Parallel Ant Colony Optimization on the University Course-Faculty Timetabling Problem in MSU-IIT

Distributed Application in Erlang/OTP

Earth B. Ugat

Jennifer Joyce M. Montemayor

Mark Anthony N. Manlimos

Dante D. Dinawanao, M.Sc.
Department of Computer Science, School of Computer Studies
Mindanao State University–Iligan Institute of Technology
Andres Bonifacio Avenue, Tibanga, Iligan City 9200, Philippines

***Abstract*—The University Course-Faculty Timetabling Problem (UCFTP) occurs in the Mindanao State University-Iligan Institute of Technology (MSU-IIT) as the delegation of classrooms for available subjects including time schedule and appropriate faculty personnel, taking into consideration constraints such as classroom capacities, location, and faculty preferences, etc. It is a more difficult variant of the classical University Course Timetabling Problem, which is an assignment problem and known to be *NP*-hard. This paper presents parallel Ant Colony Optimization Max-Min Ant System (ACO-MMAS) algorithm as an approach in solving the UCFTP instance in the institute. ACO employs virtual *ants* moving across a search space and using an indirect form of constructive feedback by depositing *pheromones* on the paths they traverse in order to influence other ants in their searches. We have developed an application to automate the timetabling process using Erlang/OTP, a functional language specializing in concurrent and distributed systems. UCFTP was successfully represented into a mathematical problem instance and solved using the ACO-MMAS algorithm applied on a distributed network setup under Parallel Independent Run and Unidirectional Ring topologies. Extensive testing was performed to properly analyze the search behavior under different parameter settings.**

Keywords—ant colony optimization; max-min ant system; parallel algorithm; distributed application; university timetabling

I. Introduction

Every start of an academic semester in the Mindanao State University–Iligan Institute of Technology (MSU-IIT), a different kind of university timetabling problem occurs. Each department in every college releases a certain number of instances of the subjects they handle. These instances, or courses, are then scheduled into timeslot schemes and rooms, and given their own faculty teachers—all done before students come and enroll.

As this scheduling is still being prepared manually, it is prone to errors and faculty personnel often end up having conflicting schedules both in timeslots and locations. Other courses are sometimes labeled as *pending* while teachers try to work out feasible timetables. This leads to delays in the actual start of classes and is a constant challenge for the institute administration every semester.

The problem, in its mathematical form, is the University Course-Faculty Timetabling Problem (UCFTP) and is adapted from a conventional definition of a university timetabling problem [10], which is known to be *NP*-hard [3].

This paper[1] describes the application of Ant Colony Optimization (ACO) to solve the UCFTP. ACO is a swarm-intelligence class, biologically-inspired algorithm introduced in the early nineties as a revolutionary multi-agent metaheuristic for combinatorial optimization problems [5]. It is inherently parallel and is implemented here as a distributed system on parallel network topologies using Erlang/OTP, a functional programming language specializing in process concurrency, fault-tolerance, and cross-processor parallelism [2].

II. University Course-Faculty Timetabling Problem

UCFTP consists of 5 sets: C, F, T, L, and Fe. Set C contains all subject instances, or course events, each having a specific college and department designation, maximum student capacity, unit equivalent value, type classifications, and feature requirements. Additionally, every course has a list of other colleges in whose buildings it can also be scheduled if there is no more room for it in its own designated college.

DOI: 10.5176_2010-2283_1.4.112

[1]This is the technical paper version of the undergraduate Computer Science thesis entitled "MyClass: MSU-IIT Faculty Workload and Class Assignment System – A Concurrent and Distributed Max-Min Ant System Application in Erlang" [16]

F is the set of all teaching faculty personnel, each one having a minimum, maximum, and targeted unit load values. A faculty has an ordered priority list of colleges and departments where she belongs to, as well as a list of courses, within those departments, that she is able to teach. *T* contains the timeslot schedules (pre-determined day-time combinations, e.g. MWF 3-5PM) classified under multiple types. Some of these elements may only be used by certain colleges or departments.

L is the set of all rooms or locations where course events can take place. Each location has a maximum student capacity, a list of features, and type classifications. A location also has an ordered priority list of colleges and departments that are allowed to use it. Set *Fe* is another event set of regular faculty activities, called *faculty-events*, that concerns all teaching personnel (e.g. weekly departmental meetings). A faculty-event from a certain college or department has type classifications and can either have a college-wide or department-wide scope.

