AI-BASED TIMETABLING ALGORITHMS: A COMPARATIVE ANALYSIS

[1]FERNANDO S. VIRAY JR., [2]MELVIN A. BALLERA

[1,2]Technological Institute of the Philippines (TIP) – Manila
E-mail: [1]skyfher@gmail.com, [2]melvin.ballera@tip.edu.ph

Abstract - AI-based timetabling algorithms are keys in programmatically providing an ideal and conflict-free university class schedules among academic institutions worldwide periodically regardless of institutional structure or complexity of offered programs. Considered as an NP-hard problem, timetabling algorithms are founded on local search and optimization techniques which are the foundations of artificial intelligence's (AI's) base algorithms. Theoretically, timetabling algorithms look for optimum solutions rather than feasible ones, thus, incorporating a significant computational power and time in relation to schedule constraints. In this paper, the researchers evaluated four of the most commonly usedAI-based timetabling algorithms, namely, Tabu Search, Greedy Algorithm, Integer Linear Programming, and Bi-Partite Graph Approach, to determine which algorithms works best in terms of number of constraints, computation time, and computation resources.

Keywords - Timetabling Problem (TP), University Class Scheduling Problem (UCSP), local search and optimization techniques

I. INTRODUCTION

The Timetabling Problem (TP), also commonly known worldwide as the University Class Scheduling Problem (UCSP)[1], has been considered as one of the most tedious and periodic endeavors encountered among academic institutions globally, be it among local grade schools or large universities.[2] The problem requires a number of courses with specified subjects, students who belong to a certain course and are required to take subjects as specified on a student's respective course curriculum, teachers who are assigned based on their expertise or specializations, and a pool of classrooms (as either ordinary classrooms or specialized facilities commonly known as laboratory rooms for distinct subjects/courses) with a goal to assign a class of students belonging to the same course subjects to classrooms and teachers while at the same time satisfying all of the university policies and optimizing the utilization of existing facilities effectively and efficiently.[3] Policies or rules in creating an academic class schedule that should be followed as part of solutions are called hard constraints while those rules or guidelines that can be forgone but a solution to the timetabling problem can still be reached are referred to as soft constraints.[4]It is of common logic and has been the acceptable standard that a timetabling solution, for it to be called either an optimum or feasible solution[5], should be, at least, conflict-free among hard-constraints[6], a criterion that held the timetabling algorithms as part of local search and optimization techniques which are one of the computational foundations of Artificial Intelligence (AI)[1]– techniques that are regarded as a nondeterministic polynomial time (NP) hard problem[5] which means that it is easier to verify in polynomial time that a solution, if one has been found, is in fact a solution of the problem rather than find the solution for the problem as there is no known efficient and express way of solving the problem, and although it can be solved, it does not guarantee optimality.[7]

This research aims to assess the solution-making background of the four most-commonly used heuristics and AI-based local search and optimization algorithms, namely: Tabu Search, Greedy Algorithm, Integer Linear Programming, and Bi-Partite Graph Approach. It also aspires to determine which among the stated timetabling algorithms work best in terms of the following factors: number of hard and soft constraints, time, and computational resources (computer memory and CPU usage) over a set of input parameters with the use of an entry-level desktop computer. This study is timely and targets to benefit application developers and academic schedule in-charge to determine which algorithm is suited for their software development endeavors or academic institution, respectively.

II. RELATED WORKS

The Timetabling Problem is refersto the problem of assigning students a set of specific subjects from their respective courses into a particular limited set of classrooms and teachers.[8] It originated from the Job-Shop Scheduling Problem (JSP) in which certain jobs are allocated to resources at distinct times, replacing jobs as subjects from course programs to be taken by students and resources as designated classroom and teachers at a particular timeslot.[9]The solution entails local search and optimization processes as it requires finding an optimum solution among possible solution sets.[5] Recently, the utilization of machine learning as heuristicsmeans to scheduling problems became an emerging method[1][10]. Under this methodology, artificial intelligence (AI) ascertains an optimum

solutionusing a computer code that mimics human intelligence and intuition that can comprehend complex calculations and causality verification.[10]

