Strategic Oscillation for Exploitation and Exploration of ACS Algorithm for Job Scheduling in Static Grid Computing

Mustafa Muwafak Alobaedy
School of Computing
College of Art and Sciences
University Utara Malaysia, 06010 Sintok, Kedah
new.technology@hotmail.com

Ku Ruhana Ku-Mahamud
School of Computing
College of Art and Sciences
University Utara Malaysia, 06010 Sintok, Kedah
ruhana@uum.edu.my

***Abstract*—Exploitation and exploration mechanisms are the main components in metaheuristics algorithms. These mechanisms are implemented explicitly in ant colony system algorithm. The rate between the exploitation and exploration mechanisms is controlled using a parameter set by the users of the algorithm. However, the rate remains unchanged during the algorithm iterations, which makes the algorithm either bias toward exploitation or exploration. Hence, this study proposes a strategic oscillation rate to control the exploitation and exploration in ant colony system. The proposed algorithm was evaluated with job scheduling problem benchmarks on grid computing. Experimental results show that the proposed algorithm outperforms other metaheuristics algorithms in terms of makespan and flowtime. The strategic oscillation has improved the exploration and exploitation in ant colony system.**

Keywords—strategic oscillation; ant colony system; job scheduling; grid computing

I. INTRODUCTION

The popularity of grid systems started in the late 1990s when Foster developed a grid system called Globus Toolkit [1]. Grid system could be classified into several types, such as grid computing, data grid, enterprise grid, sensor grid, campus grid, global grid, pc grid, and utility grid [2]–[4]. Grid computing provides a powerful processing capability which is not possible to achieve using individual computer. Grid computing is defined as "geographically distributed computers, linked through the internet in a Grid-like manner, which are used to create virtual supercomputers of vast amount of computing capacity able to solve complex problem from e-Science in less time than known before" [5]. Another definition provided by [6] is "a hardware and software infrastructure that provides transparent, dependable, pervasive and consistent access to large-scale distributed resources owned and shared by multiple administrative organizations in order to deliver support for a wide range of applications with the desired qualities of service. These applications can perform either as high throughput computing, on-demand computing, data intensive computing, or collaborative computing". From these definitions, grid computing could be defined as a collection of computing resources distributed geographically in different locations. These resources are forming a processing power that can be used to solve various complex problems in several fields, such as science, commerce, and education. The resources in grid computing could be heterogeneous in terms of hardware and software.

Grid computing has been successfully implemented to solve various real-life problems. For instance, finding Protein binding sites using DNA@Home volunteer computing project [7], grid computing for disaster mitigation implemented by Universiti Sains Malaysia [8], C-Grid based on Integrated Rule-Oriented Data System for health care community [9], and ANSYS® Commercial Suite on the EGI Grid Platform [10].

The main components in Grid are infrastructure fabric, middleware, and applications [11]. The middleware layer considered as the brain of the grid and provides many services, such as job scheduling, job enactment, monitoring, and meta-scheduling [11]. These services handled by Resource Management System (RMS) which has the responsibility to map users' submitted tasks to available and suitable resources [12]. Scheduling algorithm is the major influence on the performance of grid computing system [5]. Scheduling algorithm could be implemented using simple approach, such as first come first serve or greedy algorithm. However, grid computing system with big number of resources and tasks needs more sophisticated algorithm in order to achieve good quality of services. Therefore, more intelligent algorithms are required for scheduling algorithm implementation in RMS.

Job scheduling problem in grid computing is considered as NP-complete problem [13]. Due to the complexity nature of these types of problems, a heuristic and metaheuristics algorithms are preferred in real application [14]. Metaheuristics algorithms have the ability to produce optimal or near-optimal solution in reasonable time and resources. One of the metaheuristics algorithms branches is nature inspired algorithms, such as Ant Colony Optimization (ACO), Simulated Annealing (SA), Particle Swarm Optimization (PSO), and Artificial Bee Colony (ABC) [15]. Among them, ACO algorithm shows an excellent performance in various domains, such as routing, scheduling, classification, and optimization [16].

