Solving the Timetabling problem at FPT University

Luu Thanh Cong
Nguyen Van Hoai
Nguyen Son Ha
Hoang Minh Duc

A thesis submitted in part fulfilment of the degree of BSc. (Hons.)
in Computer Science with the supervision of
Mr. Pham Quang Dung & Mr. Nguyen Tat Trung

I confirm that the work submitted is my own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

Hoa Lac campus - FPT University
27 August 2015

Table of Contents

List of Figures

List of Tables

Abstract

University course timetabling problems are diversity combinatorial optimization class problems. Each specific case has different approaches and different solutions. In this project, we present a new approach for solving a real world university timetabling problem in FPT University. We focus on producing a service that can automatically generate a better quality timetable within acceptable time. Our web service also provides a simple Academic Portal that can handle students, teachers, courses, rooms, semesters, manual schedules, auto-checking conflict ... We conduct experiments on real dataset provided by the Academic Department which show the feasibility of the proposed approach.

Acknowledgements

In the beginning we would like to express our deep gratitude to Mr. Pham Quang Dung and Mr. Nguyen Tat Trung, our research supervisors, for their patient guidance, enthusiastic encouragement and useful critiques of this research work. I would also like to thank Mr. Tran The Trung and Ms. Le Thi Thanh Nga, for their help with administrative formalities and assistance in keeping our progress on schedule.

Our grateful thanks are also extended to Ms. Tran Thi Phuong Trang, Ms. Tran Thi Phuong Dung and FPT Academic Department for helping us in providing the real data, doing the meteorological data analysis and comparing the quality of our result.

Finally, we would like to thank our family, our friends, who supported us to complete this project.

Hoa Lac campus - FPT University

15 August 2015

Project team

Chapter 1: Introduction

In Spring 2015, many students and teachers in FPT University complain that the timetable has problems. They do not have enough teachers for some slots, some teachers have to go to the campus for only one slot on some days, some student have to study in first slot and last slot of a day. Traditionally, the timetable is scheduled manually. This task is stressful: it takes much times and cannot avoid mistakes for a typical semester with 80 classes, 300 class-courses.

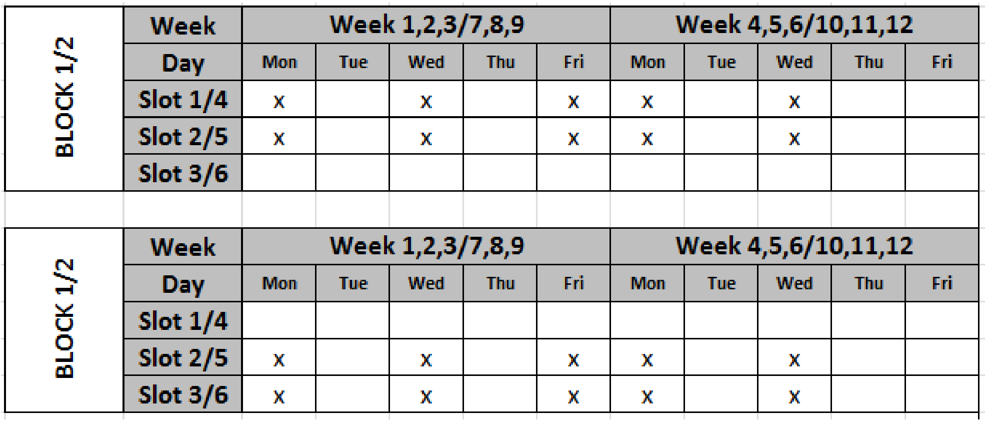
University course scheduling is one of the hard problems in combinatorial optimization. There are many variants of the problem depending on specific requirements and policies of educational institutes [2]-[9] (see [1] for extensive references). Essentially, timetabling consists of assigning a set of courses to specified slots and rooms without conflict. In [5], additional complexities of course structure and propose approach for modelling and solving the problem at hand. That paper also considered the distance between rooms, and the course scheduling takes into account the travel distance of students and teachers between consecutive classes. In [4], the author considered the situation that additional requirements may arrive after publishing the timetable and proposed an iterative forward search for re-computing the timetable minimizing perturbation. In 2013, Muller considered a particular examination timetabling problem which was common at many American universities. In that problem, one has to assign a set of examinations to different rooms at different periods satisfying given constraints. A large examination can be split into a number of rooms (up to 4 rooms). Many hard, soft constraints were considered. For example, some distribution constraints of the problem are **same room/different room** (2 exams are required/expected to be in the same/different room), **different period/same period** (2 exams are required/expected to be at different/same periods), **precedence** (one exam are required/expected to be placed in the period before another exam). In [34], the timetabling for high-school has been considered. In this context, there *m* classes $\{c_1 \ldots c_m\}$, *n* teachers $\{t_1, \ldots t_n\}$, and p periods $\{1 \ldots p\}$. A requirement matrix *r* is given in which $r_{i,j}$ indicates the number of lecturers that teacher t_j must give to class c_i. Two other matrices representing the unavailability of of teachers, classes in each period. The goal is to assign $x_{i,j,k} = 1$ if the teacher t_i is assigned to class c_j in period k, $\forall i \in \{1 \ldots n\}$, $j \in \{1 \ldots m\}$, $k \in \{1 \ldots p\}$, and $x_{i,j,k} = 0$, otherwise. The assignment must satisfy the constraints of the requirements, and the availabilities of teachers, classes and optimize an objective function which is a combination of multiple components such as idles periods between two lectures in teaching assignment of a day, moving from one site to another site between two consecutive periods, etc.

Most of the algorithms have been investigated to solve timetabling problems are heuristics and meta-heuristics, for example, memetic, genetic algorithms [35, 36, 37], Tabu search [38, 34], simulated annealing [39]. Meyers et al. in 2007 [40] proposed a very large-scale neighbourhood search algorithm for solving an exam timetabling problem which consists of scheduling a set of exams $E = \{e_1 \ldots e_n\}$ over a collection of time periods $P = \{p_1, \ldots, p_m\}$. In this paper, the problem was viewed as a partitioning problem: partition the exams of E into a number of sets; each set corresponds to a time period. The authors investigate the cycle exchange neighbourhood for this problem. In this framework, a neighbouring solution of a current solution corresponds to a cycle e_{i1}, $e_{i2}, \ldots e_{ik}$. e_{i1} in which we move e_{i1} from its current set to the set of e_{i2} , then, move e_{i2} from its current set of the set of e_{i3} ,..., and finally, move e_{ik} to the set of e_{i1} . The size of the underline neighbourhood is exponential of the size of the problem. An efficient heuristics has been applied to find a negative cost cycle on the improvement graph, thus to find quickly an improving neighbouring solution.

In the case of FPT University, the Academic Department will make a timetable based on the information from last semester including list of students, their progress statuses and the prospective information from next semester containing list of prospective fulltime teachers, list of prospective rooms and prospective semester template - how class-courses of a class are organised in weeks of a semester.

Chapter 2: Problem description

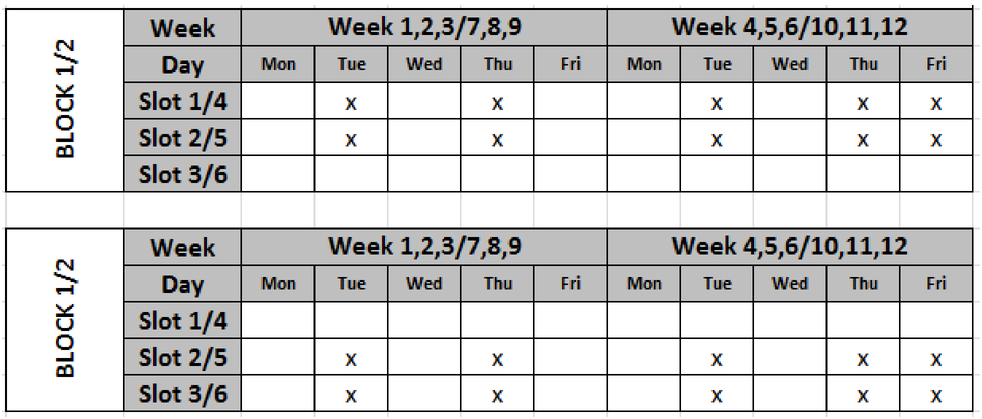
It is very difficult to formulate a general model that is applicable for all cases since every university or other educational institutions have their own special constraints and objectives. In this project, from the real-world specific requirements from FPT, we developed a system that assists FPT Academic department in making a timetable for each academic semester. In this context, each academic semester consists of 12 weeks and is divided into 2 blocks (each block has 6 weeks). There are a set of classes, each of which has at most 5 courses to be scheduled in the semester. Each day is divided into 6 slots of two sessions: slots 1, 2, 3 are in the morning, and slots 4, 5, 6, are in the afternoon (each slots lasts 1.5 hours). Each course has 30 slots. Due to the fact that several classes may have the same course to follow, we call **class-course** the course assigned to a given class. For instance, two classes IS10801 and ES10801 have the same course ITE302 to follow, we denote class-course IS10801ITE302 the course ITE302 assigned to class IS10801, and class-course ES10801ITE302 the course ITE302 assigned to class ES10801. Another situation that may happen: two class-courses i and j of the same course belong to two different classes ci and cj, but the number of students of cj attending j is too small. In this situation, all students attending j will participate in i and the class-course j is removed to save the resource. Semester **template** is how class-courses of a class are allocated in weeks of a semester. Due to specific requirements and the policy of the university, the staff define templates to allocate a class-course as illustrated in Figures 1-3. For instance, in the Figure 1, the class-course is scheduled on Monday, Wednesday, Friday of the first 3 weeks and on Monday, Wednesday of the last 3 weeks of block 1 or 2 (in slots 1, 2 in the morning or slots 4, 5 in the afternoon). Figure 4 is an example of timetable for one class with 5 class-courses M1, M2, M3, M4, M5. With these predefined templates, each class-course can be scheduled into 1 of 12 places (or possibilities)



BLOCK 1/2	Week	Week 1,2,3/7,8,9					Week 4,5,6/10,11,12				
	Day	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
	Slot 1/4	x		x		x	x		x		
	Slot 2/5	x		x		x	x		x		
	Slot 3/6										

BLOCK 1/2	Week	Week 1,2,3/7,8,9					Week 4,5,6/10,11,12				
	Day	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
	Slot 1/4										
	Slot 2/5	x		x		x	x		x		
	Slot 3/6	x		x		x	x		x		

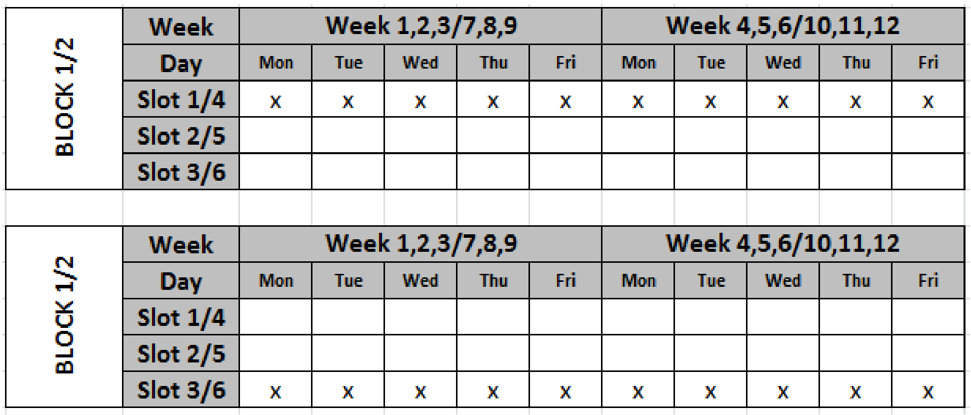
Figure 1: Example of class-course allocation 1.



BLOCK 1/2	Week	Week 1,2,3/7,8,9					Week 4,5,6/10,11,12				
	Day	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
	Slot 1/4		x		x			x		x	x
	Slot 2/5		x		x			x		x	x
	Slot 3/6										

BLOCK 1/2	Week	Week 1,2,3/7,8,9					Week 4,5,6/10,11,12				
	Day	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
	Slot 1/4										
	Slot 2/5		x		x			x		x	x
	Slot 3/6		x		x			x		x	x

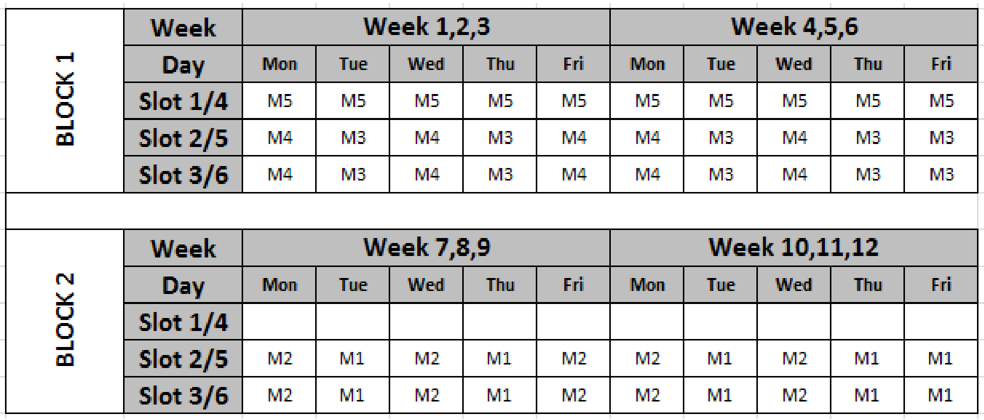
Figure 2: Example of class-course allocation 2.



