



**Automated Timetable Generation - spinTime**

**MS Project Report**

Submitted in Partial Fulfillment

Of the Requirements of the

Degree of

Master of Science (Computer Science)

AT

NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES

LAHORE, PAKISTAN

DEPARTMENT OF COMPUTER SCIENCE

By

**Usman Aftab (08-0964)**

Approved by Committee Members:

Advisor

---

Dr. Muhammad Fakhar Lodhi

Assistant Professors

FAST (NUCES)

# Table of Contents

Table of Contents .....	2
1 Abstract .....	5
2 Introduction .....	5
2.1 Base Algorithm Description [1] .....	6
2.1.1 Phase 1: Construct a feasible timetable .....	6
2.1.2 Phase 2: Optimize the feasible solution .....	8
3 Leverage Point .....	9
4 Constraints .....	10
4.1 Hard Constraints .....	10
4.2 Soft Constraints .....	10
4.3 Common Constraints .....	10
4.3.1 Student .....	10
4.3.2 Teacher .....	11
4.3.3 Class Room .....	11
4.3.4 Lesson .....	11
5 Constraint Builder .....	12
5.1 Gap Constraints .....	12
5.1.1 Operators .....	12
5.1.2 Left Operands .....	12
5.1.3 Right Operands .....	12
5.2 Limit Constraints .....	12
5.2.1 Operators .....	12
5.2.2 Left Operands .....	12
5.2.3 Right Operands .....	13
5.3 General Constraints .....	13
5.3.1 Operators .....	13
5.3.2 Left Operands .....	13
5.3.3 Right Operands .....	13
5.4 Enclosure Constraints .....	13
5.4.1 Operators .....	13

5.4.2	Left Operands.....	13
5.4.3	Right Operands .....	13
5.5	Specific Constraints.....	14
6	Leverage Point Revisited .....	15
7	Results.....	17
8	Future Work .....	18
9	Installation Guide.....	19
10	User Manual.....	20
10.1	Generate Time Table .....	20
10.1.1	Steps.....	20
10.1.2	Notes .....	20
10.2	Specific Constraints .....	21
10.2.1	Description.....	21
10.2.2	Steps.....	21
10.2.3	Notes .....	21
10.3	Gap Constraints .....	23
10.3.1	Description.....	23
10.3.2	Steps.....	23
10.3.3	Notes .....	23
10.4	Limit Constraints .....	24
10.4.1	Description.....	24
10.4.2	Steps.....	24
10.4.3	Notes .....	24
10.5	General Constraints .....	25
10.5.1	Description.....	25
10.5.2	Steps.....	25
10.5.3	Notes .....	25
10.6	Enclosure Constraints .....	27
10.6.1	Description.....	27
10.6.2	Steps.....	27
10.6.3	Notes .....	27

10.7	Migration .....	28
10.7.1	Description .....	28
10.7.2	Steps .....	28
10.7.3	Notes .....	28
11	Input File Formats .....	29
11.1	Department .....	29
11.2	Class Room .....	30
11.3	Timeslot .....	31
11.4	Course .....	32
11.5	Teacher .....	33
12	Output File Format .....	34
12.1.1	Description .....	34
13	Appendix .....	35
13.1	Default Constraints Used in Project .....	35
13.1.1	Students .....	35
13.1.2	Teachers .....	35
13.1.3	Class Rooms .....	35
13.1.4	Lessons .....	35
13.1.5	Timeslots .....	36
14	References .....	37

# 1 Abstract

In this project an approach to deal with dynamic nature of requirements in timetabling problem is implemented. As it goes from one region to another, requirements to generate timetable change. This raises a need to have a general method which can be integrated with already known timetabling algorithms without affecting their original flow and make them even more generic so they can be used anywhere without any adjustment in the code when requirements are changed.

## 2 Introduction

Different institutions have different constraints and specifications on how they want to generate timetable. Some constraints are common that everyone would want but some constraints differ as we move from one region to another or one institution to another considering the infrastructure, staff, students, capacities and other limitations. Variations in constraints and specifications make the timetabling problem even harder. A set of specifications works well for one solution and another set works better for some other solution.

This application aims to provide a solution where changing these specifications would bring minimum changes in the application and it should be easily extendable. One should be able to include/exclude constraints easily to generate timetables.

There are bunch of articles and papers have been published on automated timetable generation algorithms. A paper [1] is chosen to use as basis for the idea implemented in this paper. The aim is to extend it in terms of implementation and flexibility. The base paper uses Simulated Annealing Algorithm with a new Neighborhood Structure for Timetabling Problem to generate timetable and includes some must have constraints. An extension in the algorithm is made so that users can create constraints of their own choice and generate timetables accordingly.

	timeslot 1	timeslot 2	timeslot 3	timeslot 4	timeslot 5
class 1			F	A	B
class 2	B	G		C	
class 3	D	A	C	E	D
class 4	D	E		A	F
class 5	B	F			G

Figure 2.1 - 2D representation

Simulated Annealing Algorithm uses a 2D representation with class rooms on one side and timeslots on the other shown in Fig 2.1. Initially it allocates lessons randomly to different class rooms and timeslots. It then generates a sequence of swaps and changes allocations of lesson to

timeslots. After each swap it checks whether the solution has been improved or not; if it has, the new representation would be used further for future swaps otherwise it would be rejected. However, even the bad timetable is accepted with some probability to avoid local minima.

**Phase 1: Create a feasible timetable**

```

Initialize: allocate all the teachers to the solution matrix
//  $H(\Omega)$  returns the number of hard clashes of solution  $\Omega$ 
 $Hardclashes \leftarrow H(\Omega_c)$ 

while(  $Hardclashes > 0$  and  $T > \text{terminal temperature}$  )
    for  $k \leftarrow 0$  to  $K$  //  $K$  is the number of inner loops
        select a timeslot  $i$  with hard clashes
        select another one  $j$  randomly
        generate a sequence of swaps between  $i$  and  $j$ 
        for each swap  $m$  in the sequence
             $\Omega_n \leftarrow \text{perform } m \text{ on } \Omega_c$ 
             $newHardclashes \leftarrow H(\Omega_n)$ 
             $\Delta \leftarrow newHardclashes - Hardclashes$ 
        //  $Rand()$  returns a random real number between 0 and 1
        if(  $\Delta \leq 0$  or  $e^{-\frac{\Delta}{T}} > Rand()$  )
             $Hardclashes \leftarrow newHardclashes$ 
             $\Omega_c \leftarrow \Omega_n$ 
         $T \leftarrow \alpha * T$  //  $\alpha$  is the temperature cooling rate
    Go to Phase 2

```

Figure 2.2 - Pseudo-code of Phase 1

## 2.1 Base Algorithm Description [1]

There are two phases in our algorithm. The first phase aims to construct a feasible solution. The second phase concentrates the effort on improving the quality of the timetable while maintaining feasibility. The Pseudo-code of the algorithm is showed in Fig 2.2

### 2.1.1 Phase 1: Construct a feasible timetable

When initializing the solution, we only consider following 2 hard constraints.

- Every teacher must teach the exact number of lessons specified.
- Every class must attend the exact number of lessons specified.

Then allocate all the teachers to the matrix. For each teacher, according to the number of teaching hours specified on the class, the teacher is allocated to the same number of positions in the corresponding row. One position of the matrix can be only occupied by one teacher.

