

# University Timetabling

## Curriculum-based Course Timetabling

April 2, 2015

Course: 42137 – Optimization using Metaheuristics – Spring 2015

Michael Lindahl  
miclin@dtu.dk

Niels-Christian Fink Bagger  
nbag@dtu.dk

### Abstract

Following the success of the competitions previously held in the metaheuristics course considering timetabling problems for high schools a new competition is organized. This time the considered problem is Curriculum-based Course Timetabling (a subset of University Timetabling Problems). In this report the general rules of the competition and the details of the problem is presented.

## Contents

<b>1</b>	<b>Outline</b>	<b>2</b>
<b>2</b>	<b>Introduction</b>	<b>2</b>
<b>3</b>	<b>Problem Formulation</b>	<b>3</b>
3.1	Constraints . . . . .	4
3.2	Objectives . . . . .	4
3.3	Mathematical Model . . . . .	5
3.3.1	Sets . . . . .	5
3.3.2	Parameters . . . . .	5
3.3.3	Decision Variables . . . . .	5
3.3.4	Constraints and Functions . . . . .	6
3.3.5	Objective . . . . .	7
3.3.6	The complete model . . . . .	7
<b>4</b>	<b>Data Instances and File Formats</b>	<b>8</b>
4.1	Input Format . . . . .	8
4.2	Output Format . . . . .	10
<b>5</b>	<b>Competition Setting</b>	<b>11</b>
5.1	Rules of the Competition . . . . .	11
5.2	Submission . . . . .	12
5.3	The Judge . . . . .	12
5.4	Benchmark tool . . . . .	15

## 1 Outline

In this project you should develop a heuristic for the Curriculum-based University Timetabling. Furthermore you also have the opportunity to compete with your fellow students. In the final phase of the course, you can submit your code to CampusNet, and by the use of some extra datasets the best metaheuristic will be determined. MaCom is sponsoring a price for the best metaheuristic. The only requirement for participating in the competition is that your code is able to run on a Windows machine. It is important to mention four things:

- It is completely voluntarily whether you want to participate in the competition!
- Your performance in the competition will NOT affect your grade! Your grade is solely based on the report you hand in.
- Only the top 3 will be announced to avoid naming and shaming!
- If you do not want to compete, you can still select this project!

This project has some nice features:

- You will have a well defined problem.
- This problem is still within a very active research area.
- The mathematical model is given.
- We provide a tool which calculates the objective and checks for feasibility.
- A good variety of datasets are provided.

**If you need some guidance in the project you can write a mail to the organizers to schedule a meeting:**

Michael Lindahl  
miclin@dtu.dk

Niels-Christian Fink Bagger  
nbag@dtu.dk

## 2 Introduction

Each semester universities face the problem of creating good feasible timetables. A timetable determines when and where all the courses should take place ie. at what time and in which room.

In the planning process many things have to be taken into consideration which makes it a very complex problem. The chosen room should have capacity for all the students and many courses are attended by the same students or taught by the same lecturer which puts restrictions on the timetable. On top of this the students as well as the lecturers have preferences about how a good timetable looks like. This could for example be a certain amount of preparation time between lectures and that a class should be taught in the same room each time.

Solving this problem manually is difficult not only because of the complexity but also because a timetable includes a large amount of courses, rooms and lecturers making it very difficult to have an overview.

By providing the timetable planners with an optimization tool it can help them have a more smooth planning process as it often is a big struggle to find a feasible timetable. Also, if the universities become better at utilizing their resources they can save money and also be able to accept more students without building new lecture rooms. Finally good timetables can give a better working environment for both the lecturers and students which also is a goal worth-while to achieve.

All data for this project (including this project description) can be obtained in the CampusNet group of the course under **File sharing**. Go to the folder **Projects**. Here you can find a folder named **UniversityTimetabling** where all data resides.

### 3 Problem Formulation

There exists many different variations of this timetabling problem as many universities and institutes have their own structure. In this competition we will consider what is known as curriculum-based course timetabling. A curriculum is a set of courses that the same students follow. This problem consist of the weekly scheduling of the lectures of courses within a given number of rooms and time periods, where conflicts between courses are set according to the curricula published by the university. This problem is similar to the planning of the first semesters at DTU where the students within the same field of study have the same mandatory courses. This formulation includes some simplifications and more attributes could be included to create timetables of even higher quality.