BLOCK 1/2	Week	Week 1,2,3/7,8,9					Week 4,5,6/10,11,12				
	Day	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
	Slot 1/4	x	x	x	x	x	x	x	x	x	x
	Slot 2/5										
	Slot 3/6										

BLOCK 1/2	Week	Week 1,2,3/7,8,9					Week 4,5,6/10,11,12				
	Day	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
	Slot 1/4										
	Slot 2/5										
	Slot 3/6	x	x	x	x	x	x	x	x	x	x

Figure 3: Example of class-course allocation 3.



BLOCK 1	Week	Week 1,2,3					Week 4,5,6				
	Day	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
	Slot 1/4	M5	M5	M5	M5	M5	M5	M5	M5	M5	M5
	Slot 2/5	M4	M3	M4	M3	M4	M4	M3	M4	M3	M3
	Slot 3/6	M4	M3	M4	M3	M4	M4	M3	M4	M3	M3

BLOCK 2	Week	Week 7,8,9					Week 10,11,12				
	Day	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
	Slot 1/4										
	Slot 2/5	M2	M1	M2	M1	M2	M2	M1	M2	M1	M1
	Slot 3/6	M2	M1	M2	M1	M2	M2	M1	M2	M1	M1

Figure 4: Example of timetable of one class with 5 class-courses.

2.1 Problem formulation

We describe in this section the formulation of the problem. There are 12 weeks in total divided into 2 blocks, each of which has 6 weeks (1, 2, 3, 4, 5, 6). The timetable is repeated in 3 weeks (for example, 1, 2, 3 and 4, 5, 6). We can compute the schedule for 4 typical weeks: week 1, 4 of block 1 and weeks 1, 4 of block 2. Each week has 5 official days, each of which has 3 slots (1, 2, 3) in the morning and 3 slots (4, 5, 6) in the afternoon. Each pair day-slot is called a cell. In total, there are 120 cells of 4 typical weeks of the schedule. We denote *CELL* = {1, . . . , 120} the set of cells. The objective is to schedule a set of class-course into places satisfying a given constraints. The timetable obtained will then be submitted to the direction board where they decide to assign teachers to class-courses. The essential constraints are:

- No two class-courses of a class share the same cell because they share the same students of the class.
- Two class-courses of a class must be scheduled in the same session (morning or afternoon)

For each course, we try to schedule the class-courses corresponding to this course separately so that the teacher who is responsible for this course can teach as many as possible these class-courses. In other words, we try to minimize the number of class-courses corresponding to the course sharing a same cell. For example, if a course *M* has two class-courses *CM1* and *CM2* which are scheduled in the way that they share the same cell, then we need two different teachers for attending these two class-courses. Otherwise, if these two class-courses are scheduled in two places that share no cell, then only one teacher responsible for the course *M* can attend these two class-courses for saving resources.

2.1.1 Input

- A set of courses $C = \{1, \dots, M\}$
- A set of classes $CL = \{1, \dots, K\}$
- A set of class-courses $CC = \{1, \dots, N\}$, for each class-course $i \in CC$,
    - $c(i) \in C$ is the course of the class-course i
    - $cl(i) \in CL$ is the class following the class-course i
    - $CC(k) \subset CC$ is set of class-courses of class k.
    - $CCL(j) \subset CC$ is set of class-courses of course j, $j \in C$.
- For each pair of class-courses i and j, $diff(i, j)$ = **true** indicates that class-course i and j cannot be scheduled in the same place. For example, i and j belong to the same class or i and j belong to different classes $c(i)$ and $c(j)$ but there are students of $c(j)$ participating in class-course i (or vice versa) as mentioned above. In the second case, i and j cannot be scheduled in the same place because they have common students.
- For each class $k \in CL$, $crs(k) = \{j \in C \mid \exists i \in CC\ c(i) = j\}$: set of courses the class k has to follow in the semester
- For each class $k \in CL$, we denote $CC(k)$ the set of class-courses of class k:

- A set of places $P = \{1, ..., 12\}$ where a class-course can be scheduled.
  - For each place $p \in P$, $s(p)$ is the session (morning or afternoon) of p
  - $cfp(p1, p2)$ = **true** if place $p1$ and $p2$ share a common cell.
  - $CELL(p)$ = the set of cells of the place p, $\forall p \in P$
- $w[i]$: number of student of class-course i, $i \in CC$.
- A set of teachers: $T = \{1,.., NT\}$;
- $TL(c)$: list of teachers who can instruct course c, $c \in C$, $TL(c) \in T$.
- Given a set of room: $RO = \{1,..., NR\}$.

2.1.2 Variables

- $x\_p(i)$ represents the place to which the class-course i is scheduled, the domain of $x\_p(i)$ is P, $\forall i \in CC$.
- $x\_r[i]$ presents a room that is assigned to class-course i, $i \in CC$, $x\_r[i] \in R$.
- $x\_t[cc]$: presents teacher who will instructs class-course cc. Domain of $x\_t[cc]$ is T, $cc \in CC$.
- Auxiliary variables:
  - $occ(i, j) = 1$ if class-course i appears in cell j, $\forall i \in CC$, $j \in CELL$
  - $oc(i, j)$ represents the number of occurrences of course i in cell j, $\forall i \in C$, $j \in CELL$
  - $moc(i)$ represents the maximum number of occurrences of course i in a cell, $\forall i \in C$

2.1.3 Invariants

- $d(k) = \sum_{i \in crs(k)} moc(i), \forall k \in CL$
- $cf(i,j)$ = **true** if and only if $cfp(x\_p(i), x\_p(j))$ = **true**, $\forall i,j \in CC$. $cf(i,j)$ = **true** indicates two class-courses conflict with each other.

2.1.4 Constraints

- $cl(i) = cl(j) \rightarrow s(x\_p(i)) = s(x\_p(j)) \; \forall i,j \in CC$ (1)
- $diff(i, j)$ = **true** $\rightarrow cfp(x\_p(i), x\_p(j))$ = **false**, $\forall i,j \in CC$ (2)
- $x\_p(j) = p \rightarrow occ(j, z) = 1, \forall p \in P, z \in CELL(p)$ (3)
- $oc(i, z) = \sum_{j \in CC} occ(j, z) * (c(j) = i)$ (4)
- $cf(i,j)$ = **true** $\rightarrow x\_t[i] \neq x\_t[j]$ (5)
- $cf(i,j)$ = **true** $\rightarrow x\_r[i] \neq x\_r[j]$ (6)

Constraint (1) states that two class-courses of the same class must be scheduled in the same session. Constraint (2) states that two class-course of the same class must be scheduled at different places. Constraints (3), (4), (5) are channeling constraints that relate decision variables x and auxiliary variables. Two constraints (5) and (6) state that if two class-courses conflict with each other, they must not assigned to the same teacher and the same room.

2.1.5 Objectives function

The objective function to minimized is

$$\sum_{i \in C} moc(i) \qquad (7)$$

Chapter 3: Review of Literature for solution

3.1 Approaches to Automated Timetabling

A university timetabling problem can be defined to be the problem of assigning a number of events (courses/exams) into a limited number of time periods (day/slot). Wren (1996)[10] defines timetabling as follows:

‘‘Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives.’’

The large number of courses/exams to be scheduled and a many constraints imposed on timetabling makes the solution space very large indeed. The construction of a timetable is a difficult task and it requires much effort. This filed have attracted the attention of the scientific community from a number of disciplines (including OR and Artificial Intelligence) for haft century. A wide variety of approaches have been described in the literature and tested on real data. We will present a very brief outline of timetabling research over the years to give an overview of some of the latest approaches and research directions that have been, or are being undertaken, in the Automated Scheduling. They can be roughly divided into four types (Carter, 1986[11]; Carter, Laporte and S. Lee 1996[12]): Graph-based methods, cluster-based methods, constraint-based methods, and meta-heuristic methods.

3.1.1 Graph-based Methods[1]

Graph-based methods order courses using domain heuristic and then assigning the courses sequentially into valid time periods so that no courses in the period are no conflict with each other [13]. In this method, timetabling problems are usually represented by a graph where courses are vertices, conflicts are edges(de Werra, 1985[14]). So the problem can be represented by a colouring problem. Each time period corresponds to a colour in the graph and the vertices of a graph are coloured in such a way so that no two adjacent vertices are coloured by the same colour. Many graph colouring heuristics for constructing conflict-

[1] This method has another name is Sequential methods.

free timetables is available in the literature (Brelaz, 1979 [15]; Carter and Laporte, 1996 [16]).

- Largest degree first: Vertices that have largest degree are scheduled first. That means courses that have a large number of conflicts with other course are scheduled early. The rationale is that the courses with a large number of conflicts are more difficult to schedule and so should be tackled first.
- Largest weighted degree: This is a modification of the "Largest degree first" which weights each conflict by the number of students.
- Saturation degree: In each step, which course has the smallest number of valid slot available for scheduling in the timetable constructed is selected.
- Colour degree: This heuristic priorities those course that have the largest number of conflicts with the courses that have already been scheduled.

3.1.2 Cluster-based Methods

In cluster methods, the set of courses is split into groups which satisfy all hard constraints and then the groups are assigned to time periods to fulfil the soft constraints (White and Chan, 1979)[17]. Many optimisation techniques have been implemented to solve the assignment problem (Fisher and Shier, 1983[18]). The weakness of these methods is that the cluster are formed and fixed at the beginning of the algorithm and that may result in a bad quality timetable.

3.1.3 Constraint-based Methods

Constraint-base methods are exact methods to solve timetabling problems. The problem is modelled as a set of variables with a finite domain. The method assigns values to variables that fulfil a number of constraints. A popular method Constraint Programming (CP) [19] is an approach for solving this problem that ensures to find optimal solution if it exists. It combines between a propagation process and a branching process. The propagation process uses constraints to eliminate redundant values from the domains of variables (i.e., values that are not in any feasible solutions) for reducing the search space. The branching process partitions the original problem into smaller equivalent sub-problems (the union of the sets of solutions to sub-problems is equivalent to the set of solutions to the original problem) by splitting the domain of a selected variable into smaller domain. The propagation and branching processes are performed iteratively: after each branching, the propagation process will perform on each generated sub-problem. In CP framework, the search space is completely explored. Hence the execution time is theoretically exponential of the size of the input of the problem. Many generic CP solvers were developed in the literature, for instance, CHOCO[23], GECODE [20], Minizinc [21], etc. With these solvers, users state the model (specify variables, constraints, objective functions) and call search procedures for finding solutions. Using this generic approach has an advantage that allows users to model flexibly different problems as well as to adapt the existing model when additional constraints are added.

3.1.4 Meta-heuristic Methods

Many meta-heuristic approaches such as local search (Tabu search, simulated annealing, hill-climbing, genetic algorithms) and hybrid approaches have been implemented for timetabling. Some very good results have been reported in Burke and Ross (1996)[24], Burke and Carter (1998)[25], Burke and Erben (2001)[26]. Meta-heuristic methods begin with one or more initial solutions and employ search strategies that try to avoid local optimal. All of these search algorithms can produce high quality solutions but often have a considerable computational cost. We will mention local search methods more detail in section 3.2 and hybrid methods in section 3.3

3.2 Local Search methods

Local Search (LS) [22] is an alternative approach for solving combinatorial optimization problems. This is an incomplete approach in which the search space is partially explored. It bases on general and simple ideas. The search begins with one or more initial solutions and employs search strategies that try to avoid local optimal. The LS approach cannot ensure to find an optimal solution and prove its optimality. However, in many cases, it can finds high-quality solution in reasonable time. The search starts from an initial solution generated randomly or by a greedy algorithm (depend on particular problem at hand) and iteratively moves from a current solution to one of its neighbours. The neighbourhood of a current solution is basically generated by modifying locally the components of the current solution (e.g., changing the value of a decision variable or exchanging the values of two decision variables). Constraint-Based Local Search (CBLS) [22] is a local search framework uses constraints, objective functions to drive the search in potential region where high-quality solutions likely exist. Tabu search (TS) is a local search algorithm that uses a mechanism to avoid revisiting solutions discovered so far. One of the possible mechanism in CBLS is to use a data structure to record the move information (for example, a pair of variable, value) performed.

3.2.1 Hill-climbing algorithm

Hill-climbing [27] algorithm is based on other local search technique. Although this algorithm is simple it indicates strong and effective in large constraint satisfaction problems. Hill-climbing always selects the best candidate of all neighbours which improve the objective function. If there is no better solution, the search stuck in a local optimum. It usually restarts the search from another initial candidate. The name of this algorithm means increasing the evaluation value each step by climbing.

3.2.2 Min-conflicts algorithm

Min-conflicts algorithm (MC) [28] is a local search algorithm, a heuristic method to solve constraint satisfaction problems. MC algorithm chooses the best candidate only from a subset of the neighbour candidate. A variable is involved in a conflicted constraint, then we choose another value which minimizes the violation. If no such value exists, we pick randomly one value that does not increase the violation. MC algorithm is not able to leave a local minimum. If the algorithm is trapped by local minimum, it cannot move at all, and does not terminate. So we have to limit the number of iteration step. MC Random Walk algorithm [29] is an improved version of pure MC algorithm above. Because the pure min-conflict algorithm cannot leave the local minimum, the random walk strategy picks random a value with probability P, and apply the min-conflict heuristic with probability 1-P. The same strategy can be used in Hill-climbing also.

3.2.3 Tabu search

Tabu search [30] is a local search algorithm. It find optimal solution by moving from current solution s at iteration t to the best candidate s'. Tabu search avoid the iterations that does not improve the objective function. A tabu list, which is a special short term memory store previous configurations. Because it requires a large memory space to record all status of previous configurations, so we only record some properties in tabu list, and solutions do not have these properties, will not be considered in ***l*** iterations (that is tabu length). This strategy prevents the search from being trapped in short term cycling and allows the search process to leave out the local optimal. Tabu search is improved to reactive Tabu search and Tabu search with two tabu lists.

