

AMERICAN UNIVERSITY OF BEIRUT

HIGH SCHOOL SCHEDULING OPTIMIZATION AS
CONTINUOUS PROBLEM

by
HASSAN ALI NASSER

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Science
to Computational Science Program
of the Faculty of Arts and Sciences
at the American University of Beirut


Beirut, Lebanon
April 2016

AMERICAN UNIVERSITY OF BEIRUT


HIGH SCHOOL SCHEDULING OPTIMIZATION AS
CONTINUOUS PROBLEM

by
HASSAN ALI NASSER

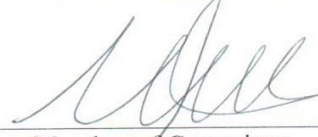
Approved by:



Dr. Mohamad Adnan Al-Alaoui, Professor
Department of Electrical & Computer Engineering
Advisor



Dr. Nabil Nassif, Professor
Department of Mathematics
Member of Committee



Dr. Michel Kazan, Associate Professor
Department of Physics
Member of Committee

Date of thesis defense: April 22, 2016

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: Nasser Hassan Ali
Last First Middle

Master's Thesis Dissertation Master's Project Doctoral

I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

I authorize the American University of Beirut, **three years after the date of submitting my thesis, dissertation, or project**, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.



Signature

25/04/2016

Date

This form is signed when submitting the thesis, dissertation, or project to the University Libraries

ACKNOWLEDGMENTS

Special thanks are for Prof. Mohamad Adnan Al-Alaoui, Prof. Nabil Nassif, and Dr. Michel Kazan for their support. In addition, special thanks for everyone supported me in my private life.

AN ABSTRACT OF THE THESIS OF

Hassan Ali Nasser for Master of Science
Major: Computational Science

Title: *High School Scheduling Optimization as Continuous Problem*

High School Scheduling is a tedious task to do manually. It is considered as NP problem where even computers have difficulties to solve. Here we introduce a new approach to solve *High School Scheduling* programmatically. We model the problem as an optimization one with multiple types of constraints where variables are continuous and functions are continuous and differentiable. Therefore, powerful tools of optimization for continuous functions would be available. Notice that such continuous optimization methods are much faster than discrete optimization methods where a huge number of iterations are usually executed to reach the solution. A lot of papers have been written about this topic and a lot of software has been designed for this purpose. However, most, if not all, interpreted such topic variables as discrete variables where every variable is considered as binary (either 0 or 1). In fact, variables' type is binary. However, we turn around this problem by inhibiting variables getting away from 0 or 1 by introducing a penalty sub-function in the continuous optimization function. In addition, a nonlinear equality constraint is also added to make the variables binary.

We build the continuous model in all its details based on required discrete input. It consists of function to be optimized (including penalty function), linear equality constraint, linear inequality constraint, and nonlinear equality constraint. Corresponding software has been implemented on Matlab. It was tested on two classes with common teachers whose schedule covers seven sessions per day over five days. Very good results were achieved. Little iteration was enough to solve the problem. Varied inputs have been tested and it took always less than one minute to be solved.

CONTENTS

ACKNOWLEDGMENTS	v
ABSTRACT.....	vi
LIST OF ILLUSTRATIONS	ix
LIST OF TABLES.....	x
Chapter	
1. INTRODUCTION.....	1
2. METHODOLOGY	3
2.1. Inputs	3
2.2 Vector X Construction.....	4
2.3 Optimization Function.....	5
2.4 Linear Equality Constraint.....	5
2.4.1 Session Uniqueness Constraint.....	6
2.4.2 Course Sessions Count Constraint.....	6
2.5 Linear Inequality Constraint.....	7
2.5.1 Teacher Time Uniqueness Constraint.....	7
2.5.2 Maximum Course Sessions Number for a Class per Day.	8
2.6 Non-Linear Equality Constraint	9
2.6.1 First Term.	9

2.6.2 Second Term.....	10
2.7 Software Flowchart.....	11
3. RESULTS	16
4. DISCUSSION.....	23
Appendix	
1. SOURCE CODE	24
BIBLIOGRAPHY	44

ILLUSTRATIONS

Figure		Page
1	Flowchart.....	12

TABLES

Table		Page
2.1	Possible combinations of AA1 and AA2.....	10
2.2	Main functions description	15

CHAPTER 1

INTRODUCTION

A constrained optimization problem is one where we try to find most convenient set of variables to achieve a given goal (an optimization function satisfying necessary constraints). When solution is built on discrete variables it is called combinatorial optimization (CO). High school scheduling belongs to this category of problems. Typical method for solving CO problems is using backtracking approach where we try many and many possible combinations of variables without repeating a previously tried one till we reach a constraint satisfied combination. However, this method failed when we had an NP hard problem where number of variables is beyond a certain limit. The main cause of this failure is that backtracking approach should pass hundred thousands of combinations that should be iterated to detect that a suggested combination is not applicable. Therefore, many other approaches were done by applying heuristic ones to improve the solution after imposing an initial suboptimal solution. But the problem is still heavy for computers because of high number of combinations which may be in the order of millions.

However, our approach is completely different one. It considers discrete variables as continuous. Then we push the continuous variables to their possible discrete values using a continuous penalty function in the function to be optimized. In addition, a nonlinear constraint is added to force final value of each variable to be either zero or one. Therefore, the problem turns to be a continuous variables and continuous

differentiable optimization function so that a lot of fast optimization algorithms could be applied. Such modeling gave surprising excellent results that were more than expected.

One of continuous optimization which we have applied is “interior point” algorithm. This algorithm accepts as input the vector of variables to be optimized based on penalty and constraints stated before to make final result binary. In addition, the algorithm accepts a set of linear equality constraints, linear inequality constraints, and non-linear equality constraints. These constraints are dedicated to guarantee many expected output conditions such as:

- Number of sessions per week for each course per class.
- Every session of a given day for each class should be assigned to a unique course.
- Every teacher could not give two simultaneous sessions except free sessions (free session is considered as free course).
- Number of sessions per day for some courses should not exceed a maximum limit.
- Some courses are not allowed to be given together on the same day.

This model was implemented using Matlab where a wide set of continuous optimization built in functions is available.

CHAPTER 2

METHODOLOGY

The implemented software is based on using interior point algorithm of the following optimization continuous problem:

Find vector X such that:

Minimize optimization function: $f(X)$

Where 1- Linear equality constraint: $A_{eq} * X = B_{eq}$.

2- Linear inequality constraint: $A * X \leq B$.

3- Non-linear equality constraint: $C_{eq}(X) = 0$.

4- lower bound $\leq X \leq$ upper bound.

Before applying interior point algorithm we should construct the vector X whose optimum value is the solution for the problem. Moreover, we should organize the input based on which vector X and the constraints would be constructed.