ACO is a framework algorithm for other variants, such as Ant System (AS), Elitist Ant System (EAS), Rank-Based Ant System (AS_{rank}), Ant Colony System (ACS), and Max-Min Ant System (MMAS) [16]. ACS algorithm is considered as one of the best among them [17]. ACS algorithm is based on two mechanisms, namely exploitation and exploration [16]. During the algorithm iteration, the selection between the two mechanisms is controlled via a fixed parameter q_0 $(0 \leq q_0 \leq 1)$. Therefore, the algorithm with small exploitation or exploration rate will never change during all iterations. In other words, the rate of any mechanism will not increase or decrease during the execution. Therefore, this study focuses on a dynamic rate specifically, adopting the strategic oscillation rate between exploitation and exploration which is proposed in Tabu Search (TS) algorithm by Glover and Laguna in [18]. This strategy makes the ACS algorithm behaves differently but strategically in each cycle.

This paper organized as follow, Section II reviews the evolution of ant colony optimization. The strategic oscillation in ant colony system is provided in Section III. Section IV illustrates the problem formulation while Section V presents the experiments and results. Finally, the conclusion is provided in Section VI.

II. Ant Colony Optimization

Ant colony optimization algorithm was proposed in 1990s by Dorigo [16] as a metaheuristics algorithm. The first version of ACO is known as ant system [19]. AS consists of two main phases, namely solution construction and pheromone update. The solution construction phase is based on probabilistic action choice rule, known as *random proportional* rule. For pheromone update phase, AS uses evaporation concept and pheromone deposit method. Compared with other ACO variants, the performance of AS algorithm decrease dramatically when the problem instance size increases [16].

Additional reinforcement to the arcs belonging to the best solution was introduced in Elitist strategy [EAS] algorithm to improve AS [20]. The implementation of the elitist strategy enables the ants to find better solution quality as well as lower number of iterations. EAS algorithm shows better performance than AS algorithm. Another improvement over AS algorithm is rank-based ant system introduced in [21]. AS_{rank} algorithm applies ranking concept to the amount of pheromone deposits on the arcs. Only the best-so-far ant and best ranked ants are allowed to deposits the pheromone. Compared to AS and EAS, AS_{rank} performed significantly better than AS and slightly better than EAS.

Another variant of ACO algorithm is max-min ant system which has direct improvement over AS algorithm [22]. MMAS provides four improvements. Firstly, it uses a stronger exploitation mechanism. Secondly, MMAS apply a range of pheromone trail values to the interval that help to avoid the premature stagnation (all ants converge early to one suboptimal solution) of the search process. Thirdly, the initial pheromone value is set to the upper pheromone limit with a small pheromone evaporation rate to increase the exploration mechanism. Finally, in MMAS, pheromones values are reinitialized whenever the algorithm is not able to find an improved solution for a certain number of iterations. For the pheromone update, only one of the two ants is allowed to add pheromone, either the best-so-far ant or the iteration-best ant.

One more important improvement over AS algorithm is ant colony system proposed in [23]. ACS algorithm improves AS algorithm in three main aspects. First, ACS implements a stronger action choice rule than AS. Second, the pheromone value is added only to the arcs belonging to the global-best solution. Third, each time an ant moves on an arc, it evaporate some pheromone from that arc. The three main phases of the ACS algorithm constitute the ants' solution construction, global pheromone trail update, and local pheromone trail update. In global pheromone update, only one ant (the best-so-far ant) is allowed to add pheromone after all ants have finished constructing their tours. In local pheromone update, all the ants in ACS algorithm apply local pheromone update rule immediately after moving on arcs during the solution construction using the evaporation concept. In ACS algorithm, the tuning between exploitation and exploration is controlled by a parameter fixed by the user. Therefore, the rate of the exploration and exploitation will never change during the algorithm execution. Thus, if the exploitation rate is high, then the algorithm will behave more toward greedy approach. In opposite, if the exploration rate is high, then the algorithm will behave more toward random approach. Hence, this study proposes strategic oscillation rate for exploitation and exploration mechanisms in ACS algorithm.

III. Strategic Oscillation in Ant Colony System

Strategic oscillation concept is proposed by Glover and Laguna in tabu search algorithm [18]. The authors state that "strategic oscillation provides a means to achieve an effective interplay between intensification and diversification over the intermediate to long term" [18]. The idea behind this concept is simply to oscillate between the exploration and exploitation in a strategic process. Fig. 1 shows the process of strategic oscillation in TS algorithm [18].