The problem is to assign every course event to a faculty, timeslot, and location, and every faculty-event to a timeslot so that the following hard constraints are satisfied:

- A faculty cannot be teaching more than one course at a time;
- A faculty can only teach her allowed courses, and only in locations her college-department designation allows;
- A faculty cannot go below or above her minimum and maximum unit load values, respectively;
- A faculty cannot be scheduled on a time (or timeslot) or location that she indicates;
- There must be no unassigned faculties;
- A timeslot or location must have all type classifications of the course or faculty-event assigned to it;
- A timeslot or location is only usable by certain colleges or departments;
- No two courses can occur in the same location at the same time;
- A location must be able to accommodate the course's capacity and meet all its feature requirements;
- Faculties within scope must not be teaching any course if a faculty-event that concerns them is being held;
- No two faculty-events of conflicting college-department designation and scope can occur at the same time.

The quality of the problem solution is penalized accordingly for every violation of the following soft constraints:

- A faculty should be scheduled only on timeslots and locations of her preference;
- A faculty or location's list of college-department designations indicates their priorities for course assignment (i.e., courses belonging to departments at the top of the list are preferred);
- Faculties should not be required to teach for more than 4.5 consecutive hours in a day;
- A faculty should not teach only 1 course in a day;
- A faculty's total unit load should match her targeted unit load value;
- A teacher's assigned courses should not vary too much in terms of subject kind;
- A course's student capacity should match the location's seating capacity;
- A faculty-event should be scheduled only on timeslots of its preference;
- Timeslots of a higher priority value are assigned first.

The objective, therefore, is to find a solution (satisfying all hard constraints) with the least number of soft constraint violations.

Formally, the definition of the UCFTP is taken from [4] as a combination of an Axial 4-index and 2-index assignment problems in the following: Let $\delta(x)$, $\varepsilon(x)$, $\phi(x)$, $\gamma(y)$ be the feasible assignments of course x to a faculty, course x to a timeslot, course x to a location, and faculty-event y to a timeslot, respectively. Let c^1_{ijkl} be the cost of assigning course i to faculty j to timeslot k in location l, and c^2_{mn} be the cost of assigning faculty-event m to timeslot n. UCFTP is finding the assignment that will minimize the total cost, as in

$$\min_{\delta,\, \varepsilon,\, \phi \in S_m,\, \gamma \in S_n} \sum_{i=1}^{m} c^1_{i\,\delta(i)\,\varepsilon(i)\,\phi(i)} + \sum_{n=1}^{n} c^2_{j\,\gamma(j)} ,$$

where S_m and S_n are the sets of all feasible assignments for courses and faculty-events, respectively, or the *search space sets*.

III. Ant Colony Optimization

In ACO, virtual colonies of ants iteratively construct solutions by moving through a search space graph and making component assignments through every *step* on the graph edges. The ants leave *pheromone* values on the paths they traverse depending on the quality of the solution they have built. Paths with higher pheromone content are more likely to be chosen by a traveling ant than ones with lower values. Additionally, pheromones on all edges are *evaporated* at a constant rate to prevent overwhelming differences between path values. This concept of pheromone trails, along with the mathematical graph representation of the problem, lies at the core of the ACO algorithm.

A. *Search Space Graph Representation*

The graphical representation of the UCFTP is taken directly from course timetabling problem in [12] where the list of events is iterated from start to end points and the elements of the set to be assigned to are replicated under each event as shown in Fig. 1. These elements are either all from *F*, *T*, or *L* if the events are courses, or from *T* if iterating through faculty-events.

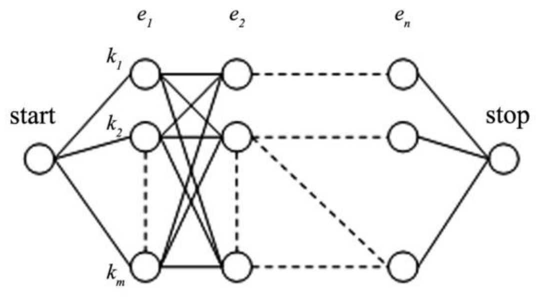

Figure 1. An ant builds a solution by travelling from start to end nodes, making probabilistic decisions at every step through the graph edges. Each step made in this way creates an assignment of event *e* to element *k*.

