Using Tabu Search Algorithm to Solve University Timetable Problem
使用禁忌算法求解大学课表排列问题

Final Report

Chongfeng Ling
1716474
Department of Applied Mathematics
Xi'an Jiaotong-Liverpool University
Chongfeng.Ling17@student.xjtlu.edu.cn

M.B.N. (Thijs) Kouwenhoven
Supervisor
Head, Department of Physics
Xi'an Jiaotong-Liverpool University
t.kouwenhoven@xjtlu.edu.cn

May 4, 2022

Abstract

The objective of this study is to give a general description and some examples to solve timetable problem in educational institutions. Due to the large scale number of elements and constraints, obtaining a feasible solution could be very time consuming. Since timetable problem can be turned into graph coloring problem, we introduce tabu search algorithm and adapted it to our model. The proposed method is evaluated using examples in XJTLU and Considering some constraints, this method can solve timetable problem efficiently.

Keywords Timetable Scheduling · Graph Coloring · Hyper-heuristic Algorithm · Tabu Search

摘要

这篇文章主要对大学课表的排列问题做了细致的阐述，并解决了几个现实中的例子。由于大量数据和约束条件的存在，得到一个排课问题的可行解是很耗时间的。考虑到排课问题可以转化为图着色问题，这篇文章介绍了禁忌搜索算法以及他在此问题上面的优化版本。通过在西交利物浦大学的几个例子，算法模型的可行性与有效性得到了验证。

Keywords 排课问题 · 图着色问题 · 超启发式算法 · 禁忌搜索算法

Contents

1 Introduction

Timetable scheduling is a practical problem with applications in several areas like transportation and plays an important roll in maintaining the orderly operation of society. In the educational institutions, a scheduled timetable is to assign students and teachers to a series of courses within a certain period of time and subject to several constraints and requirements including the capacity and types of classrooms. Every semester a university is supposed to develop timetables for teaching activities and examination to meeting students and staffs' requirements and school hardware source limitations. Finding an efficient method to make a timetable schedule terminally is necessary for all university.

Credit to pervious works, university timetable problem has been well defined as a combination optimization problem based on graph coloring and divided into two interrelated subproblem: course timetabling subproblem and stundent grouping subproblem(Hertz 1991). While a lot of studies has been spent on this topic, there is still a big gap between algorithm result and practical timetable (McCollum 2006) mainly due to various constraints of different university and large scale of data. Resources and roles to define constraints in university are unique and different with others. On the other hand, Consider these constraints with a large volume of data, timetable scheduling is a NP-hard or NP-complete problem (Even et al. 1975), which means it can not be solved in a polynomial time with some large scale problem data. This paper attempts to introduce and implement tabu search methodology with its modified version to solve the course timetabling problem in Xi'an Jiaotong-Liverpool University (XJTLU).

The structure of this paper is organized as follow. Section 2 is a review of the development of timetable scheduling including algorithms and software framework. Section 3 includes a general description of timetable problem in educational institutions and its representation by graoh theory. In section 4 and 5 we present methodologies including Tabu Search algorithm and one improved version called TABCOL. Section 6 will established the timetable in XJTLU based on above hyper-heuristic algorithm step by step. The discussion and conclusion will be contained in Section 8.

2 Literature Review

The university timetable problem is an application of combinatorial optimization to scheduling problems. Before the advent of heuristic algorithm, people solved this problem by some traditional algorithm like The Lagrangian relaxation method. Since the 1980s, the connection between timetabling problem and graph coloring has been established. With the invention of heuristic algorithms, these intelligent algorithms play an important role in the timetable problems gradually. Based on these effictive techniques and well-defined models, some software framework are developed to assign timetables for one or several universities. This section will mainly review some papers on the development of models and heuristic algorithm for the timetable problems and introduce some frameworks.

In 1975, A naive mathematical model was successfully established as description by Even et al. (1975). Though several important constraints was ignored in this simple model, this paper stated that timetable problem and all other common problems are NP-complete, which means we can hardly solve them in a polynomial time. Considering a teacher-course model with some preassigned conditions, Werra (1985) gave a formal way to represent this problem and provided formulations in both graph edge coloring and graph node coloring. Historically, this paper only implemented a lagrangean relaxation technique to solve problems and its representation of graph was not suitable for some heuristic methodology. However, since timetable problem required knowledge including combinatorial Mathematics and computer science, this paper pointed out the value of timetable problem and led the research development in the area. Based on the statement that a timetable problem can be change into a graph coloring problem, two years later, Hertz & Werra (1987) solved a large scale random graph coloring problem by tabu algorithm and the TABUCOL procedure that reduces number color from maximum value, compared with annealing algorithm, the CPU-time was much shorter and furthermore, for some unsolved graph, tabu algorithm could indicate the "bad" edges that needed to be reduced. The principles and detailed illustrations of tabu search were given by Werra & Hertz (1989). Tabu search is a type of metaalgorithm designed for finding a global optimum solution. As this paper introduced, in order to avoid entering into a circular loop that contains a set of local optimum solutions, this algorithm maintains a list of solutions that was obtained during past several iterations and will not back to these elements in the list as our global optimum solution generally. This algorithm was successfully implemented in a problem represented by a neural network, a data structure that is similiar to graph. Later in 1990, Glover (1990) mentioned that the advantage of tabu Ssarch compared with other meta-heuristic algorithm is owing to its long-term memory. Moreover, this paper offered five advice to guide people modified the general tabu search technique. By refining some strategies and processes like aspiration function and tabu list, we can generate several variations of tabu search technique and implement it efficiently under different constraints. Due to a great number of students involved in several courses, Hertz (1991) added another procedure called grouping subproblem to get the best grouping of students into sections for each course after assigning timetables. Tabu search technique was implemented in these two procedures and got a satisfied results in many practical applications including preassignment requirements, compactness constraints.Costa (1994) made a systematic description about timetable problem and mathematical formulations including elements in the university, definitions of the constraints and their categories based on the importance, mathematical expressions of the timetable problem. This paper turned the timetable problem into a combinatorial optimization problem but did not treat it as a graph coloring problem. a well-finied tabu search technique and its modified version were applied to find a global optimum and based on the comprehensive definition of problems, this model could work in a variety of assigning environments even though their constraints were quite different. However, mainly due to lack of standards and difficulty of measuring results, the value of a timetable algorithm was hard to judge. As a supplement to the above paper, Werra (1997) introduced the requirement matrix and graph coloring problem to model the above combinatorial optimization problems and proved complexity and existence of solutions under some typical constraints like preassignment in theory.

