

Style Guidelines for Final Year Project ReportsFinal Year Project Reports

Solving the timetabling problem at FPT University

A 4th Year StudentLuu Thanh Cong  
Nguyen Van Hoai  
Nguyen Son Ha  
Hoang Minh Duc

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

Hoa Lac campus - FPT University  
24 August 2015

I confirms 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.

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 expriments on real data 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 |

Table of Contents

[Chapter 1: Introduction 6](#_Toc428142614)

[1.1 Overview 6](#_Toc428142615)

[1.2 Problem description 7](#_Toc428142616)

[Chapter 2: Problem formulation 9](#_Toc428142617)

[2.1 Input 10](#_Toc428142618)

[2.2 Variables 10](#_Toc428142619)

[2.3 Invariants 11](#_Toc428142620)

[2.4 Constraints 11](#_Toc428142621)

[2.5 Objectives function 11](#_Toc428142622)

[Chapter 3: Review of Literature for Solution 12](#_Toc428142623)

[3.1 Approaches to Automated Timetabling 12](#_Toc428142624)

[3.1.1 Sequential Methods 12](#_Toc428142625)

[3.1.2 Cluster Methods 13](#_Toc428142626)

[3.1.3 Constraint Programming 13](#_Toc428142627)

[3.1.4 Meta-heuristic Methods 13](#_Toc428142628)

[3.2 Local Search Approaches 14](#_Toc428142629)

[3.2.1 Hill-climbing algorithm 14](#_Toc428142630)

[3.2.2 Min-conflicts algorithm 14](#_Toc428142631)

[3.2.3 Tabu search 14](#_Toc428142632)

[3.2.4 Genetic algorithm 15](#_Toc428142633)

[3.3 Hybrid Approaches 15](#_Toc428142634)

[Chapter 4: Solutions 16](#_Toc428142635)

[4.1 Algorithm 16](#_Toc428142636)

[4.1.1 Preprocess: Merge class-course 16](#_Toc428142637)

[4.1.2 Phase 1: Find local timetables 19](#_Toc428142638)

[4.1.3 Phase 2: Find global timetables 20](#_Toc428142639)

[4.1.4 Phase 3: Assign teachers 22](#_Toc428142640)

[4.1.5 Phase 4: Assign rooms 25](#_Toc428142641)

[4.2 Experiments 26](#_Toc428142642)

[Chapter 5: Conclusion and further work 29](#_Toc428142643)

[Chapter 6: References 30](#_Toc428142644)

[Chapter 7: Appendix - Program Code 34](#_Toc428142645)

[Chapter 8: Appendix - Other if needed 35](#_Toc428142646)

# Chapter 1: Introduction

## Overview

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 that day, some student have to study in first slot and last slot of 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 problem in combinatorial optimization. There are many variants of the problem depending on specific requirements and policy of education 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 schedule 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 recomputing the timetable minimizing perturbation. In 2013, Muller considered a particular examination timetabling problem which was common at many American universites. 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 {c1…cm}, *n* teachers {t1,…tn}, and p periods {1…p}. A requirement matrix *r* is given in which ri,j indicates the number of lecturers that teacher tj must give to class ci. Two other matrices representing the unavailabilities of of teachers, classes in each period. The goal is to assign xi,j,k = 1 if the teacher ti is assigned to class cj in period *k*, , j , k, and xi,j,k = 0, otherwise. The assignment must satisfy the constraints on 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 having been investigated for solving timetabling problems are heuristics and metaheuristics, 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 neighborhood search algorithm for solving an exam timetabling problem which consists of scheduling a set of exams E = {e1…en} over a collection of time periods P = {p1,…pm}. 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 neighborhood for this problem. In this framework, a neighboring solution of a current solution cor responds to a cycle ei1, ei2,…eik. ei1 in which we move ei1 from its current set to the set of ei2 , then, move ei2 from its current set of the set of ei3 ,..., and finally, move eik to the set of ei1 . The size of the underline neighborhood is exponential of the size of the problem. A efficient heuristics has been applied to find a negative cost cycle on the improvement graph, thus to find quickly an improving neighboring 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 courses of a class are organised in a week.

## Problem description

**Related concepts:**

* ***Class-course***: A course belongs to only one class. A class-course identifies which class studying and which subject being studied.
* ***Semester template***: Is how all courses of a class are organized in a week. Templates of two semesters may be different depending on the policies of administration. One semester may have one or more than one template.