2.1. Inputs

Inputs are organized in a manner that could be easily modified. Inputs are:

- Courses that should be offered to students.
- School Teachers.
- Each class course should be bind to a given teacher. A teacher may give two courses or more in one or more classes.

- Availability of each teacher along the week .i.e. for example, he could attend sessions one to three on Monday and four to six on Wednesday...
- Total number of sessions per week for each course. For example, Math course should be offered seven sessions per week while Geography course should be offered once per week.
- Maximum number of sessions per day and class for each course. For example, if the teacher of Math course is available on all seven sessions of a given day, we should not assign seven Math sessions on this day. In general, we should not assign more than two.
- Courses that should not be offered the same day for the same class. For example, it's preferable not to offer History and Geography on the same day because they usually need memorization.

2.2 Vector X Construction

Based on the inputs we construct the vector $X=[x_1 \ x_2 \ x_3 \ \dots \ x_n]^T$ in parallel with matrix nonzeroAvail (non zero availability) =

class ₁	course ₁	day ₁	session ₁	teacher ₁
class ₂	course ₂	day ₂	session ₂	teacher ₂
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•
class _n	course _n	day _n	session _n	teacher _n

Where every x_i corresponds to $[\text{class}_i \text{ course}_i \text{ day}_i \text{ session}_i \text{ teacher}_i]$. So every possible combination of course, day, session, class where the teacher is available (i.e. could attend the session of that day and is assigned a course for that class) is represented by an entry in X vector whose corresponding details are represented in the same entry level of “nonzeroAvail” matrix. At the end, when X vector is optimized every entry should have a zero or one value. When x_i is one we can say that the corresponding course_i given by teacher_i is assigned to session_i on day_i for class_i.

2.3 Optimization Function

The optimization function is the function to be minimized while maintaining constraints that we will explain in next sections. This function is constructed in a manner to push every entry in X vector towards either zero or one. For every entry of X the function is a parabola whose concavity is down and its peak value is at 0.6 :

$$F(X) = -(X-0.6)^T*(X-0.6)$$

When every entry x_i is below 0.6 it will move towards zero because F(X) will be decreasing and it will stop at zero because it's bounded by zero from bottom. On the other hand, when every entry x_i is above 0.6 it will move towards one because F(x) will also be decreasing and it will stop at one because it is the upper bound. You may ask why we consider 0.6 not 0.5. In fact, this will affect the constraints so x_i will not be stuck at 0.5 meanwhile 0.6 is unstable point for constraints and it will always try to move either left or right.

2.4 Linear Equality Constraint

Linear equality constraint is:

$$A_{eq} * X = B_{eq}$$

It is formed of two basic parts: session uniqueness constraint and course sessions count constraint.

2.4.1 Session Uniqueness Constraint

Every session of the same day and the same class should be assigned to a single course only. This is done by appending a row A_{eqRow} to A_{eq} matrix for each session that could be assigned to more than one course. One is assigned for corresponding locations of those session courses and zero elsewhere. The corresponding B_{eq} single element is of course one. Mathematically speaking:

$$A_{eqRow} * X = 1$$

$$A_{eqRow} = [x_1 \ x_2 \ \dots \ x_n] \quad \text{where}$$

- $x_i=1$ for all its corresponding entry in nonzeroAvail matrix belong to the same session, day, and class.
- $x_i=0$ elsewhere.

In other words for each session:

$$\sum x_i = 1 \text{ for every } x_i \text{ belongs to this session.}$$

This criterion will guarantee 100% that a given session would not be assigned twice. So, a single class will not have two courses at the same time.

2.4.2 Course Sessions Count Constraint

Another important linear equality constraint that should be appended to A_{eq}

has to guarantee that the number of sessions for each class course is as required by the input. For example, class A should be assigned 7 sessions of Math over the week, 1 session of Geography, 5 sessions of Arabic, 4 sessions of English... Another class B may be assigned 6 sessions of Math, 4 sessions of Arabic ... This could be done by appending, for each class course, to A_{eq} a row A_{eqRow} where the corresponding B_{eq} entry is equal to the number of session required by the input for this class course.

$$A_{eqRow} * X = b$$

$$A_{eqRow} = [x_1 \ x_2 \ \dots \ x_n] \quad \text{where}$$

- $x_i=1$ where all its corresponding entry in $nonzeroAvail$ matrix belong to the same course and class.
- $x_i=0$ elsewhere.
- b is the number of sessions, of this course, that should be offered over the week.

In other words for each class course:

$\sum x_i = b_j$ for every x_i belongs to this class course. b_j is the number of sessions required for this course over the week.

2.5 Linear Inequality Constraint

Linear inequality constraint is:

$$A * X \leq B$$

Formed of two basic parts.

2.5.1 Teacher Time Uniqueness Constraint

This constraint is dedicated to guarantee that a given teacher who is common between multiple classes should not be assigned two or more sessions that have the same time. Imagine that teacher Issam is assigned a session from 8:00AM to 8:50AM for class A and class B at the same time! Of course, he will not subdivide himself into two parts. This is done by appending, for each session of a given day that may belong to more than one class for the same teacher, a row ARow in matrix A where corresponding B entry should be one:

$$\text{ARow} * X \leq 1$$

$$\text{ARow} = [x_1 \ x_2 \ \dots \ x_n] \quad \text{where}$$

- $x_i = 1$ where all its corresponding entry in nonzeroAvail matrix belong to the same teacher, session number, and day.
- $x_i = 0$ elsewhere.

i.e. $\text{ARow} * X = 0$ so that the neither sessions is assigned to this teacher.

$\text{ARow} * X = 1$ so that the only one session is assigned to this teacher at a given time.

In other words:

$$\sum x_i \leq 1 \text{ for every } x_i \text{ belongs to same session of a day and same teacher.}$$

2.5.2 Maximum Course Sessions Number for a Class per Day

This constraint will guarantee that a given course should not be assigned in the schedule of a class more than a given limit per day. For example, theoretically speaking Math course of class A could be available for six sessions on Monday but practically it should not be offered more than two times on every day. This is done by appending a

row ARow for each course of a class on every day such that the corresponding B element should be equal to a given value b determined by the input requirements.

$$\text{ARow} * X \leq b$$

$$\text{ARow} = [x_1 \ x_2 \ \dots \ x_n] \quad \text{where}$$

- $x_i = 1$ where all its corresponding entry in nonzeroAvail matrix belong to the same course, day, and class.
- $x_i = 0$ elsewhere.
- b is the maximum number of sessions per day for the class course.