3.2.4 Genetic algorithm

Genetic algorithm (GA)[31] is based on the idea of natural selection. This algorithm maintains a population of candidate solutions. The candidate solutions have probability to make their child candidate solution depend on their fitness. Fitness is evaluated by an objective function. GA is used to solve constraint satisfaction problems. One of the important problems is how we implement the constraint by fitness function to control searching process.

3.3 Hybrid methods

The mixing traditional systematic search approaches have led to good results on large scale problems. There is three categories of hybrid approaches can be found in the literature [32]:

- Performing local search before or after a systematic search.
- Performing a systematic search improved with local search at some point of the search.
- performing an overall local search, and using systematic search either to select a candidate neighbour or to prune the search space

Chapter 4: Solutions

By our literature review and testing on some algorithm, we consider using hybrid algorithm that is a mixture of Constraint Based Programing and Local Search. At first we tried an exact method by using CHOCO Solver (a Constraint Programing library) but it did not give an acceptable solution in a limit time because of hugeness space (objectives function is 159, worse than manual timetable, in 2 hours runtime). We decided to use this method on smaller space of each class to find out all its possible solution. It is feasible in our experiment. Next, we use a Tabu search to find out the best feasible solution that optimizes the objective function.

4.1 Algorithm

In this section, we propose an algorithm to solve the problem stated. The algorithm consists of 4 phases. But first, we have to do a pre-processing to merge class-courses. Due to the policies of FPT University, an eligible class-course must have at least 15 students and at most 30 students. We have a lot of class-course have the number of student less than 15 that means this class-course cannot be opened. Therefore, we have to merge an ineligible class-course with another class-course in the same course to make it eligible (e.g. class-course CS701IAO101 has 10 students, class-course SE702IAO101 has 14 students, we can merge these two class-courses into one class-course and the result class-course becomes eligible because they have same course IAO101 and 24 students in total. We keep a class-course CS701IAO101 or SE702IAO101 and eliminate the other). In the first phase, we generate all local-timetables for all class-courses of each class. Given a class k, a local-timetable for k is an assignment of one place $p_i \in P$ to each class-course i of $CC(k)$ such that $p_i \neq p_j$ and $s(p_i) = s(p_j), \forall i,j \in CC(k)$. The output of the first phase, we obtain a set $S(k)$ of local-timetables for each class k; $\forall k \in CL$. In the second phase, we propose an approach using Tabu search algorithm for selecting an local-timetable from $S(k)$ for each class k such that the objective function (7) is minimal. In phase 3, 4 we assign teacher and room for each class-course.

4.1.1 Pre-processing: Merging class-course

This is a Multi-Knapsack problem. We have a list of class-courses of each course m (list of items = $CCL(m)$) with number of students of each class-course i (weight of items = $w[i]$). Our task is to divide these items to a number of bags (expected to be opened class-courses, called ECC) to satisfy a condition about weight. ($15 \leq$ weight ≤ 30). ***We will do pre-processing for each course m $\in C$.***

Problem statement for each course *m*:

Input:

- A course $m \in C$.
- *CCL(m) = {1,…,n}*.
- *w[i]*: number of student of classcourse i, $i \in CCL(m)$.

Variables:

- *x_c[i]* represents class-course which class-course *i* will be join in, domain of *x_c[i]*, $i \in CCL(m)$.
- Auxiliary variables*:*
    - *b*: is number of expected class-courses of course m.
    - *min*: lower bound of *b*. $min = \lceil (\sum_{i=1}^{n} w[i])/30 \rceil$;
    - *max*: upper bound of b. *max = n*.
- *ECC = {1,...,b}* : a set of expected class-courses to be opened of course *m*.

Invariants:

- $st(j) = \sum_{i=1}^{n} w[i]$ where $x\_c[i] = j$, $j \in ECC$.
- *violations(x_c[i])*: specifies how much *x_c[i]* violates constraints, *violations(x_c[i])* is non-negative integer.
- *sumViolations(x_c)*: $\sum_{i=1}^{n} violations(x\_c[i])$, sum of violation of all *x_c[i]*, $i \in CCL(m)$.

Constraints:

- $15 \leq st(j) \leq 30$, $j \in ECC$: This constraint states that each class-course which expected to be opened must have at least 15 students and at most 30 students.

Output:

- Best global solution: *x_c*

Algorithm:

We solve merging classcourse problem using Tabu search. We denote:

- $ns[x\_c[i] \leftarrow v]$: is a new solution of *x_c* when *x_c[i]* is reassigned to a value v, $i \in CCL(m)$, $v \in ECC$.
- $sumViolations(ns[x\_c[i] \leftarrow v])$: sum of violation of all *x_c[i]* when *x_c[i]* is reassigned to a value v, $i \in CCL(m)$, $v \in ECC$.
- *(x_c[i], v):* a pair includes *x_c[i]* and a value *v*

```
foreach course $m \in C$ do
    $b \leftarrow min$;
    $ECC \leftarrow \{1, \ldots, b\}$;
    while $b \leq max$ do
        InitRandomSolution();
        FindSolutionUsingTabuSearch();
        $b \leftarrow b + 1$;
        if $sumViolations(x\_c) = 0$ then
            break;
        end
    end
end
```

Figure 5: Algorithm 1 - FindOptimalSolution()

```
Input: $ECC = \{1, ..., b\}$
Output: A solution: $x\_c$
for $i \leq n$ do
    $x\_c[i] \leftarrow random\ element\ of\ ECC$;
end
```

Figure 6: Algorithm 2 - InitRandomSolution()

```
Input: $ECC = \{1, ..., b\}$.
       $tabu$: represents the tabu list.
       $tbl$: length of the tabu list.
       $maxIter$: limit of number of iterations.

Output: Best solution: $x\_c$
1  $it \leftarrow 1$;
2  $minViolations \leftarrow MAX\_INT$;
3  while $it \leq maxIter$ do
4      //collect all potential moves;
5      $F \leftarrow \{(x\_c[i], v) \mid$
       $(tabu[i, v] < it \vee sumViolations(ns[x\_c[i] \leftarrow v]) < minViolations)$
       $\wedge\, violations(ns[x[i] \leftarrow v]\ \ is\ \ minimal\}$;
6      $(x\_c[i], v) \leftarrow random\ \ element\ \ of\ \ F$;
7      $x\_c[i] \leftarrow v$;
8      $tabu[i, v] \leftarrow it + tbl$;
9      if $sumViolations(x\_c) < minViolations$ then
10         $minViolations \leftarrow sumViolations(x\_c)$;
11     end
12     $it \leftarrow it + 1$;
13 end
```

Figure 7: Algorithm 3 - FindFeasibleSolutionUsingTabuSearch()

In implementation, we use TabuSearch module in Open CBLS library [41].

4.1.2 Phase 1: Finding local-timetables

In this phase we apply an exact method to find all solutions (timetables) for each class.

Input:

- Class *k* ∊ *CL*.
- *CC(k)*: list of class-courses of *k*
- *P* = {1,…,12} list of places.

Variables

- *x_p(i)*: represents the places for class-course *i*, *i* ∊ *CC(k)*.

Constraints

- *s(x_p[i]) = s(x_p[j]),* ∀*i,j* ∊ *CC(k):* This constraint states that all class-courses of a class have the same session.

- $cf(x\_p[i],x\_p[j])$ = ***false***, $\forall i,j \in CC(k)$: This constraint states that all class-courses must be scheduled in different places.

Output:

- *S(k)*: A set of feasible local-timetables for each class *k*, where a feasible local-timetable is *x_p*.

Algorithm:

For each class *k*, we use CHOCO Solver to find all feasible local-timetables *S(k)*.

4.1.3 Phase 2: Finding global-timetables

Problem statement:

Input:

- *S(k)*: set of local-timetable for class *k*, $\forall k \in CL$.

Variables:

- s_k: represents selected local-timetable of class *k*, $\forall k \in CL$.

Output:

- Best global-timetable $s^* = (s_1,...,s_K)$: an array to model a global-timetable (timetable for all classes) in which $s_k \in S(k)$ represents the local-timetable selected for class *k*.

For describing the Tabu search in this phase, we give some notations:

- $s = (s_1,...,s_K)$: represents a global-timetable.
- $s^* = (s_1,...,s_K)$: represents the best global-timetable.
- *f(s)* is the objective function (7) and *g(s)* is the number of violations of constraint (2), i.e. the number of pair of class-courses (*i, j)* such that: $diff(i,j) = \textbf{true} \wedge x\_p(i) = x\_p(j)$.
- f^* represents the best value of *f(s)*.
- $s[s_k \leftarrow v]$ the array *s* in which s_k is reassigned to *v*:
- $s[s_k \leftarrow v] = (s_1,. . ., s_{k-1}, v, s_{k+1}, \ldots ,s_K)$.

Algorithm:

Input: *S(k)* is the set of local-timetables for class *k*, $\forall k \in CL$

Output: A random global-timetable

```
foreach $k = 1, \dots, K$ do
    $s_k \leftarrow$ random element of $S(k)$;
end
```

Figure 8: Algorithm 4 - InitRandom()

Input: *S(k)* is the set of local-timetables for class *k*, ∀*k* ∈ *CL*

Output: A random feasible global-timetable

```
InitRandom();
$It \leftarrow 0$;
while $it < maxIter$ do
    $F_1 \leftarrow \{k \in CL \mid \exists k' \in CL \setminus \{k\}$ such that
    $CF_s(k, k') = true\}$;
    if $F_1 = \emptyset$ then
        return $s$;
    else
        $k \leftarrow$ random element of $F_1$;
        $F_2 \leftarrow \{v \in S(k) \mid g(s[s_k \leftarrow v])$ is minimal $\}$;
        $s \leftarrow s[s_k \leftarrow v]$;
    end
    $it \leftarrow it + 1$;
end
return null;
```

Figure 9: Algorithm 5 - FindRandomFeasibleGlobalTimeTable()

Input: *S(k)* is the set of local-timetables for class *k*; ∀*k* ∈ *CL*,

Output: Generate random feasible global-timetable and initialize tabu list

```
$s \leftarrow$ FindRandomFeasibleSolution();
$nic \leftarrow 1$;
foreach $k = 1, \dots, K$ do
    $tabu[k] \leftarrow -1$;
end
```

Figure 10: Algorithm 6 - InitFeasible()

Input:

- *S(k)* is the set of local-timetables for class *k*; $\forall k \in CL$,
- Global variables:
    - *tabu*: represents the tabu list
    - *tbl*: length of the tabu list
    - *nic*: number of consecutive iterations that best global-timetable is not improved
    - *maxStable*: if the best solution is not improved after *maxStable* iterations, then the search is restarted

Output: Best global-timetable *s**

```
InitFeasible();
$f^* \leftarrow f(s)$;
$s^* \leftarrow s$;
$it \leftarrow 0$;
while $it < maxIter$ do
    $F_1 \leftarrow \{k \in CL \mid tabu[k] < it \wedge d(k)$ is maximal$\}$;
    if $F_1 = \emptyset$ then
        InitFeasible();
    else
        $k \leftarrow$ random element of $F_1$;
        $F_2 \leftarrow \{v \in S(k) \mid s[s_k \leftarrow v]$ is feasible and
        $f(s[s_k \leftarrow s'])$ is minimal$\}$;
        if $F_2 = \emptyset$ then
            InitFeasible();
        else
            $s' \leftarrow$ random element of $F_2$;
            $s_k \leftarrow s'$;
            if $f(s) < f^*$ then
                $f^* \leftarrow f(s)$;
                $s^* \leftarrow s$;
                $nic \leftarrow 1$;
            else
                $nic \leftarrow nic + 1$;
                if $nic > maxStable$ then
                    InitFeasible();
                    if $f^* > f(s)$ then
                        $f^* \leftarrow f(s)$;
                        $s^* \leftarrow s$;
                    end
                end
            end
            $tabu[k] \leftarrow it + tbl$;
        end
    end
    $it \leftarrow it + 1$;
end
return $\langle s^*, f^* \rangle$;
```

Figure 11: Algorithm 7 - TabuSearch()

4.1.4 Phase 3: Assigning teacher

Problem statement:

Input:

- A set of teachers: $T = \{1,.., NT\}$.
- $TL(c)$: list of teachers who can instruct course c, $c \in C$, $TL(c) \in T$.
- s^*: timetable of all classes from 4.1.3.
- $\forall$ (c,t), $c \in C$, $t \in T$, $ins(t,c)$ = **true** indicates teacher t can instruct course c, otherwise **false**.
- $EC(t)$: is a subset of C. $EC(t) = \{c \in C \mid ins(t,c) = \textbf{true}\}$.

Variables:

- $x\_t[cc]$: presents teacher who will instructs classcourse cc. Domain of $x\_t[cc]$ is T, $cc \in CC$.

Invariants:

- $minTarget(t)$: minimum number of class-course that teacher t will instruct.

Constraints:

- $cf(i, j) = \textbf{true} \rightarrow x\_t[i] \neq x\_t[j]$.
- $AS(t)$: subset of CC, is a list of class-courses that have been assigned to teacher $t \in T$.
- $card(AS(t))$: cardinality of $AS(t)$.

Output:

- Global solution of x_t.

