

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.) in Computer Science and Software Engineering with the supervision of   
Mr. Prof. Barry SmythNguyen Tat Trung & Mr. Pham Quang Dung

Hoa Lac campus - FPT University  
15 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 have 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 produce a service that can automatic generate a better quality timetable within acceptable time. Our web service also provide 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

[1 Introduction 6](#_Toc427416778)

[1.1 Overview 6](#_Toc427416779)

[1.2 Problem description 7](#_Toc427416780)

[2 Problem formal formulation 9](#_Toc427416781)

[2.1 Input 9](#_Toc427416782)

[2.2 Variables 10](#_Toc427416783)

[2.3 Constraints 10](#_Toc427416784)

[2.4 Objectives function 10](#_Toc427416785)

[3 Review of Literature for Solution 11](#_Toc427416786)

[3.1 Approaches to Automated Timetabling 11](#_Toc427416787)

[3.1.1 Sequential Methods 11](#_Toc427416788)

[3.1.2 Cluster Methods 11](#_Toc427416789)

[3.1.3 Constraint Based Approaches 11](#_Toc427416790)

[3.1.4 Meta-heuristic Methods 11](#_Toc427416791)

[3.2 Local Search Approaches 11](#_Toc427416792)

[3.2.1 Hill-climbing algorithm 11](#_Toc427416793)

[3.2.2 Min-conflicts algorithm 12](#_Toc427416794)

[3.2.3 Min-conflict Random Walk Algorithm 12](#_Toc427416795)

[3.2.4 Tabu search 12](#_Toc427416796)

[3.2.5 Genetic algorithm 12](#_Toc427416797)

[3.3 Hybrid Approaches 12](#_Toc427416798)

[3.3.1 Decision Repair Algorithm 13](#_Toc427416799)

[3.3.2 Constrained Local Search Algorithm 13](#_Toc427416800)

[3.3.3 Constructive Backtracking-free Algorithm[16] 14](#_Toc427416801)

[4 Solutions 14](#_Toc427416802)

[4.1 Algorithm 14](#_Toc427416803)

[4.1.1 Initialize timetable 15](#_Toc427416804)

[4.1.2 Find an optimized timetable 15](#_Toc427416805)

[4.1.3 Assign teachers 19](#_Toc427416806)

[4.1.4 Assign rooms 19](#_Toc427416807)

[4.2 Experiments 19](#_Toc427416808)

[5 Conclusion 21](#_Toc427416809)

[6 References 21](#_Toc427416810)

[7 Appendix - Program Code 22](#_Toc427416811)

[8 Appendix - Other if needed 22](#_Toc427416812)

# Introduction

## Overview

Spring 2015, many students and teachers in FPT University complain that the timetable has problems. They do not have enough teacher for some slot, 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 stressing: it takes many times and cannot avoid mistakes (example...) 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 [4], [5], [7], [6], [3], [8], [9], [2] (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). A hybrid algorithm with several phases were also proposed for solving this problem. In the case of FPT University, the academic department will make a timetable basing 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. This report is organized as follow:

Section 1: Introduction.

Section 2: Formal fomulation of the course scheduling problem(CSP) at FPT University

Section 3: Literature review of current works on CSP.

Section 4: Describe proposed solution to the problem and validation on the solution.

Section 5: Conclusion

Section 6: References

Section 7: Appendix - Program Code

Section 8: Appendix - Other

## Problem description

**Some 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)

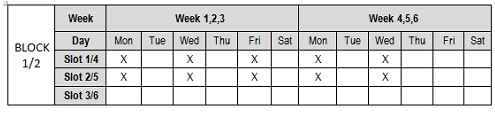


Figure 1: Template 1 for class-courses allocated to one block.

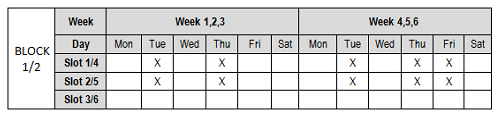


Figure 2: Template 2 for class-courses allocated to one block .

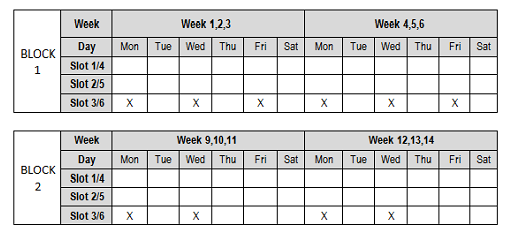


Figure 3: Template 3 for class-courses allocated to two blocks .

