Automated Timetabling Using Stochastic Free-Context Grammar Based on Influence-Mapping

Hany Mahgoub
Department of Computer Science
Faculty of Computers and Information
Menoufia University, Shebin El-Kom, EGYPT

Mohamed Altaher
Department of Information Systems
Faculty of Computer and Information Sciences
Ain Shams University, Cairo, EGYPT

***Abstract*—This paper presents a new system that solves the problem of finding suitable class schedule using strongly-typed heuristic search technique. The system is called Automated Timetabling Solver (ATTSolver). The system uses Stochastic Context-Free Grammar rules to build schedule and make use of influence maps to assign the fittest slot (place & time) for each lecture in the timetable. This system is very useful in cases of the need to find valid, diverse, suitable and on-the-fly timetable which takes into account the soft constraints that has been imposed by the user of the system. The performance of the proposed system is compared with the aSc system for the number of tested schedules and the execution time. The results show that the number of tested schedules in the proposed system is always less than that in aSc system. Moreover, the execution time of the proposed system is much better than aSc system in all cases of the sequential runs.**

Keywords*—*Heuristic Search; Automated Timetabling; Stochastic Context-Free Grammar; Influence Map

I. INTRODUCTION

The timetabling problem is a famous problem, consists in scheduling a sequence of lectures among teachers and students in a pre-fixed period of time satisfying a set of constraints of various types [1][2][3][4]. The manual solution of the timetabling problem usually requires many person-days of work. In addition, the solution obtained may be unsatisfactory in some respect; for example a student may not be able to take the courses he/she wants because they are scheduled at the same time. For the above reason, a considerable attention has been devoted to automated timetabling.

The two most common forms of this problem are exam-timetabling problems and course timetabling problems, and in reality, the constraints imposed upon these can often be quite similar. However, the crucial difference between them is usually considered to be that in exam timetables, multiple events can take place in the same room at the same time, whilst in course-timetabling problems; we are generally only allowed one event in a room per timeslot. In automated timetabling, the constraints for both types of timetabling problem generally tend to be separated into two groups: the hard constraints and the soft constraints. Hard constraints have a higher priority than soft, and will usually be mandatory in their satisfaction. Indeed, timetables will usually only be considered feasible if and only if all of the hard constraints of the problem have been satisfied [5]. The Hard constraints and soft constraints are illustrated as follows:

1) Hard Constraints cannot be violated under any circumstances (mainly due to physical restrictions). For example, conflicting lectures (i.e. those which involve common resources such as students) cannot be scheduled simultaneously. Another example is that the number of students taking a lecture cannot exceed the total seating capacity of the rooms.

Soft Constraints are desirable but are not absolutely critical. In practice, it is usually impossible to find feasible solutions that satisfy all of the soft constraints. Soft constraints vary (and sometimes conflict with each other) from one institution to another in terms of both the types and their importance. The most common soft constraint in the timetabling literature is to spread conflicting lectures as much as possible so that students can have enough revision time between exams. An example of another soft constraint which may conflict with this is to schedule all the large exams as early as possible to allow enough time for marking. The quality of timetables is usually measured by checking to what extend the soft constraints are violated in the solutions generated.

One of timetabling problem form is consists of a set of lectures or classes *E* to be scheduled in *N* timeslots (*d* days of *h* hours), a set of rooms *R* in which lectures can take place, a set of students *S* who attend the lectures, and a set of constraints *C* satisfied by rooms and required by lectures [6]. Each student attends a number of lectures and each room has a size. A feasible timetable is one in which all lectures have been assigned a timeslot and a room so that the following hard constraints are satisfied:

1) No student attends more than one lecture at the same time;
2) The room is big enough for all the attending students and satisfies all the constraints required by the lecture;
3) Only one lecture is in each room at any timeslot.
4) In addition, a candidate timetable is penalized equally for each occurrence of the following soft-constraint violations:
5) A student has a class in the last slot of a day;
6) A student has more than two classes in a row;
7) A student has a single class on a day.

Note that the soft constraints have been chosen to be representative of three different classes: the *first* one can be checked with no knowledge of the rest of the timetable; the *second* one can be checked while building a solution, taking

into account the lectures assigned to nearby timeslots; and *finally* the last one can be checked only when the timetable is complete, and all lectures have been assigned a timeslot. The objective of the problem is to minimize the number of soft constraint violations in a feasible solution. All infeasible solutions are considered worthless. In this paper, we introduce a novel technique to solve timetabling problem by making use of Stochastic Context-Free Grammar in conjunction with Influence Map for satisfying soft and hard constraints of the timetabling problem. Furthermore the performance of the approach is compared with the existing one Timetables system like aSc Timetables software.