We adopt a stochastic strategy in this phase. At each inner loop of SA, we randomly select a timeslot where the hard clashes happen, and randomly select another one (no matter whether it violates hard constraints or not) from the rest timeslots. Then we generate a sequence of swaps between these two selected timeslots and try to perform every swap in this sequence one by one. The experiment showed that this approach rarely fails in producing a feasible solution, and its average performance is higher than many heuristic construction algorithms. In Figure 2.2, we present the pseudo-code of creating a feasible timetable in phase 1.

**Phase 2: Optimize the feasible solution**

```

//  $S(\Omega)$  returns the number of soft clashes of solution  $\Omega$ 
Softclashes  $\leftarrow S(\Omega_c)$ 
while(  $T >$  terminal temperature)
  for  $k \leftarrow 0$  to  $K$  //  $K$  is the number of inner loops
    add all the timeslots in candidate list
    clear the tabu list
    for each of the timeslot  $i \neq j$  in candidate list
      generate a sequence of swaps between  $i$  and  $j$ 
      for each swap  $m$  in the sequence
         $\Omega_n \leftarrow$  perform  $m$  on  $\Omega_c$ 
        if(  $H(\Omega_n) = 0$  )
          newSoftclashes  $\leftarrow S(\Omega_n)$ 
           $\Delta \leftarrow$  newSoftclashes  $-$  Softclashes
          if(  $\Delta \leq 0$  or  $e^{-\frac{\Delta}{T}} > Rand()$  )
            Softclashes  $\leftarrow$  newSoftclashes
             $\Omega_c \leftarrow \Omega_n$ 
    remove  $j$  from candidate list and add it into the tabu list
   $T \leftarrow \alpha * T$  //  $\alpha$  is the temperature cooling rate
Output: return the final solution

```

Figure 2.3 - Pseudo-code of Phase 2

### **2.1.2 Phase 2: Optimize the feasible solution**

Once a feasible timetable is found, we turn to reduce the soft clashes. It is not allowed to bring hard clashes in this phase.

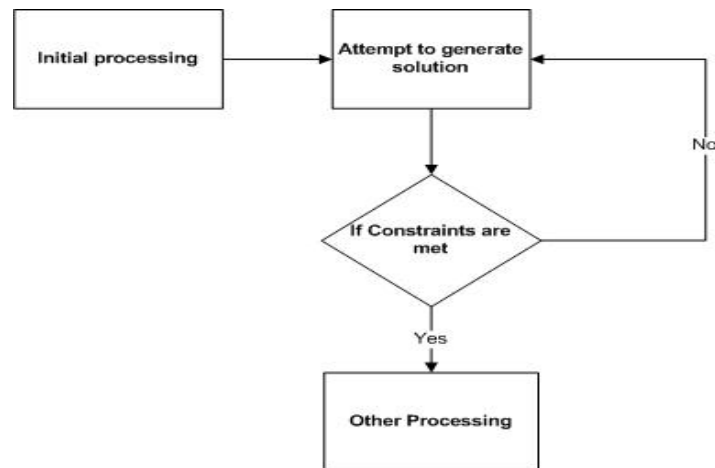
We adopt a tabu search strategy in this phase. A tabu list is used to lock the selected timeslots and a candidate list is used to contain the rest timeslots. At each inner loop, the tabu list is cleared, and all the timeslots are randomly sorted and added into the candidate list. At each iteration of inner loop, the first timeslot of candidate list is selected and coupled with one of the other in the candidate list, and then we generate a sequence of swaps between these two coupled timeslots and try to perform all the swaps in this sequence one by one while maintaining the feasibility of the timetable.

After this iteration, the first timeslot is removed from candidate list and added into the tabu list. When there is only one timeslot left in the candidate list, a new inner loop starts. The pseudo-code of phase 2 is shown in Figure 2.3.



### 3 Leverage Point

A commonality has been observed in most of the AI algorithms suggested to generate timetables. They all do some processing, then at a point in time check constraints and feasibility of solution [1] [2]. If the constraints are satisfied solution will be accepted otherwise it will start over, repeat the processing and check the constraints again. The scenario is depicted in Fig 3.1.



**Figure 3.1 - General scenario in all AI algorithms to generation timetables.**

The point where these algorithms check to see the feasibility of timetable generated so far is our leverage point. To understand the significance of this controlling point let's first talk about constraints then we will revisit the leverage point and how we can accommodate our goals within the algorithm.

## 4 Constraints

There are various definitions [3] for the timetabling problem. A popular one is that timetabling problem deals with allocating a set of events (lessons, exams) to some limited resources, such as rooms and timeslots (hours), subject to a series of constraints. Mostly these constraints are grouped into 2 categories.

### 4.1 Hard Constraints

Hard Constraints are the ones that are to be catered in any case without which the timetable is not acceptable. These are compulsory constraints and one's system must take care of these constraints in the solution. Violations of hard constraints are called hard clashes.

### 4.2 Soft Constraints

Soft constraints on the other hand are the ones that tell the quality of a timetable. Timetable would still be acceptable if any of these constraints is not incorporated in the solution. However, it would not be a very good solution if lots of constraints are violated. Violations of soft constraints are called soft clashes.

There are no. of constraints that can be incorporated in a timetable. One has to carefully choose which should be hard or soft constraint. This selection process can affect the generation of timetable in as much good way as much it can in bad way. Moreover, there should not be a conflict between any two hard constraints; the system can be stuck in a loop consequently.

The more hard constraints you add up to the problem the more time it would need to generate an acceptable timetable. So, one should pick only those hard constraints that are necessary to have in the final solution otherwise we can categorize them into soft constraints and system will try to remove as many of them as possible.

### 4.3 Common Constraints

There are five different entities on which the constraints can be applied:

1. Student
2. Teacher
3. Class Room
4. Lesson
5. Timeslot

Constraints related to each entity are outlined below:

#### 4.3.1 Student

- a. No student can attend two or more lessons in the same timeslot.

- b. Each student should have a gap of at least one timeslot between two or given no. of lessons.
- c. Each student should not have a gap of more than certain no. of timeslots in a day.
- d. Each student should have lessons on four or given no. of days in a week.

#### **4.3.2 Teacher**

- a. No teacher can teach two or more lessons in the same timeslot.
- b. Every teacher must work in his/her available timeslots.
- c. Every teacher must teach no. of lessons specified in a day.
- d. Every teacher should have a gap of at least one timeslot between two or given no. of lessons.
- e. Every teacher should not have gap of more than certain no. of timeslots in a day.
- f. Each teacher should have lessons on four or given no. of days in a week.

#### **4.3.3 Class Room**

- a. No class room can have two or more lessons in the same timeslot.
- b. Each class room must have capacity greater or equal to the class strength.
- c. Each class room should be free during one or given no. of timeslots in a week.

#### **4.3.4 Lesson**

- d. Each lesson must be taught in its respective class room. (Labs for example.)
- e. No lesson can be taught twice or more times in a day.

## 5 Constraint Builder

One can pick constraints of their choice and incorporate it in the system mentioned in section 4.3 either as hard or soft depending upon the circumstances. This choice of taking one constraint as hard or soft may differ person to person. Also, one might not want to use all the constraints mentioned earlier. Nonetheless, there can be lot many constraints not specified and perhaps have not seen yet.