Programmatically, the assignable elements, or components, under each event are generated on-demand as they depend on the previous assignments so far, which is the present state of the partial solution at every event point. Thus, the search space matrices are at least the size of *C* x *F*, *C* x *T*, *C* x *L*, and *Fe* x *T* for course-faculty, course-timeslot, course-location, and faculty-event-timeslot assignments, respectively.

B. Pheromone Trail Model

A pheromone matrix is created based on the search space graphs by designating a value to every edge on the graph. This will influence an ant's decision when making assignments. An ant will assign a component to an event by randomly moving from one node to the next within its dynamically-created *neighbor list*. The probability p^k_{ij} of ant *k* in node *i* to move to node *j* is the probabilistic decision function:

$$p^k_{ij} = \begin{cases} \dfrac{\pi_{ij}^{\ \alpha}\ \eta_{ij}^{\ \beta}}{_{l\in N_i^k}\ \pi_{il}^{\ \alpha}\ \eta_{il}^{\ \beta}}, & \text{\&if } j \in N_i^k \\ 0, & \text{otherwise,} \end{cases}$$

where π_{ij} is the pheromone value of the edge [*i*,*j*], η_{ij} is a pre-determined heuristic information value of that same edge, and N^k_i is the neighborhood of ant *k* when in node *i*. After successfully creating a solution (i.e., moving from the start to the end nodes of the graphs), the ant will update the pheromone values of the paths it used by adding a value $\Delta\pi^k$ to each graph edge, implemented as follows:

$$\pi_{ij} \leftarrow \pi_{ij} + \Delta\pi^k, \quad \forall\ i,j\ \in A^s,$$

where π_{ij} is the pheromone content of edge [*i*,*j*] and A^s is the set of all edges in the ant solution. Evaporating the pheromone trails is done in a similar manner, but on every edge of the graph. It is implemented as follows:

$$\pi_{ij} \leftarrow\ 1 - \rho\ \pi_{ij}, \quad \forall(i,j) \in A,$$

where *A* is the set of all edges the search space graph.

C. Max-Min Ant System Algorithm

The Max-Min Ant System (MMAS) [14] variant of ACO is used in the implementation. In MMAS, minimum and maximum pheromone values are imposed on the trails and the pheromone matrix is initially set to the maximum value. Evaporation is done at a slower rate to promote exploration of the search space. Strong exploitation of the search history is encouraged by allowing only the ant with the best solution to update the pheromone trails. Extended initial exploration and exploiting of the best solutions make the MMAS algorithm one of the best existing metaheuristic variant for combinatorial optimization problems like timetabling [13].

Furthermore, to prevent stagnation of solution quality when ants converge on or near a single path they perceive as optimum, the pheromone trails are reinitialized to maximum values whenever such a situation is detected. Fig. 2 shows the pseudo-algorithm for MMAS-UCFTP.

An option is added in pheromone updating wherein the algorithm can choose whether to implement an iteration-best method, where the best ant of the iteration will update the pheromone trails, or best-so-far method, which updates the trails based only on the best existing solution created by the ants so far.

IV. Parallel and Distributed Architecture

The nature of the ACO algorithm makes parallelizing it very natural [6]. Different parallelization topologies in the literature have been considered. This includes fully-connected, ring, 2D and 3D meshes, and hypercube network topologies [1][7]. The aim is to improve performance by distributing the computation load across multiple processor nodes, at the same time minimizing the communication overhead over the parallel network cluster.

A proven best-performing parallelization strategy for the ACO algorithm is the Parallel Independent Run (PIR) topology [8]. It effectively cuts all communication overhead by implementing no communication between nodes at all. The server only communicates to the slave nodes at the beginning of the search process when all necessary data are obtained, and at the end of the search when the slave nodes send their results to the server (Fig. 3). PIR is used in this implementation.

```
procedure MMAS_UCFTP
    InitializeSearchSpace
    while not terminate
        for each ant
            ConstructSolution
            EvaluateSolutionQuality
        end for
        OptionalLocalSearch
        UpdatePheromones
    end while
end procedure
```

Figure 2. High-level pseudo algorithm for the MMAS-UCFTP.