In ant colony system algorithm, the exploration and exploitation mechanisms are explicit and controlled via a fixed parameter q_0 $(0 \leq q_0 \leq 1)$ [16]. Therefore, applying the concept of strategic oscillation is direct and straightforward in ACS algorithm.

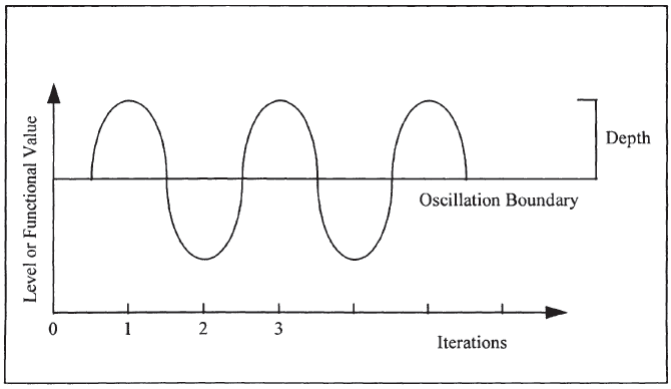

Fig. 1. Strategic oscillation process [18].

The proposed algorithm called Strategic Oscillation Ant Colony System (SOACS) starts with maximum oscillation rate $q_0 = 1$(behaves like greedy approach). During the iterations, gradually moves towards exploration by reducing the oscillation rate using the step_size parameter γ $(0 \leq \gamma \leq 1)$. Once the oscillation rate reaches the minimum ($q_0 = 0$) (behaves like random approach), the oscillation rate starts to moves toward exploitation again by increasing the oscillation rate with step-size parameter γ . Fig. 2 represents the pseudocode of the strategic oscillation in ACS algorithm.

SOACS algorithm starts with equal pheromone values on all edges. Therefore, the algorithm starts with maximum oscillation rate to exploit the heuristic information rather than pheromone information. The idea of starting with maximum exploitation will produce a good staring solution which is at least equal to the solution produced by nearest- neighbour approach. During the algorithm iterations, the oscillation rate will change using the γ parameter. The value of the γ parameter is recommended to be very small value in order to move smoothly between exploitation and exploration mechanisms. The oscillation rate for 1000 iterations using $\gamma = 0.001$ is shown in Fig. 3.

The strategic oscillation could be implemented in ACS algorithm either in iteration level or ant level. However, this study implemented the strategic oscillation in iteration level. In other words, the oscillation rate will change after all the ants finish one iteration. Fig. 4 represents the complete pseudocode for SOACS algorithm.

```
Initialize the oscillation rate q0 = 1;
Initialize the step_size parameter γ;
Initialize switch variable (switch = false);
If (q0 ≥ 1)                    //Maximum exploitation rate
        switch ← true;         //Switch to exploration
Else if (q0 ≤ 0)               //Maximum exploration rate
        switch ← false;        //Switch to exploitation
If (switch = true)             //If exploration is true
        q0 = q0 − γ;           //Decrease the rate
Else
        q0 = q0 + γ;           //Increase the rate
```

Fig. 2. Strategic oscillation pseudocode for ACS algorithm.