It is very difficult to formulate a general model that is applicable for all cases since every universities or other educational institutions have their own specical 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. Due to specific requirements and the policy of the university (for example, to avoid as much as possible the situation that one class-course must be scheduled in two blocks, and one class-course is scheduled in two consecutive days of a week), the staffs define templates for allocating a class-course as illustrated in Figures 1-6. For instance, in 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 1 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)

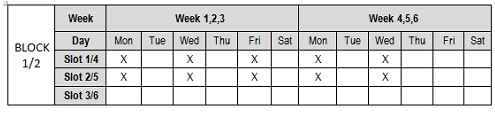


Figure 1: Template 1 for class-courses allocated to one block.

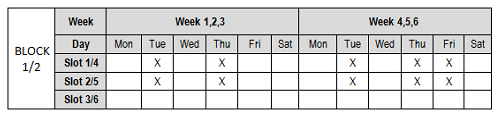


Figure 2: Template 2 for class-courses allocated to one block .

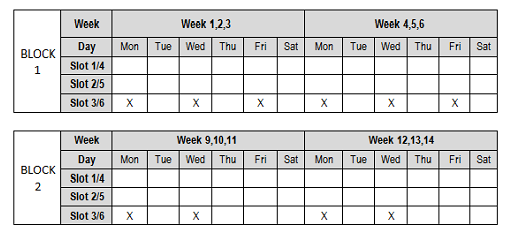


Figure 3: Template 3 for class-courses allocated to two blocks .

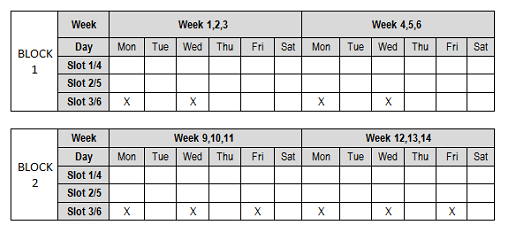


Figure 4: Template 4 for class-courses allocated to two blocks.

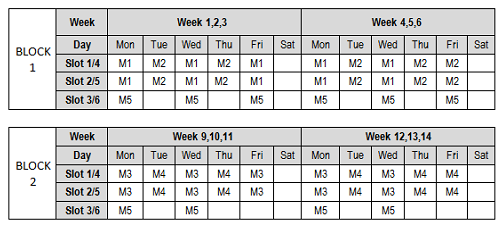


Figure 5: Example of timetable of one class with 5 class-courses.

# Chapter 2: 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.

## Input

* A set of courses *C = {1, . . . , M}*
* A set of classes *CL = {1, . . . , K}*
* A set of class-courses *CC = {1, . . . , N},* for each class-course *i ∈ CC*,
  + *c(i) ∈ C* is the course of the class-course *i*
  + *cl(i) ∈ CL* is the class following the class-course *i*
  + *CC(k)* ⊂ *CC* is set of class-courses of class k.
  + *CCL(j)* ⊂ *CC* is set of class-courses of course *j, j ∈ 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 ∈ CL, crs(k) = {j ∈ C |* ∃*i ∈ CC c(i) = j}*: set of courses the class *k* has to follow in the semester
* For each class *k ∈ 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 ∈ 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,* ∀*p ∈ P*
* *w[i]:* number of student of classcourse *i, i ∈ CC.*
* A set of teachers: *T = {1,.., NT}*;
* *TL(c):* list of teachers who can instruct course *c, c ∊ C, TL(c) T*.
* Given a set of room: *RO = {1, ..., NR}*.

## Variables

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

## Invariants

* *cf(i,j) =* **true** if and only if *cfp(x\_p(i),x\_p(j))* = **true**, ∀*i,j ∈ CC*. *cf(i,j) =* **true** indicates two class-courses conflict with each other.

## Constraints

* *cl(i)* = *cl(j)* → *s(x\_p(i))* = *s(x\_p(j))* ∀*i,j ∈ CC* (1)
* *diff(i, j)* = **true** → *cfp(x\_p(i), x\_p(j))* = **false**,∀*i,j ∈ CC* (2)
* *x\_p(j) = p* → *occ(j, z)* = 1, ∀*p ∈ P*, *z* ∈ *CELL(p)* (3)
* *oc(i, z)* = (4)
* *cf(i,j)* = **true** → *x\_t[i] ≠ x\_t[j]* (5)
* *cf(i,j) =* **true →** *x\_r[i] ≠ 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 classcourses conflict with each other, they must not assigned to the same teacher and the same room.

## Objectives function

The objective function to minimized is

(7)

# Chapter 3: Review of Literature for Solution

## 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]): sequential methods, cluster methods, constraint-based methods, and meta-heuristic methods.

### Sequential Methods[[1]](#footnote-1)

Sequential methods order courses using domain heuristic and then assigning the courses sequentially into valid time periods so that no courses in the period are inconflict 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-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 course 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.