Since the basic framework of the tabu search algorithm had been developed in 1990s, the follow-up mainly focused on ways to optimize it. Schaerf (1999) made a survey about heuristic algorithm and their implements in the timetable problems. Due to the big gap between two differen university, alhough these fancy algorithm could give a satifactory explanation of what they had said, the quality of the solutions can not be guranteed. A lot of work had been done to build a baseline for the result and design a powerful language to describe constraints. At this stage, finding a optimum solution of the timetable problem for the university was still valuable. To reduce the influence of parameter choice, a hyper-heuristic algorithm based on Tabu Algorithm was used to solved examination by Hussin (2005) and Kendall & Hussin (2005) which is parameter free. Because the combinatorial optimization problem for scheduling timetables can be modeled by graph coloring problem, Galinier & Hertz (2006) made a survey to describe coloring problem for a graph and introduce some techniques to find the numerical results. Moreover, this paper modified TABCOL algorithm and made a

comparison between four graph coloring method with different search space and search strategies. While an initial feasible solution is needed for all heuristic algorithm, Burke et al. (2007) created heuristic algorithm based on graph degree and operated algorithm in a hyper-heuristic search space to reduce the difficult of finding a feasible initial solution. Another way to initialize an feasible solution at the first step was introduced by Tuga et al. (2007). This paper used Tabu Algorithm to solved timetabling problem. However, due to some hard constraints, we can not guarantee the existence of a feasible solution in some cases. To solve this problem, they defined the feasible solution respect to soft constraints firstly and optimized it by hard constraints in the iterations.

There are also some appreciated framework of timetabling systems. Carter (2000) described a comprehensive university timetable system in Waterloo including system structure and algorithm phase. Additionally, he introduced decomposition in both student section and timetable which will reduce algorithm complexity and conflicts. In general, practical problems are more complexity than algorithm theory. McCollum (2006) gave a overview on gaps of timetabling problem between theory and practice and bridged the gaps between the two. Kristiansen & Stidsen (2013) and Johnes (2015) made a review of timetabling scheduling and student section. Kristiansen & Stidsen (2013) stated most of previous practical research in timetable were founded that they were based on simulation dataset. Therefore, they uploaded a open-source dataset of Denmark university and its format description to enrich research objects. Johnes (2015) pointed out that the qucikly development of software framework many thanks to hyper-heuristic algorithm. Due to the human-like intelligent approach, hyper-heuristic algorithm played an important role in timetable problem in theory and practice.ALTUMA (Tesfaldet 2008) was the solver in the University of Asmara which using memetic algorithm, its performance was experienced and evaluated by real data in the university successfully. Moreover, UniTime (Müller & Rudová 2016) was an open-source system and successfully implemented in a large scale university. The examples and results contained in UniTime were considered as a baseline in many competitions.

3 Problem Description

3.1 Educational Timetable Problem

According to the Wren (1996), timetable problem is defined as an allocation to arrange elements into space and time subjects to constraints such that satisfies a set of desirable objectives as many as possible. Though a lot of literatures has introduced their description of timetable problem over the past years, elements and constraints are different for educational institutions, which makes these descriptions not universal. In this section, we plan to formulate the timetable problem generally and account all requirements that we will meet in XJTLU.

In general, one curriculum contains several courses and each course could be repeated more than once to meet some constraints like the classroom can not hold so many students at once. Thus we split a course into multiple sections if necessary. Normally in every week the lecture of a course section with corresponding teachers and students is repeated several times.

A student is supposed to select several courses and thus the credits for these courses are summed to meet school regulations. Normally, there are two kinds of courses: compulsory course and optional course. Even students in the same major, or students in the same classroom, have different course selections. The sub-problem of finding the best groups of students into corresponding course set is called the grouping problem.

A teacher is often only responsible for the teaching task of one course in a semester. Hence, the upper limit of the number of course sections offered depends on the number of teachers, while the lower limit depends on the number of students and grouping results.

In order to meet some special teaching tasks of the school, some classrooms will have larger capacity or contain some special equipment, such as laboratories. In this case, the pair of classroom and course will be scheduled in advance.

A working day is divided into time slots of equal duration, called timeslots, and the duration matches the duration of a lecture. For example, a lecture will occupy a time period, while for an experimental lecture, it may take 2 or more timeslots. Simplifying the problem in this way allows us to focus only on the pairs of timeslots and matched lectures and ignore the start and end times.

Based on the definitions of sources, several constraints in the timetable problem are listed as follow:

1. student overlaps: a student cannot be involved simultaneously in more than one lecture.
2. teacher overlaps: a teacher cannot be involved simultaneously in more than one lecture.
3. course overlaps: a course cannot be involved simultaneously in more than one lecture.
4. classroom overlaps: a classroom cannot be involved simultaneously in more than one lecture.
5. period constraints: the duration of lectures could be one or two hours.
6. pre-assignment constraints: the lectures are preassigned to a set of specific periods or classrooms.
7. teacher unavailability: a lecture involving a teacher t_j cannot be scheduled at a period during which t_j is not available, including lunch break and university free afternoon (i.e. Wednesday afternoon in XJTLU)
8. geographical constraints: Two lectures given in two distant classroom should be scheduled consecutively if and only if there is sufficient time for moving one classroom to another.
9. compactness constraints: each teacher and student wants a schedule with a minimal number of holes and isolated lectures.
10. distribution constraints: the identical lectures (i.e. the lectures of a same course) should be spread as uniformly as possible in weekly scheduling.

Depend on university requirements, these constraints should be spilt into two parts: one is hard constraints like student overlaps and the other is soft constraints. A objective function is defined by hard and soft constraints to measure the quality of a timetable. Whenever we violate one hard condition, it will cause the objective function to become infinity, but when other soft constraints are violated, it will only cause the objective function to return a bad but feasible solution. While optimizing, a feasible solution is the one which satisfied all hard constraints, moreover, the optimal one is a feasible solution and satisfies all soft constraints, or minimize the objective function as samll as possible.

3.2 Mathematical formulation: A Graph Coloring Representation

In graph theory, a graph is a data structure contains a set of objects and their relations, denoted by $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is the set of elements called vertices and $E = \{(x, y) \mid x, y \in V\}$ is a pair of related vertices, named edges. Given an edge $e = (x, y)$, the vertices x and y are the endpoints of the edge e. Subject to some rules or constraints, graph coloring is an assignment of colors to vertices or edges in G. Vertex coloring is to label every vertex in graph a color such that no pair of adjacant vertices have the same color, in other words, the color of two endpoints of an edge are supposed to be different. Similiarly, edge coloring is an assignment that every edge in graph owns a color so that there are no two adjacant edges have the same color. In order to build on the initial awareness of the graph coloring problem, we list some definitions bellow.