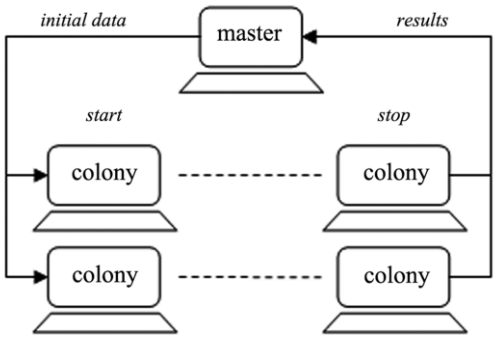

Figure 3. In a Parallel Independent Run topology, colonies get the initial data from the master at the start of the search process. No inter-node communication occurs until the search is done. During which, all colonies submit their results to the server.

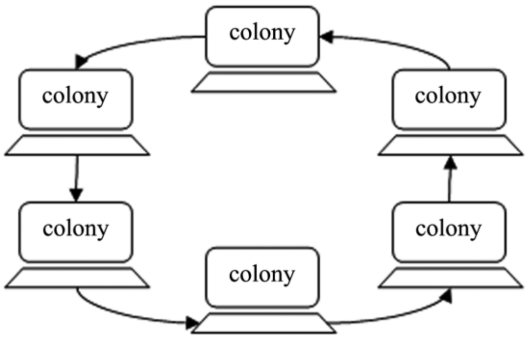

Figure 4. Unidirectional Ring topology diagram. In the MMAS-UCFTP, solution data are exchanged instead of the pheromone information. A colony will send its best solution to the next node in the ring and in the same way receives a new solution from the node behind it.

Another network topology employed to solve the UCFTP is the Unidirectional Ring (UR) topology. It has been tested in similar researches [8] and seen as very effective for ACO when data relay across slave nodes only involves the solutions being exchanged, instead of the entire pheromone matrix state. From time to time, a slave node (or ant colony) will send its best-solution-so-far to the next node within its ring. The recipient node will then compare that new solution to its own best one. If it exceeds the existing best solution in quality, it becomes the new best. This is shown in Fig. 4.

V. Search Parameter Calibration and Test Results

Seven real-world UCFTP instances were used for the test runs. These problem data were taken from groups of colleges within MSU-IIT during the 2nd semester of SY 2010-2011. The sizes of these instances are shown in Tab. I. The testing process was performed with an HP Proliant DL585 G7 AMD Opteron 6174 4x12 cores 2.20GHz 128GB RAM on CentOS 5.6 using the Erlang/OTP functional programming language. Every test was run for 20 minutes, utilizing 40 cores as ant colonies and 1 core for the main ant server. A total of 6 search classes were made and different parameters were entered for each class. Parameter *m* is the number of ants per colony, α and β represent the ants' sensitivity to the pheromone and heuristic information, respectively, and *pr* is the pheromone evaporation rate. Update method is a choice between a best-so-far update, an iteration-best, or both methods at once. An initial number of iterations, *ifs*, is given where no pheromone update or evaporation is allowed. Finally, each colony will send its best solution to the next one every *ring* iterations. A zero value indicates a PIR search.

During every test run, the algorithm was successfully able to generate feasible solutions. Thus, the objective of the testing was to observe the algorithm behavior under different settings and to find the optimum parameters for better search performance. The search parameters used for every test class are shown in Tab. II.

Test results are shown in Fig. 5-8. They are arranged to show the test search classes per problem instance. The first 3 test classes (URR, URN, and URX) are of the Unidirectional Ring network topology, while the rest use PIR.

TABLE I. Sizes of the UCFTP Instances

Problem Instances	courses	faculties	timeslots	locations	faculty-events
large1	741	150	44	58	10
large2	663	134	44	62	5
medium1	390	60	44	29	4
medium2	349	69	44	59	12
small1	278	37	44	85	0
small2	247	46	44	18	4
tiny1	175	40	44	29	4

TABLE II. Parameter Settings of the Search Classes

Search Class	m	α	β	pr	stagnation sensitivity[a]	update method	ifs	ring
URR	20	1	0	0.05	5	BSF	200	10
URN	7	1	1	0.2	10	IBU	100	10
URX	2	2	0	0.2	7	Both	100	30
PIRR	20	1	0	0.05	5	BSF	200	0
PIRN	7	1	1	0.2	10	IBU	100	0
PIRX	2	2	0	0.2	7	Both	100	0

a. A *lower* value means *more* sensitive to stagnation.

On larger problem instances **large1** and **large2**, PIR searches performed better than UR. On the other hand, UR is more likely to obtain better solutions on medium and smaller-sized instances. This behavior has been verified in literature [9] where a higher solution exchange frequency can have an adverse effect on the search due to communication overhead. It is made clearer in this implementation wherein the frequency of communication is dictated by a fixed parameter *ring*, instead of a function of the problem size, and the stopping condition is a time limit, not reaching some objective value.