The problem consists of the following entities:

**Days, Periods and Timeslots:** We are given a number of lecturing days in the week (typically 5 or 6). Each day is split in a fixed number of periods, which is equal for all days. A time slot is a pair composed of a day and a period. The total number of time slots is the product of the days times the periods.

**Courses and Lectures:** Each course consists of a fixed number of lectures to be scheduled in distinct periods, it is attended by a given number of students, and is taught by a lecturer. For each course there is a minimum number of days that the lectures of the course should be spread in, moreover there are some periods in which the course cannot be scheduled.

**Rooms:** Each room has a capacity, expressed in terms of number of available seats. All rooms are equally suitable for all courses (if large enough).

**Curricula:** A curriculum is a group of courses which have some students in common. Based on curricula, we have the conflicts between courses and other soft constraints.

The solution of the problem is a number of assignments of lectures to a time slot (day and period) and a room.

### 3.1 Constraints

The following constraints have to be obeyed in order for a timetable to be feasible.

- Lectures:** Each course has a predetermined amount of lectures that must be given. Each lecture must be scheduled in distinct time slots and the total number of lectures cannot be exceeded.
- RoomOccupancy:** Two lectures cannot take place in the same room in the same time slot.
- Conflicts:** Lectures of courses in the same curriculum or taught by the same lecturer must all be scheduled in different time slots.
- Availabilities:** Some courses cannot be scheduled at specific time slots. This can be due to the lecturer teaching the course or some students attending the course are unavailable at those time slots.

### 3.2 Objectives

The objective will be to plan as many lectures as possible and avoid having unwanted attributes. Each unwanted attribute has an associated penalty.

- Unscheduled:** Each course has a predetermined amount of lectures that must be given. As many of these lectures as possible must be scheduled. *Each course that has a lecture which is not scheduled gives a penalty of 10 points.*
- RoomCapacity:** For each lecture, the number of students that attend the course must be less or equal than the number of seats of all the rooms that host its lectures. *Each student above the capacity counts as 1 point of penalty.*
- MinimumWorkingDays:** The lectures of each course must be spread into a given minimum number of days. *Each day below the minimum counts as 5 points of penalty.*
- CurriculumCompactness:** Lectures belonging to a curriculum should be adjacent to each other (i.e., in consecutive time slots). For a given curriculum we call a lecture from the curriculum *secluded* if it is not scheduled adjacent to any other lecture from the same curriculum within the same day. *Each secluded lecture in a curriculum counts as 2 points of penalty.*
- RoomStability:** All lectures of a course should be given in the same room. *Each distinct room used for the lectures of a course, but the first, counts as 1 point of penalty.*

The overall objective is the sum of all penalties which should be minimized.

### 3.3 Mathematical Model

In this section a mathematical model of the problem is given. At first the sets and parameters needed will be described. Afterwards the variables, constraints and the objective are given followed by the entire model for completeness.

#### 3.3.1 Sets

- $C$  – The set of courses
- $L$  – The set of lecturers
- $R$  – The set of rooms
- $Q$  – The set of curricula
- $T$  – The set of time slots. i.e. all pairs of days and periods
- $D$  – The set of days
- $T(d)$  – The set of time slots that belongs to day  $d \in D$
- $C(q)$  – The set of courses that belongs to curriculum  $q \in Q$

#### 3.3.2 Parameters

- $L_c$  – The number of lectures there should be for course  $c \in C$
- $C_r$  – The capacity of room  $r \in R$
- $S_c$  – The number of students attending course  $c$
- $M_c$  – The minimum number of days that course  $c$  should be spread across
- $F_{c,t} = \begin{cases} 1 & \text{if course } c \in C \text{ is available in time slot } t \in T \\ 0 & \text{otherwise} \end{cases}$
- $\chi(c_1, c_2) = \begin{cases} 1 & \text{if course } c_1 \in C \text{ is different from course } c_2 \in C (c_1 \neq c_2) \text{ and conflicting,} \\ & \text{ie. are taught by the same lecturer or are part of the same curriculum} \\ 0 & \text{otherwise} \end{cases}$
- $\Upsilon(t_1, t_2) = \begin{cases} 1 & \text{if time slot } t_1 \text{ and } t_2 \text{ belongs to the same day and are adjacent to each other} \\ 0 & \text{otherwise} \end{cases}$

#### 3.3.3 Decision Variables

$$x_{c,t,r} = \begin{cases} 1 & \text{if class } c \in C \text{ is allocated to room } r \in R \text{ in timeslot } t \in T \\ 0 & \text{otherwise} \end{cases}$$

### 3.3.4 Constraints and Functions

This section describes the different constraints for the mathematical formulation as well as the functions used for calculating the objective.