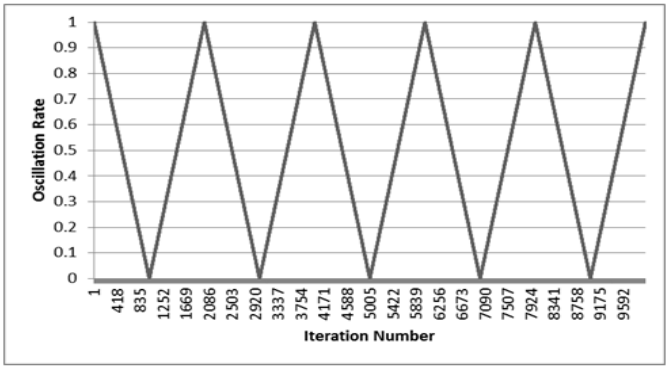

Fig. 3. Strategic oscillation rate.

```
Procedure SOACS
Initialize the number of ants n;
Initialize parameters and pheromone trails;
Initialize q = random [0, 1];
Initialize the oscillation rate q0 = 1;
Initialize the step_size parameter γ;
Initialize switch variable (switch = false);
While (Termination condition not met) Do
        For i = 1 to n Do
                Construct new solution:
                        q = random [0, 1];
                        If (q ≤ q0)
                                Exploitation;
                        Else
                                Exploration;
                Apply local pheromone update;
        End For;
        Apply pheromone evaporation;
        Apply Global pheromone update;
        If (q0 ≥ 1)
                switch ← true;
        Else if (q0 ≤ 0)
                switch ← false;
        If (switch = true)
                q0 = q0 − γ;
        Else
                q0 = q0 + γ;
        Update best-so-far solution s*;
End while;
End Procedure
```

Fig. 4. SOACS algorithm pseudocode.

In Fig. 4, the syntaxes with bold text represent the proposed strategic oscillation code in ant colony system. Moving this part of code to the location after the syntax "Apply local pheromone update" in Fig. 4, will make the strategic oscillation based on ant level in ACS algorithm. The proposed SOACS algorithm could be implemented to solve various combinatorial problems similar to ACS algorithm. However, this study has implemented the proposed SOACS algorithm to solve job scheduling problem on static grid computing system.

IV. Problem Formulation

The job scheduling problem on computational grid is known as a multi-objective problem. There are various criteria in grid computing that need to be optimized, for instance makespan, flowtime, load balancing, utilization, matching proximity, turnaround time, total weighted completion time, and average weighted response time [24]. This study has implemented two criteria, namely makespan and flowtime with the priority to makespan as the main optimization objective.

The general productivity of the grid computing is measured by makespan. The best scheduling algorithm is the one that can achieve a small value of makespan, which means that the algorithm is able to map tasks to machines in a good and efficient way. Therefore, the main objective in this study is to minimize the makespan. Makespan is defined as the time when the last task finishes execution, formally defined as:

$$minimization\ of\ makespan: min\ S_{i \in Sched}\{max_{j \in Jobs} F_j\}$$

where $Sched$ is the set of all possible schedules, $Jobs$ is the set of all jobs to be scheduled, and F_j denotes the time when task j finalizes [24]. The second criterion implemented in this study is flowtime which refers to the response time to the user submissions of task executions. Flowtime is defined as the sum of finalization times of all tasks, formally defined as:

$$flowtime: min\ S_{i \in Sched}\{\sum_{j \in Jobs} F_j\}.$$

These criteria could conflict with each other since limited resources could be the bottleneck of the system [24]

In order to evaluate the proposed SOACS algorithm, a suitable benchmark is required to reflect the robustness of the algorithm. The benchmark model should has the features to reflect the environment characteristics, such as resources and jobs heterogeneity. A benchmark for static grid computing which is based on a successful model known as Expected Time to Compute (ETC) proposed in [25] has been implemented. This model is widely accepted by researchers for algorithms evaluation in job scheduling problem [26], [27]. The benchmark model defines a matrix known as ETC matrix. Each row in the ETC $[i, j]$ matrix contains the expected time to compute task $[i]$ on machine $[j]$. Therefore, ETC has $n * m$ entries where n represents the number of tasks and m represents the number of machines. ETC matrix is again defined using three metrics, namely task heterogeneity, machine heterogeneity, and consistency. The task heterogeneity measures the variance in execution time among tasks while machine heterogeneity measures the variance in machine speed among machines. The heterogeneity of tasks and machines is represented with two values of "high" and "low" respectively. In addition, ETC matrix captures other possible features of real heterogeneous computing system using three more metrics to measure the consistencies, namely consistent, inconsistent, and semi-consistent. The ETC matrix is considered consistent whenever a machine r_j executes a task t_i faster than another machine r_k, therefore, machine r_j will execute all other tasks faster than machine r_k. ETC matrix is considered inconsistent when a machine r_j could execute some tasks faster than machine r_k and some other slower. Finally, semi-consistent ETC matrix is an inconsistent matrix which has a consistent submatrix of specific size. Combining all these matrices will generate 12 distinct types of ETC matrix [25].