Definition 1 (Line Graph)**.** *Let G be a loopless graph, The vertex set of $L(G)$ is in 1-1 correspondence with the edge set of G and two vertices of $L(G)$ are joined by an edge if and only if the corresponding edges of G are adjacent in G. The graph $L(G)$ is called the line graph of G.*

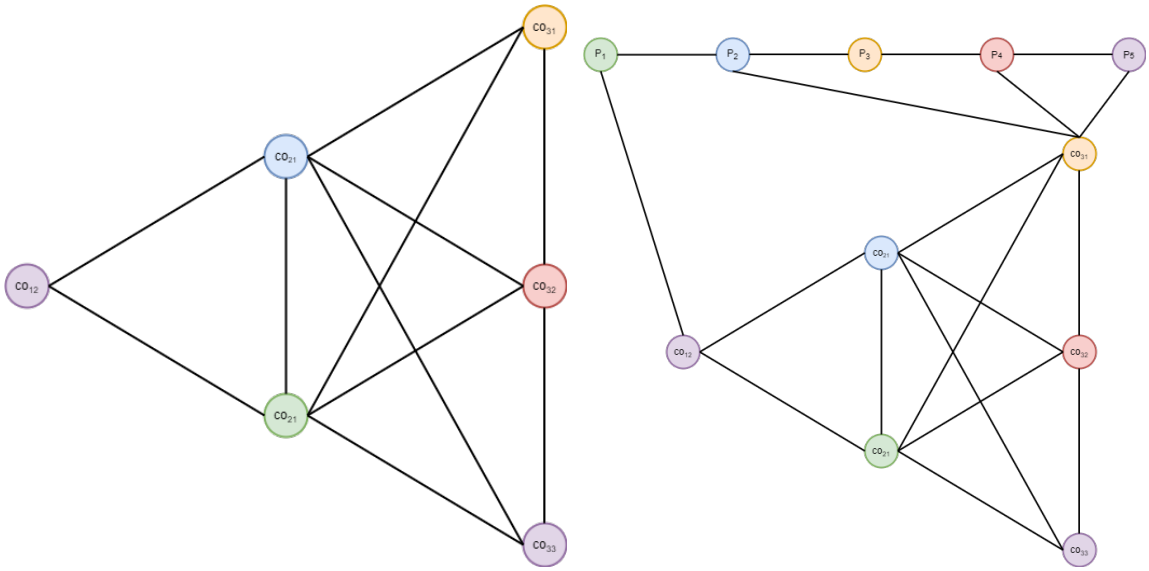
Definition 2 (Simple Graph)**.** *Graphs with no loops or multiple edges are called simple graphs.*

Definition 3 (Complete Graph)**.** *A simple graph in which each pair of distinct vertices are adjacent is a complete graph. We denote the complete graph on n vertices by K_n.*

According to the definition of line graph, an edge coloring problem can be transformed into a vertices coloring problem. It is eary to proof that he number of colors required to color a complete graph k_n equals to n. In this paper, we assume that all graph used is simple and graph coloring is the vertices coloring.

Consider a basic course scheduling model in daily scheduling. For one course section $v_a \in V$ with a set of lectures $L = \{l_1, \ldots, l_n\}$, we denote a lecture-node m_{ab} for each section v_a and lecture l_b. Due to the class overlaps constraint, all pairs of lecture note in course v_a are connected by edges. While assume all courses have only one section, if there is a student taking both courses v_{a_1} and v_{a_2}, we introduce an edge between every pair of lecture node m_{a_1b} and m_{a_2b}. The feasible course scheduling among p periods is respect to the node coloring of graph G with p colors. An example is given in Figure 3.1. Here we have 3 courses and each of them has 1, 2 or 3 lectures. Student A takes courses v_1 and v_2, another B takes v_2 and v_3. A feasible solution is drawn by 5 colors, which means at least 5 periods is needed to assign courses without conflicts.

With pre-assignment constraints and teacher unavailability, we introduce two constraints samples: 1). no v_1 at period 1; 2). one lectures of K_3 at period 1 or 3. A set of period nodes are added to Figure 1(a) and then we get Figure 1(b). Easy to see that pre-assignment of periods are equal to teacher time unavailability. The meaning of the color of different problems can have different definitions including classrooms and timeslots and it depends on what resources we want to assign. It is worth mentioning that period nodes are not necessary in our model since we can give a specific color to some course vertices in advance, which makes it be an pre-assigned problem.



(a) with some hard constraints (b) with some soft constraints (pre-assignment)

Figure 3.1: Graph Coloring representation

4 Tabu Search Methodology

Tabu Search is a hyper-heuristic algorithm designed for finding a global optimal point hopefully and has been used for solving combinatorial optimization problem including graph node coloring, large scale timetabling and TSP efficiently. It was first proposed by Glover (1990) and a series of work done by Werra and Hertz made it popular in some fields including graph coloring and timetable problem (Hertz & Werra 1987, Werra & Hertz 1989, Hertz 1991). The basic process is given a feasible solution as initial point and an objective function to evaluate results, moving the initial point to another solution in the neighbor of the initial point and making comparison and recording the better solution by aspiration function and the objective function. The algorithm does not stop until get a global optimum or reach the max number of iterations. The core of tabu search algorithm is tabu list that all moves back to current point are forbidden in the next several iterations.

The neighbor $N(s)$ is defined for each solution s that in the set of feasible solutions $s \in X$. while we get an initial solution s at the beginning of every iteration, we cometimes choose a subset $V^* \in N(S)$ to be the neighbor of point s if $N(s)$ is too large and it is impossible to traversal all solutions in $N(S)$ within acceptable time. There are two general ways to generate a neighbor of point s: move or exchange (Hertz 1991). The first method is change one of elements in s to another values. For example, if $s \in \mathbb{R}$, a neighbor $N(s)$ can be defined as $N(s) = s + a$, where $a \in \mathbb{R}$. Exchange method means $N(s)$ can be generated by exchanging the values of two elements. Let $s = [a\ b]^T \in \mathbb{R}^2$, we can generate a neighbor $N(s) = [b\ a]^T$.

After finding a new feasible solution s^* that optimize the objective function $z = f(s)$ over neighbor V^* within $|V^*|$ iterations, a move from previous solution s to the new solution s^* is made. When we generate another neighbor $V^*(s^*)$ for this new initial solution at next iteration, the intersection of two neighbor $V^*(s)$ and $V^*(s^*)$ could not be empty. In order to prevent cycling in the local optimum point, a tabu list T is used. This list permit that we can not turn back to solutions that are obtained in the previous $|T|$ iterations. The length of tabu list $|T| = k$ can be fixed or variable. This list is designed as a queue and follows the rule of "first in, first out". After moving the initial solution s to another feasible solution s^*, we add s at the end of queue T as the newest one and remove the oldest solution in T. Whenever we constructing neighbor $N(s)$ to solution s, those solutions in the tabu list will be dropped. A tabu solution s is the one contained in tabu list T and any move back to s is called tabu move.