The rest of this paper is organized as follows: Section II presents the review of literature. Section III presents the proposed approach. Experimental results are presented in section IV. Section V provides conclusion and future work.

II. Review of Literature

In timetabling literature there are many approaches that have been appeared to solve this problem; such as Graph Coloring, Tabu Search, Genetic Algorithm, Artificial Immune Systems and Simulated Annealing Algorithms. Welsh and Powell (1967) represented a very important contribution to the timetabling literature by building the bridge between graph coloring and timetabling, which led to a significant amount of later research on graph heuristics in timetabling [7].

Brailsford, Potts and Smith (1999) introduced various searching methods on constraint satisfaction problems and demonstrated that this technique could be applied to optimization problems [8]. Di Gaspero and Schaerf (2001) carried out a valuable investigation on a family of Tabu Search based techniques whose neighborhoods concerned those which contributed to the violations of hard or soft constraints [9]. Also, White and Xie (2001) developed a four-stage Tabu Search called OTTABU, where solutions were gradually improved by considering more constraints at each stage, for the exam timetabling problem at the University of Ottawa [9]. Abramson (1991) applies simulated annealing to school timetabling [11]. Duong and Lam (2004) employed Simulated Annealing on the initial solutions generated by constraint programming for the exam timetabling problem at HMCM University of Technology [12]. S. Abdullah in (2007) developed a large neighborhood search based on the methodology of improvement graph construction originally developed by Ahuja and Orlin for different optimization problems [13]. Genetic algorithms have been the most studied Evolutionary Algorithms in terms of timetabling research [14].

In particular, hybridizations of genetic algorithms with local search methods (sometimes called memetic algorithms) have led to some success in the field [15]. Also Ulker, Ozcan and Korkmaz (2007) developed a Genetic Algorithm where Linear Linkage Encoding was used as the representation method, different crossover operators were investigated in conjunction with this representation on benchmark graph coloring and exam timetabling problems with hard constraints [16].

Malim, Khader and Mustafa (2006) studied three variants of Artificial Immune systems (a Clonal Selection Algorithm, an Immune Network Algorithms and a Negative Selection Algorithm) and showed that the algorithms can be tailored for both course and exam timetabling problems [17]. Recently, R. Sutar and S. Bichkar (2012) proposed a system uses Genetic Algorithm to design a model for scheduling with challenging constraints considerations [18].

III. The Proposed Approach

In order to make feasible timetable, the aforementioned constraints should be applied by the system which is a heuristic searching technique that use Stochastic Context-Free Grammar (SCFG) parser that produces the entire schedule and selects the correct time and place slot by probabilistic influence mapping. In order to delve in the details we should take the description of the two components of our proposed approach that are the Stochastic Context-Free Grammar and the Influence Mapping.

A. Stochastic Context-Free Grammar (SCFG)

The first component of the system is constructing the schedule using SCFG rules. In order to understand the concept Stochastic Context-Free Grammar, let's describe the concept of Context-Free Grammar. Context-free grammar (CFG) is a collection of context-free phrase structure rules. Each such rule names a constituent type and specifies a possible expansion thereof [19]. These languages are described recursively in terms of each other and primitive Symbols called terminals. The rules relating the variables are called productions. The term context-free comes from the feature that all productions must have a single symbol on its left-hand side, which means that the symbol could always be replaced by the right-hand side of the rule, no matter in what context it occurs.

In general the CFG consists of the following components G = (*S*, *N*, *T*, *R*):

1) A start symbol (S), which is a special non-terminal symbol that appears in the initial sequence generated by the grammar.

2) A set of non-terminal symbols (N), which are placeholders for patterns of terminal symbols that can be generated by the non-terminal symbols.

3) A set of terminal symbols (T), which are the set of rooms, course and time slots generated by the grammar.

4) A set of productions (R), which are rules for replacing (or rewriting) non-terminal symbols (on the left side of the production) in a sequence with other non-terminal or terminal symbols (on the right side of the production).

A Stochastic context-free grammar (SCFG) is a CFG plus a probability distribution on productions: G = (*S*, *N*, *T*, *R*, *P*). The probability of each rule is a conditional probability of choosing a particular right-hand-side to rewrite a given left-hand-side. The Stochastic Context-Free Grammar Parser used here consists of two steps; the *first step* is to prepare the input structure for the actual parser as shown in Fig. 1.