In other words:

$$\sum x_i \leq b \text{ for every } x_i \text{ belongs to same course, day, and class.}$$

2.6 Non-Linear Equality Constraint

Non-linear equality constraint is:

$$\text{Ceq}(X) = \text{conflictFactor} * ((AA1 * X)' * (AA2 * X) + (X)' * (1 - X)) = 0$$

formed of two basic terms:

2.6.1 First Term

First term guaranties that every entry in X vector is either one or zero. The term is : $(X)' * (1 - X)$. Notice that the result of this term is always positive because every term of X and $(1 - X)$ is positive and between zero and one. Moreover it would not be zero unless every term of X is zero or one.

2.6.2 Second Term

Second term is dedicated to guarantee that two given courses should not be scheduled on the same day for a given class. For example, many schools ask not to put History and Geography courses on the same day for a given class. Its expression is :

$$\text{conflictFactor} * ((AA1 * X)' * (AA2 * X)).$$

Where conflictFactor is equal to number of rows in AA1 matrix which is the same as number of rows in AA2 matrix. Each entry in AA1 and AA2 corresponds to first and second courses that should not be assigned on the same day. Every couple of entries of AA1 and AA2 corresponds to every combination of two sessions that should not conflict. For example:

Course1 is available on sessions 1,2, and 3 on Monday.

Course2 is available on sessions 4 and 5 on Monday.

Six entries would be appended to AA1 and AA2 as follows:

AA1	AA2
Session 1 on Monday	Session 4 on Monday
Session 2 on Monday	Session 4 on Monday
Session 3 on Monday	Session 4 on Monday
Session 1 on Monday	Session 5 on Monday
Session 2 on Monday	Session 5 on Monday
Session 3 on Monday	Session 5 on Monday

Table 2.1 Possible combinations of AA1 and AA2.

2.7 Software Flowchart

The following is the flowchart of the software. Functions used in flowchart are described in table 2.2.

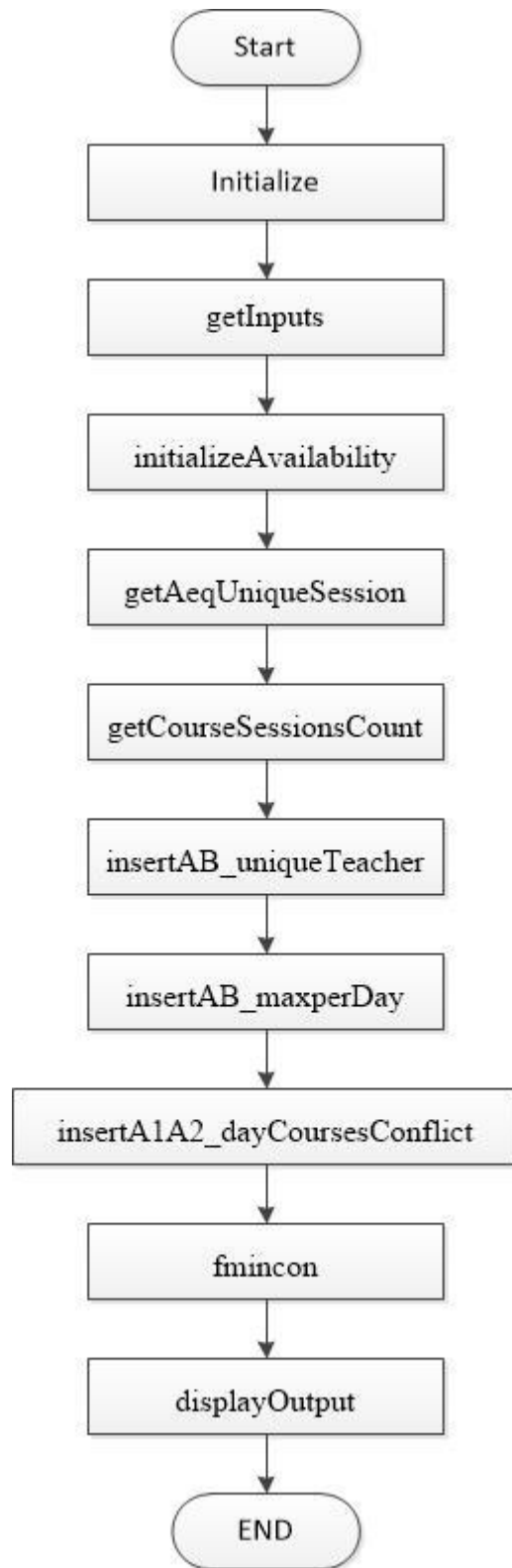


Figure 1 Flowchart

Function	Description
Initialize	No input parameters. Initialize the constants used in the software.
getInputs	No input parameters. Get all the inputs that should be given by the user to build up the output schedule as described in section 2.1.
initializeAvailability	No Input parameters. Use the global variables entered in “getInputs” function to build X vector and “nonzeroAvail” matrix as described in section 2.2.
getAeqUniqueSession	No input parameters. Append to Aeq and Beq to guarantee session uniqueness as described in section 2.4.1.
getCourseSessionsCount	No input parameters. Append to Aeq and Beq to assign a required number of sessions per week of a course for a given class as entered by “getInputs” function. This function implements section 2.4.2.
insertAB_uniqueTeacher	No input parameters. Append to A and B matrices to guarantee that the same teacher will not be in two classes at the same

	time as described in section 2.5.1.
insertAB_maxperDay	<p>Input parameters are:</p> <p>A and B to be appended</p> <p>Class id</p> <p>Course id</p> <p>Maximum number of sessions allowed per day and class</p> <p>Append to A and B corresponding rows to limit number of sessions of same course per day for a given class as described in section 2.5.2.</p> <p>This function is called multiple times as needed.</p> <p>It's called once per class course.</p>
insertA1A2_dayCoursesConflict	<p>Input parameters:</p> <p>AA1 and AA2 to be appended</p> <p>Class id</p> <p>Course 1 id</p> <p>Course 2 id</p> <p>Append to AA1 and AA2 necessary rows to guarantee that course 1 and course 2 should not be given on the same day for the class passed in parameters as described in section 2.6.1</p> <p>This function may be called many times as needed.</p>

fmincon	Matlab built in function that uses interior point optimization as default.
displayOutput	<p>Arrange the schedule from the result X vector.</p> <p>Display the schedule.</p> <p>Display elapsed time.</p> <p>Display maximum of possible combinations of inputs to get an idea how much time was saved.</p>

Table 2.2 Main functions description

CHAPTER 3

RESULTS

Software was tested on a computer with the following specifications:

- 1- 64 bit Windows 8 operating system.
- 2- 1.7 GHZ i5-3317U CPU Intel(R) Core (TM).
- 3- 4 GB Ram
- 4- Matlab version 2015.

The results were surprising. Multiple Inputs with varying situations were tested.