V. Experiment and Results

The experiments have been conducted using Intel® Core (TM) i7-3612QM CPU @ 2.10GHz and 8G RAM. A simulation is developed using C# language. The proposed SOACS algorithm was evaluated against genetic algorithm, ant system, and ant colony system provided in [28]. SOACS algorithm parameters values are given in Table I. The new parameter, γ is the step size to move from exploitation to exploration and vice versa. The parameters value of number of ants, evaporation rate, and beat are adopted from the original ACS algorithm which are recommended in [16]. The proposed SOACS algorithm was executed 10 times to calculate the best and average values. Each run is given only 90 seconds, such a time restriction is very important requirement to mimic the real grid computing environment [29], [30].

Experimental results are organized in tables. The first column of each table represents the instance name with an abbreviation code: x-yyzz as follows:

x represent the type of consistency; c means consistent, i means inconsistent, and s means semi-consistent.

yy represents the heterogeneity of the tasks; hi means high and lo means low.

zz represents the heterogeneity of the machines; hi means high and lo means low.

For example: c_hilo means consistent environment, hi heterogeneity in tasks and low heterogeneity in machines. The results show that the proposed SOACS algorithm outperforms other algorithms in terms of best makespan values on all the 12 instances as illustrated in Table II. Similar performance is shown by the proposed SOACS algorithm in terms of average makespan values as shown in Table III.

TABLE I. SOACS PARAMETERS VALUES.

Run time	Beta	Evaporation rate	No of ants	γ
90second	8	0.6	10	0.00004

TABLE II. BEST MAKESPAN

	GA	AS	ACS	SOACS
c_hihi	11215488.9	11210553.9	10794610.8	**10525341.5**
c_hilo	182232.0	184701.3	179762.4	**177747.2**
c_lohi	374686.0	367182.8	346838.4	**346627.3**
c_lolo	6138.5	6224.8	6051.8	**6031.9**
i_hihi	3995843.4	3946883.2	4066163.7	**3919048.2**
i_hilo	91682.3	90968.3	93829.0	**87510.4**
i_lohi	134151.1	133825.4	137176.5	**129994.6**
i_lolo	3045.3	3141.0	3209.0	**3020.8**
s_hihi	6223749.5	5991234.3	6119602.0	**5741578.0**
s_hilo	120447.3	118988.3	120539.1	**115123.5**
s_lohi	181155.5	176800.4	178584.8	**166583.1**
s_lolo	4246.4	4296.3	4350.4	**4131.0**

TABLE III. AVERAGE MAKESPAN

	GA	AS	ACS	SOACS
c_hihi	11266455.7	11492186.4	10947366.9	**10747849.5**
c_hilo	183264.9	186640.1	181434.4	**179875.2**
c_lohi	375322.2	373766.6	353670.8	**350654.3**
c_lolo	6152.5	6281.5	6120.0	**6062.4**
i_hihi	4029108.7	4021032.5	4261681.8	**3984413.4**
i_hilo	91682.3	92311.6	94832.7	**88536.5**
i_lohi	135625.0	136721.9	144178.5	**133200.5**
i_lolo	3051.0	3198.6	3280.0	**3045.5**
s_hihi	6317823.2	6114694.0	6322969.8	**5940008.6**
s_hilo	120664.4	121995.8	122440.4	**117386.7**
s_lohi	181734.6	178990.5	181737.4	**170489.2**
s_lolo	4249.9	4369.1	4399.4	**4186.7**

For best and average flowtime values, Tables IV and V show that the proposed SOACS algorithm was able to achieve the best results on 10 instances followed by AS algorithm on two instances.

Due to the difference scale of each instance result, the geometric mean is implemented to normalize the makespan and flowtime values in order to represent the proposed SOACS algorithm visually [31]. Figs. 5-8 represent the geometric mean for best makespan, average makespan, best flowtime, and average flowtime respectively. The visual geometric mean figures show that the proposed SOACS algorithm significantly outperforms all other algorithms for makespan and flowtime criteria. Optimizing these two criteria at the same time is not an easy task. Therefore, the proposed SOACS algorithm is considered as promising algorithm for job scheduling in grid computing.