### Cluster 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.

### Constraint Programming

Constraint Programming (CP) [19] is an approach for solving combinatorial optimization problems. The CP framework is an exact approach 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 partition the original problem into smaller equivalent subproblems (the union of the sets of solutions to subproblems 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 subproblem. 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 built-in search procedures for finding solutions. Using this generic approach has an advantage that allow users to model flexibly different problems as well as to adapt the existing model when additional constraints are added. Choco Solver is an Open-Source Library dedicated to Constraint Programming that we used in this project.

### 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

## 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 begin with one or more initial solutions and employ 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 neighbors. The neighborhood 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.

### Hill-climbing algorithm

Hill-climbing[27] algorithm is base 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.

### 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 an 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 can not move at all, and does not terminate. So we have to limit the number of interation 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 applies the min-conflict heuristic with probability 1-P. The same strategy can be used in Hill-climbing also.

### 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 candicate s’. Tabu search avoid the iterations that does not improve the object 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 considerd in ***l*** interations (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.

### Genetic algorithm

Genetic algorithm (GA)[31] is based on the idea of natural selection. This algorithm maintain a populations of candidate solutions. The candidate solutions have probability to make their child candidate solution depend on their fitness. Fitness is evaluated by a 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.

## Hybrid Approaches

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 literatural review and testing on some algorithm, we consider using hybird 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 optimize the object function.

## Algorithm

In this section, we propose an algorithm for solving 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 classcourse 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 *pi* ∊ *P* to each class-course *i* of *CC(k)* such that *pi ≠ pj* and *s(pi) = s(pj),* *i*,*j* ∊ *CC(k).* The output of the first phase, we obtain a set *S(k)* of local-timetables for each class *k*; *k* ∊ *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.

### Preprocess: Merge 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 ≤ weight ≤ 30). ***We will do pre-processing for each course m ∈ C****.*

**Problem statement for each course *m*:**

**Input**:

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

**Variables**:

* *x\_c[i]* represents class-course which class-course *i* will be join in, domain of *x\_c[i]*, *i ∈ CCL(m)*.
* Auxiliary variables*:*
  + *b*: is number of expected classcourses of course m.
  + *min*: lower bound of *b*. *min* = ⌈ (/30⌉;
  + *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)* = where *x\_c[i] = j, j ∈ 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 of violation of all *x\_c[i],* *i ∈ CCL(m).*

**Constraints**:

* 15 ≤ *st(j)* ≤ 30, *j ∈ 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

We solve merging classcourse problem using TabuSearch. We denote:

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

**Algorithm:**

|  |
| --- |
|  |
| *Algorithm 1: FindOptimalSolution()* |

|  |
| --- |
|  |
| *Algorithm 2: InitRandomSolution()* |

|  |
| --- |
|  |
| *Algorithm 3: FindFeasibleSolutionUsingTabuSearch()* |

### Phase 1: Find local timetables

In this phase we apply an exact methods for finding 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****, i,j ∊ 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).*

### Phase 2: Find global timetables

**Problem statement:**

**Input**:

* *S(k)*: set of local-timetable for class *k, k ∊ CL*.

**Output**:

* Best global-timetable *s\* = (s1,...,sK):* an array to model a global-timetable (timetable for all classes) in which *sk ∊ S(k)* represents the local-timetable selected for class *k.*

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

* *s = (s1,...,sK):* represents a global-timetable.
* *s\* = (s1,...,sK):* 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)* = **true** Λ *x\_p(i)* = *x\_p(j).*

* *f\** represents the best value of *f(s).*
* *s*[sk ← *v*] the array *s* in which *sk*is reassigned to *v*:
* *s*[*sk ← v*] = (*s1,. . ., sk-1*, *v*, *sk+1,… ,sK*).

**Algorithm :**

|  |
| --- |
| **Input**: *S(k)* is the set of local-timetables for class *k*, *k ∊ CL*  **Output**: A random global local-timetable |
| *Algorithm 4 : InitRandom()* |

|  |
| --- |
| **Input**: *S(k)* is the set of local-timetables for class *k, k ∊ CL*  **Output**: A random feasible global-timetable |
| *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 |
| *Algorithm 6 : InitFeasible()* |

|  |
| --- |
| **Input**:   * *S(k)* is the set of local-timetables for class *k*; *k ∊ 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\** |
| *Algorithm 7: TabuSearch()* |

### Phase 3: Assign teachers

**Problem statement:**

**Input**:

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

**Variables**:

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

**Invariants**:

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

**Constraints**:

* *cf(i, j)* = **true** → *x\_t[i] ≠ x\_t[j]*.
* *AS(t)*: subset of *CC*, is a list of classcourses that have been assigned to teacher *t* *∊* *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)* *∊* *R*, indicates the suitability between teacher *t* *∊* *TL(c(cc))* and classcourse *cc* *∊ CC*. *suitableScore(t,cc)* is calculated based on *relativeScore(cc,cc’)*, *cc’* *∊* *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 classcourses 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 classcourses at slot 3 and 4 consecutively on the same day.