Each course can at most be assigned one room for any given time slot and only if the course is available for that time slot

$$\sum_{r \in R} x_{c,t,r} \leq F_{c,t} \quad \forall c \in C, t \in T \quad (1)$$

Each room can accommodate at most one course in any given time slot

$$\sum_{c \in C} x_{c,t,r} \leq 1 \quad \forall t \in T, r \in R \quad (2)$$

Each course can at most be assigned to the maximum allowed number of lectures

$$\sum_{t \in T, r \in R} x_{c,t,r} \leq L_c \quad \forall c \in C \quad (3)$$

Conflicting courses are not allowed to be scheduled in the same time slot

$$\sum_{r \in R} x_{c_1,t,r} + \sum_{r \in R} x_{c_2,t,r} \leq 1 \quad \forall c_1, c_2 \in C, t \in T : \chi(c_1, c_2) = 1 \quad (4)$$

The function  $V_{t,r}(x)$  indicates the amount of capacity that room  $r \in R$  is exceeded in time slot  $t \in T$ :

$$V_{t,r}(x) = \max \left\{ 0, \sum_{c \in C} S_c \cdot x_{c,t,r} - C_r \right\}$$

The function  $U_c(x)$  indicates the amount of lectures by which course  $c \in C$  is scheduled below the specified value  $L_c$ :

$$U_c(x) = \max \left\{ 0, L_c - \sum_{t \in T, r \in R} x_{c,t,r} \right\}$$

The number of room changes (number of violations of the room stability) by a course  $c \in C$  is calculated by the function  $P_c(x)$ :

$$P_c(x) = \max \left\{ 0, \left\| \left\{ r \in R \mid \sum_{t \in T} x_{c,t,r} \geq 1 \right\} \right\| - 1 \right\}$$

The number of days that the course is scheduled below the minimum number of working days is calculated by the function  $W_c(x)$ :

$$W_c(x) = \max \left\{ 0, M_c - \left\| \left\{ d \in D \mid \sum_{t \in T(d), r \in R} x_{c,t,r} \geq 1 \right\} \right\| \right\}$$

The (binary) indicator function  $A_{q,t}(x)$  determines if a curriculum in a time slot has a secluded lecture i.e. there is no adjacent lecture from the same curriculum. It is 1 if  $x_{c,t,r} = 1$  for some  $c \in C(q), r \in R$  and there does not exist  $t' \in T, c' \in C(q), r' \in R$  such that  $x_{c',t',r'} = 1$  where  $t$  and  $t'$  belongs to the same day and are adjacent to each other. More formally this can be stated as:

$$A_{q,t}(x) = \begin{cases} 1 & \text{if } \sum_{c \in C(q), r \in R} x_{c,t,r} = 1 \wedge \sum_{\substack{c \in C(q), r \in R, \\ t' \in T: \Upsilon(t,t')=1}} x_{c,t',r} = 0 \\ 0 & \text{otherwise} \end{cases}$$

### 3.3.5 Objective

The objective is the sum of all objectives including their penalty and becomes:

$$\begin{aligned} & 10 \cdot \sum_{c \in C} U_c(x) + 5 \cdot \sum_{c \in C} W_c(x) + 2 \cdot \sum_{q \in Q, t \in T} A_{q,t}(x) \\ & + 1 \cdot \sum_{c \in C} P_c(x) + 1 \cdot \sum_{t \in T, r \in R} V_{t,r}(x) \end{aligned} \quad (5)$$

### 3.3.6 The complete model

For the sake of completeness the entire mathematical formulation is given in Model 1 and the descriptions of the constraints follows.

---


$$\begin{aligned} \min \quad & 10 \cdot \sum_{c \in C} U_c(x) + 5 \cdot \sum_{c \in C} W_c(x) + 2 \cdot \sum_{q \in Q, t \in T} A_{q,t}(x) \\ & + 1 \cdot \sum_{c \in C} P_c(x) + 1 \cdot \sum_{t \in T, r \in R} V_{t,r}(x) \end{aligned} \quad (1a)$$

$$\text{s. t.} \quad \sum_{r \in R} x_{c,t,r} \leq F_{c,t} \quad \forall c \in C, t \in T \quad (1b)$$