Whenever there is a solution it was found. The elapsed time is less than one minute for two classes multiple situations. The results for appendix 1 software version is as follows:

```
vectorSize = 311
```

```
options =
```

```
fmincon options:
```

```
Options used by current Algorithm ('interior-point'):
```

```
(Other available algorithms: 'active-set', 'sqp', 'trust-region-reflective')
```

```
Set by user:
```

```
FinDiffType: 'central'
```

MaxFunEvals: 1288000

MaxIter: 5000

TolCon: 1.0000e-09

TolFun: 1.0000e-09

TolX: 1.0000e-30

Default:

Algorithm: 'interior-point'

AlwaysHonorConstraints: 'bounds'

DerivativeCheck: 'off'

Diagnostics: 'off'

DiffMaxChange: Inf

DiffMinChange: 0

Display: 'final'

FinDiffRelStep: 'eps^(1/3)'

FunValCheck: 'off'

GradConstr: 'off'

GradObj: 'off'

HessFcn: []

Hessian: 'bfgs'

HessMult: []

InitBarrierParam: 0.1000

InitTrustRegionRadius: 'sqrt(numberOfVariables)'

MaxProjCGIter: '2*(numberOfVariables-numberOfEqualities)'

ObjectiveLimit: -1.0000e+20

OutputFcn: []

PlotFcns: []

ScaleProblem: 'none'

SubproblemAlgorithm: 'ldl-factorization'

TolProjCG: 0.0100

TolProjCGAbs: 1.0000e-10

TypicalX: 'ones(numberOfVariables,1)'

UseParallel: 0

Show options not used by current Algorithm ('interior-point')

Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than the selected value of the step size tolerance and constraints are satisfied to within the selected value of the constraint tolerance.

<stopping criteria details>

output =

iterations: 138

funcCount: 86058

constrviolation: 1.6290e-14

stepsize: 2.2023e-30

algorithm: 'interior-point'

firstorderopt: 2.9733

cgiterations: 59

message: 'Local minimum possible. Constraints satisfied.'

fmincon stopped because the size of the current step is less than
the selected value of...'

Class 1:

str =

'Day 1: 2-Math., 2-Math., 5-Biol., 3-Comp., 4-Engl.,14-Phil., 7-Arab.,Courses

Number:7;'

str =

'Day 2: 8-Sport, 5-Biol.,13-Geog., 9-Fren., 7-Arab., 4-Engl.,10-Chem.,Courses

Number:7;'

str =

'Day 3: 4-Engl., 4-Engl.,12-Phys.,11-Reli., 2-Math., 7-Arab., 7-Arab.,Courses

Number:7;'

str =

'Day 4: 2-Math., 2-Math.,10-Chem., 4-Engl., 4-Engl., 7-Arab., 9-Fren.,Courses

Number:7;'

str =

'Day 5: 6-Hist., 4-Engl., 2-Math.,12-Phys.,16-Art., 7-Arab.,15-Educ.,Courses

Number:7;'

str =

Day 6:Courses Number:0;

Class 2:

str =

'Day 1:17-Math.,17-Math.,19-Engl.,19-Engl.,18-Biol., 3-Comp., 6-Hist.,Courses
Number:7;'

str =

'Day 2:13-Geog.,18-Biol.,17-Math.,23-Arab.,23-Arab., 9-Fren.,20-Chem.,Courses
Number:7;'

str =

'Day 3:19-Engl.,19-Engl.,21-Reli.,22-Phys.,17-Math.,23-Arab.,23-Arab.,Courses
Number:7;'

str =

'Day 4:17-Math.,17-Math.,20-Chem.,14-Phil.,19-Engl., 9-Fren.,23-Arab.,Courses
Number:7;'

str =

'Day 5:19-Engl.,19-Engl., 8-Sport,22-Phys.,15-Educ.,16-Art...,23-Arab.,Courses
Number:7;'

str =

Day 6:Courses Number:0;

elapsedTime =

24.2797

possibleCombinations =

3.0487e+32

Notice how the following constraints are satisfied:

6 sessions of Math course per week for each class distributed on the valid available sessions of Math teacher.

1 session of Computer course per week for each class.....

No Geography, History, or Philosophy is given on the same day for the two classes.

No more than two sessions of Math are given per day.

No more than two sessions of English are given per day.

Notice that although Computer, History, Sport, French, Geography, Philosophy, Education, and Art have common teachers for both classes, the software didn't assign the same session on the same day for both classes because the same teacher couldn't attend both classes at the same time.

The following output of the class 2:

```
"Day 5:19-Engl., 19-Engl., 8-Sport,22-Phys.,15-Educ.,16-Art.,23-Arab.,Courses  
Number:7;"
```

Means for day 5 of class 2:

The following output of the class 2:

```
'Day 5:19-Engl.,19-Engl., 8-Sport,22-Phys.,15-Educ.,16-Art.,23-Arab.,Courses  
Number:7;'
```

Means for day 5 of class 2:

session 1 is English whose teacher id is 19.

session 2 is English whose teacher id is 19.

session 3 is Sport whose teacher id is 8.

session 4 is Physics whose teacher id is 22.

session 5 is Education whose teacher id is 15.

session 6 is Art whose teacher id is 16.

session 7 is Arabic whose teacher id is 23.

Total number of assigned sessions for this day is 7.

In addition, important results are that:

Elapsed time is 24.3 sec.

Number of iterations is 138.

CHAPTER 4

DISCUSSION

As a conclusion, the software succeeded to find the solution despite the complicated practical conditions, in little iteration with little time. This makes it eligible to solve more complicated real cases where interrelated fifteen classes are asking for a convenient schedule. These results make this new approach of solving school scheduling problem quite competitive and promising of excellent results in future. Interior point algorithm is a powerful tool for linear and nonlinear constraints. Applying it in this approach has proven more and more how much this algorithm is powerful in solving large constrained problems.