|  |
| --- |
|  |
| *Algorithm 8: AssigningTeacher* |

|  |
| --- |
|  |
| *Algorithm 9: IsFeasibleToAssign(t,cc)* |

|  |
| --- |
|  |
| *Algorithm 10: calculateMinTarget(t)* |

### Phase 4: Assign rooms

**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**}, repsents 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

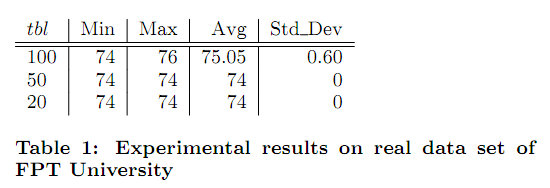
**Algorithm :**

|  |
| --- |
|  |
| *Algorithm 10: calculateMinTarget(t)* |

## Experiments

We conduct an experiment on a real data of the first semester in 2015 [[2]](#footnote-2)with different values of tabu tenure *tbl*. The data consists of unknow classes, unknow courses, and unknow class-courses. For this data, the timetable was made manually and the value of the objective function (7) is 91.

For each value of *tbl*, we execute the program 20 times and report the minimum, the maximum, the average, and the standard deviation of the best objective value found among 20 times. The experimental results with 3 values of *tbl* are presented in Table 4. The table shows that the solution computed by our proposed algorithm is better than that made by human. The performance is stable and good for the value 10, 20 of *tbl*. Figure 4 presents the best objective values found in 10 last executions. Figure 4 shows the behaviour of the tabu search.



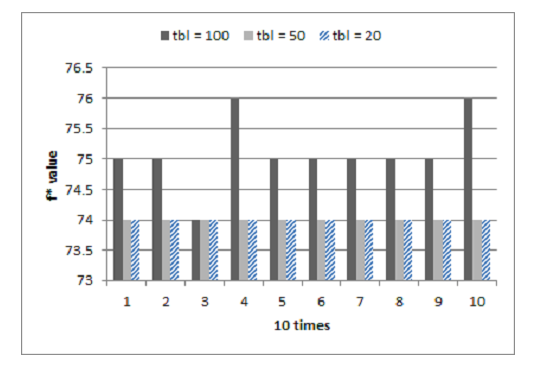


Figure 6: Objective values of 10 executions

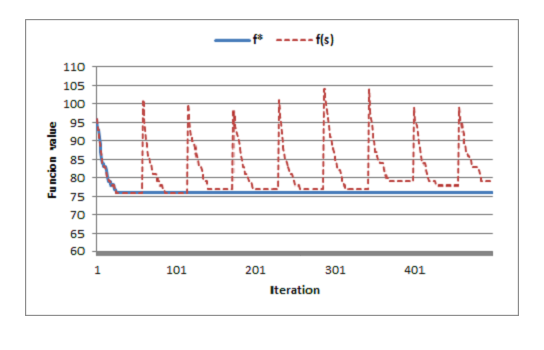


Figure 7: Behaviour of the tabu search

# Chapter 5: Conclusion and further work

In this project, we produced a web service that support Academic Department in scheduling, class arrangement, merging class-course, handle teacher, student and resources by manual. The most important automatic scheduling is achieved some positive results. Generated timetable satisfy the hard condition: no conflict resources was founded. It also make profit by reducing the resource cost( about teacher, room) and increasing teachers productivity. We presented a new hybird approach to solve the specific case of FPT University. Our system is currently being introduced in FPT Academic Department. The results are quite appealing and method used shows good promise for the future. With the limited time, knowledge and man power, our automated scheduling can not cover all conditions of Academic Department, some specific case of small group of student is missing. We propose a bias timetable automatically then user can manual it as they want. The future work should concentrate on dealing with other conditions, improve the usability and add some optimize function. We should find a way to improve diversity of final population and coverage of search space. Finally, more test data sets have to be obtained in oder to prove our method.

# Chapter 6: References

|  |  |
| --- | --- |
| [1] | University TimeTabling, http://www.unitime.org/. |
| [2]  [29]  [32] | 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]  [13] | M. W. Carter. A Survey of Practical Applications of Examination Timetabling Algorithms. Operations Research 34, p193-202, 1986. |
| [12]  [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 |
| [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 |
| [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. |
| [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. |
| [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. |

# Chapter 7: Appendix - Program Code

# Chapter 8: Appendix - Other if needed

1. This method has another name is Greedy Constructive Search. [↑](#footnote-ref-1)
2. The new policy with two blocks described above has just applied this year, so there is only one data. [↑](#footnote-ref-2)