Algorithm:

Idea: For each classcourse cc we pick the most appropriate teacher t in $TL(c(cc))$ and assign to this classcourse. Picking the most appropriate teacher is based on a score between cc and t, called *pickScore*. The teacher with highest *pickScore* among candidates (teacher in $TL(c(cc))$) will be picked:

- $pickScore(t,cc) = targetScore(t) + suitableScore(t,cc)$.
  Where:
  - $targetScore(t) = minTarget(t) - card(AS(t))$.
  - $suitableScore(t,cc) \in R$, indicates the suitability between teacher $t \in TL(c(cc))$ and classcourse $cc \in CC$.
    - $suitableScore(t,cc) = \textbf{min}(relativeScore(cc,cc'))$, $cc' \in AS(t)$.
  - $relativeScore(cc1,cc2)$ indicates relationship of classcourse $cc1$ and classcourse $cc2$ in timetable.

- *relativeScore(cc1,cc2)* = ***w1*****sameday(cc1,cc2)* + ***w2*** * *samenoon(cc1,cc2).*

Where:

- *sameday(cc1,cc2)* = 1 if *cc1* and *cc2* are taken place on the same day, otherwise equal 0.
- *samenoon(cc1,cc2)* = 1 if *cc1* and *cc2* are taken place on the same day and both at slot 3 or slot 4, otherwise equal 0.
- ***w1***: weight of *sameday(cc1,cc2)*. This weight specifies how much we favour the case that a teacher will instruct two class-courses on the same day.
- ***w2***: weight of *samenoon(cc1,cc2)*. This weight specifies how much we against the case that a teacher will instruct two class-courses at slot 3 and 4 consecutively on the same day.

```
foreach $cc \in CC$ do
    $F \leftarrow \emptyset$;
    $course \leftarrow c(cc)$;
    if $TL(course) \neq \emptyset$ then
        foreach $t' \in TL(course)$ do
            if $isFeasibleToAssign(t', cc) = true$ then
                $F \leftarrow F \cup \{t'\}$;
            end
        end
        if $F \neq \emptyset$ then
            $F1 \leftarrow \{t \in F \mid pickScore(t, cc)\ is\ maximal\}$;
            $t' \leftarrow random\ element\ of\ F1$;
            $x\_t[cc] \leftarrow t'$;
            $AS(t') \leftarrow AS(t') \cup \{cc\}$;
        end
    end
end
return $x\_t$;
```

Figure 12: Algorithm 8 - AssigningTeacher()

```
Input: Classcourse $cc$.
       Teacher $t$.

Output: Return **true** if it is possible to assign teacher $t$ to
        classcourse $cc$, else return **false**.
1 foreach $cc' \in AS(t)$ do
2     if $cf(cc, cc') = true$ then
3         return $false$;
4     end
5 end
6 return $true$;
```

Figure 13: Algorithm 9 - IsFeasibleToAssign(t,cc)

```
Input: Teacher $t$.

Output: $minTarget(t)$.
1 foreach $t \in T$ do
2     $minTarget(t) \leftarrow 0$;
3     $F \leftarrow \{c \in C \mid ins(t, c) = true\}$;
4     foreach $c \in F$ do
5         $minTarget(t) \leftarrow minTarget(t) + \left\lceil |CCL(c)| \,/\, |TL(c)| \right\rceil$;
6     end
7 end
8 return $true$;
```

Figure 14: Algorithm 10 - calculateMinTarget(t)

4.1.5 Phase 4: Assigning room

Problem statement:

Input:

- Given a set of room: *RO = {1,...,NR}*.
- *s**: timetable of all classes from 4.1.3
- *spl(r)* ⊂ *P*, represents set of places that room *r* can supplies. *spl(r)[i]* represents place *i* of room *r* ∈ *RO*, *spl(r)[i]* ∈ *P*.
- *x_p(cc):* place of classcourse *cc*.

Variables:

- *x_r[i]* presents a room that is assigned to class-course *i*, *i* ∈ *CC*, *x_r[i]* ∈*R*.

Invariants:

- *mark(r)[i]* in {**true**, **false**}, represents status of place *i* of room *r*. *mark(r)[i]* = **true** if place *i* of room *r* is acquired, else equal **false**.

Constraint

- *cf(i,j)* = **true** → *x_r[i]* ≠ *x_r[j]*

Output:

- Global solution *x_r*

Algorithm :

```
Input: As problem statement
Output: As problem statement
foreach $k \in CL$ do
    foreach $cc \in CC(k)$ do
        $p \leftarrow x\_p(cc)$;
        foreach $r \in RO$ do
            if $p \in spl(r) \wedge mark(r)[p] = false$ then
                $x\_r[cc] = r$;
                $mark(r)[p] = true$;
                break;
            end
        end
    end
end
```

Figure 15: Algorithm 11 - AssignRoom()

4.2 Experiments and Results

Our project has many independent phases, so we decide to experiment and collect result of each phase. In phase 1 and 2, we conduct an experiment on a real data of Summer 2015[2] semester. This data consists of 57 classes, 112 courses, and 200 class-courses. In the other phases including pre-processing (merging class-courses), we experiment on partitioning Fall 2015 semester dataset to test the possibility of the result. This data consist of 27 classes, 35 courses and 135 class-courses.

In pre-processing: Merging classes

For each *tbl* (*tbl* =10, 20, 50), we execute the program 20 times. The results are all the same: 9 bins were packed or 9 merged class-courses in all experiments. This means that our algorithm is stable.

Phase 1 and phase 2: Finding timetables

For this data, the timetable was made manually and the value of the objective function (7) is **91**.

In our experiments, we execute the program 20 times and report the minimum, maximum, average, and standard deviation of the best objective function value found of 20 times for each *tbl*. The experimental results with 3 values of *tbl* are presented in Table 1. Table 2 shows that the solution produced by our proposed algorithm is better than manually-making timetable. The performance is stable and good for the value 10, 20 of *tbl*. Figure 16 presents the best objective values found in 10 last executions. Figure 17 shows the behaviour of the Tabu search.

tbl	min	max	avg	std
100	74	76	75.05	0.6
50	74	74	74	0
20	74	74	74	0

Table 1: Objective function results in 20 executions each tbl, Summer dataset

[2] The new policy with two blocks described above has just applied this year, so there is only one data.

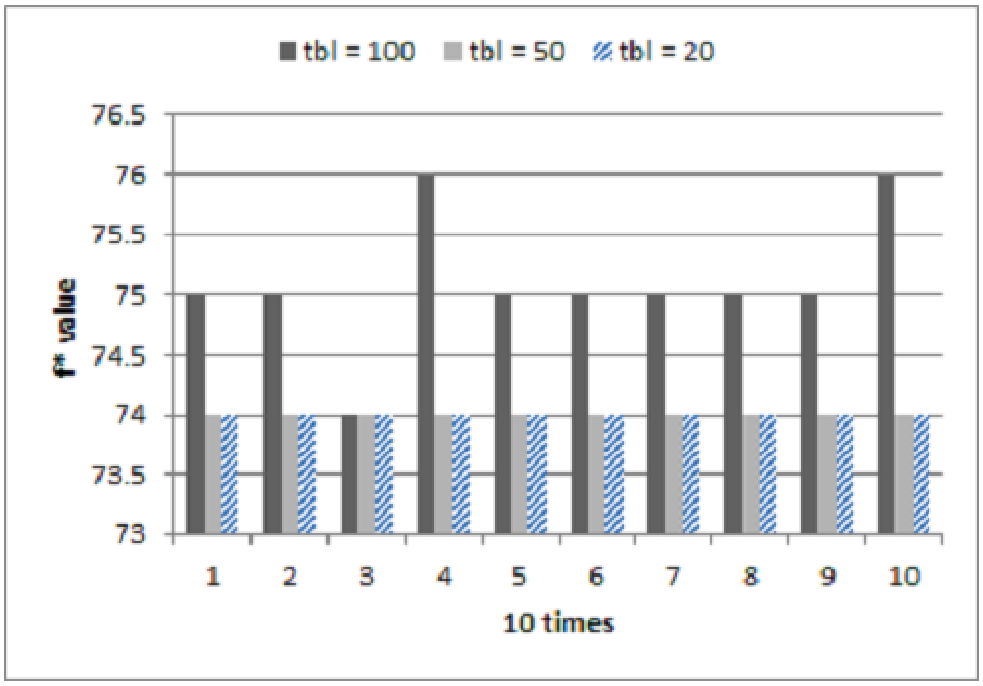


Figure 16: Objective function results in last 10 executions each tbl, Summer dataset

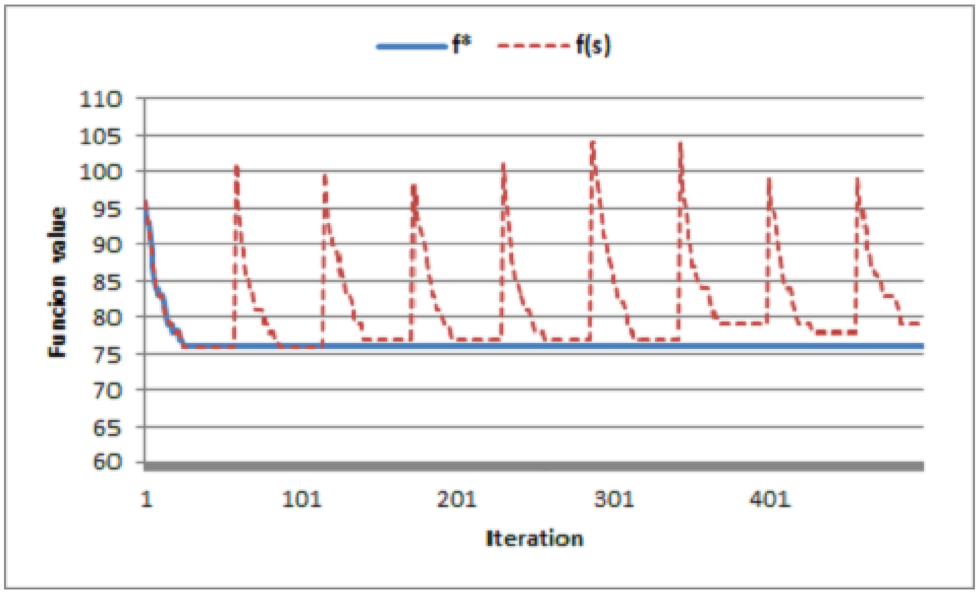


Figure 17: Behaviour of the Tabu search

	Manual	Complete method (using CHOCO)	Tabu search
Objective function value	91	159	74-76
Running time	2 weeks	2 hours	10 minutes

Table 2: Comparing with other methods

Phase 3: Assigning teacher

On Fall dataset, we try (w_1,w_2) = { (10,-5) ; (5,-5) ; (5,-10) } and collect results of 2 functions: number of class-course that are assigned room and percentage of times a teacher working on slot 3,4 in total of his working days. Program is executed 10 times for each pair of (w_1,w_2). Results are presented below:

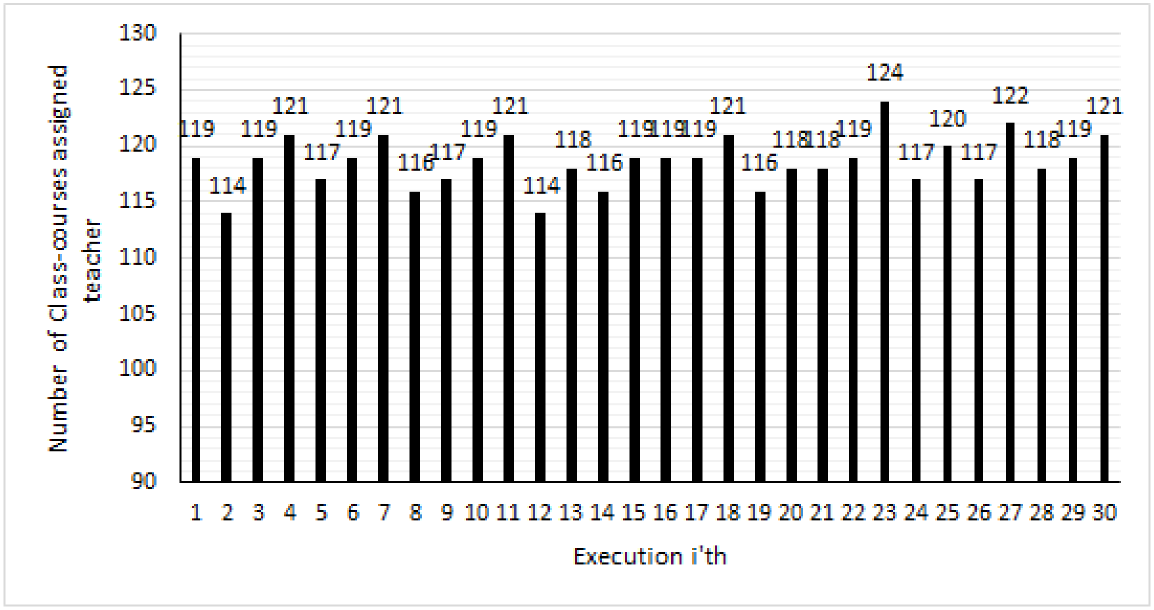


Figure 18: Number of class-course assigned teacher in 30 executions, Fall dataset

min	max	avg	std
114	124	118,6	2,252967

Table 3: Statistics on number of class-course assigned teacher in total 135 class-courses