S	<SHEDULE>
N	{ <SHEDULE>, <SLOT>, < LECTURE> , <ROOM, <TIMESLOT >}
T	{ $room_1$-$room_n$, $group_1$- $group_n$, $professor_1$- $professor_n$, $course_1$ – $course_n$,$timeslot_1$- $timeslot_n$}
Rs_{ch1}	< SCHEDULE > → < SLOT>
Rs_{ch2}	< SCHEDULE > → < SLOT> < SCHEDULE >
Rs_{lot}	< SLOT> → {<LECTURE><ROOM><TIMESLOT>}
Rl_{ec}	<LECTURE> → {<COURSE><PROFESSOR><GROUP>}
Rc_1	< COURSE > → $course_1$
Rc_n	< COURSE > → $course_n$
Rp_1	< PROFESSOR > → $professor_1$
Rp_n	< PROFESSOR > → $professor_n$
Rg_1	<GROUP> → $group_1$
Rg_n	<GROUP> → $group_n$
Rr_1	<ROOM> → $room_1$
Rr_n	<ROOM> → $room_n$
Rt_1	<TIMESLOT> → $timeslot_1$
Rt_n	<TIMESLOT> → $timeslot_n$

Fig. 1. The input structure for the Stochastic Context-Free Grammar Parser. The structure is composed of terminals, nonterminals and rules for a timetable.

As Fig. 1 shows, the input structure for the main algorithm composed of a set of terminals and non-terminals.

- The non-terminal <schedule> is taken as the start symbol of parsing.
- The rest of non-terminals are {<SLOT>, < LECTURE>, <ROOM, <TIMESLOT >}. Where the non-terminal < SLOT> represents the lecture in addition to the place and the time, the non-terminal <LECTURE> represents the elements of the lecture which are the course, the groups that take the course, the professor teaching that course. And finally <ROOM> <TIMESLOT > are self-descriptive.

After the previous structure is built, it is used as input for the next stage of the approach. The *second step* is the design of the Context-Free Grammar algorithm as shown in Fig. 2.

- In general, the algorithm applies one of the productions with the start symbol on the left hand size, replacing the start symbol with the right hand side of the production.
- The non-terminal < SHEDULE> is replaced by the rule (Rs_{ch1}) to < SLOT> or to < SLOT>and another < SHEDULE > by the rule (Rs_{ch2}).
- The non-terminal < SLOT> which represents the time slot for a lecture in the schedule is replaced by the rule (Rs_{lot}) into the sequence of non-terminals <LECTURE> <ROOM><TIMESLOT>

```
Algorithm: Context-Free Grammar
----------------------------------------
1:  let S := the start symbol.
2:  let N := a set of non-terminals.
3:  let T := a set of terminals.
5:  let Schedule[ ] := {}empty array of all lectures
6:  let Lecture[ ]={} empty slot
7:  Input: P: productionType
8:  Output: Schedule
9:  let Rules:: = a set of rules that contain P.
10: If TypeOf (P) is Course
11:       let Rules [ ] ← InfluenceMap( )
12: If TypeOf (P) is Professor|| Group|| Room || TimeSlot
13:     let rand:= RandomNumber()%SizeOf(Rules)
14:       let Rules [ ] ← Rules [rand]
15: For each rule in Rules
16:   If rule is nonTerminal
17:       For Each productionType in rule
18:    CFG(productionType)
19:       End for
20:   Else
21:       let Schedule [ ] ←rule.productionType
22: End for
23: Return Schedule
```

Fig. 2. The recursive SCFG parsing algorithm

- The non-terminal <LECTURE> is re-written with the sequence {<COURSE> <PROFESSOR> <GROUP>}; each one of the non-terminals, in the sequence, is re-written to its terminal by selecting its production rule with the higher probability given to that rule to be chosen.
- Lines [1:6] define the storage for the start symbol from which the beginning of the sequence, array of non-terminals , another one for terminals and two empty arrays on for the output schedule and another one for temporary storage for lecture components.
- Line [7] the input for the parser is a productionType which can be any elements of the problem (e.g. professor**,** group, room, timeslot or even a rule that maps non-terminal with another non-terminal or with another terminal).
- Line [8] the output schedule.