Though tabu list is a simple and efficient way to reduce the risk of cycling in local optimum, it may also forbid us to move to some points which we do not obtain. Additionally, deciding a move is tabu by the number of iterations may be too absolute. Hence, we introduce the aspiration function $A(z)$ for all possible values of the objective function z. Whenever a move from s to another point s' is a tabu move but gives

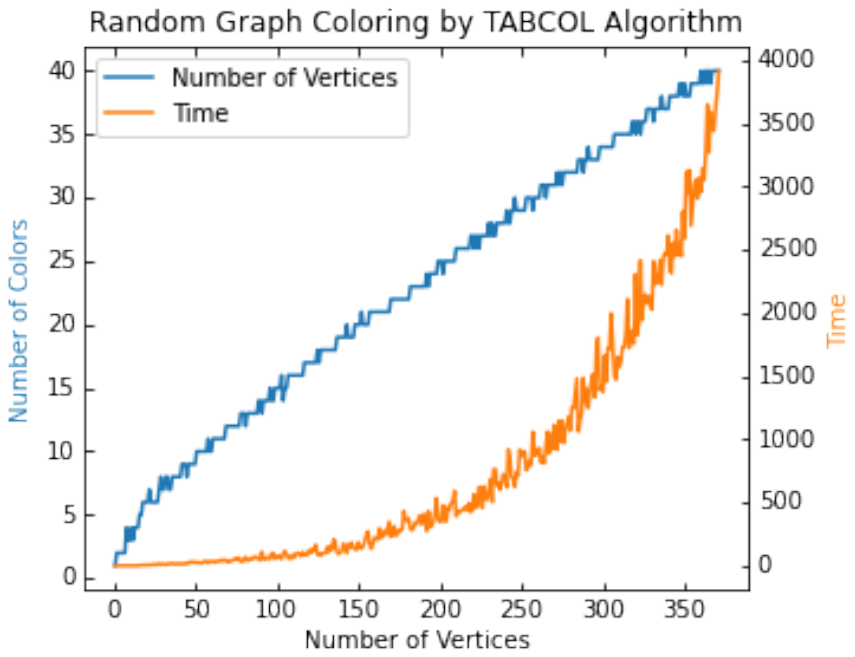


Figure 4.1: Time and color needs to color a graph with different number of vertices.

$f(s') \leq A(z = f(s))$, we remove the point s' from the tabu list and deal with it as a normal element in the neighbor V^*. A simple example of implement is setting aspiration function $A(f(s)) = f(s^\circ)$ where s° is the best solution we have reached so far. With this criterion, a tabu move is allowed if it leads a better solution s in the neighbor $N(s)$ than the best solution obtained in the previous iterations.

In theory, a tabu search implemented in a combination problem will leads to a global optimum solutions after many iterations. However, mainly depending on the way to construct neighborhood of element in feasible solutions set, the number of iterations to traverse all possible solutions can be large and runtime is unacceptable. We create a series of graph that the probability of one edge exists between two random vertices is 0.5 and run it on two-core CPU machine. As Figure 4 tells that with the increasing of number of vertices, running time grows exponentially and the numer of color needs increases linearly. It remains us that a stopping citerion is necessary.

In order to interrupt the tabu search process, we define two stopping criterions. The first is to definite a maximum number of iterations *nbmax* and interrupt the loop if the performance of the solution does not get improved in the last *nbmax* iterations. Secondly, for many combination problems, the bound of the objective function $f = f^*$ is a definite value. Hence we may stop the iterations if the objective function has led to the lower bound f^* for the objective function f or close enough to f^*.

ALGORITHM 1 gives a discription of general tabu search algorithm. The definitions of variables is listed:
X : the set of feasible solutions;
f : the objective function;
$N(s)$: the neighbor of a solution s in X;
$|T|$: the size of the tabu list;
$V^*(s)$: the neighbor of a solution s generated at each iteration;
f^* : a lower bound for the objective function f;
A : the aspiration function;
nbmax: the maximum number of iterations between two improvements of the best solution.

Algorithm 1 General Tabu Search Algorithm (Hertz 1991)

Input
Initialization
$s :=$ initial solution in X.
$nbiter := 0$. (current iteration)
$bestiter := 0$. (iteration when the best solution has been found)
$bestsol := 0$. (best solution)
$T := \varnothing$.
$A :=$ aspiration function.
while $(f(s) > f^*)$ **a**nd (nbiter−bestiter<nbmax) **do**
  $nbiter := nbiter + 1$
  generate a set V^* of solutions s_i in $N(s)$ which are either not tabu or such that $A(f(s)) \geq f(s_i)$;
  choose a solution s^* minimizing f over V^*;
  update the aspiration function A and the tabu list T;
  if $f(s^*) < f($ *bestsol* $)$ **then**
    $bestsol := s^*$;
    $bestiter := nbiter$;
  end if
  $s := s^*$;
end while

5 TABCOL: Tabu Search for Coloring

TABCOL (Hertz & Werra 1987) is a modified version of Tabu search for coloring some large graphs. It first initializes a random solution with a fixed number k of colors, then use tabu search algorithm to reduce the conflicts of edges and obtain a k-color graph as a feasible solution. The number of colors k may change for achieving our goals.

Considering a graph $G = (V, E)$, a feasible solution is any partition $(C_1, \ldots, C_k)$ of the node set V into a fixed number k of subsets. Let E_i denote the set of edges having both endpoints in C_i. The objective function $f(s)$ is

$$f(s = (C_1, \ldots, C_k)) = \sum_{i=1}^{k} |E_i|$$

and need to be minimized. The neighbor of solution s is defined as follow: choose a random node x from the previous solution $s = (C_1, \ldots, C_k)$ and it is an endpoint of one edge in $\cup_{i=1}^{k} E$. Let x belong to a node set partition C_i and reset it in another color $j \neq i$, then we obtain a different solution $s' = (C'_1, \ldots, C'_k)$ where

$$C'_i := C_i \backslash \{x\};$$

$$C'_j := C_j \cup \{x\};$$

$$C'_r := C_r \quad \text{for} \quad r = 1, \ldots, k, \quad r \neq i, j$$

s' is a neighbor solution of s. We repeat this process $|V^*|$ times at one iteration and move to the best solution to minimize objective function $f(s)$ over V^*.

Maintaining a list with $|T|$ tabu solutions is generally not a partical way. For a graph contains hundreds of vertices, the storage space required is exceedingly large (Glover 1990). On the other hand, since we need to make sure if a solution s' is in the tabu list, it will cost much computation time to compare. Therefore, in practice we define tabu list as follows: once we move a vertices x from color set C_i to C_j and return a best solution, this vertie-color pair (x, i) becomes tabu. Any move back to this pair would be treated as tabu move and is forbidden for next $|T|$ iterations. According to the experience of Hertz (1991), the length of tabu list can be set to $|T| = 7$ and $|V^*| = \frac{1}{2}n$ for a graph with n vertices.

Aspiration function is used to identify whether the tabu status should be dropped or not. When we have current value $z = f(s)$ and a move from s to s' is tabu, however, if the value of $f(s')$ smaller than the minimum value of f we has got so far, the tabu status of this solution s' is dropped and consider it as a new element of generated neighbor normally. let us put it all together, firstly we initialize the aspiration function that set $A(z) = z - 1$ for all valueso of z. Then as a new neighbor solution s' is generated and $f(s') \leq A(f(s))$, we update the aspiration function by setting $A(f(s)) = f(s') - 1$.