TABLE IV. BEST FLOWTIME

	GA	AS	ACS	SOACS
c_hihi	175890174.2	170869481.0	167168928.0	**164883402.9**
c_hilo	2885387.6	2839818.7	2839974.6	**2803190.5**
c_lohi	5862262.0	5600439.3	5481314.1	**5468218.9**
c_lolo	97154.5	95877.0	95871.5	**94910.8**
i_hihi	63759167.6	**60169758.2**	64092691.0	61094410.9
i_hilo	1461297.4	1403670.4	1451182.0	**1378783.4**
i_lohi	2141505.9	**2032456.4**	2150374.0	2038837.0
i_lolo	48547.9	48773.5	50707.6	**47652.4**
s_hihi	98814397.0	90312215.7	95998535.0	**89287719.8**
s_hilo	1909954.1	1832927.6	1893970.7	**1816923.6**
s_lohi	2867157.9	2682621.5	2800124.8	**2635073.7**
s_lolo	67508.1	65545.5	68232.0	**64960.5**

TABLE V. AVERAGE FLOWTIME

	GA	AS	ACS	SOACS
c_hihi	176638718.7	174513587.9	171594188.4	**168542040.5**
c_hilo	2893345.6	2866863.1	2865314.2	**2831223.1**
c_lohi	5867869.1	5712409.2	5587489.2	**5533250.4**
c_lolo	97298.9	96857.6	96697.1	**95703.2**
i_hihi	64261850.8	**61409716.3**	66654183.7	62585921.8
i_hilo	1461683.7	1422434.6	1489277.2	**1396360.0**
i_lohi	2163840.8	**2068376.5**	2256605.3	2093506.3
i_lolo	48579.5	49416.3	51606.3	**48107.0**
s_hihi	99887497.7	92951306.3	98799209.7	**92752032.6**
s_hilo	1915659.2	1867344.1	1934073.4	**1852245.3**
s_lohi	2871564.9	2738879.1	2869869.2	**2691070.5**
s_lolo	67548.4	67048.3	69185.3	**65793.0**