- Line [9] define a variable the holds the rules that the input productionType contains.
- The first non-terminal is <COURSE>, which has many rules as the total number of courses in the schedule; the algorithm selects the rule that contains the course returned from the Influence Map algorithm which returns the course with the higher priority this algorithm will be introduced later. the second non-terminal is <PROFESSOR> which has many rules as the total number of professors who gives the course, selected previously, then the professor rule is selected randomly .the last non-terminal is <GROUP> which has the rule that re-write this non-terminal to the group that takes the selected course. this is implemented as following:
- Lines [10:11] the if-condition checks if productionType is the 'Course' element ,if so, then invoke the InfluenceMap procedure to get the course with the highest probability in the form of course-to-availableSlot rule(s).
- Lines [12:14] the if-condition checks if the input productionType is the 'Professor' element (contains the available professors), if so, then selects one of the available professors randomly; this is applied the same for the group, room or timeslot elements.
- Lines [15:22] this is a loop where we iterates over the rules; if the rules of the current productionType are non-terminal (set of professors**,** set of groups, set of rooms or set of timeslot) then call the algorithm recursively against each productionType in that non-terminal; else if the rules are terminal (professor, group, room or timeslot) then add it to the output array.
- Finally, we end up with a sequence of group, professor, course which is the element of the lecture.
- Repeating the process of selecting non-terminal symbols in the sequence, and replacing them with the right hand side of some corresponding production.

B. Influence Maps

In the previous section we constructed the schedule from its elements (professors, groups, course and random slot) using stochastic context free grammar. But, how each production rule determines the probability by which it selects the correct course, professor, etc. in other words how each individual (lecture) is placed in its fittest time and place? For answering this question let's understand and make use of a great concept in the field of artificial intelligence in games, the Influence mapping. Influence mapping is an invaluable and proven game artificial intelligence technique for performing tactical assessment. Influence maps have been used most often in strategy games, but are also useful for many other types of games that require an aspect of tactical analysis [20].

As there is no standard implementation of Influence Map we use simple implementation consists of a grid which has values assigned to each cell based on some function which represents a spatial feature or concept. The Stochastic Influence Mapping Algorithm used in this work is a 2D grid contains the time slots in its column headers and the available days in the headers of the rows representing the time table. In general, every time the algorithm is invoked we fill the grid with zeros the lecture gets its slot (place/time) based on its probability. This means that the lecturer with the higher probability, the higher priority it takes to proceed reserving his slot. The probability value for each lecturer is assigned based on the space available for him, and the less space available to a lecturer increases its probability/priority, and vice versa. Every time the map is constructed we reserve a slot (place/time) for the lecture with the highest probability. The Influence-Mapping algorithm is shown in Fig. 3.

Algorithm: **Influence Map**

1: **let** Courses[] := Array of all courses.
2: **let** ReservedCourses[] := an global array of reserved lectures.
3: **let** tempMaxProb **:= 0**
4: **let** crsWithMaxProb := empty
5: **Input:** Courses
6: **Output:** Lecture with valid room, day and time.
7: **for each** course **in** Courses
8: **if** course **is not in** ReservedCourses
9: **let** nRooms := number of rooms available for the group which the current course belongs to.
10: **let** nDays **:=** number of days available for the professor who gives the course.
11: **let** nTimeSlots **:=** number of time slots per day available for the professor(s) who give(s) the course.
12: **let** nRequiredSlots **:=** number of required time slots for all courses.
13: **let** probabiliy:= nRequiredSlots / (nRooms * nDays * nTimeSlots).
14: **if** probability > tempMaxProb
15: **let** tempMaxProb:= probability
16: **let** crsWithMaxProb:= course
17: **end for**
18: **let** ReservedLecs**[]** ← crsWithMaxProb
19: **Return** crsWithMaxProb.

Fig. 3. The Influence Mapping Algorithm

We can illustrate the work of the Influence Map algorithm as follows:

- Lines [1:4] the array "Courses" represents the array that holds all courses in the problem, the global array

"ReservedCourses" which keeps track of the lectures that has been already selected before, the tempMaxProb and crsWithMaxProb are variables to hold the temporary maximum probability and the course with this maximum.

- Line [5] the input array of all courses.
- Line [6] the output course with highest probability.
- Lines [7:17] the loop iterates over all the courses and checks if the course is not selected before then make simple computation to calculate priority of this course this priority equals the division of the required resources (timeslot and room) by the available resources (timeslot of the available professor and available room)
- Lines [18:19] add the result course to list of the reserved courses and returns.
- Note: The returned type is a productionType this is an abstract type of all the elements in our implementation of the context-free grammar parser.

Usually if we design an algorithm, the best way to present its efficiency is to compute its time complexity. Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of time to perform. Where each time the algorithm is invoked on the same set of courses will iterate over the courses, subtracting the courses that have been already parsed before, that is given by the formula $n*(n+1)/2$; Thus, the time complexity of the Stochastic Influence Mapping algorithm presented here is $O(n^2)$.

C. The Proposed Approach and its Advantages

The solution proposed here to the timetabling problem is a heuristic search method that might not always find the best solution but it is guaranteed to find a good solution in reasonable time. The Stochastic Context Free Grammar Parser takes the structure of the timetable in a suitable format and generates a formal representation of one lecture; each lecture contains professor, course, group, time and room with the constraints imposed by the user. The parser runs as many times as the number of courses in the schedule producing a set of lectures, and each production rules are rewritten based on a specific probability produced by probabilistic Influence Map .