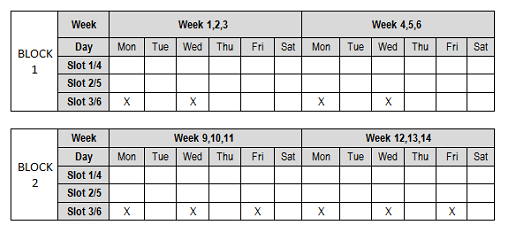


Figure 4: Template 4 for class-courses allocated to two blocks.

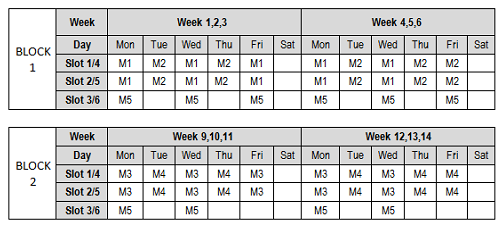


Figure 5: Example of timetable of one class with 5 class-courses.

# Problem formal 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 submit 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.

## 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*
* For each pair of class-courses *i* and *j*, *cf(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*
  + *cf p(p1, p2)* = true if place *p1* and *p2* share a common cell.
  + CELL*(p)* = the set of cells of the place *p,* ∀*p* ∈ P

## Variables

* *x(i)* represents the place to which the class-course *i* is scheduled, the domain of *x(i)* is *P,* ∀*i ∈ 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*

## Constraints

* *cl(i)* = *cl(j)* ⇒ *s(x(i))* = *s(x(j))* (1)
* *cf(i, j)* = **true** ⇒ *cf p(x(i), x(j))* = **true**(2)
* *x(j) = p* ⇒ *occ(j, z)* = 1, ∀*p ∈ P*, *z* ∈ CELL*(p)* (3)
* *oc(i, z)* = (4)
* *moc(i)* = maxz∈CELL{*oc(i, z)*} (5)

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.

## Objectives function

The objective function to minimized is

(6)

# Review of Literature for Solution

## Approaches to Automated Timetabling

In general, problem-specific heuristic methods can make good timetables, but the size and complexity of modern university timetabling is big problems. It require general problem solving algorithms, or metaheuristics algorithm, such as simulated annealing, genetic algorithms, and tabu search. Problem-specific heuristics is used to reduce the number of candicates, or to locally optimise a solution. Constraint Logic Programming also is another popular approach.

### Sequential Methods

Sequential methods order events using domain heuristic and then assign the events sequentially into valid time periods so that no events in the period are inconflict with each other[1]. Timetabling problems are usually represented by a graph where events are vertices, conflicts between the events are edges. 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.

### Cluster Methods

In cluster methods, the set of events is split into groups which satisy all hard constraints and then the groups are assigned to time periods to fulfil the soft constraint [2]. Many optimisation techniques have been implemented to solve the assignment problem. 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 Based Approaches

In these methods, the problem is modelled as a set of variables. Assigned values have to satisfy a number of constraints [3][4][5][6] . A number of rules is defined for assigning value to variable. If it has no rule, a backtracking search is performed until a solution is found that satisfies all constraints.

### Meta-heuristic Methods

Many meta-heuristic approaches such as tabu search, simulated annealing, Hill-climbing, genetic algorithms and hybrid approaches have been implemented for timetabling. We will mention these method more detail in section 3.2 and section 3.3.

## Local Search Approaches

Local search is a metaheuristic method for solving computationally hard optimization problems. It base on general and simple idea. Suppose that we have a combinatorial optimization problem **P**, current candicate solution ***s***, cost funtion ***f(s)*** . N(***s)*** is space of candicate solution. From solution ***s*** we can ***move*** to the neighbor candicate solution ***s’*** by a ***move*** ***m.*** This process will repeat until we find a solution maximizing a criterion. This solution is close to optimal solution.

### Hill-climbing algorithm

Hill-climbing[7] algorithm is base of other local search technique. Although this algorithm is simple it indicate strong and effective in large constraint satisfaction problems. Hill-climbing always selects the best candicate of all neighbours which minimizes the violation or the objective function. If there is no better solution, the search stuck in a local optimum. It usually restarts the search from another initial candicate. The name of this algorithm means increasing the evalutation value each step by climbing.