Parameter settings from search classes suffixed -R and -N (URR, URN, PIRR, PIRN) are designed to encourage the ants to explore the search space, instead of over-emphasizing the use of pheromones and quickly converging on some local optimum (URX, PIRX). Researches have pointed out the importance of exploration over exploitation especially during the initial part of the search [5]. Tests indicate, however, that there must be a balance between exploration and exploitation as shown by the results of URR and PIRR, which are designed to make ants as purely explorative as possible. Both searches got the worst results in almost all instances.

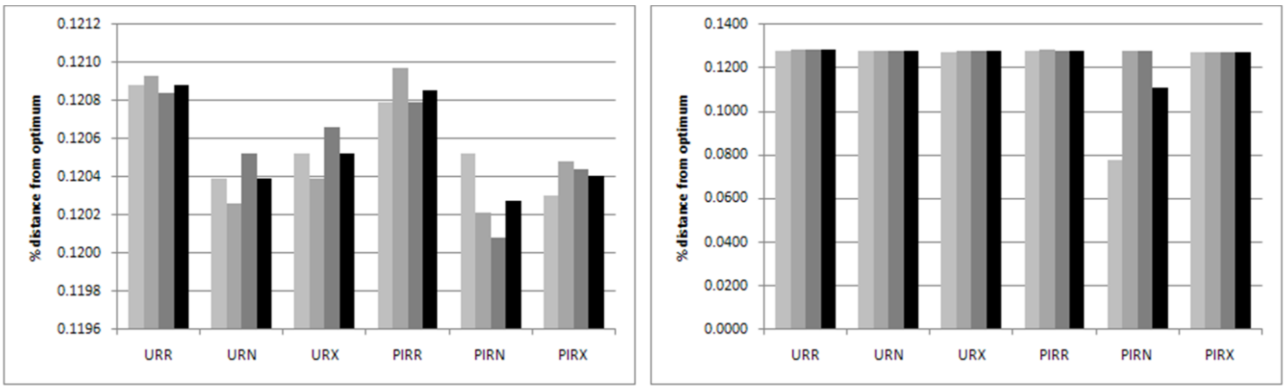

Figure 5. Search results for problem instances **large1** (left) and **large2** (right).

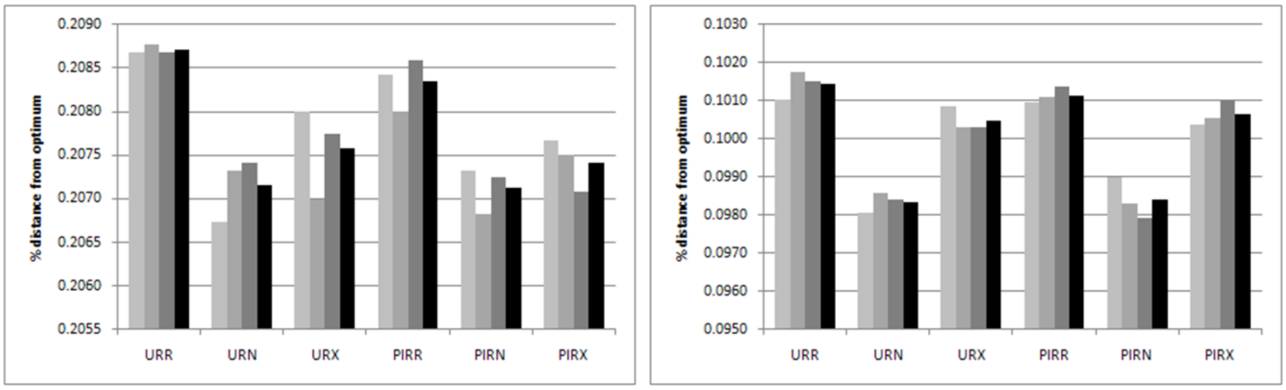

Figure 6. Search results for problem instances **medium1** (left) and **medium2** (right).