The algorithm should use influence map for lecturers to prioritize the lectures to take the precedence in the assigning slots operation. Sometimes the conflicts occur when all the appropriate slots for a low-probability lecture are reserved for high-probability lecture, to handle such cases; another influence map for slots is constructed. After the parser invokes the influence map for the lecturer to get the lecturer with the highest probability in order to bind his lectures to appropriate and available slots and here comes the role of the influence map for the slots but on the contrary of the lecturer influence map, the slot influence map returned the slot with low-probability. In order to make things simpler we use influence map only for lecturers and whenever the conflicts occur the algorithm re-assign different slot.

One of the advantages of this approach lies in the using of the Context-Free Grammar to construct the time table; using this implies strongly-typed contents to the behavior of the algorithm. Strongly-Typed behavior means specifying one or more restrictions on how operations involving values of different data types can be intermixed. Also, the algorithm gives various, on the fly and feasible schedule each time the algorithm runs. As each time the influence map picks a free slot from the available slots for underlying lecture it does so randomly; thus, every time it assigns different available slot to the lecture.

D. Case Study

This case study illustrates the work of the new approach. Suppose there is a schedule that is composed of four courses as shown in Fig. 4.

SCHEDULE	→	SLOT
SLOT	→	{ LECTURE, ROOM, TIMESLOT }
LECTURE	→	{ GROUP,PROFESSOR,COURSE }
GROUP	→	Diploma
PROFESSOR	→	David | John
COURSE	→	Math | Physics | Calculus | C.Science
ROOM	→	R19
TIMESLOT	→	Sun (8:11)……..Thu(17:20)

Fig. 4. Example of a schedule composed of four courses.

We can apply the re-writing rules of the SCFG timetabling as shown in Fig. 5, where the output is four lectures.

S1	{{ Diploma, Alan, Calculus}, {R19}, Wed (15:18)}
S2	{{ Diploma, John, Physics }, { R19}, Wed (15:18)}
S3	{{ Diploma, David, Eng. Fund. }, { R19}, Sat (8:11)}
S4	{{ Diploma, Alan, Math }, {R19}, Mon (11:14)}

Fig. 5. The output of the re-writing rules is four lectures.

To determine which lecture is chosen the Influence mapping algorithm is applied. The Influence mapping algorithm is giving the precedence to the lecture S2 to reserve the slot R19-Wed (15:18), based on its superior probabilistic value which is indicated by red circle as shown in Fig. 6.

S2	8:11	11:14	15:18
Sun.	0.05555	0	0
Mon.	0	0.05555	0
Wed.	0	0	0.166

Fig. 6. The Influence mapping algorithm is giving the precedence to the lecture S2 based on its superior probabilistic value.

IV. Experimental Results

The performance results for the proposed system are presented by designing and implementing a desktop application program using Java programming language. We called the program "Automated Timetabling Solver (ATTSolver)" and the output is produced in PDF file format.

The implementation of the system is tested with the following specifications:

- Operating system windows 7 ultimate with system type 32-bit
- Hardware specification processor 2 GHz and memory 1.5 GB.
- To evaluate the results and the performance of the ATTSolver system, we compare it with the aSc Timetables software system. aSc timetable generator uses novel in-house developed algorithm. It is loosely based on backtracking with plenty of heuristics and special data structure optimized for maximum performance [21].
- The experiment that is used in the comparison is a timetable consists of one class group with 67 courses and 63 professors and one or more time-constraints on each professor. Fig. 7, 8, and 9 show some of the snapshots screen of the graphical user interface of the ATTSolver system.

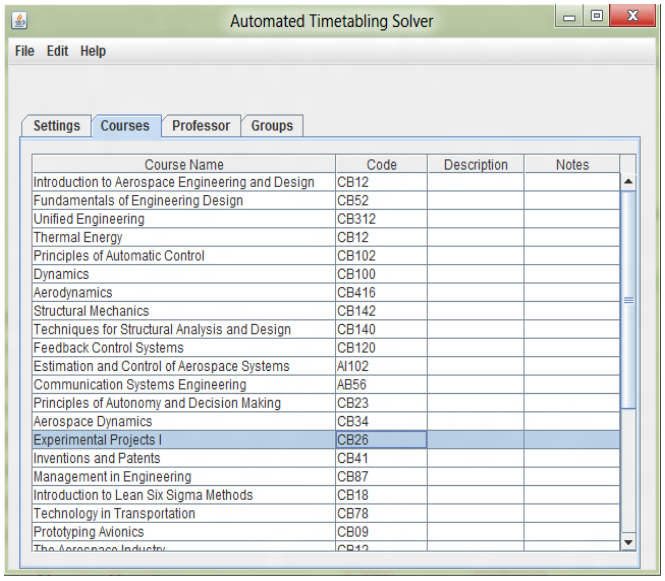

Fig. 7. The form used in the data entry for the information about the courses.