$$\sum_{c \in C} x_{c,t,r} \leq 1 \quad \forall t \in T, r \in R \quad (1c)$$

$$\sum_{t \in T, r \in R} x_{c,t,r} \leq L_c \quad \forall c \in C \quad (1d)$$

$$\sum_{r \in R} x_{c_1,t,r} + \sum_{r \in R} x_{c_2,t,r} \leq 1 \quad \forall c_1, c_2 \in C, t \in T : \chi(c_1, c_2) = 1 \quad (1e)$$

$$x_{c,t,r} \in \mathbb{B} \quad \forall c \in C, t \in T, r \in R \quad (1f)$$


---

Model 1: The complete model

## Constraints

- (1b) – Each course can at most be assigned one room for any given time slot and only if the course is available for that time slot
- (1c) – Each room can accommodate at most one course in any given time slot
- (1d) – Each course can at most be assigned to the maximum allowed number of lectures
- (1e) – Conflicting courses are not allowed to be scheduled in the same time slot

## 4 Data Instances and File Formats

This section describes the format of the provided data instances as well as how the format of output of the solutions should be. 13 data instances of different sizes and complexity is provided. The data can be downloaded from CampusNet. The data comes from real timetabling problems from a university.

### 4.1 Input Format

Each data instance consists of 7 files. Each file has a layout similar to a database table where the first line is the header naming the columns and the following lines are the rows of the table. Everything is separated by white spaces.

basic.utt	Includes basic information about the data instance and its size. It contains the following columns:	
	Courses	The total number of courses
	Rooms	The total number of rooms
	Days	The total number of days
	Periods_per_day	The number of periods per day
	Curricula	The total number of curricula
	Constraints	The total number of unavailable time slots for all the courses
	Lecturers	The total number of lecturers
courses.utt	Information about each course, how many lectures there should be etc. The columns are:	
	Course	The ID of the course
	Lecturer	The ID of the lecturer teaching the course
	Number_of_lectures	The number of lectures that the course can at most be scheduled for and that it is preferred to schedule for according to the objective <b>Unscheduled</b>



	<b>Minimum_working_days</b>	The preferred minimum amount of working days for the course according to the objective <b>MinimumWorkingDays</b>
	<b>Number_of_students</b>	The number of students attending the course
<b>lecturers.utt</b>	A list of all lecturers. The file only contains one column with the ID of the lecturers	
<b>rooms.utt</b>	A list of all rooms and their capacity where each line has the format <RoomID> <Capacity>	
<b>curricula.utt</b>	A list of all curricula. Each line in the file correspond to one curriculum with the ID of the curriculum and the number of courses that the curriculum contains	
<b>relation.utt</b>	Each line in this file specifies a relation between a curriculum and a course. Each line has the format <CurriculumID> <CourseID>	
<b>unavailability.utt</b>	A list of certain time slots where a course cannot be planned. Each line has the format <CourseID> <Day> <Period>	

All IDs are strings without white spaces. Days and periods start from 0. For example, the constraint C0001 3 2 states that course C0001 cannot be scheduled in the third (i.e., 2) period of Thursday (i.e., 3)

On the machine used for the competition your program will be given as input the location of the files and the time limit of 300 seconds, i.e. your code gets the following arguments:

```
basic.utt courses.utt lecturers.utt rooms.utt curricula.utt relation.utt unavailability.utt 300
```

The reason for giving the time limit as input even though it is fixed at 300 seconds is to give you the opportunity to use this instead of hard coding the time limit. This has the advantage that you can change the time limit during the parameter tuning on your own computer and then submit the code without having to remember to change the time limit in the code.

One last thing to note is the time slots. In the **basic.utt** file two values are given; **Days** and **Periods\_per\_day**. These values specifies the time slots where each time slot is a combination of day and period, e.g. if the value of **Days** is 4 and the value of **Periods\_per\_day** is 3 then there are 12 time slots as illustrated in Table 1.

Time slot	Day	Period
(0,0)	0	0
(0,1)	0	1
(0,2)	0	2
(1,0)	1	0
(1,1)	1	1
(1,2)	1	2
(2,0)	2	0
(2,1)	2	1
(2,2)	2	2
(3,0)	3	0
(3,1)	3	1
(3,2)	3	2

Table 1: Illustration of the 12 time slots as a result of `Days` having value 4 and `Periods_per_day` having value 3.

Two time slots are adjacent to each other if they have the same day and their period numbers are consecutive, e.g. time slot (2,1) and (2,2) in Table 1 are adjacent since they both have day 2 and time slot (2,1) has period 1 and time slot (2,2) has period 2.