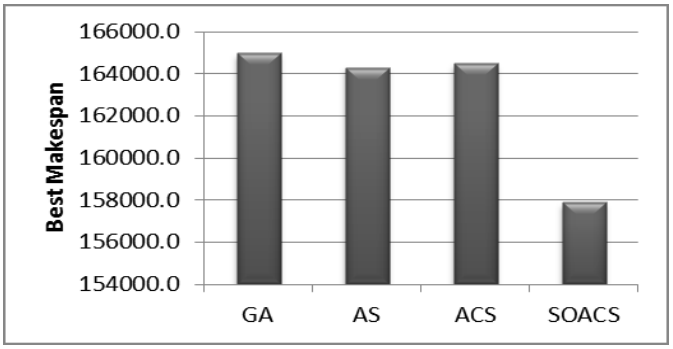

Fig. 5. Geometric mean for best makespan values

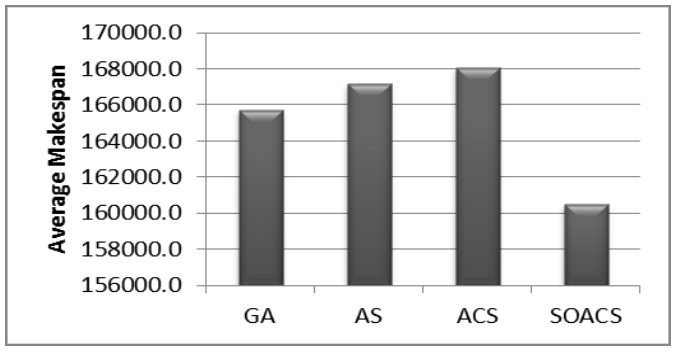

Fig. 6. Geometric mean for average makespan values

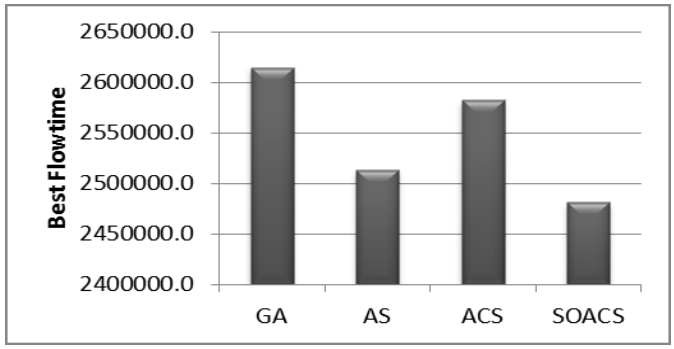

Fig. 7. Geometric mean for best flowtime values

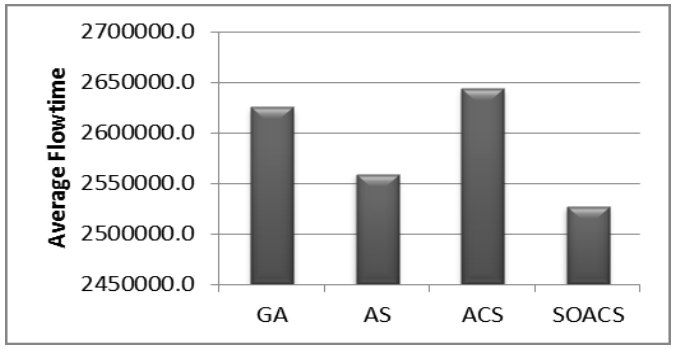

Fig. 8. Geometric mean for average flowtime values

VI. CONCLUSION

Assigning tasks to suitable resources is a very critical process which influences the performance of grid system. This study has enhanced the exploitation and exploration mechanisms in ant colony system. The enhancement is based on implementing the strategic oscillation concept which is adopted from tabu search algorithm. The proposed SOACS algorithm was evaluated against other metaheuristics algorithm using expected to compute model for job scheduling. Results show that the proposed algorithm outperforms all other algorithms in terms of makespan and flowtime values. Future work could focus on the strategic oscillation concept which can be implemented the ant level which gives different rate for each ant. In addition, future work can be on the implementation of the proposed SOACS algorithm to solve other combinatorial problems such as routing and job shop scheduling.

ACKNOWLEDGMENT

The authors wish to thank the Ministry of Higher Education Malaysia for funding this study under the Fundamental Research Grant Scheme, S/O codes 12819 and 11980, and RIMC, Universiti Utara Malaysia, Kedah, for the administration of this study.

REFERENCES

[1] I. Foster and C. Kesselman, "Globus: a Metacomputing Infrastructure Toolkit," *Int. J. High Perform. Comput. Appl.*, vol. 11, no. 2, pp. 115–128, Jun. 1997.

[2] J. Kolodziej, *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems*. New York: Springer, 2012.

[3] O. Babafemi, M. Sanjay, and M. Adigun, "Towards Developing Grid-Based Portals for E-Commerce on-Demand Services on a Utility Computing Platform," *J. Procedia*, vol. 4, no. 1, pp. 81–87, Jan. 2013.

[4] N. Z. C. Fulop, "A Desktop Grid Computing Approach for Scientific Computing and Visualization," (Doctoral dissertation), 2008.

[5] F. Xhafa and A. Abraham, "Computational Models and Heuristic Methods for Grid Scheduling Problems," *J. Futur. Gener. Comput. Syst.*, vol. 26, no. 4, pp. 608–621, Apr. 2010.

[6] F. Magoules, T.-M.-H. Nguyen, and L. Yu, *Grid Resource Management : Toward Virtual and Services Compliant Grid Computing*. Boca Raton: CRC Press, 2009.

[7] T. Desell, L. A. Newberg, M. Magdon-Ismail, B. K. Szymanski, and W. Thompson, "Finding Protein Binding Sites Using Volunteer Computing Grids," in *Proceedings of the 2nd International Congress on Computer Applications and Computational Science*, F. L. Gaol and Q. V. Nguyen, Eds. Berlin Heidelberg: Springer, 2012, pp. 385–393.

[8] H. Koh, S. Teh, T. Majid, and H. Aziz, "Grid Computing for Disaster Mitigation," in *Data Driven e-Science*, S. C. Lin and E. Yen, Eds. New York: Springer, 2011, pp. 445–456.

[9] N. Sukhija and A. K. Datta, "C-Grid: Enabling iRODS-based Grid Technology for Community Health Research," in *Information Technology in Bio- and Medical Informatics*, M. Bursa, S. Khuri, and M. E. Renda, Eds. Berlin Heidelberg: Springer, 2013, pp. 17–31.

[10] A. Costantini, "Implementation of the ANSYS® Commercial Suite on the EGI Grid Platform," in *Computational Science and Its Applications*, vol. 7971, B. Murgante, S. Misra, M. Carlini, C. Torre, H.-Q. Nguyen, D. Taniar, B. Apduhan, and O. Gervasi, Eds. Berlin Heidelberg: Springer, 2013, pp. 84–95.