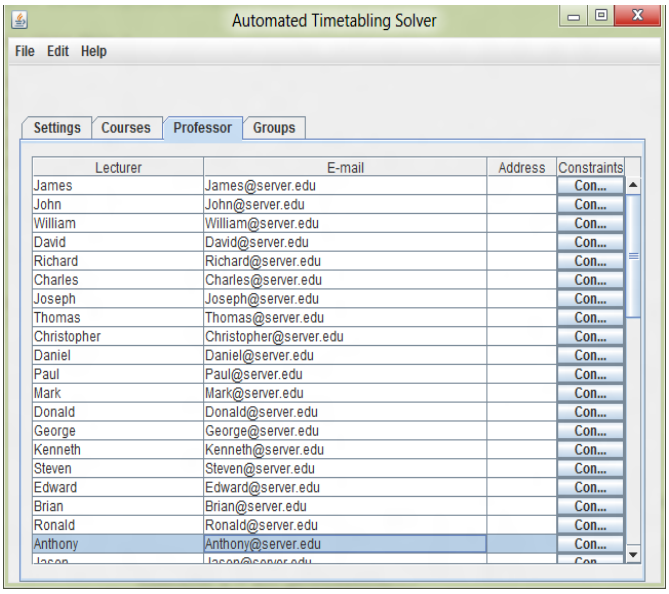

Fig. 8. The form used in the data entry for the information about the professors and their constraints.

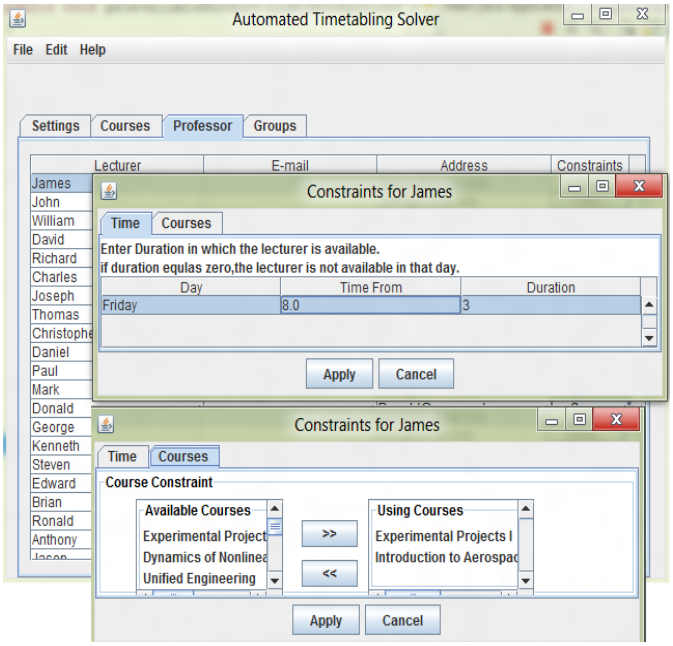

Fig. 9. The dialogs which are responsible for setting constraint for both time and courses assigned for specific professor.

The output of the ATTSolver system is exported in PDF file format as shown in Fig. 10.