Aside from the required inputs for Timetabling algorithms, rules from inputs are set which controls and dictates the computational complexity of the problem. These rules are called constraints which are classified as either hard constraints or soft constraints.[3]Hard constraintsare the required parameters or factors in which a solution cannot be derived without taking these parameters into full consideration. Conversely, soft constraints as optional parameters used to improve the solution's quality because satisfying one of them means the solution meets some particular preference; therefore, a feasible solution can be found satisfying or not a soft constraint.[4]A solution to timetable programming that violates even a single hard constraint is not feasible, whereas a solution that violates any number of soft constraints remains feasible.[7]

The inputs and constraints applied to Timetabling problems make the problem unique to a particular academic institution as the number of students, offered programs, academic resources and provision of degree of stakeholder satisfaction vary from school to school.This scenario gives rise to custom-built and case-based computerized timetabling software and the difficulty to develop a generalized model and "one-size-fits-all" timetabling computer program given our regional, ethnic, curricular and time zone differences. Nevertheless, a generalized model of scheduling algorithms are being continuously developed to serve as backbones and foundations of school timetabling computer applications.[11]

2.1Sub-Problems of the Timetabling Problem

a. Teacher Assignment

The teacher assignment sub-problem involves assigning and scheduling teachers to specific courses by taking some factors, such as the abilities of the teachers and number of courses offered, into consideration. The aim is to balance the full time teachers' load, where the load refers to the total number of credits assigned to each teacher.[2] For example, if a 3-credit course is assigned to two teachers, the total load of each teacher will be incremented by 1.5 credits.

The requirements imposed on the teacher assignment sub-problem which are listed as follows:

- Each course section has a minimum and maximum number of teachers to teach.
- Part time teachers are required to teach courses that cannot be taught by full time teachers. Due to internal regulations, certain part time teachers, especially those from government universities, have to be involved in the teaching process.
- There are both a maximum and minimum number of teachers that can be assigned to a particular course. This depends on the number of course sections offered.
- The total number of different courses taught by each full time teacher should not exceed a certain number. The aim of this constraint is to reduce the amount of preparation time for each teacher.
- The teaching load for full time teachers is prioritized to be balanced as compared to part time teachers.

The first four requirements are regarded as hard constraints that need to be satisfied, while the last one is treated as a soft constraint or preference that is considered to be equivalent to minimizing the total weighted variance of the teachers' load.[2]

b. Course Scheduling

The course scheduling sub-problem comprises of assigning and scheduling the subjects to be offered by school on a specific time period and the number of sections for a particular course by following a course curriculum on how subjects are to be offered or to be taken by students systematically. The aim in solving the course scheduling sub-problem is to create a schedule of subjects and subject/course sections based on the subjects to be offered as specified by the course curriculum per year level and the number of registered students.[2]

The Course Scheduling sub-problem have the following factors and constraints:[2]

- Each student of the school has its own curriculum
- A curriculum consists of a given set of courses/subjects.
- The courses are divided into lectures of given length, and each lecture must be taught during consecutive time periods. The length of a lecture is its duration which is supposed to be at most two periods. The set of available periods of the week within which all curricula must be completed is given.
- Each day consists of at most two half days (i.e., morning and afternoon) with a fixed lunch time in-between. The periods in each half day are consecutive and their number is called the length of the half day. Some periods (for example in the late afternoon) are called bad periods in that sense that important courses should not be scheduled at these periods.
- The students are classified according to their level. The curricula of two students with the same level can be considerably different. In general, a course is followed by students with the same level. The level of a course (respectively level of a lecture) is defined as the level of the students following this course (respectively lecture). Some courses appear in the curriculum of each student having the same level. Such courses are called compulsory courses as opposed to optional courses.