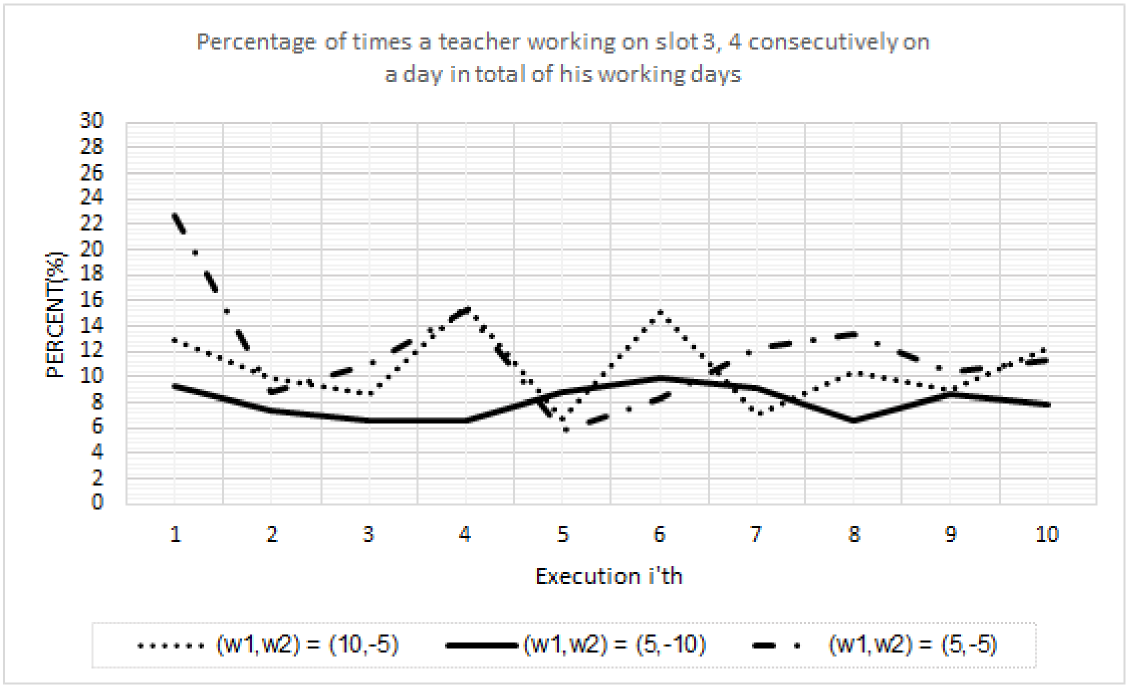


Figure 19: Percentage of times a teacher working on slot 3, 4 consecutively on a day in total of his working days

	min	max	avg	std
(w_1,w_2) = (10,-5)	6,75	15,52	10,762	3,122594
(w_1,w_2) = (5,-10)	6,53	9,85	8,059	1,259007
(w_1,w_2) = (5,-5)	5,8	22,63	11,921	4,623533

Table 4: Statistics on percentage of times a teacher working on slot 3, 4 consecutively on a day in total of his working days

	min	max	avg	std
(w_1,w_2) = (10,-5)	114	121	118,2	2,20101
(w_1,w_2) = (5,-10)	117	124	119,5	4,752304
(w_1,w_2) = (5,-5)	114	121	118,1	2,233582

Table 5: Number of class-courses assigned teacher in each case (w_1,w_2)

Phase 4: Assigning room

We collected number of rooms assigned in total 30 executions on Fall dataset. Results are presented in Table 6 and Figure 20.

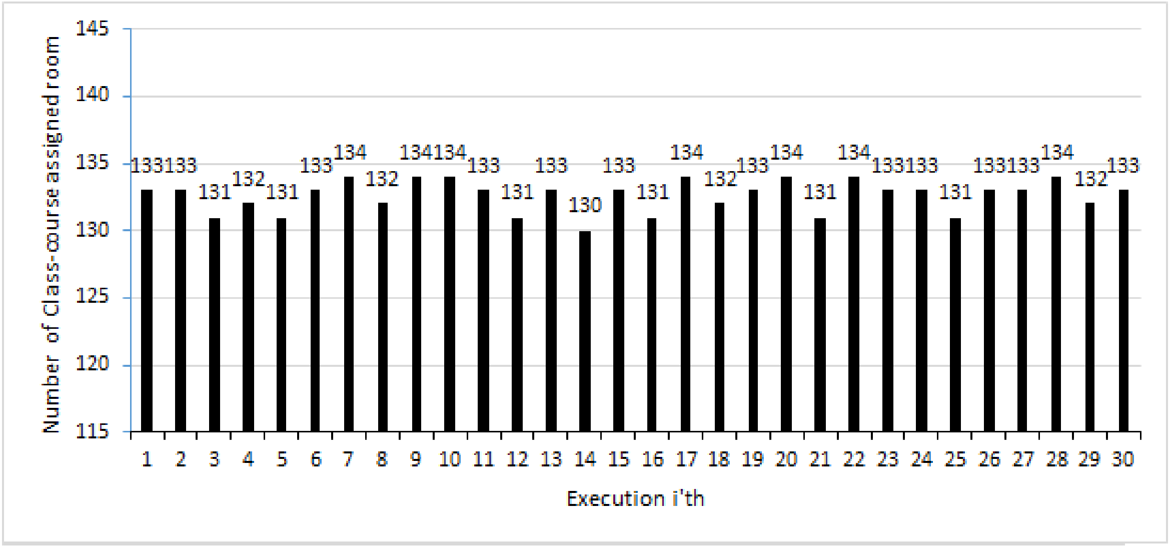


Figure 20: Number of class-course assigned room tested on Fall dataset

min	max	avg	std
130	134	132,6	1,162637

Table 6: Statistics number of class-course assigned room in total 135 class-courses

Chapter 5: Conclusion and further work

In this project, we produced a web service that support Academics Department in scheduling, class arrangement, merging class-course, handling teacher, student and resources automatically and manually. The achieved results of automatic-scheduling objective are positive. Generated timetable satisfies the hard constraints such that no conflict resources were founded. It also makes profit by reducing the resource cost (about teachers) and increasing teachers' productivity. We presented a new hybrid approach to solve the specific case at FPT University. Our system is currently being introduced in FPT Academics Department. The results are quite appealing and used method shows a good promise in the future. With limited time, knowledge and human effort, our automatic scheduling system cannot cover all requirements of Academics Department; some specific cases of small group of student are not yet covered. The future works should concentrate on dealing with other requirements, improving the usability and trying some more optimization functions. We also investigate other approaches such as Large Neighborhood Search and other large scale techniques. Finally, more test datasets need to be tested in order to improve our method.

Chapter 6: References

[1]	University Timetabling, http://www.unitime.org/.
[2]	T. Muller. Constraint-based Timetabling, PhD thesis. KTIML MFF UK, Prague, 2005.
[3]	T. Muller. Real-life examination timetabling. In MISTA 2013 - Proceedings of the 6th Multidisciplinary International Scheduling Conference, 2013.
[4]	T. Muller, R. Bartak, and H. Rudova. Minimal perturbation problem in course timetabling. In Edmund Burke and Michael Trick, editors, Practice and Theory of Automated Timetabling, Selected Revised Papers, pages 126–146, 2005.
[5]	T. Muller, K. Murray, and S. Schluttenhofer. University course timetabling and student sectioning system. In ICAPS 07, The International Conference on Automated Planning and Scheduling, 2007.
[6]	T. Muller and H. Rudova. Real-life curriculum-based timetabling. In In PATAT 2012 - Proceedings of the 9th international conference on the Practice And Theory of Automated Timetabling, 2012.
[7]	T. Muller, H. Rudova, and K. Murray. Interactive course timetabling. In In MISTA 2009 - Proceedings of the 4th Multidisciplinary International Scheduling Conference, 2009.
[8]	K. Murray, T. Muller, and H. Rudova. Modeling and solution of a complex university course timetabling problem. In In Edmund Burke and Hana Rudova, editors, Practice and Theory of Automated Timetabling, Selected Revised Papers, Springer-Verlag LNCS 3867, pages 189–209, 2007.
[9]	H. Rudova and T. Muller. Rapid development of university course timetables. In In MISTA 2011 - Proceedings of the 5th Multidisciplinary International Scheduling Conference, 2011.

[10]	Wren, A. Scheduling, Timetabling and Rostering – A Special Relationship, in Burke and Ross (eds.), The Practice and Theory of Automated Timetabling, Springer LNCS, Vol.1153, 1996
[11]	M. W. Carter. A Survey of Practical Applications of Examination Timetabling Algorithms. Operations Research 34, p193-202, 1986.
[12]	Michael W. Carter, Gilbert Laporte and Sau Yan Lee. Examination Timetabling: Algorithmic Strategies and Applications. The Journal of the Operational Research Society. Vol. 47, No. 3 Mar., 1996
[13]	M. W. Carter. A Survey of Practical Applications of Examination Timetabling Algorithms. Operations Research 34, p193-202, 1986.
[14]	D. de Werra. An Introduction to Timetabling. European Journal of Operations Research 19, 1985
[15]	Brélaz, D. "New methods to color the vertices of a graph", Communications of the ACM 22 (4) ,1979
[16]	Michael W. Carter, Gilbert Laporte and Sau Yan Lee. Examination Timetabling: Algorithmic Strategies and Applications. The Journal of the Operational Research Society. Vol. 47, No. 3 Mar., 1996
[17]	G. M. White, P. W. Chan. Towards the Construction of Optimal Examination Timetables. INFOR 17, p219-229, 1979.
[18]	Fisher, J.G. and R.R. Shier, A Heuristic Procedure for Large-Scale examination scheduling problems. Congressus Numerantium, 1983
[19]	F.Rossi, P.VanBeek, and T.Walsh. Handbook of Constraint Programming. Elsevier, 2006.
[20]	GECODE, http://www.gecode.org/
[21]	Minizinc, http://www.minizinc.org/.
[22]	P. V. Hentenryck and L. Michel. Constraint-based Local Search. The MIT Press, 2005.
[23]	http://choco-solver.org/

[24]	Edmund Burke, Peter Ross.(eds) Practice and Theory of Automated Timetabling First International Conference Edinburgh, U.K., August 29–September 1, 1995
[25]	Edmund Burke, Michael Carter.(eds) Practice and Theory of Automated Timetabling II. Second International Conference, PATAT'97 Toronto, Canada, August 20–22, 1997
[26]	Burke, Edmund, Erben, Wilhelm.(eds) Practice and Theory of Automated Timetabling III Third International Conference, PATAT 2000 Konstanz, Germany, August 16-18, 2000
[27]	Zbigniew Michalewicz and David B. Fogel. How to Solve It: Modern Heuristics. Springer, 2000.
[28]	Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence, 58:161–205, 1992.
[29]	T. Muller. Constraint-based Timetabling, PhD thesis. KTIML MFF UK, Prague, 2005.
[30]	Philippe Galinier and Jin-Kao Hao. Tabu search for maximal constraint satisfaction problems. In Proceedings 3rd International Conference on Principles and Practice of Constraint Programming, pages 196–208. Springer-Verlag LNCS 1330, 1997.
[32]	T. Muller. Constraint-based Timetabling, PhD thesis. KTIML MFF UK, Prague, 2005.
[31]	Stephanie Forrest, ACM Computing Surveys (CSUR) Volume 28 Issue 1, Pages 77-80, March 1996
[33]	Tomáš Müller. Constraint-based Timetabling, Ph.D. Thesis, KTIML MFF UK, Prague, p 26-30, 2005
[34]	A. Schaerf. Tabu search techniques for large high-school timetabling problems. In AAAI-96 Proceedings, 1996.
[35]	E. Burke, D. Elliman, and R. Weare. A hybrid genetic algorithm for highly constrained timetabling problems. In Proceedings of the Sixth International Conference on Genetic Algorithms, pages 605-610, 1995.

[36]	E. K. Burke, J. P. Newall, and R. F. Weare. A memetic algorithm for university exam timetabling. pages 241- 250. Springer-Verlag, 1996.
[37]	H. Fang. Genetic algorithms in TimeTabling and Scheduling. PhD thesis. University of Edinburgh, 1994.
[38]	A. Colorni, M. Dorigo, and V. Maniezzo. Metaheuristics for high-school timetabling. Computational Optimization and Applications, 9-1998.
[39]	B. Bullnheimer. An examination scheduling model to maximize students's study time. In Proceedings of the Second International Conference on the Practice and Theory of Automated Timetabling, number 1408 in Lecture Notes in Computer Science, pages 78-91,1998.
[40]	C. Meyers and J. B. Orlin. Very large-scale neighborhood search techniques in timetabling problems. In Practice and Theory of Automated TimeTabling VI, Lecture Notes in Computer Science Volume 3867, pages 24-39, 2007.
[41]	OpenLS: A Local Search Library for Combinatorial Optimization. Algorithms & Optimization LAB (SoICT/HUST)