APPENDIX 1

SOURCE CODE

```
function schedule01()
clear;
global Math
global Computer
global English
global Biologie
global History
global Arabic
global Sport
global French
global Chemistry
global Religion
global Physics
global Geography
global Philosophy
global Education
global Art

global nonzeroAvail
global vectorSize
global AA1
global AA2
global A
global B
global Aeq
global Beq
global conflictFactor
global midX
global X
global elapsedTime

initialize();
getInputs();
```

```

% initialize positive and negative availability of each teacher per session
% per day so that variable vector that should be optimized and its class,
% course,day,session,teacher positive availability matrix are constructed
initializeAvailability();

% this is the size of the variable vector to be optimized
vectorSize=size(nonzeroAvail,1)

% lower bound vector of X variable vector to be optimized
lb=zeros(vectorSize,1);
% higher bound vector of X variable vector to be optimized
ub=ones(vectorSize,1);
% initial value of X variable vector
X0=midX*ones(vectorSize,1);

% Aeq*X=Beq
[Aeq1,Beq1]=getAeqUniqueSession();
[AeqArray,BeqArray]=getCourseSessionsCount();
Aeq=[Aeq1;AeqArray];
Beq=[Beq1;BeqArray];

% Build A and B where A*X<=B
[A,B]=insertAB_uniqueTeacher();
%class 1
[A,B]=insertAB_maxperDay(A,B,1,Math,2);
[A,B]=insertAB_maxperDay(A,B,1,English,2);
[A,B]=insertAB_maxperDay(A,B,1,Biologie,1);
[A,B]=insertAB_maxperDay(A,B,1,Arabic,2);
[A,B]=insertAB_maxperDay(A,B,1,French,1);
[A,B]=insertAB_maxperDay(A,B,1,Chemistry,1);
[A,B]=insertAB_maxperDay(A,B,1,Physics,1);
%class 2
[A,B]=insertAB_maxperDay(A,B,2,Math,2);
[A,B]=insertAB_maxperDay(A,B,2,English,2);
[A,B]=insertAB_maxperDay(A,B,2,Biologie,1);
[A,B]=insertAB_maxperDay(A,B,2,Arabic,2);
[A,B]=insertAB_maxperDay(A,B,2,French,1);
[A,B]=insertAB_maxperDay(A,B,2,Chemistry,1);
[A,B]=insertAB_maxperDay(A,B,2,Physics,1);

```

```

% AA1,AA2,and conflictFactor are terms to be used in nonlinear
% equality constraint
AA1=[];
AA2=[];
[AA1,AA2]= ...
    insertA1A2_dayCoursesConflict(AA1,AA2,1,History,Geography);
[AA1,AA2]= ...
    insertA1A2_dayCoursesConflict(AA1,AA2,1,Philosophy,History);
[AA1,AA2]= ...
    insertA1A2_dayCoursesConflict(AA1,AA2,1,Philosophy,Geography);
[AA1,AA2]= ...
    insertA1A2_dayCoursesConflict(AA1,AA2,2,History,Geography);
[AA1,AA2]= ...
    insertA1A2_dayCoursesConflict(AA1,AA2,2,Philosophy,History);
[AA1,AA2]= ...
    insertA1A2_dayCoursesConflict(AA1,AA2,2,Philosophy,Geography);
conflictFactor=size(AA1,2);

% Select non default optimization before applying it
warning('off','MATLAB:nearlySingularMatrix');
options = optimoptions('fmincon','MaxFunEvals',1288000, ...
    'TolCon',1.0e-09,'TolFun', 1.0e-09, ...
    'TolX',1.0e-30,'MaxIter',5000, ...
    'FinDiffType','central')

% Apply optimization and get results in X and status in output
tic;
[X,fval,exitflag,output] = fmincon(@objectivefun,X0,A,B,Aeq,Beq,lb,ub, ...
    @nonLinearConst,options);
elapsedTime=toc;
X=round(X);

% display optimization status ( if fail or success)
display(output)

% display outcome of software
displayOutput();

```

```

end
function returnVar=objectivefun(X)
% returnVar is function to be optimized
    global midX
    Y=X-midX;
    returnVar=-Y*Y ;
end
function [c,ceq ] = nonLinearConst(X)

% This is non linear constraint where ceq should be zero when evaluated.
% evaluation of ceq should be zero to consider non linear constraints
% equality valid c is used for non linear constraints inequality that
% is not needed in our modeling of problem to be solved.
    global AA1
    global AA2
    global conflictFactor

    ceq = conflictFactor*((AA1*X)')*(AA2*X)+(X')*(1-X);
    c = [];

end
function [AeqReturn,BeqReturn]=getAeqUniqueSession()
% This function determine the linear equality constraint of form :
% AeqReturn*X = BeqReturn
% such that the the same session on the same day of the same class should
% be assigned once and not assigned twice ore more.

    global vectorSize
    global nonzeroAvail
    global SessionDim
    global DayDim
    global ClassDim

    AeqReturn=[];
    zeroRow=zeros(1,vectorSize);
    for classCounter=1:ClassDim
        for sessionCounter=1:SessionDim
            f3=find(nonzeroAvail(:,4)==sessionCounter);

```

```

for dayCounter=1:DayDim
    tempo=zerosRow;
    f1=find(nonzeroAvail(:,1)==classCounter);
    f2=find(nonzeroAvail(:,3)==dayCounter);
    for f1Counter=1:size(f1,1)
        for f2Counter=1:size(f2,1)
            for f3Counter=1:size(f3,1)
                if ((f1(f1Counter)==f2(f2Counter))&& ...
                    (f1(f1Counter)==f3(f3Counter)))
                    tempo(f1(f1Counter))=1;
                end
            end
        end
    end
    if (sum(tempo)>=2)
        AeqReturn=[AeqReturn;tempo];
    end
end
end
end
end
end

```

```

BeqReturn=ones(size(AeqReturn,1),1);

```

```

end
function [AeqReturn,BeqReturn]=getCourseSessionsCount()
% This function determine the linear equality constraint of form :
% AeqReturn*X = BeqReturn
% such that every course is assigned a determined number of sessions for
% each class all around the week.

```

```

global vectorSize;
global nonzeroAvail
global ClassDim
global CourseDim
global Empty
global Math
global Computer
global English
global Biologie

```

```
global History
global Arabic
global Sport
global French
global Chemistry
global Religion
global Physics
global Geography
global Philosophy
global Education
global Art
global classCourseNumber
```

```
AeqReturn=[];
BeqReturn=[];
zeroRow=zeros(1,vectorSize);
for classCounter=1:ClassDim
    for courseCounter=1:CourseDim
        tempo=zeros(1,vectorSize);
        match=false;
        for counter=1:vectorSize
            if ((nonzeroAvail(counter,2)==courseCounter)&& ...
                (nonzeroAvail(counter,1)==classCounter))
                tempo(1,counter)=1;
                match=true;
            end
        end
        if (match)
            AeqReturn=[AeqReturn;tempo];
            BeqReturn=[BeqReturn;classCourseNumber(classCounter, ...
                courseCounter)];
        end
    end
end
end
```



```

BReturn=ones(size(AReturn,1),1)+0.01;

end
function [Areturn,Breturn]=insertAB_maxperDay(A,B,classNb, ...
    courseNb,maxDaySession);
% This function determine the constraint of form : Areturn*X <= Breturn
% such that every course whose number is courseNb should not be assigned
% a number of sessions greater than maxDaySession per day for class classNB.
% For eexample if Math course is available five sessions for a day we
% should assign it not more than two hours a day.

    global vectorSize
    global nonzeroAvail
    global DayDim

    Atempo=[];
    zeroRow=zeros(1,vectorSize);

    for dayCounter=1:DayDim
        tempo=zeroRow;
        for counter=1:vectorSize
            if ((nonzeroAvail(counter,2)==courseNb)&& ...
                (nonzeroAvail(counter,3)==dayCounter)&& ...
                (nonzeroAvail(counter,1)==classNb))
                tempo(1,counter)=1;
            end
        end
        if (sum(tempo)>0)
            Atempo=[Atempo;tempo];
        end
    end

    Btempo=(ones(size(Atempo,1),1)+0.01)*maxDaySession;
    Areturn=[A;Atempo];
    Breturn=[B;Btempo];

end
function [A1return,A2return]=insertA1A2_dayCoursesConflict(A1,A2,classNb,
courseNb1,courseNb2);