Large courses on which numerous students are enrolled have to be repeated several times during the

week. Students registered in such courses must then be assigned to a specific course section and that a course which is not repeated has only one section. All sections of a course need not be taught by the same teacher. Hence, two sections of a course may have periods in common. The availabilities of the teachers as well as the set of course sections in which they are involved must be known in advance (i.e., subjects for higher year levels can be assigned several sections based on the assumed number of returning students).[3]

c. Class-Teacher Timetabling

The class-teacher timetabling sub-problem concerns scheduling teachers and classes over a set of periods, typically covering five or six week days depending upon the institutional policy. The basic constraints that must be satisfied are: i) professors can only be in one class at any given time; and ii) students belonging to a section or class can only attend to a teacher's lecture at any given time. This sub-problem focuses on a) assigning substitute teacher to a particular class in the absence of the regular assigned teacher; and b) merging of classes/sections having the same subject to take under the same curriculum.[12]

The class-teacher timetabling sub-problem is different from the teacher assignment problem as this specifically sets the class section in which a specific teacher will handle for a given period. Additionally, this problem is different from the course scheduling problem as it focuses more on opening or closing subject offerings based on number of students and teachers as per curriculum standards.[12]

d. Student Scheduling

The student scheduling sub-problem deals with the creation of a balanced, almost gap-free or compact student schedules for a specific time period based on the curriculum the student is under with. A balance and ideal student schedule speaks of a well-divided number of subject hours for the student in a week while gap-free or compact means that, as much as possible, subjects are contiguously scheduled (except for lunch break, morning and afternoon breaks or recess times) in order to maximize the usual 8-hour whole day class period.[13]

The hard constraints of this sub-problem are the students' list of selected courses and the master schedule of course offerings with their multiple sections, each with possibly a number of meeting times, rooms and instructors.[13]

e. Room Assignment

The room assignment sub-problem covers allocating classes or block sections to classrooms which in mathematical terms, is the same as the problems of hotel room assignments, shipyard scheduling, job scheduling on machines with varying capabilities, the assigning of airport gates to flights, etc. The only constraint of this problem is the list of available classrooms, however, since some subjects require specific facilities (laboraties, Audio-Visual rooms, etc.), the nature on how the room will be used attaches to the subject as a form of hard constraint and are based on specific criteria as follows, in decreasing order of priority: [9] (1) use of general use lecture classrooms; (2) use of specialized classrooms /facilities (laboratory or AVP rooms, etc.); (3) student capacity of each classroom; (4) estimated enrollment turnaround; (5) length of period the classroom will be used for a particular subject; and (6) use of lecture rooms by respective college/department.

2.2Commonly Used AI-based Timetabling Algorithms

A lot of works and heuristic algorithms have been proposed to solve the Timetabling Problem which are based on local search techniques.[5] This section provides different analytical timetabling algorithms based on local search optimizations.

a. Tabu Search

The TabuSearch algorithm was created in 1986 by Fred W. Glover and was formalized three years thereafter. It is a metaheuristic search method which employsmathematical local search optimizationsschemes.[14]

Local searches means taking a candidate solution by checking its immediate solutions (called neighbors) from a solution set that are similar but has very minute details hopingto find an enhanced solution by easing its basic rules (aspiration criteria). However, when there are many similar solutions, called suboptimal regions, are a fit to the possible solution being compared, this method becomes stuck as each candidate solution matches the relaxed criteria it is looking for.

If no possible similar solution among neighbors are present and that neighbors do not satisfy the strict local minimum, the neighborhood is marked and remembered and prohibitions are introduced (appending its address and values in the Tabu List), thus the term "tabu", to prevent searching again the visited and marked neighborhood.[10] Below is a pseudo code for the Tabu Search.

- Set TabuList={}; TabuL=TL; k=1;
- Choose an initial solution s ∈ S
- ŝ ∈ s;
- Repeat
- Generate N(s,k) Є N(s);
- Evaluate each s Є N(s,k);
- Modify the neighborhood N'(s,k)=N(s,k)-TabuList;
- Choose the best solution s' Є N'(s,k);
- Move to s=s';
- If solution s is better than ŝ then ŝ=s;
- Update the TabuList and aspiration criteria;
- k=k+1;

- Until stopping condition is met;