Appendix: Program Guide

I. How to build

a. Environment:

i. OS: Windows 7 or later

ii. Browser: Google Chrome 44 or later

iii. Tomcat 7.0 or later

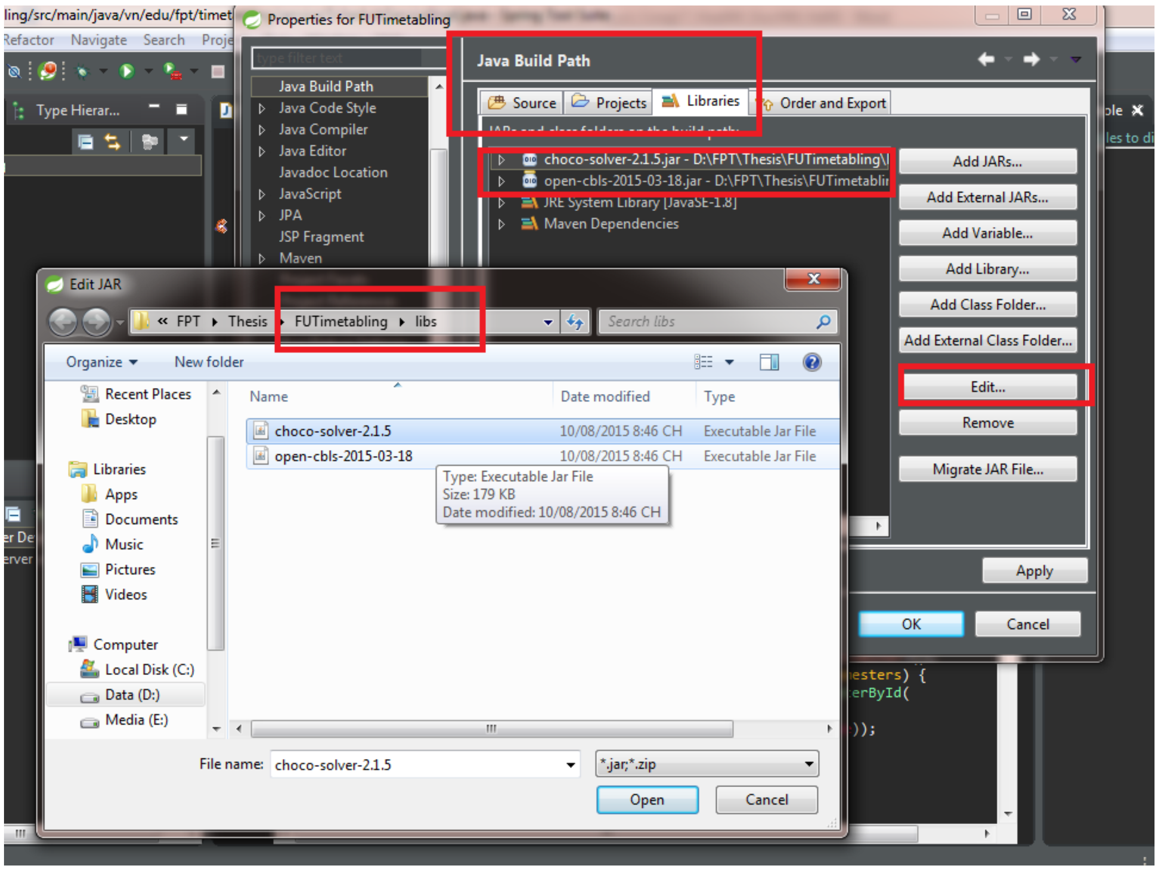
iv. XAMPP 3.2.1 or later (for phpmyadmin)

b. Developer Tools:

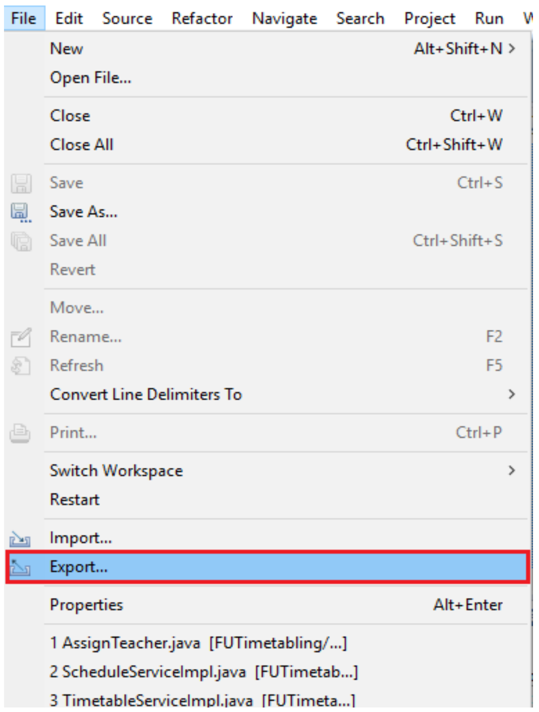
i. Spring Tool Suite 3.7.0 or later

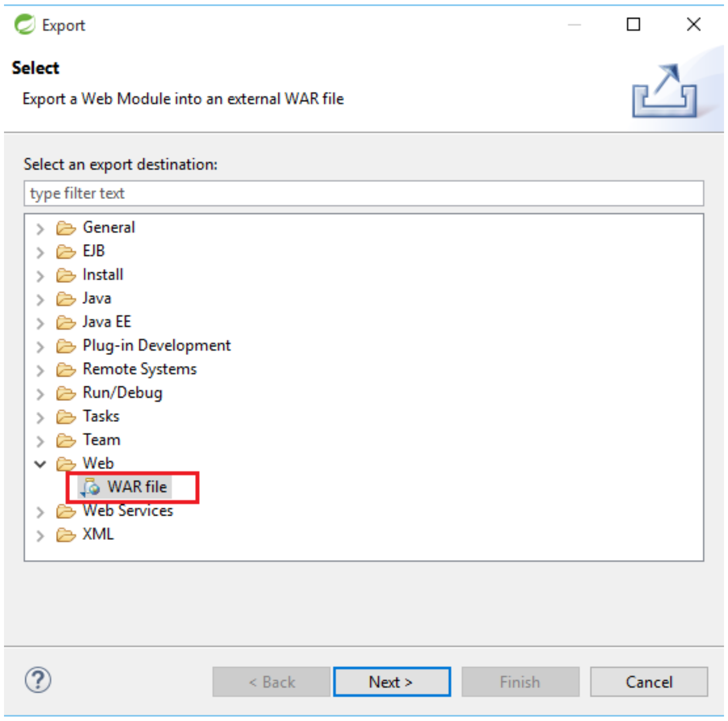
c. How to build

i. Config build path for project, choose library choco-solver-2.1.5 & open–cbls-2015-03-18

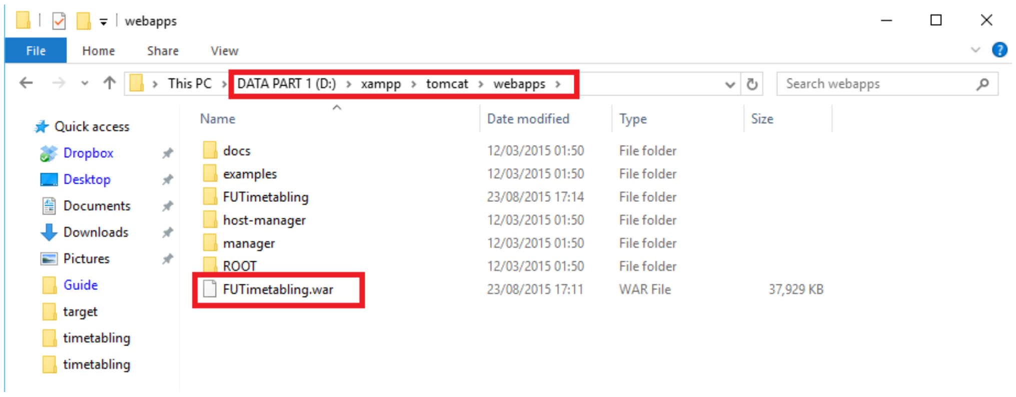


ii. Use Spring Tool Suite to build project into file 'FUTimetabling.WAR'



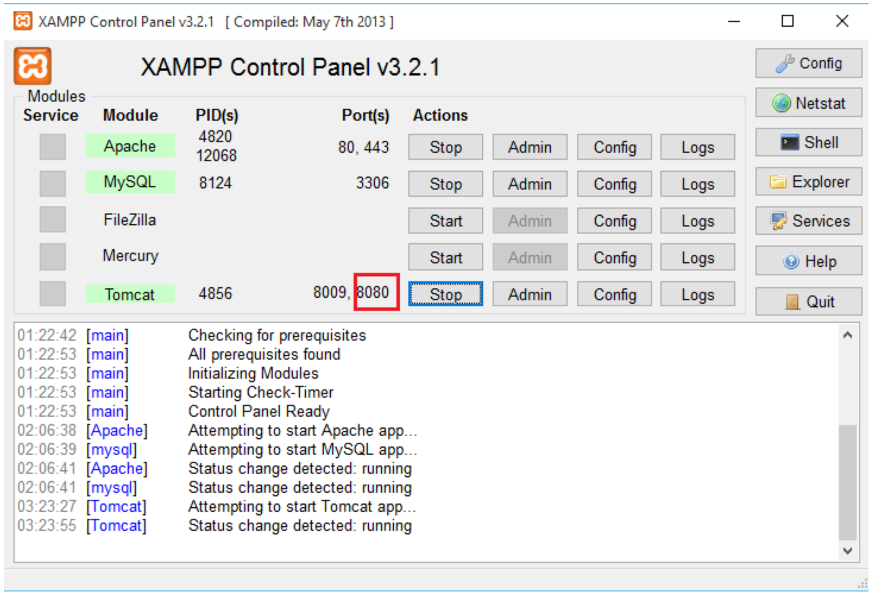


iii. Copy file 'FUTimetabling.WAR' into 'Tomcat's install folder/webapps/'



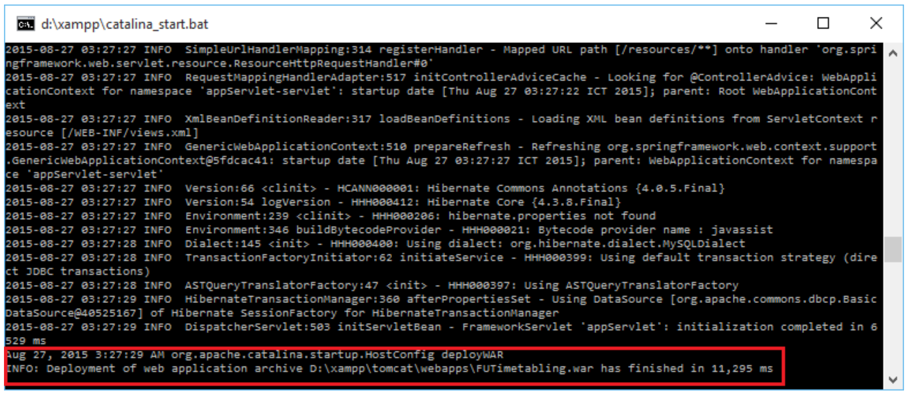
Name	Date modified	Type	Size
docs	12/03/2015 01:50	File folder	
examples	12/03/2015 01:50	File folder	
FUTimetabling	23/08/2015 17:14	File folder	
host-manager	12/03/2015 01:50	File folder	
manager	12/03/2015 01:50	File folder	
ROOT	12/03/2015 01:50	File folder	
FUTimetabling.war	23/08/2015 17:11	WAR File	37,929 KB

iv. Wait for Tomcat deploy project (Tomcat must keep port 8080)


XAMPP Control Panel v3.2.1 [Compiled: May 7th 2013]
XAMPP Control Panel v3.2.1
Config
Netstat
Shell
Explorer
Services
Help
Quit
Modules

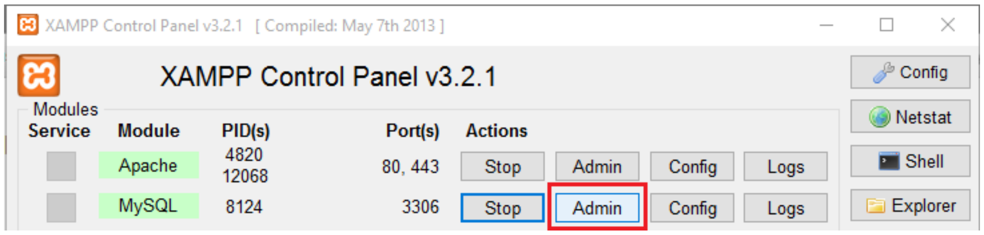
Service	Module	PID(s)	Port(s)	Actions			
	Apache	4820 12068	80, 443	Stop	Admin	Config	Logs
	MySQL	8124	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat	4856	8009, 8080	Stop	Admin	Config	Logs

```
01:22:42  [main]      Checking for prerequisites
01:22:53  [main]      All prerequisites found
01:22:53  [main]      Initializing Modules
01:22:53  [main]      Starting Check-Timer
01:22:53  [main]      Control Panel Ready
02:06:38  [Apache]    Attempting to start Apache app...
02:06:39  [mysql]     Attempting to start MySQL app...
02:06:41  [Apache]    Status change detected: running
02:06:41  [mysql]     Status change detected: running
03:23:27  [Tomcat]    Attempting to start Tomcat app...
03:23:55  [Tomcat]    Status change detected: running
```