If seen closely, aforementioned constraints can be categorized in four different classes. Each category can be viewed as a combination of operators and operands. All operators are binary but a class may contain more than one operators. Classes' short description along with their operators and operands are discussed below:

### 5.1 Gap Constraints

Type of constraints that define timeslot gaps between entities:

#### 5.1.1 Operators

- a. Gap

#### 5.1.2 Left Operands

- a. Student
- b. Teacher
- c. Lesson
- d. Class Room

#### 5.1.3 Right Operands

- a. Timeslot
- b. Day

### 5.2 Limit Constraints

Type of constraints that limit the occurrences of entities in timeslots:

#### 5.2.1 Operators

- a. Limit

#### 5.2.2 Left Operands

- a. Student
- b. Teacher
- c. Lesson
- d. Class Room

### **5.2.3 Right Operands**

- a. Timeslot
- b. Day
- c. Week

## **5.3 General Constraints**

Type of constraints that compares properties of entities:

### **5.3.1 Operators**

- a. Equal
- b. Not Equal
- c. Less Than
- d. Less Than & Equal
- e. Greater Than
- f. Greater Than & Equal

### **5.3.2 Left Operands**

- a. Teacher
- b. Lesson

### **5.3.3 Right Operands**

- a. Timeslot
- b. Class Room

## **5.4 Enclosure Constraints**

Type of constraints that define preferable allocation of entities:

### **5.4.1 Operators**

- a. Inclusion
- b. Exclusion

### **5.4.2 Left Operands**

- a. Teacher
- b. Lesson

### **5.4.3 Right Operands**

- a. Timeslot
- b. Class Room

Apart from these constraint types we can add another category which can cater exceptional cases.

## **5.5 Specific Constraints**

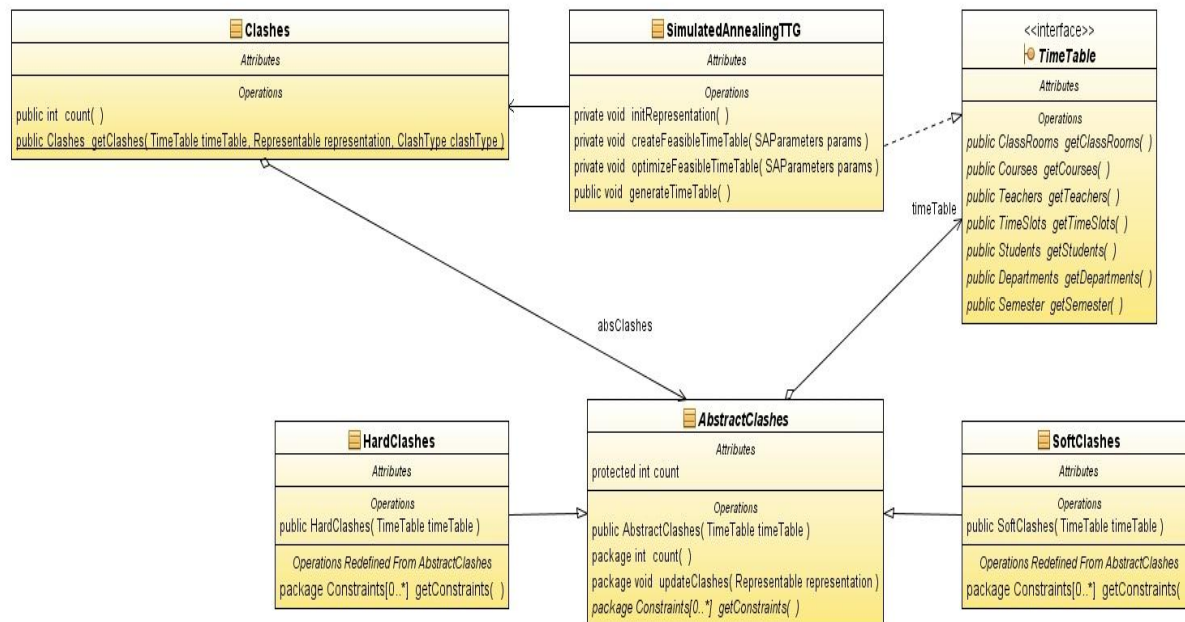
This constraint type is used to fix lessons to particular timeslots or classrooms or both. Algorithm needs to make special arrangements to provide this type of constraints.

However, with given classification one can easily construct a constraint builder which provides a mechanism to allow users create constraints of their own choices and save them in a database. Whenever required constraints can be modified or removed. Moreover a sequence to which constraints should be applied can also be provided. Also, at any point in time users can specify through the constraint builder which constraint to consider in creation of timetable and which to not.

## 6 Leverage Point Revisited

Now that constraint types and their construction are discussed, it is easy to see how we can integrate it with our algorithm and make it immune to dynamic requirements.

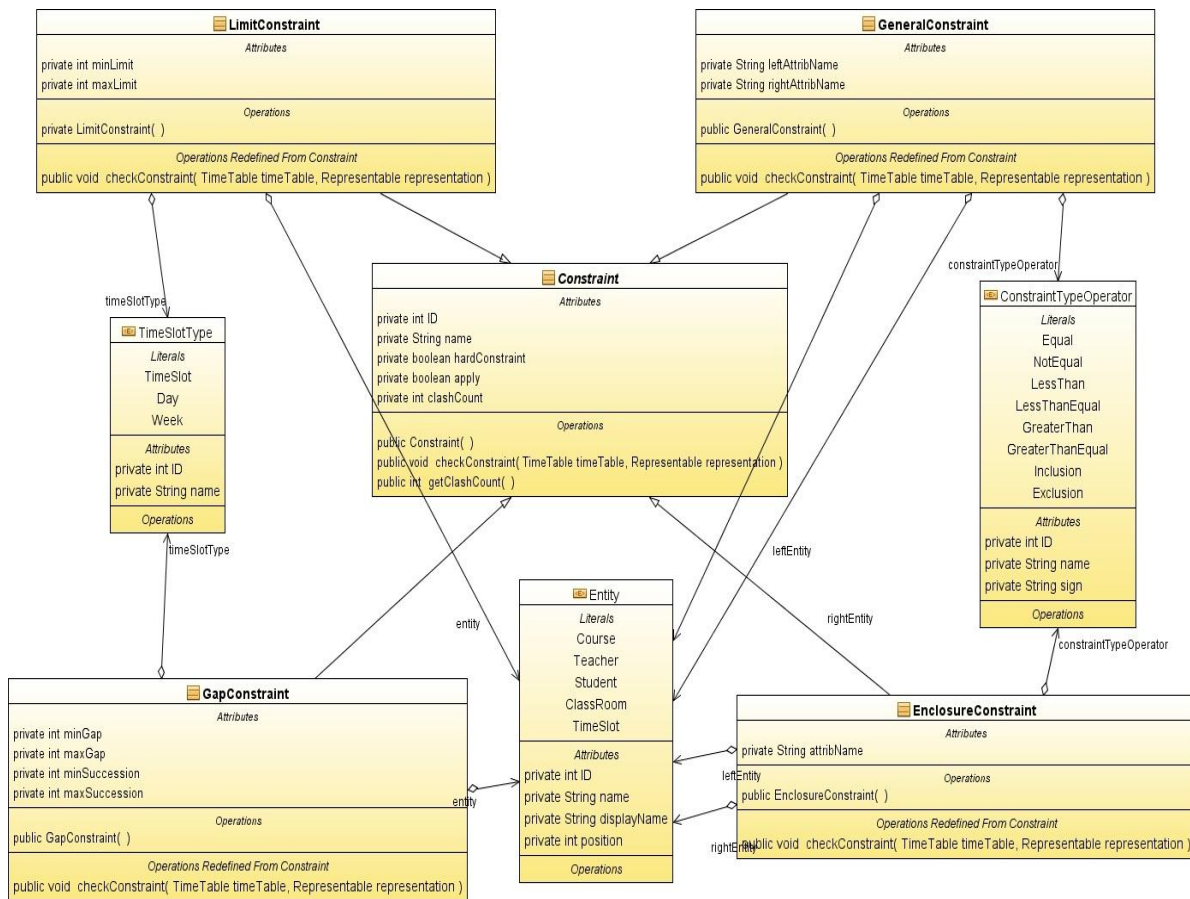
Since, all constraint types (except for specific constraints) have two operands against each operator so it is possible to use concepts of polymorphism and inheritance to isolate main algorithm's logic with the point where we check whether all constraints are met or not. Class diagram shown in Fig 6.1 can explain things better which is used on top of Simulated Annealing Algorithm.



