tsutsui. cas: Ant colony optimization with cunning ants. *In Proc. of the 9th Int. Conf. on Parallel Problem Solving from Nature* (PPSN IX, pages 162–171, 2006.

[8] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L.,Woodall, T. S.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, 2004, pp. 97-104.

[9] D.-W. Huang and J. Lin. Scaling populations of a genetic algorithm for job shop scheduling problems using mapreduce. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 780–785, Washington, DC, USA, 2010. IEEE Computer Society.

[10] M. Manfrin, M. Birattari, T. Sttzle, and M. Dorigo. Parallel multicolony aco algorithm with exchange of solutions.

[11] M. Craus and L. Rudeanu. Parallel framework for ant-like algorithms. In *Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, ISPDC '04, pages 36–41, Washington, DC, USA, 2004. IEEE Computer Society.

[12] S. Tsutsui and N. Fujimoto. Parallel ant colony optimization algorithm on a multi-core processor. In *Proceedings of the 7th international conference on Swarm intelligence*, ANTS'10, pages 488–495, Berlin, Heidelberg, 2010. Springer-Verlag.