Theimplementation of the Tabu Search algorithm to solve the Timetabling problem in the hope to ensure that hard constraints are satisfied and obtain more feasible solutions through swapping of lectures to avoid conflict among lectures of instructors, has been noted that although the Tabu Search algorithm is more time consuming as complexity increases with directly proportional increase in constraints, it is an efficient algorithm in terms of using limited computing resources and has effectively avoided loops in cycling back to previously visited solutions through memory structure called Tabu List.[15]

b. Greedy Algorithm

Greedy algorithm is an algorithmic model that adheres to the problem-solving heuristic of making the local optimal choice at each phase with hoping to find anoverall optimum. Generally, a greedy technique does not search for an optimal solution, however, it may produce optimal solutions from each locals that estimates an overall optimum solution at a significant period of time.[6]

The greedy algorithm finds the optimum solution in timetabling problems by: 1) choosing the interval x that will finish first; 2) removing the interval x and all other intervals that intersect with the interval x from a set of candidate intervals; and 3) repeating until a set of candidate intervals is emptied. When choosing an interval at step 1, it is opted to remove a lot of intervals in step 2. However, all given intervals may intersect the time when x will be finished, and as such, these intervals will intersect each other at the same time period. Thus, at least 1 of these candidate intervals can be considered as an optimum solution, which is a proof that this algorithm will really indeed searchfor an optimum solution, hence, the term "greedy".[7]Below is the pseudo code for the standard Greedy Algorithm:

```
Algorithm Greedy (a,n)
//a[1:n] contains the n inputs,
{
solution:=0; //initialize the solution
  for i:=1 to n do
  {
x:=Select(a);
    if Feasible(solution,x) then
solution:=Union(solution,x);
  }
 return solution;
}
```

The utilization of the greedy algorithm to solve the timetabling problem by considering a complicated multiple hard constraint conditions such as 1) a uniform teaching resources, which include teachers, students, and classrooms, with practicable timeslots, and 2) a balance for all the curricular loads of students and teachers' satisfaction of preferred lecture schedule, has shown that the greedy algorithm is indeed effective and that the runtime is significantly shortened as compared to other techniques used in the timetabling system.[8]

c. Integer Linear Programming Algorithm

Linear programming (LP), which is also called as linear optimization, is a technique to attain the best result in a mathematical model that requires to be characterizedthrough linear relationships.[7]It is a special case of mathematical optimization with a feasible region called a convex polytope composed of a set of finitely many intersecting half spaces derived using linear equality and inequality constraints. The main goal of this algorithm is to search for a point in the polyhedron's function where the largest or smallest value exists. If the value being searched for and values to choose from the feasible region needs to be integers, then the the problem is referred to as an integer programming (IP) or integer linear programming (ILP) problem.[16]

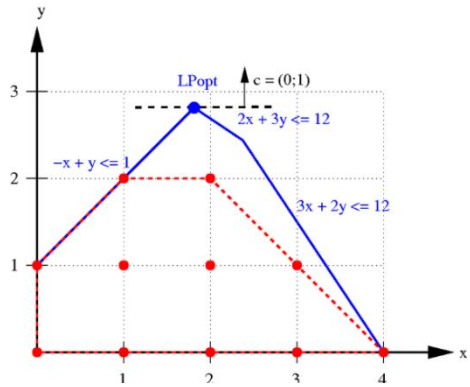

Figure 1. A graph showing Integer Programming Polytope from a relaxed Linear Programing Polyhedron derived from multiple inequality formulae(image source: www.mathcs.emory.edu).

Figure 1 illustrates how best result (largest value) are derived using integer programming. Given a polyhedron (blue line) defined by several inequality formulae, the goal (black dotted line above the polyhedron) is to find the biggest integer values when the highest point (LPopt) of the polyhedron is extended upwards. Relaxing the polyhedron (to attain integer values) is done by creating a polytope (red dots) by marking all the integers inside the original polyhedron which will form the smallest polyhedron (red dotted line) possible inside the original polyhedron. Among the points in the polytope, highest integers [points (1,2) and points (2,2)] will be selected base from the goal. These values are different if points from the original polyhedron are rounded up or down.