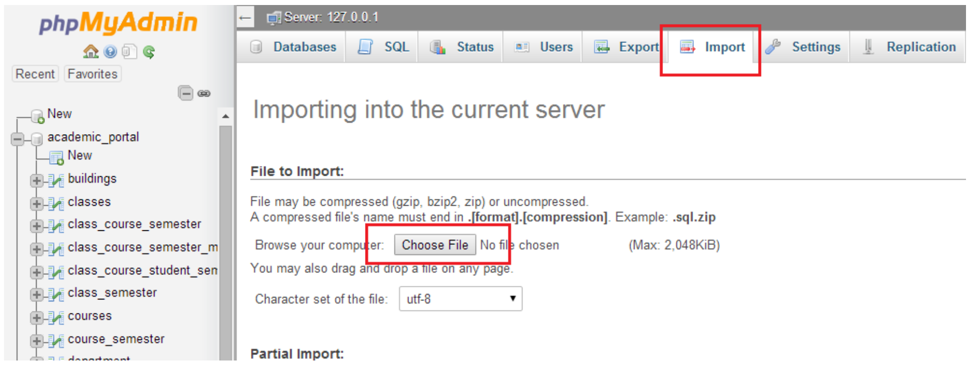


```
2015-08-27 03:27:27 INFO  SimpleUrlHandlerMapping:314 registerHandler - Mapped URL path [/resources/**] onto handler 'org.spri
ngframework.web.servlet.resource.ResourceHttpRequestHandler#0'
2015-08-27 03:27:27 INFO  RequestMappingHandlerAdapter:517 initControllerAdviceCache - Looking for @ControllerAdvice: WebAppli
cationContext for namespace 'appServlet-servlet': startup date [Thu Aug 27 03:27:22 ICT 2015]; parent: Root WebApplicationCont
ext
2015-08-27 03:27:27 INFO  XmlBeanDefinitionReader:317 loadBeanDefinitions - Loading XML bean definitions from ServletContext r
esource [/WEB-INF/views.xml]
2015-08-27 03:27:27 INFO  GenericWebApplicationContext:510 prepareRefresh - Refreshing org.springframework.web.context.support
.GenericWebApplicationContext@5fdcac41: startup date [Thu Aug 27 03:27:27 ICT 2015]; parent: WebApplicationContext for namespa
ce 'appServlet-servlet'
2015-08-27 03:27:27 INFO  Version:66 <clinit> - HCANN000001: Hibernate Commons Annotations {4.0.5.Final}
2015-08-27 03:27:27 INFO  Version:54 logVersion - HHH000412: Hibernate Core {4.3.8.Final}
2015-08-27 03:27:27 INFO  Environment:239 <clinit> - HHH000206: hibernate.properties not found
2015-08-27 03:27:27 INFO  Environment:346 buildBytecodeProvider - HHH000021: Bytecode provider name : javassist
2015-08-27 03:27:28 INFO  Dialect:145 <init> - HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
2015-08-27 03:27:28 INFO  TransactionFactoryInitiator:62 initiateService - HHH000399: Using default transaction strategy (dire
ct JDBC transactions)
2015-08-27 03:27:28 INFO  ASTQueryTranslatorFactory:47 <init> - HHH000397: Using ASTQueryTranslatorFactory
2015-08-27 03:27:29 INFO  HibernateTransactionManager:360 afterPropertiesSet - Using DataSource [org.apache.commons.dbcp.Basic
DataSource@40525167] of Hibernate SessionFactory for HibernateTransactionManager
2015-08-27 03:27:29 INFO  DispatcherServlet:503 initServletBean - FrameworkServlet 'appServlet': initialization completed in 6
529 ms
Aug 27, 2015 3:27:29 AM org.apache.catalina.startup.HostConfig deployWAR
INFO: Deployment of web application archive D:\xampp\tomcat\webapps\FUTimetabling.war has finished in 11,295 ms
```

d. Use phpmyadmin in XAMPP to import database (database.sql)


XAMPP Control Panel v3.2.1 [Compiled: May 7th 2013]
XAMPP Control Panel v3.2.1
Config
Netstat
Shell
Explorer

Service	Module	PID(s)	Port(s)	Actions			
	Apache	4820 12068	80, 443	Stop	Admin	Config	Logs
	MySQL	8124	3306	Stop	Admin	Config	Logs



Importing into the current server

File to Import:

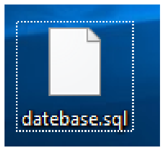
File may be compressed (gzip, bzip2, zip) or uncompressed.
A compressed file's name must end in **.[format].[compression]**. Example: **.sql.zip**

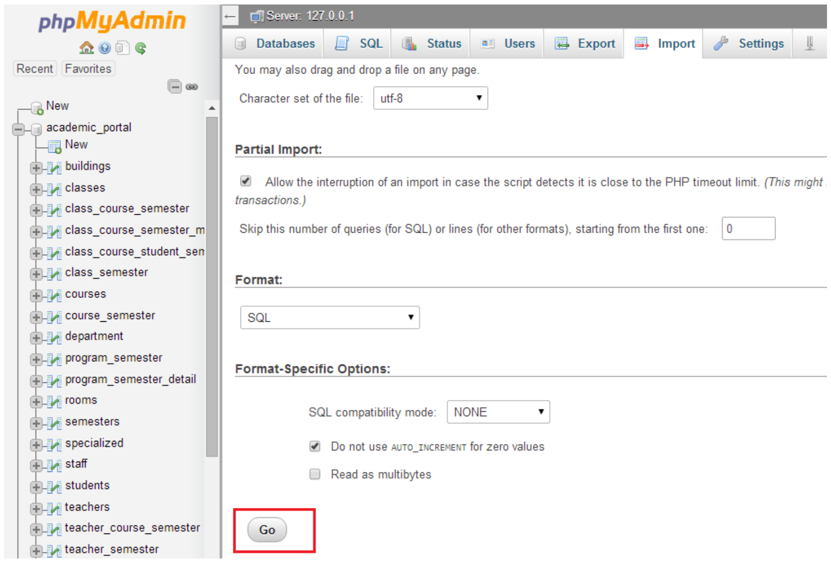
Browse your computer: Choose File No file chosen (Max: 2,048KiB)

You may also drag and drop a file on any page.

Character set of the file: utf-8

Partial Import:





You may also drag and drop a file on any page.

Character set of the file: utf-8

Partial Import:

☑ Allow the interruption of an import in case the script detects it is close to the PHP timeout limit. *(This might transactions.)*

Skip this number of queries (for SQL) or lines (for other formats), starting from the first one: 0

Format:

SQL

Format-Specific Options:

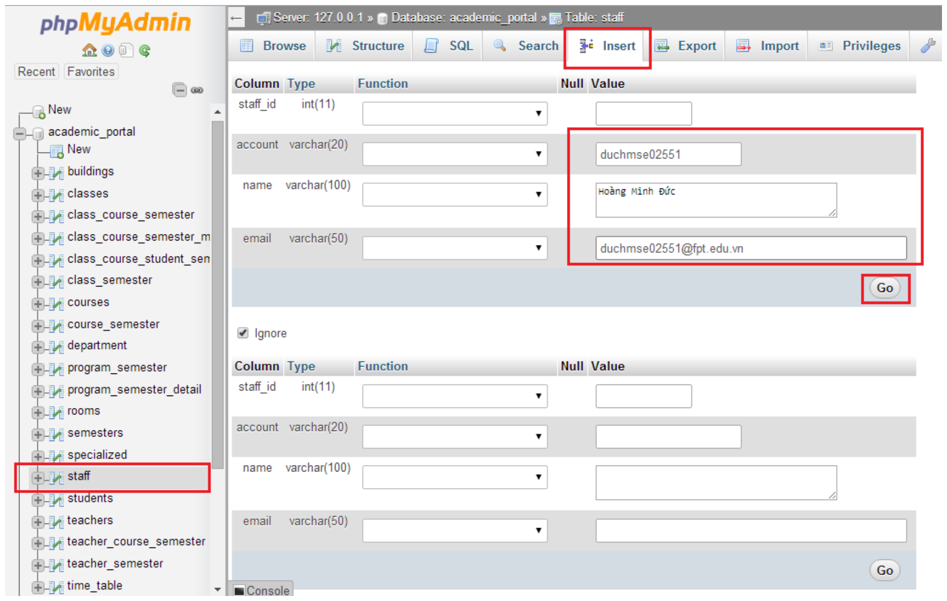
SQL compatibility mode: NONE

☑ Do not use AUTO_INCREMENT for zero values

☐ Read as multibytes

Go

e. Use phpmyadmin in XAMPP to add staff accounts (must be @fpt.edu.vn email)



Column	Type	Function	Null	Value
staff_id	int(11)			
account	varchar(20)			duchmse02551
name	varchar(100)			Hoàng Minh Đức
email	varchar(50)			duchmse02551@fpt.edu.vn

Go

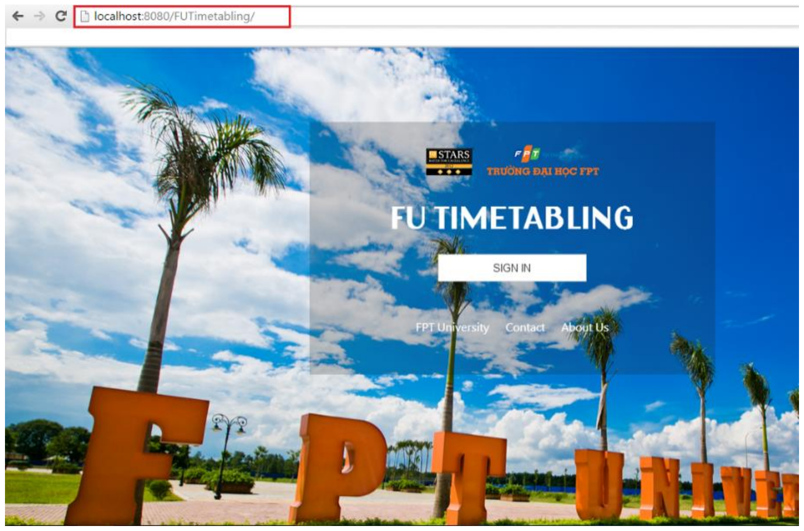
✓ Ignore

Column	Type	Function	Null	Value
staff_id	int(11)			
account	varchar(20)			
name	varchar(100)			
email	varchar(50)			

Go

II. How to run

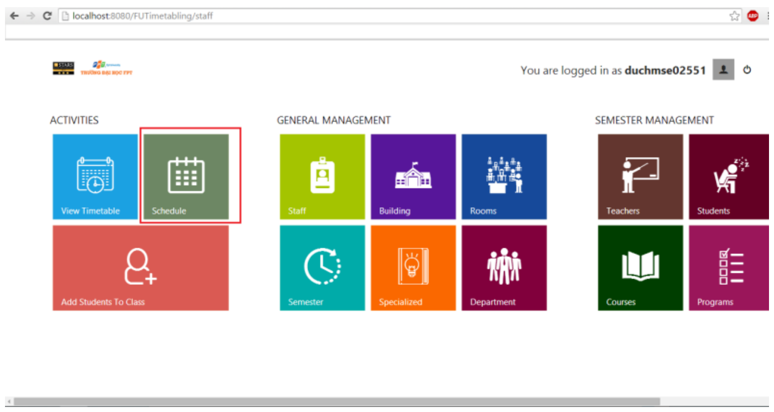
a. Use link 'http://localhost:8080/FUTimetabling/' to run project



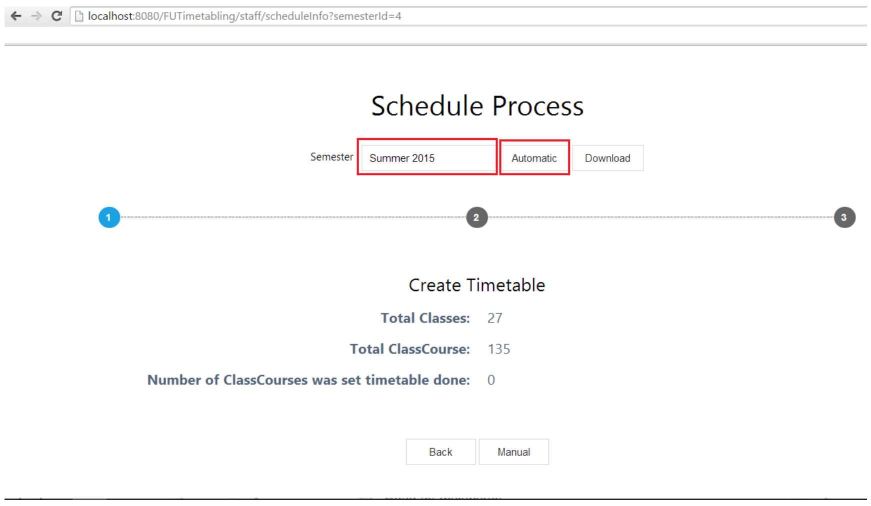
b. Sign in by email registered above

c. Run automatic timetabling:

1. Click to 'Schedule'



2. Click to Automatic


localhost:8080/FUTimetabling/staff/scheduleInfo?semesterId=4
Semester
Automatic
Download
1
2
3
Back
Manual

Schedule Process

Semester: Summer 2015

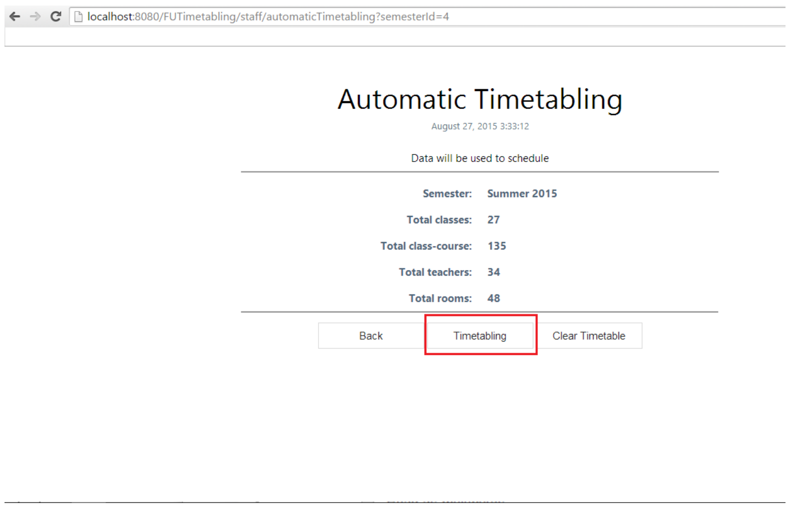
Create Timetable

Total Classes: 27

Total ClassCourse: 135

Number of ClassCourses was set timetable done: 0

3. Click to ‘Timetabling’


localhost:8080/FUTimetabling/staff/automaticTimetabling?semesterId=4
Back
Timetabling
Clear Timetable