Similiarly to the raw tabu search, the iteration introduced above will not stop until it reachs its maximum number $nbmax$ of iterations, or it return a solution s with $f(s) = f^*$, where f^* is the lowest bound of the objective funtion.

One modification is applied to speed up the computation process. While we are generating the set of solutions in the neighbor of the initial solution s, we may get $f(s') < f(s)$ at some stage s' and $f(s')$ is not in the tabu list. We hence move to the solution s' directly rather than generate the rest element in the neighbor $N(s)$.

ALGORITHM 2 gives the comprehensive formulation of TABCOL algorithm: the tabu search algorithm for coloring graphs.

Algorithm 2 TABCOL Algorithm

Input
$G = (V, E)$;
$K :=$ number of colors.
$|T| :=$ size of tabu list.
$rep := |V^*|$ number of neighbors in each iteration.
Initialization
$s :=$ initial solution in X.
$nbiter := 0$. (current iteration).
$bestiter := 0$. (iteration when the best solution has been found)
$bestsol := 0$. (best solution)
$T := \varnothing$.
$A :=$ aspiration function.
1: **while** $(f(s) > 0)$ **a**nd $(nbiter - bestiter < nbmax)$ **do**
2: $nbiter := nbiter + 1$
3: generate rep neighbors s_i in $N(s)$ with move $s \to s_i \notin T$ or $f(s_i) \leq A(f(s))$
4: choose a solution s^* minimizing f over V^*;
5: update the aspiration function A and the tabu list T;
6: **if** $f(s^*) < f($ $bestsol$ $)$ **then**
7: $bestsol := s^*$;
8: $bestiter := nbiter$;
9: **end if**
10: **end while**
11: **if** $f(s) > 0$ **then**
12: **return** No coloring has been found with k colors.
13: **end if**
14: **return** s

6 Application in XJTLU Timetable Problem

In this section we take the timetable assignment problem in Xi'an Jiaotong-Liverpool University as an example and simulate the real situation step by step. With the rapid development in recent years, now XJTLU has more than ten thousand students. Since some hardwares facilities did not increase with the number of students, nowadays stuff faces a problem in scheduling timetable under limited resources. For the timetable problein in educational institutions, a feasible solution of with graph representation is to assign every course section to specific timeslot and classroom with constraints as little as possible, in other words, every node in graph is assigned one color from color set and minimizes the number of edges that have endpoints in the same color set. Based on the requirements in university, we here list a part of simplified assumptions that will be involved in our examples:

- Every student takes four courses every semester, including compulsory and optional courses.
- Every teacher is responsible for one course only during one semester.
- A teaching week contains five working days and each day has four timeslots. Weekly timetable keeps the same in all teaching weak.
- Courses are made up of multiple classes and each class lasts for two hours, one timeslot in a day.
- Every week there are two class of one course and should be assigned into different days. We divide one teaching week into two teaching parts: the one from Monday morning to Wednesday noon and the rest belongs to another part. Every teaching part has ten timeslots and they have the same class in different order.
- According to the capacity, the classroom in the university can be divided into three types: less than 50 seats, 50 to 200 seats, and more than 200 seats.

These above assumptions along with the constraints in different examples will build graphs to coloring. Tabu Search algorithm with adapted aspiration function and objective function will be introduced to this assignment problem.

6.1 Example I: Basic Model

We first assign timetables for three major students from year 1 to 3, semester 2, including Applied Mathematics (AM), Finance Mathematics (FM), Actuarial Science (AS). The intersection of courses set offered to three major students is not empty. For example, these three majors contains one common compulsory course (MTH301) and one common compulsory or optional course (MTH302), others have no overlaps with each course. We set a round number of students for each major which in direct proportion to the number in reality and use it to simulate the real data. Table 1 shows the basic information of course sections.

Table 1: Course set with corresponding numbers of studennts For AM, FM, AS (year 1 - 3).

Major	**Compulsory**	**Optional**	**Number**
AMY3S2	MTH301	MTH302, MTH308,PHY302,MTH310,MTH318	150
FMY3S2	MTH301, MTHH302, ECO310	FIN302, MTH316	200
ASY3S2	MTH301, MTH302, MTH306	ECO310, ECO304	80
AMY2S2	MTH208, MTH210	MTH209, MTH224, MTH203	300
FMY2S2	MTH203, FIN202, CPT206	FIN206, MTH208, MTH222	400
ASY2S2	MTH202, MTH214, MTH223	ECO216, FIN206	100
AMY1S2	MTH106, MTH108, MTH118, MTH122		300
FMY1S2	MTH116, MTH106, ECO120, FIN104		400
ASY1S2	MTH120, MTH116, ECO120, FIN104		100

We assume that students choose their optional courses without preference and generate course selection data by uniformly distribution. While consider the particularity of MTH301, Final Year Project, the capacity of classroom is ignored and therefore we assume that there is no multiple sections for all courses.

Based on the course list, we get a graph with 11 vertices and denote is as $V = \{v_i | i = 1, ..., 11\}$, each vertex v_i of graph is corresponding to a course. In addition to the assumption that students have the same chance

to choose their optional courses, course set c_i of a student i is the combination result of compulsory and optional courses and the length of course set $|c_i|$ is 4.

Take AM students as an example, students are supposed to select three optional course out of five in addition to one compulsory course, thus the number of all course sets for AM students is $|C| = C_5^3$, 10 different course sets in total and each course set c has four vertices (courses) $v_i \in V$.

Under the assumption of student overlaps, a edge exists between two vertices if and only if these two vertices contain at least one same student, that is, there exists a course set c and $v_i, v_j \in c$. If there is more than one optional course needs to be selected among several courses by a student, then a edge exists between every two optional course. These vertices of optional courses plus the corresponding compulsory courses of the student build a complete graph K_n and hence we need at least n colors to avoid breaking the constraints.

In this example, we aim to assign courses into different timeslots in a week. Based on the assumption of timeslots in XJTLU, there are 9 timeslots for assigning courses and hence, the maximum number of colors we can use in graph coloring is 9.

To measure the quality of results in iterations, we set the objective function f equals number of edges having two endpoints with the same color and the aspiration function to $A(f) = f - 1$. Two hyperparameter *nbmax* and $|T|$ equal to 10000 and 7. We initial the number of colors as 9 and reduce it until the TABCOL algorithm fails to return a feasible solution.

6.2 Example II: with soft constraints

Based on the first model, in the second example we will add more soft constraints including pre-assignment constraints, geographical constraints and distribution constraints to make the model closer to reality . Course list and the distribution of students of each course is the same as the last model. Since some soft constraints are taken into account in our model, we would have to design a new objective function to measure results. On the other hand, with the increasing of number of constraints, some hyperparameters are necessary to decide which soft constraint is more important than others.