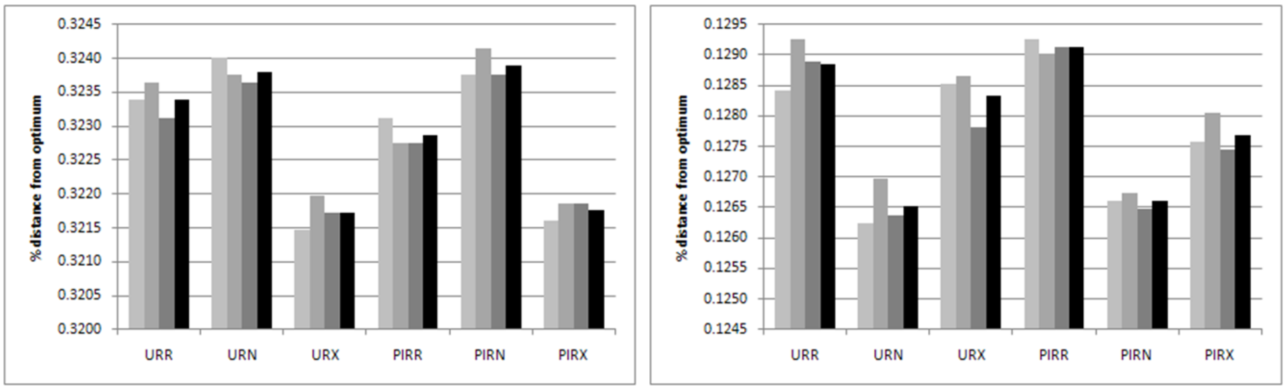

Figure 7. Search results for problem instances **small1** (left) and **small2** (right).

It is better to let ants explore "normally" (URN, PIRN) and not set the *ifs* value too high as to make the ants almost never converge at all.

Interestingly, exploitative ants on a Unidirectional Ring topology had the best results on instance **small1**. Upon manually examining this problem instance, it was determined that a solution without at least 30% soft constraint violations is mathematically impossible to obtain because of the problem elements' individual properties. Thus, in this case, it is more advantageous to converge on a quickly-found local optimum than to keep exploring a search space that would only yield solutions of no better, if not worse, quality, anyway.

VI. Conclusions

Past works [11] have confirmed that each ACO configuration will behave differently, even unexpectedly, in different problem instances or sizes. This is why it is very difficult to find an optimal, "one-size-fits-all" configuration best for all future cases of real-world UCFTP instances. For an effective MMAS-UCFTP, the algorithm implementation's parameter setting will then have to be calibrated and the problem instance's characteristics examined when such a case occurs.

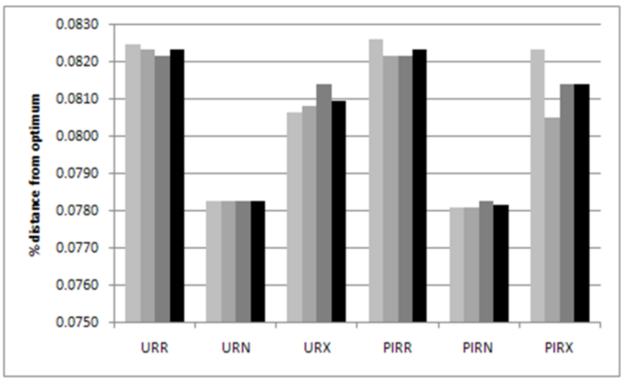

Figure 8. Search results for problem instance **tiny1**.

The following conclusions are made from this work:

- The ACO virtual ants must be given enough initial time during which they are encouraged to explore the search space. This has been verified in [5] and established that exploration is an integral part of the ACO-MMAS algorithm. The ants must also be able to finally exploit the best-solution memory and converge on a solution path upon concluding the search.
- Communication overhead should be taken into account when parallelizing an ACO algorithm. A policy where only solution information is exchanged within the ant colony cluster is best [15], although the most essential factor is the frequency of this exchange, which should be proportional to the search duration and problem instance size.

ACKNOWLEDGMENT

We would like to thank the Computer Science Department, the Computer Facilities and Support Services, and the High Performance Computing Facility at the MSU-IIT for allowing us to conduct this study; and the MSU-IIT Computer Science Alumni Association for the financial help extended.

REFERENCES

[1] Alba, E., "Parallel metaheuristics: A new class of algorithms," in Wiley Series on Parallel and Distributed Computing. Hoboken, NJ. Wiley-Interscience, 2005.

[2] Armstrong, J., Programming in Erlang: Software for a Concurrent World. Raleigh, NC, USA: The Pragmatic Bookshelf, 2007.