There are various fields of study where linear programming can be utilized. Among these is scheduling which involves vehicle service scheduling among transportation networks which then can be adapted to university class timetabling by modifying

the routes to courses/subjects, vehicles to classrooms, vehicle drivers to teachers,and passengers to students.[16] Integer programming techniques can be used to model university class scheduling which can integrate several academic operational requirements and constraints and generate a feasible outcome that can be used for practical use.[9]

d. Bi-Partite Graph Approach

A bi-partite graph, also called a bigraph, is a graph that showcases the relative connections (relations) of the elements of two or more sets. Each set is called a part of the graph or vertex sets while elements of the sets are called vertices. Graph colouring approach, an adaptation of the bi-partite graph, uses a unique color for each vertex set for easy visualizations.[6]

In a bi-partite graph, an element from a set is linked through a line to one or all of the elements of another set, thus, forming a bi-partite relations.[7] Figure 2 exhibits two vertex sets (U and V) in which each element from both sets are related to all elements (vertices) from the other set.

Bi-partite graph approach can be used in timetabling problems by converting the list of teachers, classrooms, subjects, students, courses into vertex sets where each element in the list will become an element of a vertex set. Afterwhich,

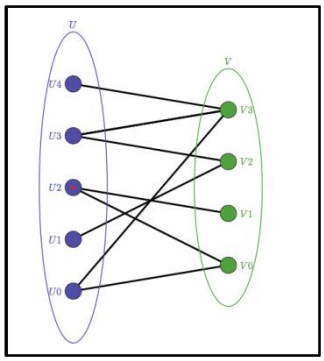

Figure 2: A simple bi-partite graph of vertex sets U and V showing the relationships of their corresponding elements as vertices.(image source: stackexchange.com).

relations will be linked from corresponding vertices. From these data, vertex value matching, collisions and non-collisions, and possible optimizations can be performed.[10]Results of experimental studies demonstrated that utilizing bi-partite graph approachcan solve university timetabling problems.[17]

III. METHODOLOGY

The evaluations in this paper were conducted into two ways: constraint evaluation and computational hardness evaluation. The constraint evaluation assessed how much of each subject local search and optimization algorithm satisfies all hard constraints and the number of soft constraints disregarded. Meanwhile, the computational hardness evaluation calculated the total time and computer resources (memory and processor)consumed by each algorithm until it finds a feasible solution.

To conduct the evaluations, the researchers reviewed similar online scheduling algorithm sources and created a VisualBasic.NET tool (Figure 3) using Microsoft Visual Studio 2010 to simulate each algorithm. The researchers incorporated the requirements, factors and guidelines discussed in each sub-problem of the Timetabling Problem. The researchers also devised a mechanism to read test data from an input xml file.

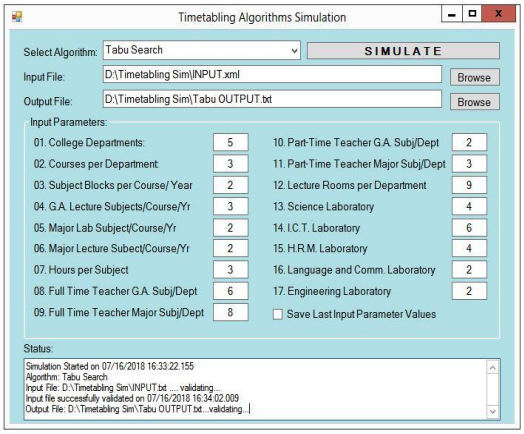

Figure 3:Simulation of AI-based local search and optimization algorithms.

A dummy set of hard and soft constraints, list of teachers, classrooms, subjects, block sections, and students were prepared by the researchers which served as test data for the simulation of each algorithm. The test data is equivalent to an enrollment scheduling data for a semester of 5 college departments, each with four 4-year degree courses for morning shift. Tables 1 and 2 display the details of constraints and input parameters used in this study.

Table 1: Input Constraints

#	Constraint	Type
1	Subjects to follow are based on curriculum	Hard
2	Number of hours per subject per week is equal to the corresponding subject's number of units	Hard
3	Laboratory subjects are to be held in respective laboratory rooms	Hard
4	Full time teachers' daily teaching load minimum is 5 and maximum of 6	Hard
5	Daily student lesson hours' minimum is 4 and maximum of 8	Hard
6	Teachers should teach a subject based on expertise/specialization	Hard
7	Classes are to be held from Monday to Friday	Hard
8	Part-time teachers daily minimum teaching load is 2 and maximum of	Soft

	4	
9	Lecture classes are prioritized to held at respective departments	Soft
10	Each subject is to be taught at least twice a week	Soft

Listing of Hard and soft constraints used in the simulation of the four AI-based local search and optimization algorithms.