```

```

% This function determine the constraint of form : ((A1return*X)')*(A2return*X)=0
% such that every course whose number is courseNb1 should not be assigned
% a session the same day with another course whose number is courseNb1.
% For example some schools may not prefer to give History and Geography the
% same day

```

```

global vectorSize
global nonzeroAvail
global DayDim

```

```

A1tempo=[];
A2tempo=[];
zeroRow=zeros(1,vectorSize);

```

```

for dayCounter=1:DayDim
    for counter1=1:vectorSize
        for counter2=1:vectorSize

```

```

            if ((nonzeroAvail(counter1,2)==courseNb1) && ...
                (nonzeroAvail(counter2,2)==courseNb2) && ...
                (nonzeroAvail(counter1,3)==dayCounter) && ...
                (nonzeroAvail(counter2,3)==dayCounter) && ...
                (nonzeroAvail(counter1,1)==classNb) && ...
                (nonzeroAvail(counter2,1)==classNb))

```

```

                tempo=zeroRow;
                tempo(1,counter1)=1;
                A1tempo=[A1tempo;tempo];

```

```

                tempo=zeroRow;
                tempo(1,counter2)=1;
                A2tempo=[A2tempo;tempo];

```

```

            end
        end
    end
end

```

```

A1return=[A1;A1tempo];

```

```
A2return=[A2;A2tempo];
```

```
end
function initializeAvailability()
% initialize availability of each teacher per session and per day
% construct the vector of variables x from availability only so it could
% be assigned yes (one) or no (zero). Of course a session that is not
% available it is not necessary to seek its status and consider it as a
% variable since it's always no (zero).
global CourseDim
global nonzeroAvail
global ClassDim
global SessionDim
global DayDim
global TeacherDim
global possibleCombinations
global classCourseTeacher
global Avail

nonzeroAvail=[];
for courseCounter=1:CourseDim
    for classCounter=1:ClassDim
        for teacherCounter=1:TeacherDim
            if (classCourseTeacher(classCounter,courseCounter)== ...
                teacherCounter)
                for dayCounter=1:DayDim
                    for sessionCounter=1:SessionDim
                        if (Avail(teacherCounter,dayCounter, ...
                            sessionCounter)~=0)
                            nonzeroAvail=[nonzeroAvail;classCounter ...
                                courseCounter dayCounter ...
                                sessionCounter teacherCounter];
                        end
                    end
                end
            end
        end
    end
end
end
end
end
end
```

```

possibleCombinations=1;
for dayCounter=1:DayDim
    for sessionCounter=1:SessionDim
        count=0;
        for counter=1:size(nonzeroAvail,1)
            if ((nonzeroAvail(counter,3)==dayCounter)&& ...
                (nonzeroAvail(counter,4)==sessionCounter))
                count=count+1;
            end
        end
        if count>1
            possibleCombinations=possibleCombinations*count;
        end
    end
end
end

```

```

end
function getInputs()
    global courseName
    global Empty
    global Math
    global Computer
    global English
    global Biologie
    global History
    global Arabic
    global Sport
    global French
    global Chemistry
    global Religion
    global Physics
    global Geography
    global Philosophy
    global Education
    global Art

    global CourseDim

```

```
global ClassDim
global SessionDim
global DayDim
global TeacherDim
```

```
global Monday
global Tuesday
global Wednesday
global Thursday
global Friday
```

```
global classCourseTeacher
global Avail
global classCourseNumber
```

```
courseName=cellstr(['Empty';'Math.';'Comp.';'Engl.';'Biol.';
                   'Hist.';'Arab.';'Sport';'Fren.';'Chem.';
                   'Reli.';'Phys.';'Geog.';'Phil.';'Educ.';
                   'Art..'];]);
```

```
Avail=zeros(TeacherDim,DayDim,SessionDim);
```

```
Avail(1,Monday,:)= [1 1 1 1 1 1 1];
Avail(1,Tuesday,:)= [1 1 1 1 1 1 1];
Avail(1,Wednesday,:)= [1 1 1 1 1 1 1];
Avail(1,Thursday,:)= [1 1 1 1 1 1 1];
Avail(1,Friday,:)= [1 1 1 1 1 1 1];
```

```
Avail(2,Monday,:)= [1 1 1 1 1 1 1];
Avail(2,Tuesday,:)= [0 1 1 1 0 0 0];
Avail(2,Wednesday,:)= [0 0 0 0 1 1 1];
Avail(2,Thursday,:)= [1 1 0 0 0 0 0];
Avail(2,Friday,:)= [1 1 1 1 1 1 1];
```

```
Avail(3,Monday,:)= [1 1 1 1 1 1 1];
Avail(3,Tuesday,:)= [0 0 0 0 0 0 0];
Avail(3,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(3,Thursday,:)= [1 1 1 0 0 0 0];
Avail(3,Friday,:)= [0 0 0 0 0 0 0];
```

Avail(4,Monday,:)= [0 0 1 1 1 1 1];
Avail(4,Tuesday,:)= [0 0 0 0 0 1 1];
Avail(4,Wednesday,:)= [1 1 1 1 1 1 1];
Avail(4,Thursday,:)= [0 0 0 1 1 0 0];
Avail(4,Friday,:)= [1 1 1 0 0 0 0];

Avail(5,Monday,:)= [0 0 1 1 1 0 0];
Avail(5,Tuesday,:)= [0 1 0 0 0 0 0];
Avail(5,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(5,Thursday,:)= [0 0 0 0 0 0 0];
Avail(5,Friday,:)= [0 0 0 0 0 0 0];

Avail(6,Monday,:)= [0 0 0 0 1 1 1];
Avail(6,Tuesday,:)= [0 0 0 0 0 0 0];
Avail(6,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(6,Thursday,:)= [0 0 0 0 0 0 0];
Avail(6,Friday,:)= [1 1 1 0 0 0 0];

Avail(7,Monday,:)= [0 0 0 0 0 0 1];
Avail(7,Tuesday,:)= [0 0 0 1 1 0 0];
Avail(7,Wednesday,:)= [0 0 0 0 0 1 1];
Avail(7,Thursday,:)= [0 0 0 0 0 1 1];
Avail(7,Friday,:)= [0 0 0 0 1 1 1];