Here we state the definition of objectives functions. Let a be a color and $h(a)$ represents the number of courses labelled by color a with distribution $H(a)$. Hence, we consider a function f_1 measeure the ditribution constraint by variance and set $f_1 = Var(H)$.

We assume that color set $A = \{a_i | i = 1, ..., 9\}$ is a sorted set and let two courses v_m, v_n be labeled by two color v_i and v_j respectively. Geographical constraint works if $|i-j| = 1$ and $v_m, v_n \in c$. If there are q students involved in both two courses, the second objective function is written as $f_2(v_m, v_n) = \begin{cases} q & \text{if } |i-j| = 1 \\ 0 & \text{otherwise} \end{cases}$.

Pre-assignment requirements will not create conflicts that violate hard constraints in general. So when we generate the initial solution s at the beginning of the loop, a specific color is given to every pre-assignment course and would not be changed in the loop.

Hyperparameter q indicate the importance of two soft constraints and a conbinated objective function for a feasible solution s is $f(s) = q_1 f_1 + \sum_{\substack{v_m \neq v_n \\ v_m, v_n \in V}} q_2 f_2(v_m, v_n)$. this hyperparameter value affcets the objective function and similiar to another hyperparameter $|T|$, p is supposed to given in advance. How large p it is totally depends on our experience.

To find a optimal solution s^* that minimize objective function $f(s)$, the strategy to find a neighbor is modified: at each stage we generate a neighbor $N(s)$, in addition to the feasible solutions, the solutions have one conflicts to our hard constraints will be contained in $N(s)$. This modification helps to enlarge the search space in the loop. Whenever we get a solution with a minimized objective function but it is not feasible, we can add another color to avoid breaking hard constraints. Other ingredients of the example have been introduced in Section 6.1.

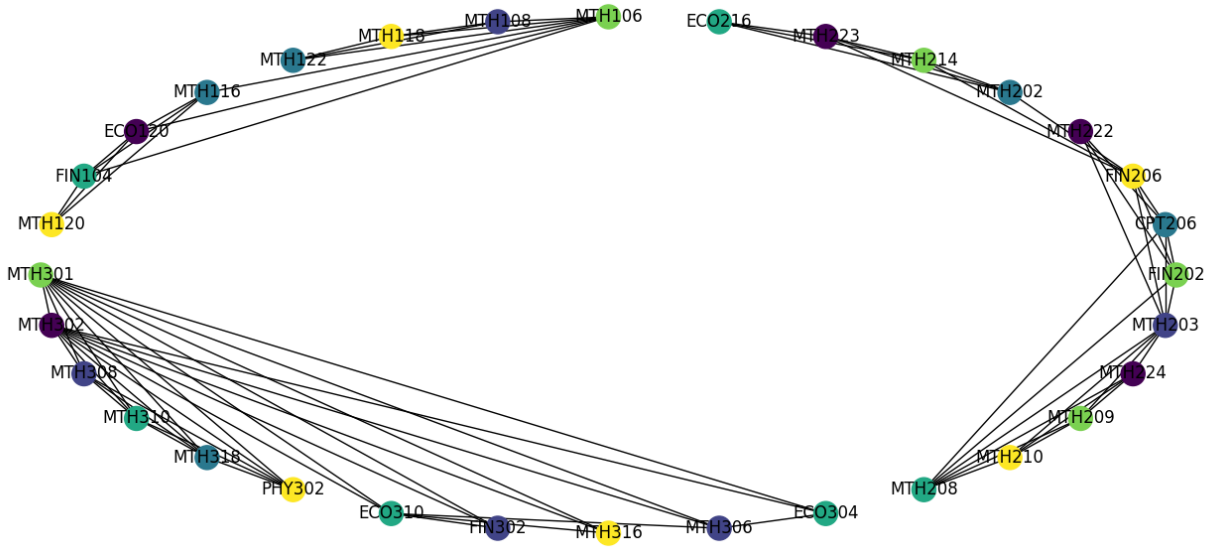


Figure 6.1: Graph Coloring Result. The big graph is composed of three isolated small graph.

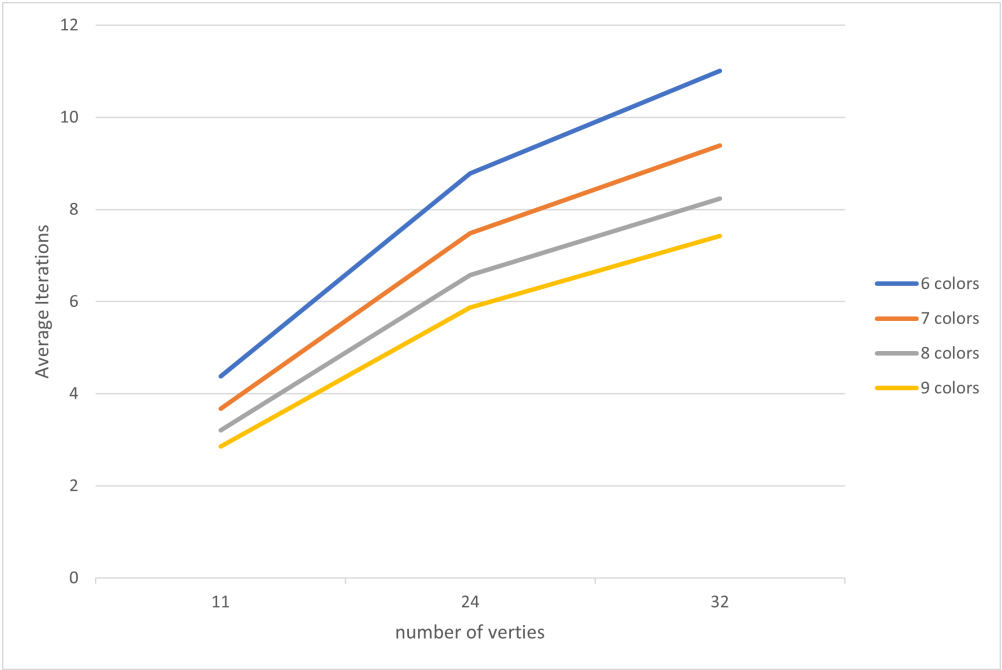


Figure 6.2: Average iterations of n colors, per 10000 attempts. $n = 5$ is not listed since its value is $nbmax$.

7 Numrical Analysis

The code written in Python is listed on appendix A, contains data generation and TABCOL algorithm .Figure 6.2 shows the structure of the graph under student overlaps and the graph is labelled by a minimum 6 colors, which meets our analysis and smaller than the number of timeslots. Taking into account the gap of computing performance between the different computers, we choose the number of iterations as a standard to measure the calculation time. Figure 6.2 shows the average running iterations to find a feasible solution in 10000 attempts of three graphs that contain year 3, year 3 and year 2, year 1 to year 3 of three majors. It might be noticed that the running iteration of coloring graphs within 5 colors equals to max iterations $nbmax$. In addition to the average running iterations grows exponentially with the number of vertices, obtaining a better feasible solution with less colors requires more time in general and proving whether a number is a solution to a problem is time consuming, expecially when this number is not a feasible solution. This result meet the definition of NP problem. In terms of the scale of the number of courses in a semester, TABCOL algorithm can work out the timetable problem in an acceptable time under student overlaps.