### Min-conflicts algorithm

Min-conflicts algorithm (MC) [8] is a local search algorithm, a heuristic method to solve constraint satisfaction problems. MC algorithm chooses the best candicate only from a subset of the neighbour candicates. A variable that 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.

### Min-conflict Random Walk Algorithm

MC Random Walk algorithm[9] 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 [10] is a method to avoid getting trapped in a local minimum. A tabu list, which is a special short term memory store previous configurations. Because it require a large memory space to record all status of previous configurations, so we only record some properties of previous solution in tabu list, and all solutions in this list are banned, 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 optima. Tabu search is improved to reactive tabu search and tabu search with two tabu lists.

### Genetic algorithm

Genetic algorithm (GA)[11] base on the idea of natural selection. This algorithm maintain a populations of candicate solutions. The candicate solutions have probability to make their child candicate solution depending 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[12]:

* 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

### Decision Repair Algorithm

Decision repair algorithm is presented in falls into the third category above.

***Procedure*** *decision-repair(V,D,C)*

*//a CSP problem is the parameter*

*CD = any initial set of decisions;*

*//decisions are constraints as well*

***while*** *conditions of failure not satisfied* ***do***

*C’ = C ∪ CD;*

***switch*** *obviousInferences(Ф(V,D,C’))*

***case*** *no solution:*

*k = conflict explaining the failure;*

*CD = neighbour(CD, k);*

***case*** *solution:*

*return C’;*

***default****:*

*CD = extend(CD);*

***end******switch***

***end******while***

***return*** *failure;*

***end procedure***

*The DR Algorithm [13]*

The decision-repair algorithm starts with a partial solution (CD) . It first applies a filtering technique Ф. When no inconsistency is detected, we adds a decision that extends CD, and the search continues (case default). When we find no solution, the algorithm tries to repair that set of decisions. A conflict is identified (k) better than current conflict, and the conflict is used to choose a neighbour of the current set of decisions. (neighbour function). The function obviousInference is able to examine a set of constraints in order to decide whether to stop the computation or not.

### Constrained Local Search Algorithm

The Constrained Local Search algorithm [14] is constructed by randomizing the backtracking component of a systematic search algorithm, that is allowing backtracking to occur on arbitrary chosen variables. An integer parameter called the noise level(*ε)* is the number of backtracked variables (selected by backtrackVars function).

***Procedure*** *cls(V,D,C,ε) // ε is the noise level*

*σ = {}; //current assignment*

***while*** *σ is not complete* ***do***

*assigned = {A∈V | A assigned in σ}*

*unassigned = V – assigned;*

*A = selectVariable(unassigned);*

*values = {a∈DA; σ ∪ {A/a} is consistent};*

***if*** *(values is empty)* ***then***

***for*** *all v in backtrackVars(assigned,ε)* ***do***

*unassign v in σ and unpropagate;*

***end******for***

***else***

*a = selectValue(DA);*

*σ = σ ∪ {A/a};*

***end******if***

***end******while***

***return*** *σ;*

***end******procedure***

*The Constrained Local Search Algorithm [15]*

### Constructive Backtracking-free Algorithm[16]

The algorithm iteratively extends a feasible partial candicate until we cant find any solution. At this point, it performs a local search phase which makes local changes on the current partial solution. After that, the construction continues. The algorithm stops either when a complete solution is reached or when a predetermined number of local search phases have been reached.

***Procedure*** *cbf(V,D,C)*

*σ = {}; //current assignment*

***while*** *σ is not complete* ***do***

*assigned = {A∈V | A assigned in σ}*

*unassigned = V – assigned;*

*A = selectVariable(unassigned);*

*values = {a∈DA; σ ∪ {A/a} is consistent};*

***if*** *(values is empty)* ***then***

***if*** *last trial* ***then******return*** *failure;*

*β = σ;*

***repeat***

*move = selectMove(σ);*

*makeMove(move, σ);*

*if improves(σ, β) then β = σ;*

***until*** *last iteration or lower bound reached;*

*σ = β;*

***else***

*a = selectValue(values);*

*σ = σ ∪ {A/a}*

***end******if***

***end******while***

***return*** *σ;*

***end******procedure***

*The Constructive Backtracking-free algorithm[17]*

# Solutions

## Algorithm

In this section, we propose an algorithm for solving the problem stated. The algorithm consists of 4 phases. In the first phase, we generate all solutions for class-courses of each class. Given a class *k*, a solution 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 solution for each class *k*; *k* ∊ *CL*. In the second phase, we propose a tabu search algorithm for selecting a solution from *S(k)* for each class k such that the objective function (6) is minimal. In phase 3,4 we assign teachers and rooms for each class.