**Figure 6.1 - Class Diagram to integrate Constraint into existing algorithm.**

After integration of the idea Simulated Annealing Algorithm works exactly the way before, all it has to do is to get clashes and check the count. Rest of the working remains intact. Further, Clashes class is implemented on player role pattern to make main algorithm independent of the fact that there are any soft or hard constraints and we can always add more types of clashes to it.

Fig 6.2 puts some light on how different classes of constraints are implemented, what attributes have been used and how they access entities to carry out their purpose. They all are inherited from Constraint class and use different classes of their need accordingly. All information regarding entities can be accessed through the Entity class. The interesting point here is users would be allowed to create custom properties of their choice and save them in database. Later on, we can save those properties in hash tables, hashed against their property names. Hence General constraints and Enclosure constraints provide a very good flexibility and make the system immune to dynamic nature of requirements.



**Figure 6.2 - Class Diagram to show how different constraint types are implemented, what attributes have used and how access entities to achieve goals.**



## 7 Results

Strategy discussed above have also been implemented and tested on a good real time university dataset with different constraints. Dataset's statistics are outlined below in Table 1. Average time to generate timetable without soft constraints has been observed around 1 to 2 minutes. On the other hand if soft constraints are also used average time increases 10 to 15 minutes.

Table 1. DATASET STATISTICS.

Class Rooms	36
Teachers	135
Students	2000
Average courses took per students	5
Courses	110
Average sections per course	3
Departments	4

## 8 Future Work

The model presented is used for only timetable generation algorithms and entities. Nonetheless, it has very much potential to deal with scheduling problems on the whole. If we can identify entities along with their dependencies and map them to respective constraint type we can generate a schedule using the same model.

## 9 Installation Guide

Follow the steps listed below to install and run the application.

1. Install xampp Server.
  - a. Start xampp control panel.
  - b. Start Apache server.
  - c. Start MYSQL server.
  - d. Start your browser and open url: <http://localhost/phpmyadmin/>
  - e. Create a new database “timetable” (You can name the database of your choice but make sure you use it accordingly in the guide ahead.)
2. Copy source code from the disc to any folder.
3. Configure config.txt
  - a. Set **db.url**= jdbc:mysql:///timetable (timetable is name of the database).
  - b. Set **db.user**= <your apache server username default is “root”>
  - c. Set **db.password**= <your apache server password default is empty>
4. Run initial DB script
  - a. In your browser open url: <http://localhost/phpmyadmin/>
  - b. Select “timetable” option from the left panel.
  - c. Select “import” tab.
  - d. Click browse and open file under your source code:  
    \dal\dal\timetable\db\script\MySQL-create.sql
  - e. The script should execute with no errors.
5. Install and setup netbeans
  - a. Install Java JDK (latest version.)
  - b. Install netbeans (latest version.)
  - c. Start netbeans and open existing project and select the folder you copied.
  - d. Make sure you have following folders listed in your project pane;
    - i. business
    - ii. com
    - iii. dal
    - iv. gui
    - v. migration
    - vi. test
    - vii. Libraries
6. You are set to go, run the application. See [User Manual](#) section to know how to create timetables. (Note: before you can create timetables, you first need to perform migration of required entities and constraints.)

## 10 User Manual

Manual is organized according to the menu options you can see to the left of your main window.

### 10.1 Generate Time Table

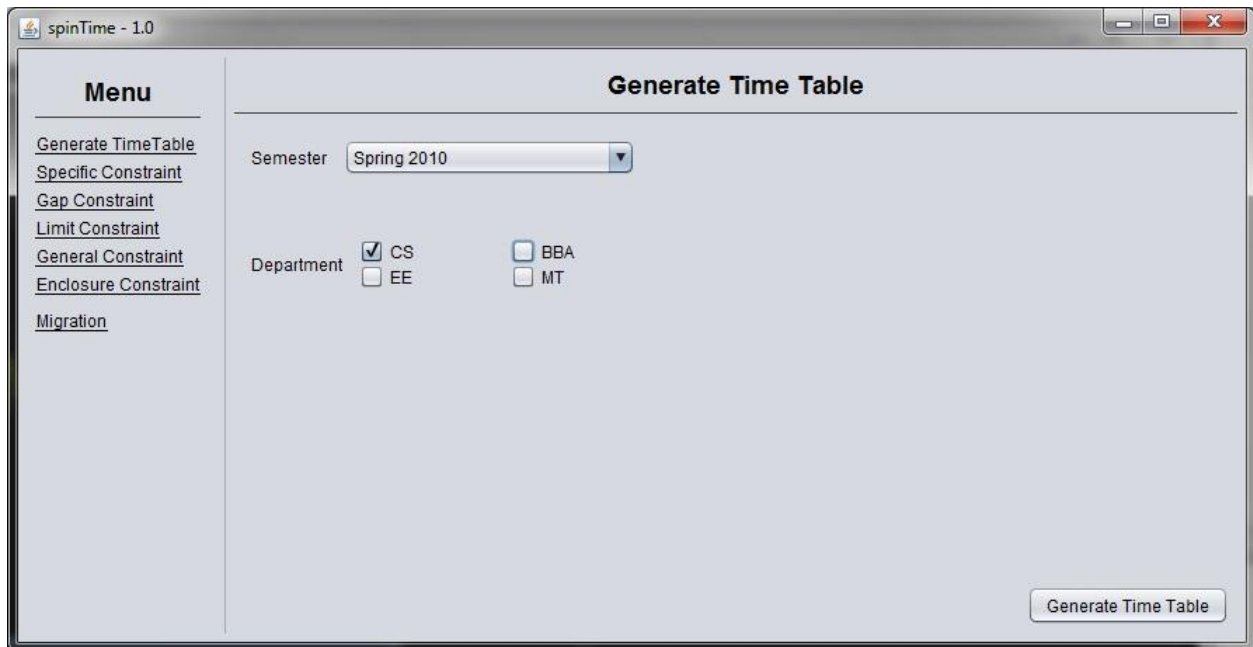


Figure 10. 1.1 - Generate Time Table

#### 10.1.1 Steps

1. Check department checkbox for which you want to generate the timetables.
2. Click Generate Time Table button.
3. Application will save the timetable in DB and generate an excel file for you at the root of you source code folder with name timetable.xls.

#### 10.1.2 Notes

1. While the application is creating timetable you can click the cancel button and stop processing.
2. Application will remove previously generated timetables and save new ones in the DB only for departments with marked checkboxes.