	8.0 AM	9.0 AM	10.0 AM	11.0 AM	12.0 PM	1.0 PM	2.0 PM	3.0 PM	4.0 PM	5.0 PM	6.0 PM	7.0 PM
Saturday	DOROTHY Type Theory Room12	LISA Compiler Design Room12	NANCY Programming Languages Room12	KAREN Computer Vision Room12	BRENDA Computer Security Room12	SARAH Robotics Room12	JESSICA Microarchitecture Room12	DEBORAH Digital Logic Room12	SANDRA Pattern Recognition Room12	CYNTHIA Operating Systems Room12	MARY Automata Theory Room12	MARIA Algorithms Room12
Sunday	John Dyanmics of Nonlinear Systems Room12	Charles Compressible Flow Room12	William Inventions and Patents Room12	Smith Principle of Optimal Control Room12	George Aircraft Stability and Control Room12	Daniel Aerodynamics Room12	Brown Computational Methods in Aerospace Engineering Room12	Miller Unified Engineering Room12	Miller Plates and Shells Room12	David Fundamentals of Engineering Design Room12		
Monday	Jones Space System Engineering Room12	Joseph Estimation and Control of Aerospace Systems Room12	Edward Aerospace Dynamics Room12	Paul Technology in Transportation Room12	Mark Structural Mechanics Room12	Paul Thermal Eneregy Room12	Ritchard Prototypin Avionics Room12	Steven Principle of Automatic Control Room12	Thomas Stochastic Estimation and Control Room12	Anthony Dynamic Systems and Control Room12		
Tuesday	Kennet Principles of Autonomy and Decesion Making Room12	James Introduction to Aerospace Engineering and Design Room12	Davis Aerodynamics of Viscous Fluids Room12	Christopher The Aerospace Industry Room12	Kennet Communication Systems Engineering Room12	Jeff Feedback Control Systems Room12	Johnson Techniques for Structural Analysis and Design Room12	James Experimental Projects Room12	Brian Dynamics Room12	Donald Management in Engineering Room12	Jason Computational Mechanics of Matreials Room12	Ronald Introduction to Lean Six Sigma Methods Room12
Wednesday	CAROL Data Mining Room12	PATRICIA Computability Theory Room12	HELEN Image Processing Room12	LAURA Natural Language Processing Room12	MELISSA Computer Networks Room12	SHARON Information retrieval Room12	VIRGINIA Bioinformatics Room12	ANGELA Databases Room12	AMY Ubiquitous Computing Room12	MICHELE Knowledge Representation Room12	JENNIFER Analysis of Algorithms Room12	
Thursday	ELIZABETH Quantum Computing Theory Room12	LINDA Computational ComplexityTheory Room12	RUTH Evolutionary Computation Room12	ANNA Systems Architecture Room12	REBECCA Numerical Analysis Room12	KIMBERLY Medical Image Processing Room12	BETTY Machine Learning Room12	MARGARET Computational Geometry Room12	SHIRLEY Mutiprocessing Room12	DONNA Cognitive Science Room12	SUSAN Data Structures Room12	BARBARA Cryptography Room12
Friday												

Fig. 10. The output table of the ATTSolver system in pdf file format

The experiments are performed to compare the performance of both aSc software system and ATTSolver system for the number of tested schedules, until the fittest solution in each run is satisfied, and the execution time through five sequential runs. The ATTSolver gives best results where the number of tested schedules of the ATTSolver is fewer in compared to the number of tested schedules of the aSc software as shown in Fig. 11. In addition, the execution time of the ATTSolver system is much better than that of the aSc in all cases of the sequential runs. The reason of these results returns to the using of Stochastic Context-Free Grammar rules to build schedule and make use of influence maps to assign the fittest slot (place & time) for each lecture in the timetable.