* Build timetable for each class
* Find an optimal aggregate timetable for all class
* Assign teachers
* Assign rooms

### Initialize timetable

Input:

List of classes, class-course

Template of distibution lecture

Output:

Timetable for 1 class

Section (morning or afternoon)

Task:

Modeling the template

Search solution for each class

Initialize session for each class

Explain: We model the problem as a constraint satisfaction problem with these integer arrays variable : day, slot, block, section, week. All constraint must limit the variable satisfied the template.

We use a backtracking search to find all acceptable solution and store in a list for the next step.

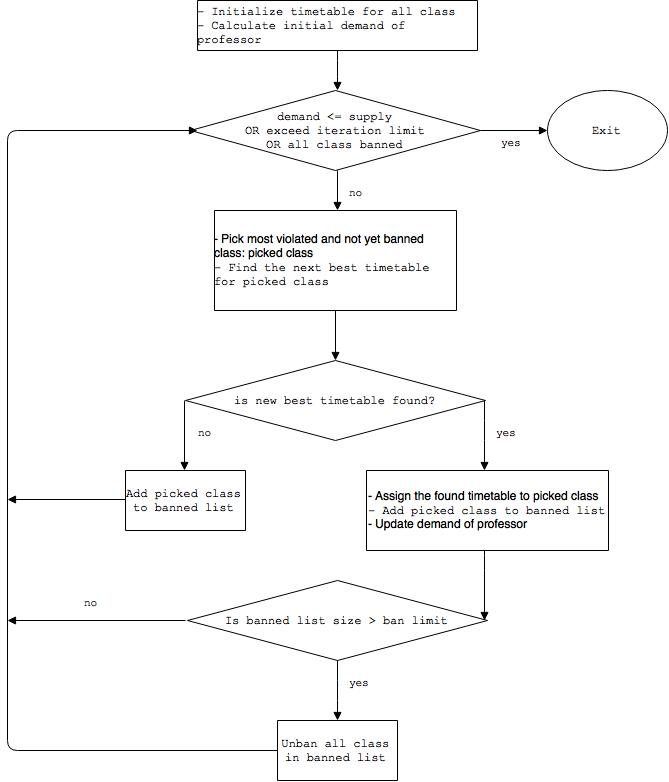
### Find an optimized timetable

Input: A list of aggregate timetables for all class.

A list of prospective teachers of each course

Output: An optimal aggregate timetable with minimum required teacher.

Task: Our task is presented in this figure:



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

* We use an array s = (s1, . . ., sK) to model a global solution (solution for all classes) in which sk ∊ *S(k)* represents the solution selected for class *k*.
* We denote *f(s)* is the objective function (6) and *g(s)* the number of violations of constraint (2), i.e., the number of pair of class-courses ‹ *i, j* › such that *cf( i , j)* = **true** *x(i)* = *x(j).*
* We denote *s*[sk ← *v*] the array *s* in which *sk*is reassigned to *v*:

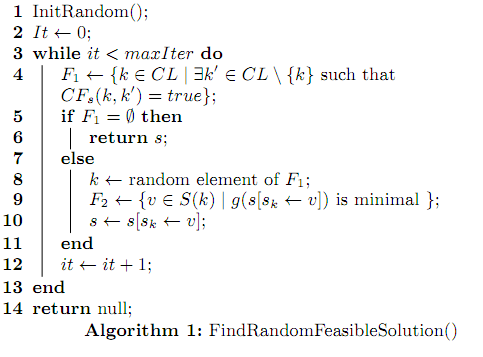
*s*[*sk ← v*] = (*s1,. . ., sk-1*, *v*, *sk+1,. . . ,sK*)

* *CFs(k1, k2)* = **true** if in solution *s*, there exists a class-course *i* of class *k1* and a class-course *j* of class *k2* such that *cf(i, j)* = **true** and *i, j* are scheduled in the same place.