## 10.2 Specific Constraints

The screenshot shows the 'spinTime - 1.0' application window. On the left is a 'Menu' sidebar with links: 'Generate TimeTable', 'Specific Constraint', 'Gap Constraint', 'Limit Constraint', 'General Constraint', 'Enclosure Constraint', and 'Migration'. The main window is titled 'Specific Constraint'. It features a list box on the left (currently empty) and a form on the right. The form includes: 'ID\*' with value '-1', 'Name\*' with value 'Specific Constraint', and checkboxes for 'Hard Constraint' and 'Apply'. Below these are four rows of input fields: 'Course\*', 'Lesson No.\*' (a dropdown menu), 'Class Room', and 'Timeslot'. Each of these four rows has a 'Browse' button and a 'Reset' button to its right. At the bottom right of the form are two buttons: 'New Constraint' and 'Save'.

Figure 10.2.1 - Specific Constraints

### 10.2.1 Description

Specific constraints are used to bind courses to a class room or timeslot or both. No matter what other constraints you specify these constraints will be bound to the timetable in the beginning of the timetable generation process and will not be changed throughout all the processing. So, you have to make sure that these constraints don't conflict; either with each other and with rest of the constraints classes.

If there is a scenario that requires your timetable to always have a course's lesson in a particular class room to time slot or both you can use specific constraints.

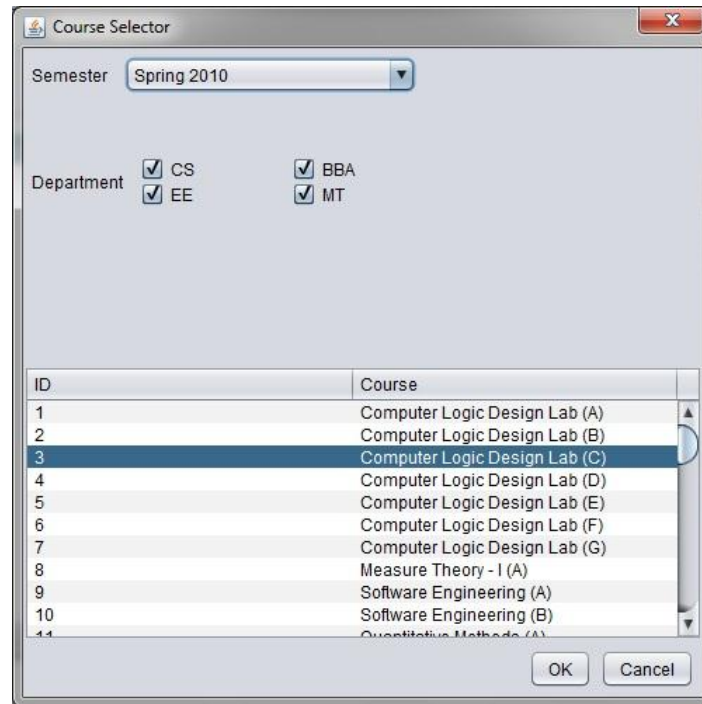
### 10.2.2 Steps

1. Click on 'New Constraint' button to enable text options.
2. Click on 'Browse' button against Course and a selector will be popped up shown in figure 10.2.2.
3. Select lesson no.
4. Select Class Room (optional).
5. Select Timeslot (optional).
6. Save.

### 10.2.3 Notes

1. To understand specific constraint in detail, see section 5.5

2. Class Room and Timeslot options are optional but you have to provide at least one of them.
3. These constraints will not be applied on already generated timetables.
4. Make sure there are no conflicts with other specific constraints.
5. Make sure specific constraints don't conflict with other constraint classes as well. i.e gap constraints, limit constraints, general constraints, and enclosure constraints.



**Figure 10.2.2 - Course Selector**

## 10.3 Gap Constraints

The screenshot shows a window titled "spinTime - 1.0" with a "Menu" on the left and a "Gap Constraint" configuration area on the right. The menu includes: Generate TimeTable, Specific Constraint, Gap Constraint, Limit Constraint, General Constraint, Enclosure Constraint, and Migration. The "Gap Constraint" area contains a list of entities with "teacher" selected. Below the list, there are fields for ID\* (5), Name\* (teacher), and checkboxes for Hard Constraint and Apply. There are also dropdown menus for Entity\* (Teacher) and Timeslot type\* (TimeSlot). At the bottom, there are input fields for Min Gap (1), Max Gap (3), Min Succession (-1), and Max Succession (2). Buttons for "New Constraint" and "Save" are at the bottom right.

Figure 10.3.1 - Gap Constraint

### 10.3.1 Description

Gap Constraints are used to control entity's gap between timeslots/days.

In the figure 10.3.1 'teacher' constraint says that whenever a teacher has a gap between 2 allocated timeslots (timeslots in a day) it should be at least 1 and at most 3. In simpler words the gap of a lecture after another should not be more than 3. The constraint also says whenever a teacher has allocated timeslots in succession it should not be more than 2. In simpler words no teacher should teach more than 2 lectures consecutively.

### 10.3.2 Steps

1. Click 'New Constraint' button.
2. Select an entity from the entity drop down menu.
3. Select a timeslot type from the timeslot type drop down menu.
4. Provide values for 'Min Gap', 'Max Gap', 'Min Succession', and 'Max Succession'. If you don't want to mention a value use '-1'.
5. Save.

### 10.3.3 Notes

1. There are few gap constraint already defined for you. You can change them and create new gap constraints however be sure that you don't repeat them. It'll generate the time table anyways but might affect the performance.

## 10.4 Limit Constraints

The screenshot shows a software window titled "spinTime - 1.0". On the left is a "Menu" panel with links: "Generate TimeTable", "Specific Constraint", "Gap Constraint", "Limit Constraint", "General Constraint", "Enclosure Constraint", and "Migration". The "Limit Constraint" link is selected. The main area is titled "Limit Constraint" and contains a list of constraints: "teacher timeslot" (selected), "teacher day", "student timeslot", "student day", and "course day". To the right of the list are input fields: "ID\*" with value "1", "Name\*" with value "teacher timeslot", "Entity \*" with a dropdown menu showing "Teacher", "Timeslot type \*" with a dropdown menu showing "TimeSlot", "Min Limit" with value "-1", and "Max Limit" with value "1". There are checkboxes for "Hard Constraint" and "Apply", both of which are checked. At the bottom right are buttons for "New Constraint" and "Save".

Figure 10.4.1 - Limit Constraint

### 10.4.1 Description

Limit constraints are used to limit/control entity's occurrences in timeslots/day/week.

In figure 10.4.1 'teacher timeslot' constraint says that every teacher is allowed only 1 occurrence in a particular timeslot of a day.

### 10.4.2 Steps

1. Click 'New Constraint' button.
2. Select entity from the entity drop down menu.
3. Select timeslot type from the timeslot type drop down menu.
4. Provide 'Min Limit' and 'Max Limit'. If you don't want to mention a value use '-1'.
5. Save.

### 10.4.3 Notes

1. There are few limit constraint already defined for you. You can change them and create new limit constraints however be sure that you don't repeat them. It'll generate the time table anyways but might affect the performance.