Table 2: Input Parameters

#	Parameter	Count
1	College Departments	5
2	Courses per College Department (4-Year)	3
3	Subject Blocks per course per year	2
4	General Academic Lecture Subject per course / yr	3
5	Specialization Laboratory Subject per course / yr	2
6	Specialization Lecture Subject per course / year	2
7	Hours per subject (Laboratory and Lecture)	3
8	General Academic Subject Fulltime Teacher per Dept.	6
9	Specialized (Major) Subject Fulltime Teacher per Dept.	8
10	General Academic Subject Part-time Teacher per Dept.	2
11	Specialized (Major) Subject Part-time Teacher per Dept.	3
12	Lecture Classrooms per Department	9
13	Science Laboratory	4
14	I.C.T. Laboratory	6
15	H.R.M. Laboratory	4
16	Language and Communication Laboratory	2
17	Engineering Laboratory	2

Listing of input parameters used in the simulation of the four AI-based local search and optimization algorithms.

The researchers devised and used the following formulae to check the validity of the count of each input parameter in
connection with the given constraints. Equation 1 pertains to the Subject Hours Sh formula:

$$Sh = C \times Y \times S \times D \times B \times H \quad (1)$$

to get the number of subject hours in a week where C equals to the umber of Course per Department, Y equals the number of Year Level per Course, S equal s the number of Subject (Lab/Lec) per Course, D equals the number of Departments, B equals the Number of Blocks per Year Level, and H is the allotted Hours per Subject.

Equation 2, Rooms formula, is devised to obtain the minimum number of rooms required:

$$Rooms = \frac{Sh}{Rh} \quad (2)$$

where Sh refers to the theresult of Subject Hours formula and
Rh is the allotted hours to a room per week.

To get the number of weekly teaching hours per teacher ITH, Equation 3, the Individual Teacher Hours formula, is devised:

$$ITH = Th \times d \quad (3)$$

where Th refers to the minimum or maximum number of teaching hours per day and d as the number of days per week. Meanwhile, to get the minimum required number of teachers, Equation 4 is used:

$$Teachers = \frac{Sh}{ITH} \quad (4)$$

where Sh is the result of Equation 1 and ITH equals the result of Equation 3.

The goal of the evaluation is for algorithms to prepare a conflict-free schedule for 95 teachers, 53 rooms and 840 class blocks.

Meanwhile, to compute the total percentage of constraints satisfied/violated, a point deduction is used. Each schedule to be created for individual teachers, rooms, and blocks automatically has 100% hard constraint satisfaction score and another 100% soft constraint satisfaction score. Violations that occurred for the respective constraint for each schedule will be counted and deducted in the default score. The scores are then averaged and ranked from highest to lowest satisfactionscores.
During each scheduling algorithm simulation, the VB.NET tool that the researchers created records the start, end and elapsed times in ticks in order to monitor which among the algorithms performed well in accordance with time.A single tick represents one ten-millionth of a second or one hundred nanoseconds, this means that there are 10,000 ticks in a millisecond, or 10 million ticks in a second[13].To convert the number of ticks covered by the hash function in generating hash values into seconds, the Equation 5 is used:

$$time_{seconds} = \frac{time_{ticks}}{10{,}000\ ticks/second} \quad (5)$$

Meanwhile, the researchers used Microsoft's Process Monitor version 3.50 to monitor and log the real-time effect of the VB.NET tool created with regards to computer memory and CPU usage.[18] Microsoft's Process Monitor version 3.50 is part of Microsoft's

Windows Sysinternals utilities that help users in managing, diagnosing, and troubleshooting Windows systems and applications in real-time.[19] The simulation was run on a regular desktop computer with 2.4 Gigahertz Intel Core i5 processor, 4 gigabytes of random access memory, 256 gigabyte of hard disk space, and installed with Microsoft Windows 8 64-bit operating system. Results of the simulation are saved via a regular text output file in order to allow manual review of each schedule created.