Moreover, the graph showed in Figure 6.2 can be divided into three smaller isolated graph and each graph can represent a group of courses designed for the same level stundets, Which means we can reduce our calculation by coloring samll graphs and repeating it several times. Obviously, the decomposition rules is related the level of courses. When we extend our data to the whole university, depending on the course level and majors, the big graph coloring problem can be decomposed into small one and solved by our model one by one within an acceptable running time.

On the other hand, when we are supposed to assign classrooms to courses, this big graph can be used by removing all edges and adding the new edge between a aet of vertices which have the same colors, while vertices remains. If we only consider hard constraints including student overlaps and classroom overlaps and label a set of courses that assigned to the same classroom by the same color, this problem can also be solved by TABCOL algorithm.

8 Conclusion and Discussion

This study introduces the timetable problem in educational institutions and its solution using the hyper-heuristic algorithm. Timetabling means an allocation to assgin elements into a series of timeslots subjects to some requirements and rules. However, given that resources and requirements vary widely from school to school, there is rarely a comprehensive system to address the problem. Moreover, considering the large scale of data in a university and the complexity of some existing algorithms, it is necessary to develop and adapt a model for timatabling.

Since the timetable problem can be formulated by the combination optimization problem, we firstly take XJTLU as an example and discuss the definitions of some elements and requirements in general. Then we illustrate how to represent a timetable problem by graph coloring. What's more, tabu search algorithm is a popular hyper-heuristic to solve combination optimization problems. Here we introduce it and its modified version called TABCOL which is designed for graph coloring. Under some specific constraints, TABCOL algorithm is applied to assign he timeslots to a set of courses in XJTLU. The results have no conflicts to hard constraints and can be promoted to other courses set in university.

However, since we ignore classroom capacity constraint and assume there is only one section for every course, with the number of students and the number of optional courses increasing, this absolute grouping method conflicts to the real situation and will lead to a large number of edges in the graph, or even generate a complete graph. In the future plan, the limitation mentioned above, named student grouping problem, is required more research. Furthermore, when given a graph G, using some graph decomposition algorthm before coloring could reduce the complexity of the problem and make the runtime acceptable. We briefly illustrate how important it is to reduce complexity and more research ahout decomposition algorithm for a general graph coloring is welcomed.

9 References

References

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S. & Qu, R. (2007), 'A graph-based hyper-heuristic for educational timetabling problems', *European Journal of Operational Research* **176**(1), 177–192.

Carter, M. (2000), 'A Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo', *PATAT* .

Costa, D. (1994), 'A tabu search algorithm for computing an operational timetable', *European Journal of Operational Research* **76**(1), 98–110.

Even, S., Itai, A. & Shamir, A. (1975), On the complexity of time table and multi-commodity flow problems, *in* '16th Annual Symposium on Foundations of Computer Science (Sfcs 1975)', IEEE, USA, pp. 184–193.

Galinier, P. & Hertz, A. (2006), 'A survey of local search methods for graph coloring', *Computers & Operations Research* **33**(9), 2547–2562.

Glover, F. (1990), 'Tabu Search: A Tutorial', *Interfaces* **20**(4), 74–94.

Hertz, A. (1991), 'Tabu search for large scale timetabling problems', *European Journal of Operational Research* **54**(1), 39–47.

Hertz, A. & Werra, D. (1987), 'Using tabu search techniques for graph coloring', *Computing* **39**(4), 345–351.

Hussin, N. M. (2005), Tabu Search Based Hyper-Heristic Approachies to Examination Timetabling, PhD thesis.

Johnes, J. (2015), 'Operational Research in education', *European Journal of Operational Research* **243**(3), 683–696.

Kendall, G. & Hussin, N. M. (2005), An Investigation of a Tabu-Search-Based Hyper-Heuristic for Examination Timetabling, *in* G. Kendall, E. K. Burke, S. Petrovic & M. Gendreau, eds, 'Multidisciplinary Scheduling: Theory and Applications', Springer-Verlag, New York, pp. 309–328.

Kristiansen, S. & Stidsen, T. J. R. (2013), 'A Comprehensive Study of Educational Timetabling - a Survey'.

McCollum, B. (2006), 'A Perspective on Bridging the Gap Between Theory and Practice in University Timetabling', *PATAT* .

Müller, T. & Rudová, H. (2016), 'Real-life curriculum-based timetabling with elective courses and course sections', *Annals of Operations Research* **239**(1), 153–170.

Schaerf, A. (1999), 'A Survey of Automated Timetabling', *Artificial Intelligence Review* p. 41.

Tesfaldet, B. T. (2008), 'Automated lecture timetabling using a memetic algorithm', *ASIA-PACIFIC JOURNAL OF OPERATIONAL RESEARCH* **25**(4), 451–475.

Tuga, M., Berretta, R. & Mendes, A. (2007), 'A Hybrid Simulated Annealing with Kempe Chain Neighborhood for the University Timetabling Problem', *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)* pp. 400–405.

Werra, D. (1985), 'An introduction to timetabling', *European Journal of Operational Research* **19**, 151–162.

Werra, D. (1997), 'The combinatorics of timetabling', *European Journal of Operational Research* **96**(3), 504–513.

Werra, D. & Hertz, A. (1989), 'Tabu search techniques: A tutorial and an application to neural networks', *Operations-Research-Spektrum* **11**(3), 131–141.

Wren, A. (1996), Scheduling, timetabling and rostering — A special relationship?, *in* G. Goos, J. Hartmanis, J. Leeuwen, E. Burke & P. Ross, eds, 'Practice and Theory of Automated Timetabling', Vol. 1153, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 46–75.