## 10.5 General Constraints

The screenshot shows the 'General Constraint' dialog box in the 'spinTime - 1.0' application. The dialog is divided into a 'Menu' on the left and a main configuration area on the right. The 'Menu' contains links to 'Generate TimeTable', 'Specific Constraint', 'Gap Constraint', 'Limit Constraint', 'General Constraint', 'Enclosure Constraint', and 'Migration'. The main area is titled 'General Constraint' and contains the following fields and controls:

- ID\***: 10
- Name\***: course duration
- Operator\***: Equal (dropdown menu)
- Left Entity\***: Course (dropdown menu)
- Right Entity\***: TimeSlot (dropdown menu)
- Left Entity Attribute\***: duration (dropdown menu)
- Right Entity Attribute\***: duration (dropdown menu)
- Left Entity Attribute Type\***: Time (dropdown menu)
- Right Entity Attribute Type\***: Time (dropdown menu)
- Hard Constraint**: ☒
- Apply**: ☒
- New Constraint**: button
- Save**: button

Figure 10.5.1 - General Constraint

### 10.5.1 Description

General constraints are defined to apply different arithmetic operations on entities.

Consider a case that you want each timeslot allocated to a course should be equal to the course duration defined in the database. You can achieve this by applying general constraint as shown in figure 10.5.1. This is a sort of constraint that everyone would want in their timetable that's why it's already been created for you. You can define as many properties as you want in the migration process explained in section 10.7 and 11.

### 10.5.2 Steps

1. Click 'New Constraint' button.
2. Select the operator you from the operator drop down menu.
3. Select left entity
4. Select left entity property
5. Select right entity
6. Select right entity property
7. Save.

### 10.5.3 Notes

1. Attribute type of both entities should match.

2. You can provide/create new properties in the migration process elaborated in section 10.7 and 11.
3. Only single value properties are shown in general constrain window. For multi-value properties consult section 10.6.

## 10.6 Enclosure Constraints

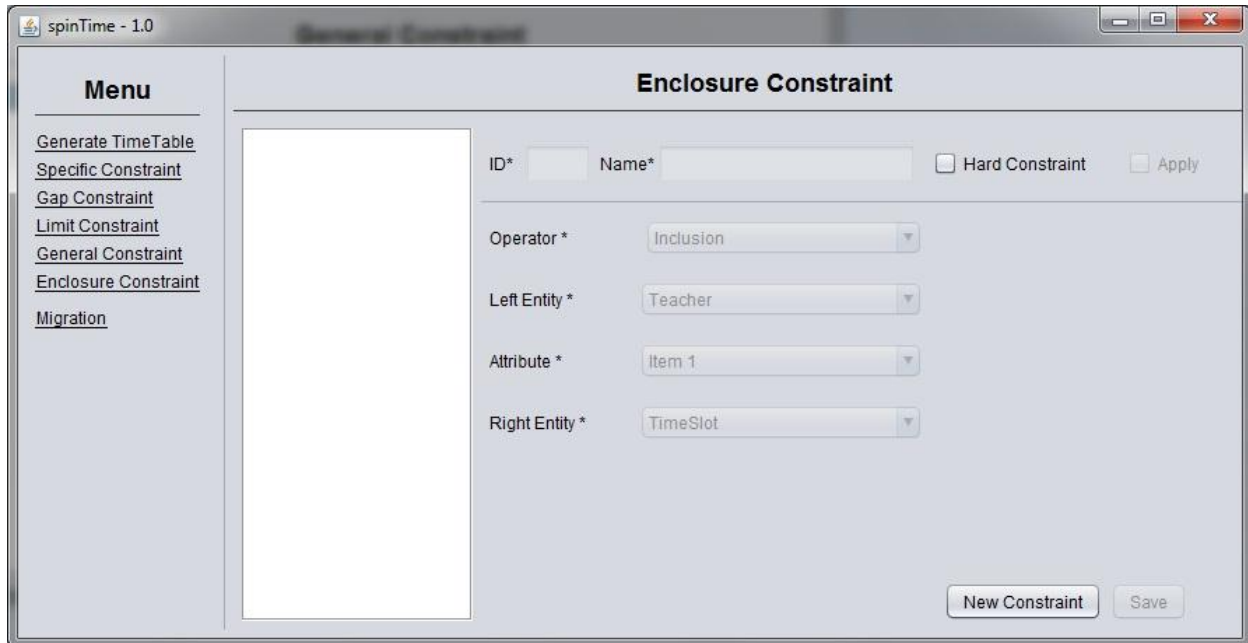


Figure 11.6.1 - Enclosure Constraint

### 10.6.1 Description

Enclosure constraints are defined for multi-value properties only.

Consider a case that you want all teachers to teach lesson in their available timeslots. You define their available timeslots in the migration process and then 'available timeslot' multi-value property would start showing up in Enclosure Constraint window after you restart the application. Create a constraint and save it as you do in rest of the constraint types and then generate timetable.

### 10.6.2 Steps

1. Click 'New Constraint' button.
2. Select Operator type from the operator drop down menu.
3. Select Left entity
4. Select attribute
5. Select Right entity
6. Save.

### 10.6.3 Notes

1. Entity's attribute type and Right entity should match
2. You can provide/create new multi-value properties in the migration process elaborated in section 10.7 and 11.
3. Only multi-value properties are shown in enclosure constrain window.

## 10.7 Migration

The screenshot shows a software window titled "spinTime - 1.0". On the left is a "Menu" sidebar with the following items: [Generate TimeTable](#), [Specific Constraint](#), [Gap Constraint](#), [Limit Constraint](#), [General Constraint](#), [Enclosure Constraint](#), and [Migration](#). The "Migration" menu item is selected. The main window area is titled "Migration" and contains the following fields and buttons:

- Semester \* (text input field)
- Department \* (text input field) with a "Browse" button to its right.
- Class Room \* (text input field) with a "Browse" button to its right.
- Timeslot \* (text input field) with a "Browse" button to its right.
- Course \* (text input field) with a "Browse" button to its right.
- Teacher \* (text input field) with a "Browse" button to its right.
- A "Start Migration" button at the bottom left.

Figure 11. 7.1 - Migration

### 10.7.1 Description

To populate data in the database you need to go through the migration process. Here you provide data in excel files in specific format and the application would do the rest for you. Consult section 11 for input file formats.

### 10.7.2 Steps

1. Fill in all the information and then click start migration button.
2. Restart applications for the changes to take effect.

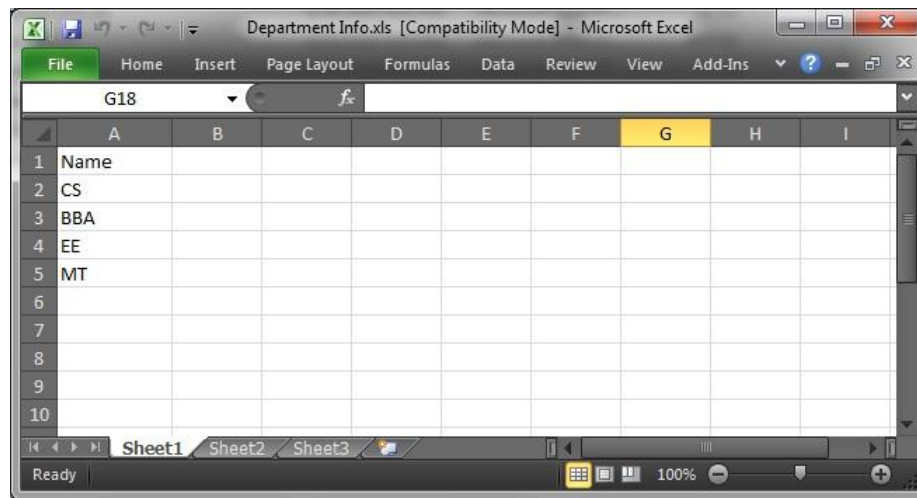
### 10.7.3 Notes

1. You'll you have provide all the information at once.
2. You can provide information only semester wise.
3. Input data contains information about all the departments.
4. Upon migration of data for a semester, has already time table generated and saved in DB; all previous timetables will be removed.
5. Migrations module will itself distribute students to different sections of courses provided.
6. See input file format section 11 to understand the input format.