IV. RESULTS AND DISCUSSIONS

Figure 4 shows the results of the constraints evaluation of the 4 subject AI-based scheduling algorithms in terms of the number of hard constraints satisfied and the number of soft constraints ignored. It exhibits that all the subject algorithms have successfully found a feasible solution without violating any of the 7 hard constraints. It also showcases the superiority

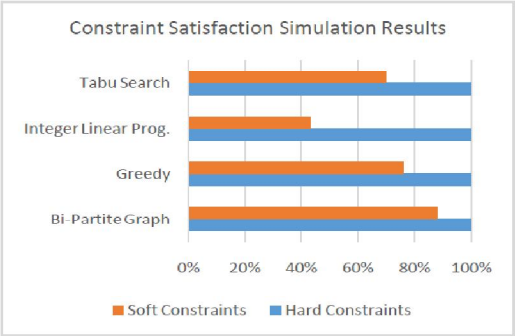

Figure 4: Constraints Evaluation Results of the 4 subject AI-based Timetabling Algorithms.

of Bi-Partite graph approach in terms of satisfying most of the soft constraints (869 out of 988) followed by Greedy algorithm (751) and Tabu Search (691) and leaving behind Integer Linear Programming with the lowest number (427) of soft constraints satisfied.

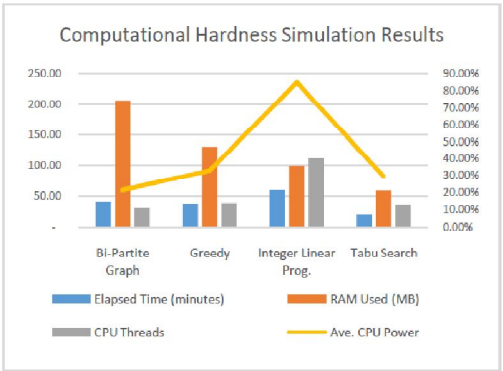

Figure 5: Computational Hardness Evaluation Results of the 4 subject AI-based Timetabling Algorithms in terms of computer memory and processor consumption.

Figure 5highlightsthe computational hardness evaluation regarding the total time and computer resources (memory and processor) consumed by each algorithm until it finds a feasible solution.Tabu Search ranks first in terms of the fastest algorithm (finishing after 20 minutes, 31 seconds and 188 milliseconds) to find a feasible solution without violating a hard constrained and satisfying a significant number of soft constraints while Integer Linear Programming performed to be the slowest (finishing after 1 hour, 8 seconds and 5 milliseconds). In terms of computer memory consumption, Tabu Search leads other contending algorithms in consuming the least RAM at 58.951 megabytes while Bi-partite graph algorithm managed to consume most of the RAM at 206.091 megabytes. Finally, in terms of processor speed, Bi-partite graph approach used the smallest number CPU threads (21) and average CPU processing power (31.21%) while Integer Linear Programming recorded the highest CPU threads (112) and average CPU processing (85.11%).

CONCLUSION

All of the subject AI-based Timetabling algorithms found a feasible solution in finding a conflict-free schedule for 95 teachers, 53 rooms and 840 block classes with a 100% hard constraints satisfaction.Although no algorithm has 100% satisfied the soft constraints, the Bi-partite graph approach has shown a significant lead of 87.96% in soft constraints satisfaction. Simulation results also proved that among the 4 subject algorithms, Tabu Search performed the fastest algorithm to find a feasible solution in no less than 21 minutes while consuming the least computer RAM at less than 98 MB. Finally, the Bi-partite graph approach has consumed the smallest number of CPU threads and least average processing power at 21 and 31.21%, respectively.

REFERENCES

[1] Poole, David; Mackworth, Alan (2017). Artificial Intelligence: Foundations of Computational Agents (2nd ed.). Cambridge University Press. ISBN 9781107195394.

[2] Domenech, B. &Lusa, A. (2016). A MILP model for the teacher assignment problem considering teachers' preferences. European Journal of Operational Research Volume 249, Issue 3, 16 March 2016, Pages 1153-1160. Elsevier.