Avail(8,Monday,:)= [1 1 1 1 1 1 1];
Avail(8,Tuesday,:)= [1 1 1 1 1 1 1];
Avail(8,Wednesday,:)= [1 1 1 1 1 1 1];
Avail(8,Thursday,:)= [1 1 1 1 1 1 1];
Avail(8,Friday,:)= [1 1 1 1 1 1 1];

Avail(9,Monday,:)= [0 0 0 0 0 0 0];
Avail(9,Tuesday,:)= [0 1 1 1 1 1 0];
Avail(9,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(9,Thursday,:)= [0 0 1 1 1 1 1];
Avail(9,Friday,:)= [0 0 0 0 0 0 0];

Avail(10,Monday,:)= [0 0 0 0 0 0 0];
Avail(10,Tuesday,:)= [0 0 0 0 0 1 1];

Avail(10,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(10,Thursday,:)= [0 0 1 1 0 0 0];
Avail(10,Friday,:)= [0 0 0 0 0 0 0];

Avail(11,Monday,:)= [0 0 0 0 0 0 0];
Avail(11,Tuesday,:)= [0 0 0 0 0 0 0];
Avail(11,Wednesday,:)= [0 0 1 1 0 0 0];
Avail(11,Thursday,:)= [0 0 0 0 0 0 0];
Avail(11,Friday,:)= [0 0 0 0 0 0 0];

Avail(12,Monday,:)= [0 0 0 0 0 0 0];
Avail(12,Tuesday,:)= [0 0 0 0 0 0 0];
Avail(12,Wednesday,:)= [0 0 1 1 0 0 0];
Avail(12,Thursday,:)= [0 0 0 0 0 0 0];
Avail(12,Friday,:)= [0 0 0 1 1 0 0];

Avail(13,Monday,:)= [0 0 0 0 0 0 0];
Avail(13,Tuesday,:)= [1 1 1 0 0 0 0];
Avail(13,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(13,Thursday,:)= [0 0 0 1 1 1 1];
Avail(13,Friday,:)= [0 0 0 0 0 0 0];

Avail(14,Monday,:)= [0 0 0 0 1 1 1];
Avail(14,Tuesday,:)= [0 0 0 0 0 0 0];
Avail(14,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(14,Thursday,:)= [1 1 1 1 1 1 1];
Avail(14,Friday,:)= [0 0 0 0 0 0 0];

Avail(15,Monday,:)= [0 0 0 0 0 0 0];
Avail(15,Tuesday,:)= [0 0 0 0 0 0 0];
Avail(15,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(15,Thursday,:)= [0 1 0 0 0 0 0];
Avail(15,Friday,:)= [1 1 1 1 1 1 1];

Avail(16,Monday,:)= [0 0 0 0 0 0 0];
Avail(16,Tuesday,:)= [0 0 0 0 0 0 0];
Avail(16,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(16,Thursday,:)= [0 0 0 0 0 0 0];
Avail(16,Friday,:)= [1 1 1 1 1 1 1];

Avail(17,Monday,:)= [1 1 1 1 1 1 1];
Avail(17,Tuesday,:)= [0 1 1 1 0 0 0];
Avail(17,Wednesday,:)= [0 0 0 0 1 1 1];
Avail(17,Thursday,:)= [1 1 0 0 0 0 0];
Avail(17,Friday,:)= [0 0 0 0 1 1 0];

Avail(19,Monday,:)= [0 0 1 1 1 1 1];
Avail(19,Tuesday,:)= [0 0 0 0 0 1 1];
Avail(19,Wednesday,:)= [1 1 1 1 1 1 1];
Avail(19,Thursday,:)= [0 0 0 1 1 0 0];
Avail(19,Friday,:)= [1 1 1 0 0 0 0];

Avail(18,Monday,:)= [0 0 1 1 1 0 0];
Avail(18,Tuesday,:)= [0 1 0 0 0 0 0];
Avail(18,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(18,Thursday,:)= [0 0 0 0 0 0 0];
Avail(18,Friday,:)= [0 0 0 0 0 0 0];

Avail(20,Monday,:)= [0 0 0 0 0 0 0];
Avail(20,Tuesday,:)= [0 0 0 0 0 1 1];
Avail(20,Wednesday,:)= [0 0 0 0 0 0 0];
Avail(20,Thursday,:)= [0 0 1 1 0 0 0];
Avail(20,Friday,:)= [0 0 0 0 0 0 0];

Avail(21,Monday,:)= [0 0 0 0 0 0 0];
Avail(21,Tuesday,:)= [0 0 0 0 0 0 0];
Avail(21,Wednesday,:)= [0 0 1 1 0 0 0];
Avail(21,Thursday,:)= [0 0 0 0 0 0 0];
Avail(21,Friday,:)= [0 0 0 0 0 0 0];

Avail(22,Monday,:)= [0 0 0 0 0 0 0];
Avail(22,Tuesday,:)= [0 0 0 0 0 0 0];
Avail(22,Wednesday,:)= [0 0 1 1 0 0 0];
Avail(22,Thursday,:)= [0 0 0 0 0 0 0];
Avail(22,Friday,:)= [0 0 0 1 1 0 0];

Avail(23,Monday,:)= [0 0 0 0 0 0 1];


```

Avail(23,Tuesday,:)= [0 0 0 1 1 0 0];
Avail(23,Wednesday,:)= [0 0 0 0 0 1 1];
Avail(23,Thursday,:)= [0 0 0 0 0 1 1];
Avail(23,Friday,:)= [0 0 0 0 1 1 1];

```

```

classCourseTeacher=zeros(ClassDim,CourseDim);

```

```

%classCourseTeacher(1,Empty)=1;
classCourseTeacher(1,Math)=2;
classCourseTeacher(1,Computer)=3;
classCourseTeacher(1,English)=4;
classCourseTeacher(1,Biologie)=5;
classCourseTeacher(1,History)=6;
classCourseTeacher(1,Arabic)=7;
classCourseTeacher(1,Sport)=8;
classCourseTeacher(1,French)=9;
classCourseTeacher(1,Chemistry)=10;
classCourseTeacher(1,Religion)=11;
classCourseTeacher(1,Physics)=12;
classCourseTeacher(1,Geography)=13;
classCourseTeacher(1,Philosophy)=14;
classCourseTeacher(1,Education)=15;
classCourseTeacher(1,Art)=16;