# 11 Input File Formats

First line of all input files is header of the file. You define values to the header columns below them. Each input file is assumed to provide data for one semester. You don't have to provide data that has already been given in previous semesters. For example, you will only have to provide new class rooms, new departments, and new timeslots. But you ll always have to provide courses and teacher since they also contain the information about when courses are taught by who and students enrolled in courses. Make sure you use the same name for teachers and courses that you have used before.

## 11.1 Department



	A	B	C	D	E	F	G	H	I
1	Name								
2	CS								
3	BBA								
4	EE								
5	MT								
6									
7									
8									
9									
10									

Figure 11.1.1 - Department Input File Format.

1. **Mandatory Column:** Name
2. **Single Value Properties:** Not Allowed.
3. **Multi-Value Properties:** Not Allowed.

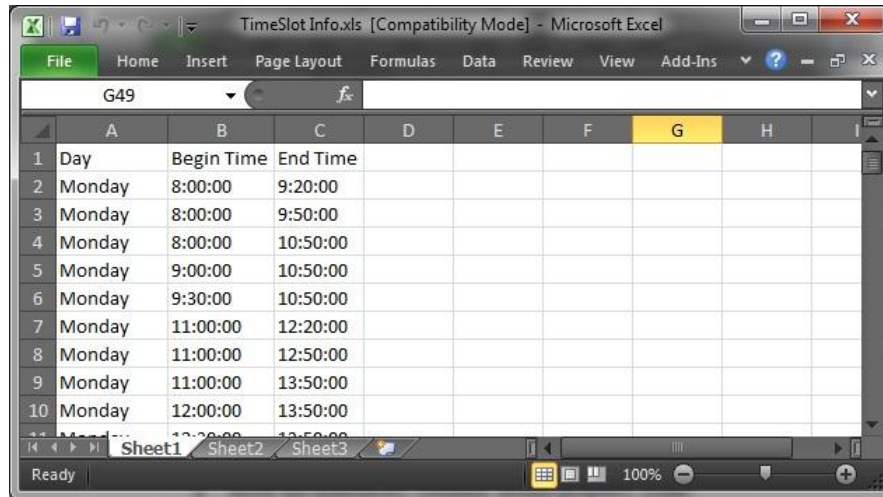
## 11.2 Class Room

	A	B	C	D	E	F	G	H	I
1	Name	Department	Capacity	property count	SV1	String	MV1	TimeSlot	
2	M1	CS	50	1	m1 sv1		m1 mv1 1		
3							m1 mv1 2		
4									
5	M2	CS	50						
6					m2 sv1		m2 mv1 1		
7							m2 mv1 2		
8	M3	CS	50						
9	M4	CS	60						
10	M5	CS	50						

Figure 11.2.1 - Class Room Input File Format.

1. **Mandatory Column:** Name, Department, Capacity
2. **Single Value Properties:**
  - a. After the mandatory columns define property count.
  - b. For each property you are required to fill two columns; property name and property type. Example shown in figure 11.2.1.
  - c. Available Property types are (case sensitive): Integer, Double, String, Time, Day
  - d. You have to mention the property type only once in the header in the column next to property name as shown in the figure 11.2.1.
3. **Multi-Value Properties:**
  - a. After Single value properties define multi-value properties.
  - b. If there are no single value-properties mention '0' in property count column.
  - c. As like single value properties, you are required to fill in 2 columns for each multi-value property; property name and property type. Example shown in figure 11.2.1.
  - d. Available Property types are (case sensitive): TimeSlot.
  - e. You have to mention the property type only once in the header in the column next to property name as shown in the figure 11.2.1.
  - f. **Important HACK:** In multi-value properties the values are timeslot Ids. For now I am afraid you'll have to go to the database and see timeslot ids. And if you don't know how to see database values go to specific constraints and browse timeslots and there you can see timeslot ids.

## 11.3 Timeslot

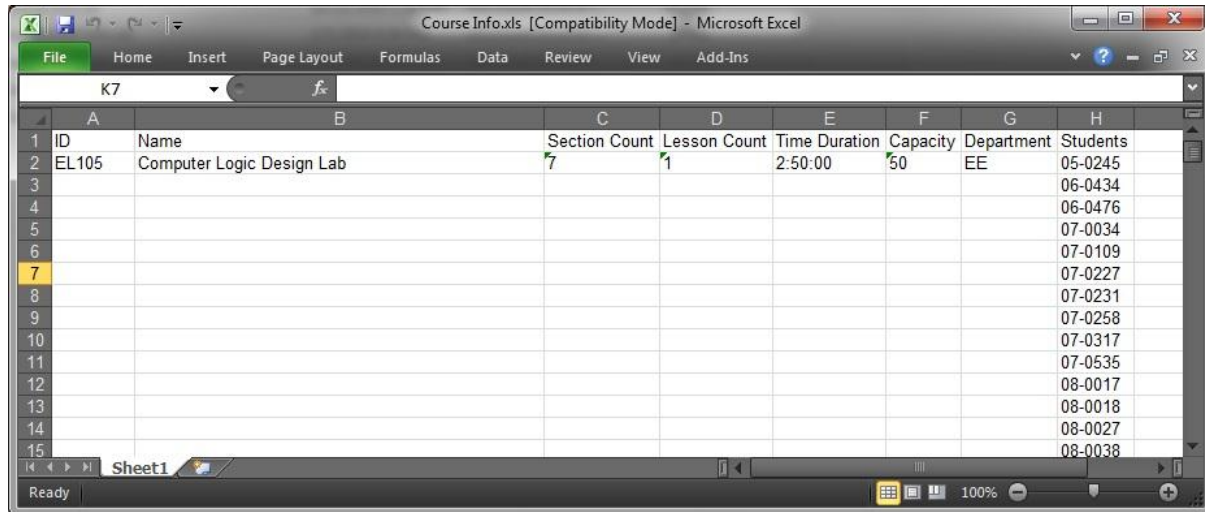


	A	B	C	D	E	F	G	H	I
1	Day	Begin Time	End Time						
2	Monday	8:00:00	9:20:00						
3	Monday	8:00:00	9:50:00						
4	Monday	8:00:00	10:50:00						
5	Monday	9:00:00	10:50:00						
6	Monday	9:30:00	10:50:00						
7	Monday	11:00:00	12:20:00						
8	Monday	11:00:00	12:50:00						
9	Monday	11:00:00	13:50:00						
10	Monday	12:00:00	13:50:00						

Figure 11.3.1 - Timeslot Input File Format.

1. **Mandatory Column:** Day, Begin Time, End Time
2. **Single Value Properties:** Allowed (see section 11.2 for details you can provide single value properties the same way).
3. **Multi-Value Properties:** Not Allowed.