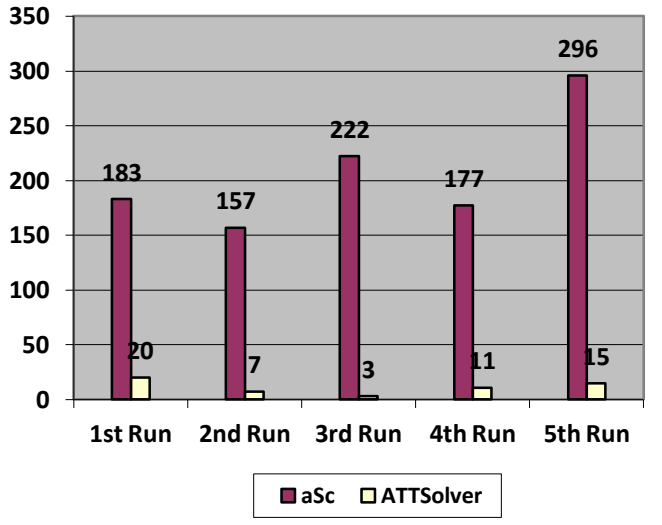

Fig. 11. The number of tested schedules of aSc and ATTSolver

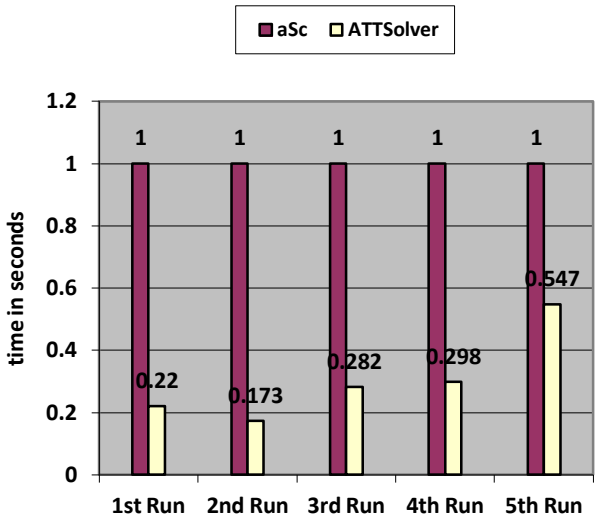

Fig. 12. Execution time of aSc and ATTSolver systems

V. Conclusion and Future Work

This paper presented a solution of the problem of finding suitable class schedule by using strongly-typed heuristic search technique. The system uses Stochastic Context-Free Grammar rules to build schedule and make use of influence maps to assign the fittest slot (place and time) for each lecture in the timetable. The system is taking the soft constraints as well as the hard constraint into the consideration based on its priority making various, instant, and suitable timetable. The results of comparing ATTSolver and aSc systems reveal that the number of tested schedules in ATTSolver system is always less than that in aSc system. Moreover, the execution time for ATTSolver system is much better than that of aSc system in all cases of the sequential runs. In future work we intend to apply the ATTSolver system in the field of software engineering as an approach of refactoring the code and apply the fittest design pattern.

References

[1] A. Schaerf, "A survey of automated timetabling," Artificial Intelligence Review, Kluwer Academic Publishers, vol. 13, pp. 87-127, 1999.

[2] D. Montana, "Strongly typed genetic programming," ACM, vol.3, Issue 2, 1995.

[3] M. Carter and D. Johnson, "Extended clique initialization in examination timetabling," Journal of Operational Research Society, vol. 52, pp. 538-544, 2001.

[4] L. Reis and E. Oliveira, "A language for specifying complete timetabling problems, *Selected Papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling, 2001*.

[5] E. Burke, and S. Petrovic, "Recent research directions in automated timetabling," European Journal of Operational Research, vol. 140, pp. 266–280, 2002..

[6] O. Rossi-Doria, et al. , "A comparison of the performance of different metaheuristics on the timetabling problem," *In Proc. 4th International Conference on the Practice and Theory of Automated Timetabling IV, PATAT02, 2002.*

[7] A. Welsh, and B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," The Computer Journal, vol. 10, issue 1, pp. 85–86, 1967.

[8] S. Brailsford, C. Potts and B. Smith, "Constraint satisfaction problems: algorithms and applications," ELSEVIER; European Journal of Operational Research,vol. 119, pp. 557-581, 1999.

[9] D. Gaspero, and S. Tabu, "Search techniques for examination timetabling," *Selected Papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling, 2000*.

[10] G. White, B Xie, and S. Zonjic, "Using tabu search with longer-term memory and relaxation to create examination timetables," ELSEVIER, European Journal of Operational Research, vol. 153, Issuo 1 pp. 80-91, Feb. 2004.

[11] D. Abramson, "Constructing school timetables using simulated annealing: sequential and parallel algorithms," Management Science, vol. 37, No. 1, pp. 98-113, Jan. 1991.

[12] T. Duong and K. Lam, "Combining constraint programming and simulated annealing on university exam timetabling," *In Proceedings of the 2nd International Conference in Computer Sciences, Research, Innovation & Vision for the Future (RIVF2004), Hanoi, Vietnam, February 2-5, 2004*.

[13] S. Abdullah, "Heuristic Approaches for university timetabling problems," PhD Thesis, School of Computer Science and Information Technology, University of Nottingham, UK , July 2006.

[14] A. Colorni, M. Dorigo, and V. Maniezzo, "A genetic algorithm to solve the timetable problem," *Technical Report. 90-060 revised, Politecnico di Milano, Italy, 1992.*

[15] Burke and H. Rudova, "Practice and theory of automated timetabling," *Selected Papers from the 6th International Conference. Lecture Notes in Computer Science, vol. 3867, 2007*.

[16] B. Bilgin, E. Ozcan and E. Korkmaz, "An experimental study on hyper-heuristics and exam timetabling," *Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling, 2006.*

[17] M. Malim, A Khader, A Mustafa, "Artificial immune algorithms for university timetabling," *Proceedings of the 6th International Conference on the Practice & Theory of Automated Timetabling, September, 2006.*

[18] S. Sutar and R. Bichkar, "University timetabling based on hard constraints using genetic algorithm," International Journal of Computer Applications, Vol 42, No.15, March 2012.

[19] C. Antunes and A. Oliveira, "Using context-free grammars to constrain apriori-based algorithms for mining temporal association rules," *Workshop Proceedings on Temporal Data Mining*, 2002.

[20] P. Tozour, "Influence mapping game programming Gemes 2," Chalres River Media, 2001.

[21] http://help.asctimetables.com/index.php