```

```

%classCourseTeacher(2,Empty)=1;
classCourseTeacher(2,Math)=17;
classCourseTeacher(2,Computer)=3;
classCourseTeacher(2,English)=19;
classCourseTeacher(2,Biologie)=18;
classCourseTeacher(2,History)=6;
classCourseTeacher(2,Arabic)=23;
classCourseTeacher(2,Sport)=8;
classCourseTeacher(2,French)=9;
classCourseTeacher(2,Chemistry)=20;
classCourseTeacher(2,Religion)=21;
classCourseTeacher(2,Physics)=22;
classCourseTeacher(2,Geography)=13;

```

```
classCourseTeacher(2,Philosophy)=14;  
classCourseTeacher(2,Education)=15;  
classCourseTeacher(2,Art)=16;
```

```
%number of sessions for each class and course  
classCourseNumber=zeros(ClassDim,CourseDim);  
classCourseNumber(1,Empty)=0;  
classCourseNumber(1,Math)=6;  
classCourseNumber(1,Computer)=1;  
classCourseNumber(1,English)=7;  
classCourseNumber(1,Biologie)=2;  
classCourseNumber(1,History)=1;  
classCourseNumber(1,Arabic)=6;  
classCourseNumber(1,Sport)=1;  
classCourseNumber(1,French)=2;  
classCourseNumber(1,Chemistry)=2;  
classCourseNumber(1,Religion)=1;  
classCourseNumber(1,Physics)=2;  
classCourseNumber(1,Geography)=1;  
classCourseNumber(1,Philosophy)=1;  
classCourseNumber(1,Education)=1;  
classCourseNumber(1,Art)=1;
```

```
classCourseNumber(2,Empty)=0;  
classCourseNumber(2,Math)=6;  
classCourseNumber(2,Computer)=1;  
classCourseNumber(2,English)=7;  
classCourseNumber(2,Biologie)=2;  
classCourseNumber(2,History)=1;  
classCourseNumber(2,Arabic)=6;  
classCourseNumber(2,Sport)=1;  
classCourseNumber(2,French)=2;  
classCourseNumber(2,Chemistry)=2;  
classCourseNumber(2,Religion)=1;  
classCourseNumber(2,Physics)=2;  
classCourseNumber(2,Geography)=1;  
classCourseNumber(2,Philosophy)=1;
```

```

classCourseNumber(2,Education)=1;
classCourseNumber(2,Art)=1;

end
function initialize()
clear;
global Monday
global Tuesday
global Wednesday
global Thursday
global Friday
global Saturday
global CourseDim
global Empty
global Math
global Computer
global English
global Biologie
global History
global Arabic
global Sport
global French
global Chemistry
global Religion
global Physics
global Geography
global Philosophy
global Education
global Art
global ClassDim
global TeacherDim
global SessionDim
global DayDim
global midX

midX=0.6;% 0.55 has shown very good results

ClassDim=2;
SessionDim=7;

```

```
DayDim=6;  
TeacherDim=23;  
CourseDim=16;
```

```
Monday=1;  
Tuesday=2;  
Wednesday=3;  
Thursday=4;  
Friday=5;  
Saturday=6;
```

```
Empty=1;  
Math =2;  
Computer=3;  
English=4;  
Biologie=5;  
History=6;  
Arabic=7;  
Sport=8;  
French=9;  
Chemistry=10;  
Religion=11;  
Physics=12;  
Geography=13;  
Philosophy=14;  
Education=15;  
Art=16;
```

```
end  
function displayOutput()
```

```
global nonzeroAvail  
global vectorSize  
global courseName  
global ClassDim  
global TeacherDim  
global SessionDim  
global DayDim  
global possibleCombinations
```

```

global classCourseTeacher
global X
global elapsedTime
% display the output schedule
for classCounter=1:ClassDim
    display(sprintf('Class %d:',classCounter));
    for dayCounter=1:DayDim
        str=sprintf('Day %d:',dayCounter);
        dayCoursesNb=0;
        for sessionCounter=1:SessionDim
            for counter=1:vectorSize
                if X(counter)==1
                    if ((nonzeroAvail(counter,3)==dayCounter)&&...
                        (nonzeroAvail(counter,4)==sessionCounter)&&...
                        (nonzeroAvail(counter,1)==classCounter))

                        dayCoursesNb=dayCoursesNb+1;
                        courseNum=nonzeroAvail(counter,2);
                        str=strcat (str ,sprintf('%2d-', ...
                            classCourseTeacher(classCounter, ...
                                courseNum)),courseName(courseNum,1),',');
                    end
                end
            end
        end
        str=strcat (str ,sprintf('Courses Number:%d;',dayCoursesNb));
        display(str);
    end
end
% display elapsed time
display(elapsedTime)
% display backtracking possible combinations number
display(possibleCombinations)

end

```

BIBLIOGRAPHY

- [1] Nocedal, Jorge, and Stephen J. Wright “Interior-Point Methods for Nonlinear Programming” in *Numerical Optimization*, 563-597, second ed. New York: Springer Science+Business Media, LLC., 2006.
- [2] Ferris, Michael C., Olvi L. Mangasarian, and Stephen J. Wright, “Interior-Point Methods,” in *Linear Programming with Matlab*, 195-215, first ed. Philadelphia: Society for Industrial and Applied Mathematics, 2007.
- [3] Chorbev, Ivan, Ivica Dimitrovski, Dragan Mihajlov, and Suzana Loskovska, “Hybrid Heuristics for Solving the Constraints Modeled High School Scheduling Problem,” *The International Conference on “Computer as a Tool”* (September 2007): 2242-2249.
- [4] Ghaemi,, Sehraneh, Mohammad Taghi Vakili, and Ali Aghagolzadeh, *Using a Genetic Algorithm Optimizer Tool to Solve University Timetable Scheduling Problem* (2007).
- [5] Aldasht, Mohamed, Mahmoud Alsaheb, Safa Adi, and Mohammad Abu Qopita, “University Course Scheduling Using Evolutionary Algorithms,” *Fourth International Multi-Conference on Computing in the Global Information Technology* (2009): 47-51.
- [6] Bhaduri, Antariksha, “University Time Table Scheduling using Genetic Artificial Immune Network,” *International Conference on Advances in Recent Technologies in Communication and Computing* (2009): 289-292.
- [7] YuZheng, lingfa Liu, “A Novel Quantum-inspired Genetic Algorithm For a Weekly University Scheduling Optimization,” *International Conference on Information Science and Technology Nanjing, Jiangsu, China* (March 2011): 373-376.
- [8] Oner Adalet, Sel Ozcan, and Derya Dengi, *Optimization Of University Course Scheduling Problem With A Hybrid Artificial Bee Colony Algorithm*, (2011): 339-346.
- [9] Ferdoushi Tania, Prodip Kumer Das, and M.A. H. Akhand, “Highly Constrained University Course Scheduling using Modified Hybrid Particle Swarm Optimization,” *International Conference on Electrical Information and Communication Technology (EICT)* (2013).