## 11.4 Course



The screenshot shows a Microsoft Excel spreadsheet titled 'Course Info.xls [Compatibility Mode]'. The spreadsheet has the following columns: ID, Name, Section Count, Lesson Count, Time Duration, Capacity, Department, and Students. The data is as follows:

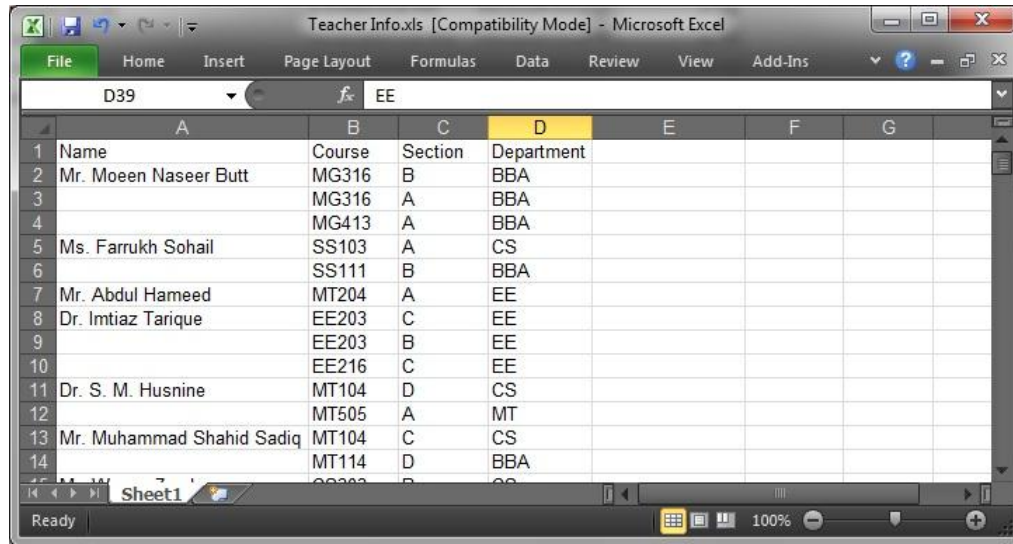
ID	Name	Section Count	Lesson Count	Time Duration	Capacity	Department	Students
EL105	Computer Logic Design Lab	7	1	2:50:00	50	EE	05-0245
							06-0434
							06-0476
							07-0034
							07-0109
							07-0227
							07-0231
							07-0258
							07-0317
							07-0535
							08-0017
							08-0018
							08-0027
							08-0038

Figure 11.4.1 - Course Input File Format.

1. **Mandatory Column:** ID, Name, Section Count, Lesson Count, Time Duration, Capacity, Department, Students.
2. **Single Value Properties:** Allowed (see section 11.2 for details you can provide single value properties the same way).
3. **Multi-Value Properties:** Allowed (see section 11.2 for details. You can provide multi-value properties the same way).
4. **Note:** Migration process will divide and create course sections for you. You don't have to mention them here.



## 11.5 Teacher



	A	B	C	D	E	F	G
1	Name	Course	Section	Department			
2	Mr. Moeen Naseer Butt	MG316	B	BBA			
3		MG316	A	BBA			
4		MG413	A	BBA			
5	Ms. Farrukh Sohail	SS103	A	CS			
6		SS111	B	BBA			
7	Mr. Abdul Hameed	MT204	A	EE			
8	Dr. Imtiaz Tarique	EE203	C	EE			
9		EE203	B	EE			
10		EE216	C	EE			
11	Dr. S. M. Husnine	MT104	D	CS			
12		MT505	A	MT			
13	Mr. Muhammad Shahid Sadiq	MT104	C	CS			
14		MT114	D	BBA			

Figure 12.1.1 - Teacher Input File Format.

1. **Mandatory Column:** Name, Course, Section, Department
2. **Single Value Properties:** Allowed (see section 11.2 for details you can provide single value properties the same way).
3. **Multi-Value Properties:** Allowed (see section 11.2 for details. You can provide multi-value properties the same way).
4. **Note:** Since you cannot define course sections in course input file and here you have to provide sections so you can simply assume A → section 1, B → section 2 so on and so forth.

## 12 Output File Format

Output file will be generated in the root directory and is pretty much self-explanatory. Figure 12.1 shows a snapshot of output file.

	A	B	C	D	E	F
1			08:00:00 - 09:20:00	08:00:00 - 09:50:00	08:00:00 - 10:50:00	09:00:00 - 10:50:00
2		LAB-1	Arabic Language (A) A 2 Ms. Farrukh Sohail,			
3		LAB-2				
4		LAB-3				
5		M1	Differential Equations (B) B 115 Ms. Saeeda Zia,			
6		M2				
7		M3				
8		M4	Physics - I (A) A 119 Dr. Mahmud Ahmad,			
9		M5	Introduction to Computer Science (B) B 9 Ms. Zareen Alamgir,			
10	Monday	M6	Web Programming (B) B 56 Mr. Awais Athar,			
11		LAB-1	English Composition (C) C 82 Ms. Fasiha Khanum Shirwany,			
12		LAB-2	Software for Mobile Devices (A) A 26 Mr. Waqar Ahmad,			
13		LAB-3	Computer Programming (D) D 41 Mr. Aamir Wali,			
14		M1				

Figure 12.1 - Output File Format.

### 12.1.1 Description

Top bar tells about the timeslot and as you can see it covers all the overlapping timeslots as well.

Left most bar is for weekdays and against each weekday rooms are mentioned. Hence each cell in the format tells about allocated timeslot day and room no.

# **13 Appendix**

## **13.1 Default Constraints Used in Project**

There are five different entities on which the constraints can be applied in our system: Students, Teachers, Class Rooms, lessons, and Timeslots.

### **13.1.1 Students**

Constraints related to students are:

1. No class can attend two or more lessons in the same timeslot.
2. Every class must attend the exact no. of lessons specified.
3. Each class should have a gap of at least one timeslot between two/given lessons.
4. Each class should not have a gap of more than specified timeslots in a day.
5. Each student should have lesson on four/given days a week.

### **13.1.2 Teachers**

Constraints related to teachers are:

1. No teacher can teach two or more lessons in the same timeslot.
2. Every teacher must teach the exact no. of lessons specified.
3. Every teacher must work in his/her available timeslots.
4. Teacher should not be allocated to a timeslot he/she is not willing to work in.
5. Every teacher must teach no. of lessons specified in a day.
6. Every teacher should have a gap of at least one timeslot between two/given lessons.
7. Every teacher should not have gap of more than specified timeslots in a day.
8. Every teacher must teach for a specific no. of days in a week.
9. Every teacher must teach same lesson to different classes on the same day.

### **13.1.3 Class Rooms**

Constraints related to Class Rooms are:

1. No class room can have two or more lessons in the same timeslot.
2. Each class room must have capacity greater or equal to the class strength.
3. Each class room should be free during one/given timeslot in a week.

### **13.1.4 Lessons**

1. Each lesson must be taught in its respective class room. (Labs for example.)
2. No lesson can be taught twice or more times in a day.

### **13.1.5 Timeslots**

Constraints related to timeslots are:

1. There should be a timeslot in which no teacher in permanent faculty is teaching lesson.
2. Each class should follow the same lesson sequence on every alternative day of the week.

## 14 References

- [1] Yongkai Liu, Defu Zhang, Stephen C.H. Leung. 2009. A Simulated Annealing Algorithm with a new Neighborhood Structure for the timetabling problem, pp 381-386.
- [2] Edmund Burke, David Elliman, Rupert Weare. A Genetic Algorithm Based University Timetabling System
- [3] Burke, E.K., Jackson, K.S., Kingston, J.H, Weare, R.F. 1997. Automated timetabling: The state of art. Comput. J. 40(9), 565-571