[11] M. Siddiqui and T. Fahringer, "Model," in *Grid Resource Management*, vol. 5951, Berlin Heidelberg: Springer, 2010, pp. 17–44.

[12] M. Siddiqui and T. Fahringer, "Grid Resource Management and Brokerage System," in *Grid Resource Management*, vol. 5951, Berlin Heidelberg: Springer, 2010, pp. 47–78.

[13] M. B. Qureshi, M. M. Dehnavi, N. Min-Allah, M. S. Qureshi, H. Hussain, I. Rentifis, N. Tziritas, T. Loukopoulos, S. U. Khan, C.-Z. Xu, and A. Y. Zomaya, "Survey on Grid Resource Allocation Mechanisms," *J. Grid Comput.*, vol. 12, no. 2, pp. 399–441, Apr. 2014.

[14] G. Zapfel, R. Braune, and M. Bogl, *Metaheuristic Search Concepts a Tutorial with Applications to Production and Logistics*. Heidelberg: Springer, 2010.

[15] X.-S. Yang, *Nature-Inspired Optimization Algorithms*. Amsterdam: Elsevier, 2014.

[16] M. Dorigo and T. Stutzle, *Ant Colony Optimization*. Cambridge, Mass: MIT Press, 2004.

[17] M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*. New York: Springer, 2010.

[18] F. Glover and M. Laguna, *Tabu Search*. Boston: Kluwer Academic, 1997.

[19] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed Optimization by Ant Colonies," in *Proceedings of the European Conference on Artificial Life*, 1991, pp. 134 – 142.

[20] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents," *J. IEEE Trans. Syst. Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Jan. 1996.

[21] B. Bullnheimer, R. F. Hart, and C. Straub, "A New Rank-Based Version of the Ant System: A Computational Study," *Cent. Eur. J. Oper. Res. Econ.*, vol. 7, no. 1, pp. 25 – 38, 1999.

[22] T. Stutzle and H. H. Hoos, "MAX-MIN Ant System," *J. Futur. Gener. Comput. Syst.*, vol. 16, no. 8, pp. 889–914, 2000.

[23] M. Dorigo and L. M. Gambardella, "Ant Colony System: a Cooperative Learning Approach to the Traveling Salesman Problem," *J. IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, 1997.

[24] F. Xhafa and A. Abraham, "Meta-heuristics for Grid Scheduling Problems," in *Metaheuristics for Scheduling in Distributed Computing Environments*, F. Xhafa and A. Abraham, Eds. Heidelberg: Springer, 2008, pp. 1–37.

[25] T. D. Braun et al, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001.

[26] J. Kołodziej, F. Xhafa, and J. Kolodziej, "Enhancing the Genetic-Based Scheduling in Computational Grids by a Structured Hierarchical Population," *J. Futur. Gener. Comput. Syst.*, vol. 27, no. 8, pp. 1035–1046, 2011.

[27] G. Ritchie and J. Levine, "A Hybrid Ant Algorithm for Scheduling Independent Jobs in Heterogeneous Computing Environments," in *Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group*, 2004, pp. 1–7.

[28] M. M. Alobaedy and K. R. Ku-Mahamud, "Scheduling Jobs in Computational Grid using Hybrid ACS and GA Approach," in *Proceedings of the International Conference on Computing, Communications and IT Applications*, 2014, pp. 223–228.

[29] J. Carretero, F. Xhafa, and A. Abraham, "Genetic Algorithm Based Schedulers for Grid Computing Systems," *Int. J. Innov. Comput. Inf. Control*, vol. 3, no. 6, pp. 1–19, 2007.

[30] F. Xhafa and B. Duran, "Parallel Memetic Algorithms for Independent Job Scheduling in Computational Grids," in *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, C. Cotta and J. van Hemert, Eds. Heidelberg: Springer, 2008, pp. 219–239.

[31] H. Izakian, A. Abraham, and V. Snsel, "Performance Comparison of Six Efficient Pure Heuristics for Scheduling Meta-Tasks on Heterogeneous Distributed Environments," *J. Neural Netw. World*, vol. 6, no. 09, pp. 695–711, 2009.