A Code

```
import pandas as pd
import numpy as np
import random
from collections import deque
from random import randrange
import networkx as nx
import itertools
import matplotlib.pyplot as plt

def create_data(uni_info, student, num_course = 4):
    # generate course list for students from compulsory and optional data
    res = pd.DataFrame(columns=[])
    for i in student.keys():
        all_course = uni_info[i]['Compulsory'] + uni_info[i]['Optional']
        df_tmp = pd.DataFrame(columns = all_course)
        for j in range(0, student[i]):
            optional_course = random.sample(uni_info[i]['Optional'], num_course -
                len(uni_info[i]['Compulsory']))
            single_choice = uni_info[i]['Compulsory'] + optional_course
            single_choice_index = [1 if c in single_choice else 0 for c in all_course]
            df_tmp.loc[len(df_tmp.index)] = single_choice_index
        res = res.append(df_tmp, ignore_index=True).fillna(0)
    return res

def build_constraints(df):
    # generate conflict matrix P, conflict exists if P_ij=1
    columns = df.columns
    print(columns)
    graph = np.zeros((len(columns), len(columns)))
    for i in range(len(columns)-1):
        c1 = df.loc[:, columns[i]]

        for j in range(i+1, len(columns)):
            c2 = df.loc[:, columns[j]]
            c3 = c2.add(c1)
            if (2 in c3.values):
                graph[i][j] = 1
                graph[j][i] = 1
    return graph, dict(itertools.zip_longest([*range(0,len(columns))], columns))

def tabucol(graph, number_of_colors, tabu_size=7, reps=100, max_iterations=10000, debug=False,
    statistics=False):
    # graph is assumed to be the adjacency matrix of an undirected graph with no self-loops
    # nodes are represented with indices, [0, 1, ..., n-1]
    # colors are represented by numbers, [0, 1, ..., k-1]
    colors = list(range(number_of_colors))
    # number of iterations of the tabucol algorithm
    iterations = 0
    # initialize tabu as empty queue
    tabu = deque()

    # solution is a map of nodes to colors
    # Generate a random solution:
    solution = dict()
    for i in range(len(graph)):
        solution[i] = colors[randrange(0, len(colors))]

    # Aspiration level A(z), represented by a mapping: f(s) -> best f(s') seen so far
    aspiration_level = dict()

    while iterations < max_iterations:
        # Count node pairs (i,j) which are adjacent and have the same color.
```

```
move_candidates = set() # use a set to avoid duplicates
conflict_count = 0
for i in range(len(graph)):
    for j in range(i+1, len(graph)):
        if graph[i][j] > 0:
            if solution[i] == solution[j]: # same color
                move_candidates.add(i)
                move_candidates.add(j)
                conflict_count += 1
move_candidates = list(move_candidates) # convert to list for array indexing

if conflict_count == 0:
    # Found a valid coloring.
    break

# Generate neighbor solutions.
new_solution = None
for r in range(reps):
    # choose a conflicted node to move.
    node = move_candidates[randrange(0, len(move_candidates))]

    # Choose color other than current.
    new_color = colors[randrange(0, len(colors) - 1)]
    if solution[node] == new_color:
        # swapping last color with current color
        new_color = colors[-1]

    # Create a neighbor solution
    new_solution = solution.copy()
    new_solution[node] = new_color
    # Count adjacent pairs with the same color in the new solution.
    new_conflicts = 0
    for i in range(len(graph)):
        for j in range(i+1, len(graph)):
            if graph[i][j] > 0 and new_solution[i] == new_solution[j]:
                new_conflicts += 1
    if new_conflicts < conflict_count: # found an improved solution
        # if f(s') <= A(f(s)) [where A(z) defaults to z - 1]
        if new_conflicts <= aspiration_level.setdefault(conflict_count, conflict_count - 1):
            # set A(f(s) = f(s') - 1
            aspiration_level[conflict_count] = new_conflicts - 1

            if (node, new_color) in tabu: # permit tabu move if it is better any prior
                tabu.remove((node, new_color))
                if debug:
                    print("tabu permitted;", conflict_count, "->", new_conflicts)
                break
        else:
            if (node, new_color) in tabu:
                # tabu move isn't good enough
                continue
        if debug:
            print (conflict_count, "->", new_conflicts)
        break

# At this point, either found a better solution,
# or ran out of reps, using the last solution generated

# The current node color will become tabu.
# add to the end of the tabu queue
tabu.append((node, solution[node]))
if len(tabu) > tabu_size: # queue full
    tabu.popleft() # remove the oldest move

# Move to next iteration of tabucol with new solution
```

```
            solution = new_solution
            iterations += 1
            if debug and iterations % 500 == 0:
                print("iteration:", iterations)

    # At this point, either conflict_count is 0 and a coloring was found,
    # or ran out of iterations with no valid coloring.
    if conflict_count != 0:
        print("No coloring found with {} colors.".format(number_of_colors))
        return None
    elif statistics:
        return iterations
    else:
        print("Found coloring:\n", solution, "in ", iterations, 'iterations')
        return solution

def test(graph, k, columns, draw=False):
    ls_graph = graph.astype(int).tolist()
    coloring = tabucol(ls_graph, k, debug=True)
    if draw:
        values = [coloring[node] for node in range(len(ls_graph))]
        nx_graph = nx.from_numpy_matrix(graph)
        # nx_graph.add_nodes_from(columns)
        nx.draw(nx_graph, node_color=values, pos=nx.shell_layout(nx_graph))
        nx.draw_networkx_labels(nx_graph, pos=nx.shell_layout(nx_graph), labels=columns)
        plt.show()

def statistics(graph, k, columns, draw=False, example=10000):
    ls_graph = graph.astype(int).tolist()
    num = 0
    for i in range(example):
        num += tabucol(ls_graph, k, debug=False, statistics=True)

    return num/example

if __name__ == '__main__':
    uni_info = {
        "AM_Y3S2":{"Compulsory": ["MTH301"], "Optional": ["MTH302", "MTH308", "PHY302", "MTH310",
            "MTH318"]},
        "FM_Y3S2":{"Compulsory": ["MTH301","MTH302", "ECO310"], "Optional": ["FIN302", "MTH316"]},
        "AS_Y3S2":{"Compulsory": ["MTH301", "MTH302", "MTH306"], "Optional": ["ECO310", "ECO304"]},

        "AM_Y2S2":{"Compulsory": ["MTH208", "MTH210"], "Optional": ["MTH209", "MTH224", "MTH203"]},
        "FM_Y2S2":{"Compulsory": ["MTH203", "FIN202", "CPT206"], "Optional": ["FIN206", "MTH208",
            "MTH222"]},
        "AS_Y2S2":{"Compulsory": ["MTH202", "MTH214", "MTH223"], "Optional": ["ECO216", "FIN206"]},

        "AM_Y1S2":{"Compulsory": ["MTH106", "MTH108", "MTH118", "MTH122"], "Optional": []},
        "FM_Y1S2":{"Compulsory": ["MTH116","MTH106", "ECO120", "FIN104"], "Optional": []},
        "AS_Y1S2":{"Compulsory": ["MTH120", "MTH116", "ECO120", "FIN104"], "Optional": []}
    }
    student = {
        "AM_Y3S2": 150,
        "FM_Y3S2": 200,
        "AS_Y3S2": 80,
        "AM_Y2S2": 300,
        "FM_Y2S2": 400,
        "AS_Y2S2": 100,
        "AM_Y1S2": 300,
        "FM_Y1S2": 400,
        "AS_Y1S2": 100
    }

    res = create_data(uni_info, student)
    graph,columns = build_constraints(res)
```

```
color_num = 6 # available timeslots
test(graph, color_num, columns, True)

# for i in range(6,10): # statistic number of iterations
#     print(statistics(graph, i, columns, True))
```