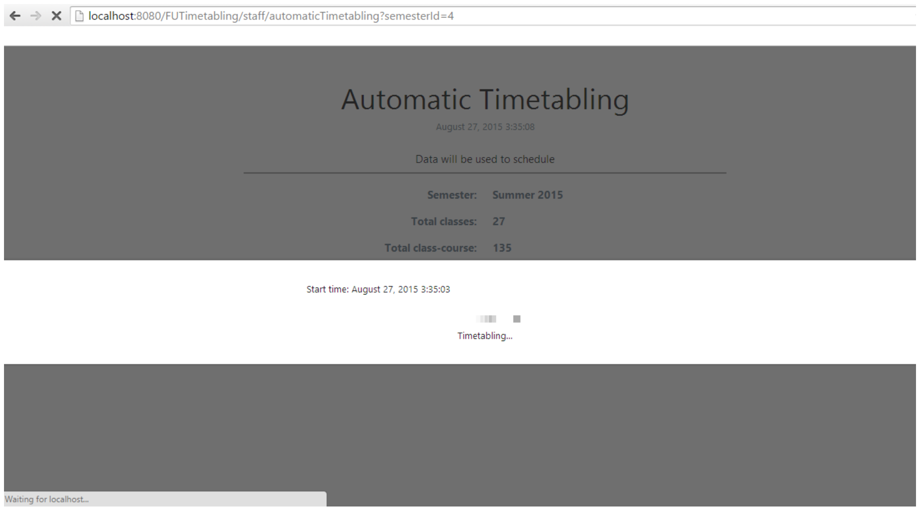
Automatic Timetabling

August 27, 2015 3:33:12

Data will be used to schedule

Semester:	Summer 2015
Total classes:	27
Total class-course:	135
Total teachers:	34
Total rooms:	48

4. Wait while application running


localhost:8080/FUTimetabling/staff/automaticTimetabling?semesterId=4

Automatic Timetabling

August 27, 2015 3:35:08

Data will be used to schedule

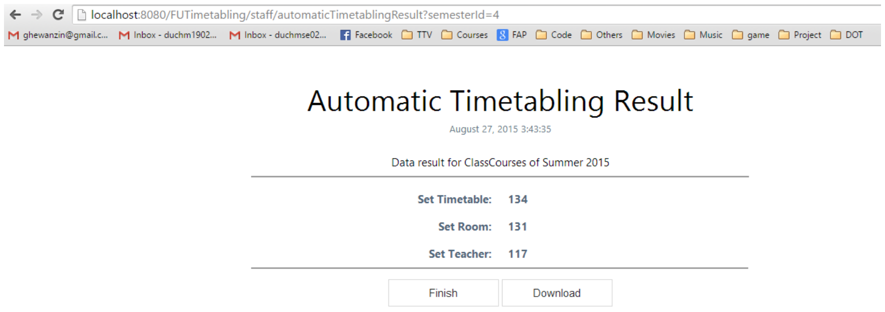
Semester:	Summer 2015
Total classes:	27
Total class-course:	135

Start time: August 27, 2015 3:35:03

Timetabling...

Waiting for localhost...

5. See result, click download


localhost:8080/FUTimetabling/staff/automaticTimetablingResult?semesterId=4

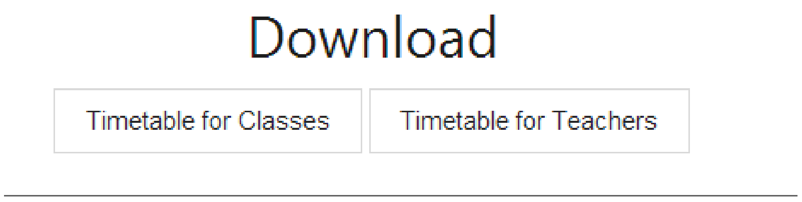
Automatic Timetabling Result

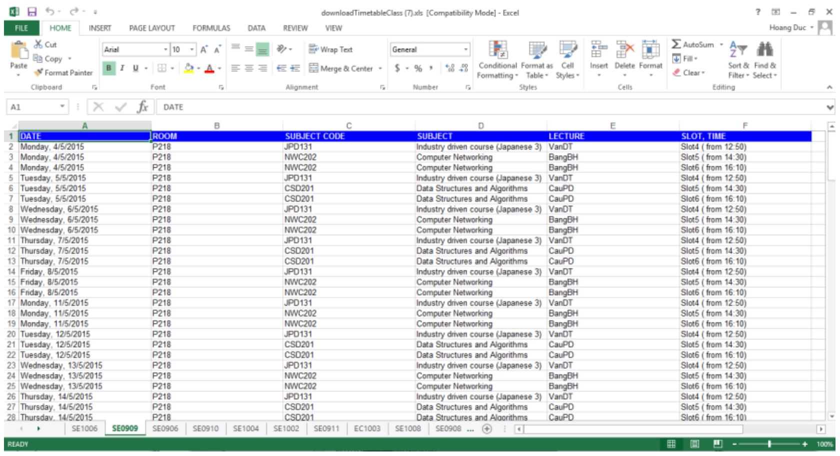
August 27, 2015 3:43:35

Data result for ClassCourses of Summer 2015

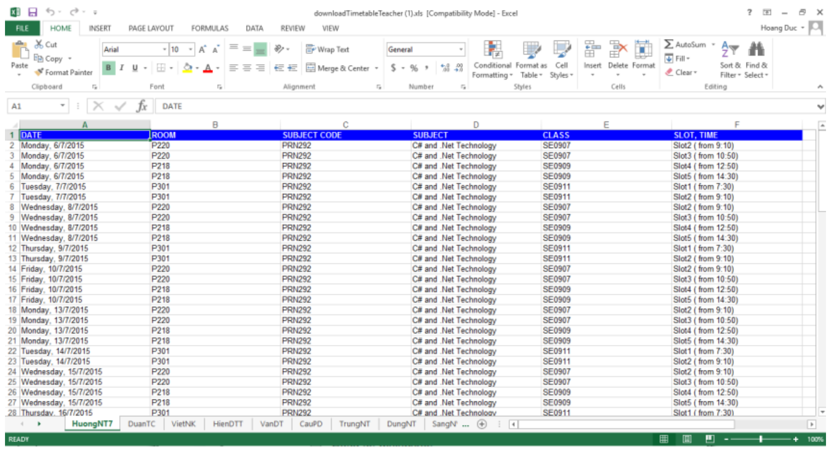
Set Timetable:	134
Set Room:	131
Set Teacher:	117

6. Choose download timetable group by class or teacher


Download


downloadTimetableClass (7).xls [Compatibility Mode] - Excel

	DATE	ROOM	SUBJECT CODE	SUBJECT	LECTURE	SLOT, TIME
2	Monday, 4/5/2015	P218	JPD131	Industry driven course (Japanese 3)	VanDT	Slot4 (from 12:50)
3	Monday, 4/5/2015	P218	NWC202	Computer Networking	BangBH	Slot5 (from 14:30)
4	Monday, 4/5/2015	P218	NWC202	Computer Networking	BangBH	Slot6 (from 16:10)
5	Tuesday, 5/5/2015	P218	JPD131	Industry driven course (Japanese 3)	VanDT	Slot4 (from 12:50)
6	Tuesday, 5/5/2015	P218	CSD201	Data Structures and Algorithms	CauPD	Slot5 (from 14:30)
7	Tuesday, 5/5/2015	P218	CSD201	Data Structures and Algorithms	CauPD	Slot6 (from 16:10)
8	Wednesday, 6/5/2015	P218	JPD131	Industry driven course (Japanese 3)	VanDT	Slot4 (from 12:50)
9	Wednesday, 6/5/2015	P218	NWC202	Computer Networking	BangBH	Slot5 (from 14:30)
10	Wednesday, 6/5/2015	P218	NWC202	Computer Networking	BangBH	Slot6 (from 16:10)
11	Thursday, 7/5/2015	P218	JPD131	Industry driven course (Japanese 3)	VanDT	Slot4 (from 12:50)
12	Thursday, 7/5/2015	P218	CSD201	Data Structures and Algorithms	CauPD	Slot5 (from 14:30)
13	Thursday, 7/5/2015	P218	CSD201	Data Structures and Algorithms	CauPD	Slot6 (from 16:10)
14	Friday, 8/5/2015	P218	JPD131	Industry driven course (Japanese 3)	VanDT	Slot4 (from 12:50)
15	Friday, 8/5/2015	P218	NWC202	Computer Networking	BangBH	Slot5 (from 14:30)
16	Friday, 8/5/2015	P218	NWC202	Computer Networking	BangBH	Slot6 (from 16:10)
17	Monday, 11/5/2015	P218	JPD131	Industry driven course (Japanese 3)	VanDT	Slot4 (from 12:50)
18	Monday, 11/5/2015	P218	NWC202	Computer Networking	BangBH	Slot5 (from 14:30)
19	Monday, 11/5/2015	P218	NWC202	Computer Networking	BangBH	Slot6 (from 16:10)
20	Tuesday, 12/5/2015	P218	JPD131	Industry driven course (Japanese 3)	VanDT	Slot4 (from 12:50)
21	Tuesday, 12/5/2015	P218	CSD201	Data Structures and Algorithms	CauPD	Slot5 (from 14:30)
22	Tuesday, 12/5/2015	P218	CSD201	Data Structures and Algorithms	CauPD	Slot6 (from 16:10)
23	Wednesday, 13/5/2015	P218	JPD131	Industry driven course (Japanese 3)	VanDT	Slot4 (from 12:50)
24	Wednesday, 13/5/2015	P218	NWC202	Computer Networking	BangBH	Slot5 (from 14:30)
25	Wednesday, 13/5/2015	P218	NWC202	Computer Networking	BangBH	Slot6 (from 16:10)
26	Thursday, 14/5/2015	P218	JPD131	Industry driven course (Japanese 3)	VanDT	Slot4 (from 12:50)
27	Thursday, 14/5/2015	P218	CSD201	Data Structures and Algorithms	CauPD	Slot5 (from 14:30)
28	Thursday, 14/5/2015	P218	CSD201	Data Structures and Algorithms	CauPD	Slot6 (from 16:10)


downloadTimetableTeacher (1).xls [Compatibility Mode] - Excel

	DATE	ROOM	SUBJECT CODE	SUBJECT	CLASS	SLOT, TIME
2	Monday, 6/7/2015	P220	PRN292	C# and .Net Technology	SE0907	Slot2 (from 9:10)
3	Monday, 6/7/2015	P220	PRN292	C# and .Net Technology	SE0907	Slot3 (from 10:50)
4	Monday, 6/7/2015	P218	PRN292	C# and .Net Technology	SE0909	Slot4 (from 12:50)
5	Monday, 6/7/2015	P218	PRN292	C# and .Net Technology	SE0909	Slot5 (from 14:30)
6	Tuesday, 7/7/2015	P301	PRN292	C# and .Net Technology	SE0911	Slot1 (from 7:30)
7	Tuesday, 7/7/2015	P301	PRN292	C# and .Net Technology	SE0911	Slot2 (from 9:10)
8	Wednesday, 8/7/2015	P220	PRN292	C# and .Net Technology	SE0907	Slot2 (from 9:10)
9	Wednesday, 8/7/2015	P220	PRN292	C# and .Net Technology	SE0907	Slot3 (from 10:50)
10	Wednesday, 8/7/2015	P218	PRN292	C# and .Net Technology	SE0909	Slot4 (from 12:50)
11	Wednesday, 8/7/2015	P218	PRN292	C# and .Net Technology	SE0909	Slot5 (from 14:30)
12	Thursday, 9/7/2015	P301	PRN292	C# and .Net Technology	SE0911	Slot1 (from 7:30)
13	Thursday, 9/7/2015	P301	PRN292	C# and .Net Technology	SE0911	Slot2 (from 9:10)
14	Friday, 10/7/2015	P220	PRN292	C# and .Net Technology	SE0907	Slot2 (from 9:10)
15	Friday, 10/7/2015	P220	PRN292	C# and .Net Technology	SE0907	Slot3 (from 10:50)
16	Friday, 10/7/2015	P218	PRN292	C# and .Net Technology	SE0909	Slot4 (from 12:50)
17	Friday, 10/7/2015	P218	PRN292	C# and .Net Technology	SE0909	Slot5 (from 14:30)
18	Monday, 13/7/2015	P220	PRN292	C# and .Net Technology	SE0907	Slot2 (from 9:10)
19	Monday, 13/7/2015	P220	PRN292	C# and .Net Technology	SE0907	Slot3 (from 10:50)
20	Monday, 13/7/2015	P218	PRN292	C# and .Net Technology	SE0909	Slot4 (from 12:50)
21	Monday, 13/7/2015	P218	PRN292	C# and .Net Technology	SE0909	Slot5 (from 14:30)
22	Tuesday, 14/7/2015	P301	PRN292	C# and .Net Technology	SE0911	Slot1 (from 7:30)
23	Tuesday, 14/7/2015	P301	PRN292	C# and .Net Technology	SE0911	Slot2 (from 9:10)
24	Wednesday, 15/7/2015	P220	PRN292	C# and .Net Technology	SE0907	Slot2 (from 9:10)
25	Wednesday, 15/7/2015	P220	PRN292	C# and .Net Technology	SE0907	Slot3 (from 10:50)
26	Wednesday, 15/7/2015	P218	PRN292	C# and .Net Technology	SE0909	Slot4 (from 12:50)
27	Wednesday, 15/7/2015	P218	PRN292	C# and .Net Technology	SE0909	Slot5 (from 14:30)
28	Thursday, 16/7/2015	P301	PRN292	C# and .Net Technology	SE0911	Slot1 (from 7:30)