#### Find random feasible global solution

**Input**: *S(k)* is the set of solutions for class *k, k ∊ CL*

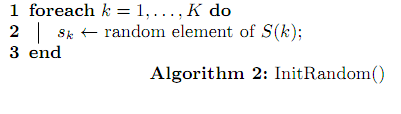
**Output**: A random feasible global solution



#### Initial random global soltion

**Input**: *S(k)* is the set of solutions for class *k*; *k ∊ CL*

**Output**: A random global solution



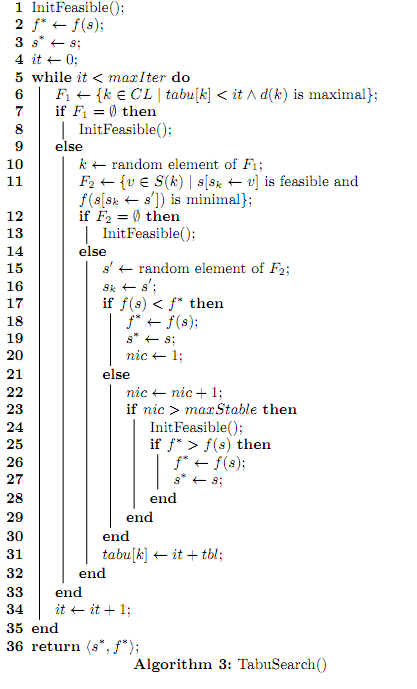
#### Tabu Search

**Input**: *S(k)* is the set of solutions 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 solution is not improved
  + maxStable: if the best solution is not improved after maxStable iterations, then the search is restarted

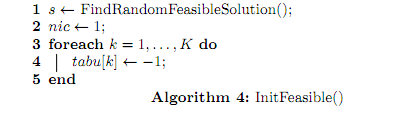
**Output**: Best global solution.



#### Initialize feasible solution and tabu list

**Input**: *S(k)* is the set of solutions for class *k*; *k ∊ CL*,

**Output**: Generate random global feasible solution and initialize tabu list



### Assign teachers

Input: An optimal timtable from previous step.

Conflict matrix get from this timetable

List of prospective teachers of each course.

Output: Which teacher is assigned to which class-course.

### Assign rooms

Input: Fixed-optimal timetable from previous step.

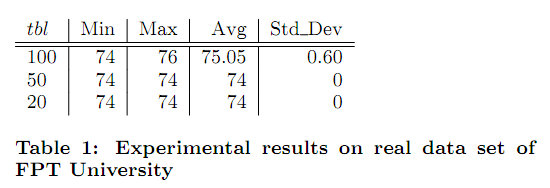
List of rooms (+ their buildings)

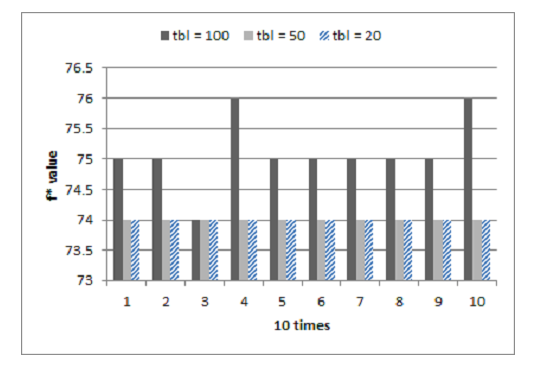
Output Which room is assigned to which class-course

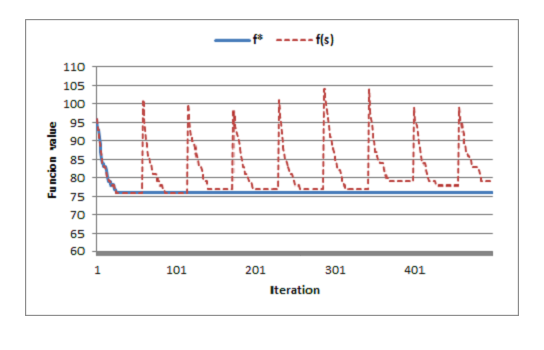
## Experiments

We conduct an experiment on a real data of the first semester in 2015 [[1]](#footnote-1)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 (6) 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.







# Conclusion

# 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] |  |
| [11] |  |
| [12] |  |
| [13] |  |
|  |  |

# Appendix - Program Code

# Appendix - Other if needed

1. The new policy with two blocks described above has just applied this year, so there is only one data. [↑](#footnote-ref-1)