[3] Burkard, R. E. and Cella, E., "Linear assignment problems and extensions," in Handbook of Combinatorial Optimization, supp. vol. A, P. M. Pardalos and D. Za. Du, Eds. Kluwer Academic Publishers, 1999, pp. 75-149.

[4] Burkard, R. E., Dell'Amico, M., and Martello, S., Assignment Problems. Philadelphia, USA: Society for Industrial and Applied Mathematics, 2009, pp. 305-306.

[5] Dorigo, M. and Stützle, T., Ant Colony Optimization. Cambridge, MA, USA: The MIT Press, 2004.

[6] Dorigo, M. and Stützle, T., "The ant colony optimization metaheuristic: Algorithms, applications, and advances," in Handbook of Metaheuristics, F. Glover and G. Kochenberger, Eds. Kluwer, 2002.

[7] Janson, S., Merkle, D., and Middendorf, M., "Parallel ant colony algorithms," in Parallel Metaheuristics. Wiley, 2005.

[8] Manfrin, M., Birattari, M., Stützle, T., and Dorigo, M., "Parallel ant colony optimization for the traveling salesman problem," in Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006, vol. 4150, Lecture Notes in Computer Sciences, M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle, Eds. Berlin, Germany. Springer-Verlag, 2006.

[9] Manfrin, M., Birattari, M., Stützle, T., and Dorigo, M., "Parallel multicolony ACO algorithm with exchange of solutions." Brussels, Belgium: IRIDIA, CoDE, Université Libre de Bruxelles, 2006.

[10] Mayer, A., Nothegger, C., Chwatal, A., and Raidl, G. R., "Solving the post enrolment course timetabling problem by ant colony optimization." 2007.

[11] Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., et al., "A comparison of the performance of different metaheuristics on the timetabling problem". Metaheuristics Network, 2002.

[12] Socha, K., Knowles, J., and Sampels, M., "A max-min ant system for the university course timetabling problem," in The Proceedings of the International Workshop on Ant Algorithms, ANTS 2002, vol. 2463, pp. 1-13, Springer Lecture Notes in Computer Science. Springer-Verlag, 2002.

[13] Stützle, T. and Hoos, H., "Max-min ant system," in Future Generation Computer Systems, vol. 16, M. Dorigo, T. Stützle, and G. DiCaro, Eds. 2000, pp. 889-914.

[14] Stützle, T. and Hoos, H., "Improvements on the ant system: Introducing the max-min ant system," in Proc. Int. Conf. Artificial Neural Networks and Genetic Algorithms. Wien: Springer-Verlag, 1997.

[15] Twomey, C., Stützle, T., Dorigo, M., Manfrin, M., and Birattari, M., "An analysis of communication policies for homogenous multi-colony ACO algorithms," in Information Sciences Direct 180(2010), pp. 2390-2404. Information Sciences, Elsevier, Inc., 2010.

[16] Ugat, E., Montemayor, J.J., and Manlimos, M.A., "MyClass: MSU-IIT Faculty Workload and Class Assignment System – A Concurrent and Distributed Max-Min Ant System Application in Erlang," unpublished.

ABOUT THE AUTHORS

Earth B. Ugat
earth.ugat@g.msuiit.edu.ph

Earth is a developer and administrator of small to large scale network systems. He is a senior BS Computer Science student in MSU-IIT who also finds interest in PHP, C++, Java, skateboarding, and reading fantasy novels.

Jennifer Joyce M. Montemayor
jenniferjoyce.montemayor@g.msuiit.edu.ph

Jennifer is an experienced programmer, web designer, mobile applications developer and movie buff hailing from Baguio City. She received her BS in Computer Science at MSU-IIT in 2011 where she now works as an assistant lecturer for the Computer Science Department.

Mark Anthony N. Manlimos
markanthony.manlimos@g.msuiit.edu.ph

Mark earned his BS in Computer Science at MSU-IIT in 2011 and has since worked there as an assistant lecturer for the Computer Science Department. He is a PHP, MySQL, and web frontend engineer who is also into experimental cooking, dancing, and reading.

Dante D. Dinawanao, M.Sc.
dante.dinawanao@g.msuiit.edu.ph

Dante is a systems administrator and assistant professor in the MSU-IIT Computer Science Department. He obtained his Master's degree at De La Salle University in 2003. He finds programming, distributed computing, music, education, and beer terribly exciting.