[3] Pereira, Valdecy and Costa, Helder Gomes (2016). Linear Integer Model for the Course Timetabling Problem of a Faculty in Rio de Janeiro. Advances in Operations Research Volume 2016. doi: http://dx.doi.org/10.1155/2016/7597062.

[4] Almeida, Maria Weslane S., Medeiros, João Paulo S. and Oliveira, Patrícia R. (2015). Solving the Academic Timetable Problem Thinking on Student Needs. 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). Florida: IEEE. ISBN: 978-1-5090-0287-0.

[5] Babaei, Hamed; Karimpour, Jaber; and Hadidi, Amin (2015). A Survey of Approaches for University Course Timetabling Problem. Computers and Industrial Engineering Volume 86. Elsevier.

[6] Christian, Brian and Griffiths, Tom (2016). Algorithms to Live By: The Computer Science of Human Decisions. Henry Holt and Company. ISBN: 1627790373.

[7] Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. (2014). Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill. pp. 966–1021. ISBN 0-262-03293-7

[8] Zhang, D., Guo, S., Zhang, W. & Yan, S. (2014). A novel greedy heuristic algorithm for university course timetabling problem. Intelligent Control and Automation (WCICA), 2014 11th World Congress On Intelligent Control and Automation. IEEE. Shenyang, China

[9] Phillips, Antony, Waterer, Hamish, Ehrgott, Matthias, and Ryan, David (2014). Integer Programming Methods for Large Scale Practical Classroom Assignment Problems. Lancaster EPrints 2015. Accessed February 12, 2018.

[10] Russell, Stuart J.; Norvig, Peter (2015), Artificial Intelligence: A Modern Approach 3rd Edition (Paperback). New Jersey: Prentice Hall, ISBN: 9332543518, 978-9332543515.

[11] Fonseca, George H.G., Santos, Haroldo G., Carrano, Eduardo G., Stidsen, Thomas J.R. (2017). Integer programming techniques for educational timetabling. European Journal of Operational Research Volume 262, Issue 1, 1 October 2017, Pages 28-39. Elsevier.

[12] Bahel, M. & Thomas A. (2017). Innovative Evolutionary Algorithm Approach for Class-Teacher Timetabling Problem.Bhilai Institute of Technology, Durg District, India.

[13] Cheng, E., Kruk, S., &Lipman, M.J. (2013). Flow Formulations for the Student Scheduling Problem. Practice and Theory of Automated Timetabling IV, 4th International Conference, PATAT 2013, Gent, Belgium.

[14] Rovatsos, Michael, Vouros, George & Julian, Vicente (2015). Multi-Agent Systems and Agreement Technologies. 13th European Conference, EUMAS 2015, and Third International Conference, AT 2015, Athens, Greece. Springer. ISBN: 9783319335094.

[15] Bindra, Richa, Modi, Nishank, Shah, Rahil and Puri ,Simran (2014). University Course Scheduling using Tabu Search Algorithm. Thesis: University of Waterloo. Retrieved 20 March 2018.

[16] Neapolitan, Richard; Jiang, Xia (2018). Artificial Intelligence: With an Introduction to Machine Learning. Chapman & Hall/CRC. ISBN 978-1-13850-238-3.

[17] Ganguli, Runa and Roy, Siddhartha (2017). A Study on Course Timetable Scheduling using Graph Coloring Approach. International Journal of Computational and Applied Mathematics Volume 12, Number 2 (2017), pp. 469-485. Research India Publications. ISSN 1819-4966.

[18] Russinovich, Mark (2018). Process Monitor v3.50. Retrieved from https://docs.microsoft.com/en-us /sysinternals /downloads/procmon.

[19] Russinovich, M., Kim, L., and Sharkey, K. (2017) Windows Sysinternals. Retrieved from https://docs. microsoft.com/en-us/sysinternals/.

[20] Microsoft Developer Network. Article: DateTime.TicksProperty.https://msdn.microsoft.com/en-us/library/ system.datetime.ticks(v=vs.110).aspx, accessed 2017-03-09.

★★★