## 4.2 Output Format

The solution must be provided as output to the standard console/terminal. By output to the standard console/terminal is meant as using the following or similar approaches:

```

C      printf("output a line example\n");
C#     System.Console.WriteLine("output a line example");
C++    std::cout << "output a line example" << std::endl;
Java   System.out.println("output a line example");
Python print "output a line example"

```

Each line in the output represents the assignment of one lecture of a course in a room and a time slot (lines can be in any order) in the following format

`<CourseID> <Day> <Period> <RoomID>`. A solution could look like this:

```

C0000 3 0 R0001
C0001 3 1 R0000
C0002 4 0 R0000
C0003 0 1 R0001
C0004 1 1 R0001
C0005 1 2 R0001
C0006 0 0 R0001
C0007 0 1 R0000
C0008 3 0 R0001
C0009 3 1 R0000
C0010 4 2 R0000

```

For example, the first line states that a lecture of C0000 takes place on Thursday (i.e., 3) in

the first period (i.e., 0) in room R0001.

It is possible to include the calculated values of the different objectives described in Section 3.2. These values are put in separate lines in any order before the solution where each line is in the format `<Name> <Value>` where `<Name>` is the name of the objective, i.e. `Unscheduled`, `RoomCapacity`, `MinimumWorkingDays`, `CurriculumCompactness` or `RoomStability`, furthermore `<Name>` can also be `Objective` which is the total penalty for the solution. Not all objective values need to be provided, e.g. the output could look like the following:

```
ROOMSTABILITY 400
UNSCHEDULED 22000
C0000 3 0 R0001
C0001 3 1 R0000
C0002 4 0 R0000
C0003 0 1 R0001
C0004 1 1 R0001
C0005 1 2 R0001
C0006 0 0 R0001
C0007 0 1 R0000
C0008 3 0 R0001
C0009 3 1 R0000
C0010 4 2 R0000
```

## 5 Competition Setting

The idea of the competition is to encourage healthy sportsmanship.

**Remember that your grade will NOT be affected by your rank in the competition.**

### 5.1 Rules of the Competition

In order to have a fair competition the following rules apply:

1. All code will be run in a single core Windows environment. This means that your code will not benefit from a parallel implementation and only one of the following programming languages should be used: C, C++, C#, Java or Python
2. The code should not require the use of third party libraries, e.g. BLAS/LAPACK or the like. These libraries are not on the competition machine and therefore the code might not be able to run.
3. A benchmark program is provided to ensure that you tune your parameters for the same amount of iterations as you get on the computer used for the competition
4. The rank in the competition will be determined based on some hidden datasets.
5. The code is allowed to run on the competition machine for 300 seconds plus a few extra seconds to import data and export the solution. If an algorithm has not finished after

the time limit is reached, the process will be terminated

**Note:** When using the time limit in the code then it includes all algorithms, i.e. both your initial solution, your main algorithm and some post process if any. The time limit should not include reading the data files and printing the solution.

6. For each hidden dataset the algorithm will run 10 times and the average objective value and variance will be calculated. The one with the lowest average objective value will get rank 1, the second lowest rank 2 and so on. If there is a tie between two teams the one with the lowest variance will get the lowest rank of the two. The final results is the sum of all ranks over all datasets. The team with the lowest sum of ranks wins
7. The winner will receive a price sponsored by MaCom A/S followed by eternal fame and glory

If you have any doubts about the rules please contact the organizers.

## 5.2 Submission

When submitting your solution to CampusNet please upload the code in a zip file. If your code is in C, C++ or C# then a compiled executable for Windows should also be provided. If your code is written in JAVA please provide a compiled Java archive (jar-file) as well. If you are not able to compile a Windows executable, e.g. due to no access to a Windows machine, please write the organizers to get help on this matter. If we are not able to run your executable we will try to compile the code ourself to obtain an executable, so please write in the comment field on CampusNet if the code has been compiled with some non-standard compiler flags or arguments.

## 5.3 The Judge

A program called "Judge" is provided to analyze solutions and determine feasibility and penalty. It is provided as a command line interface Windows executable program. To obtain the program go to **File sharing** on the CampusNet group of the course and go to the folder **Project/BenchmarkMachineAndJudge**. The program is compiled into the Windows executable file **Judge.exe** which resides in this folder. To use the program simply give it two argument; the directory of the test instance and the path to a file containing the solution you want to analyze. Example:

```
Judge Test01 test01.sol
```

This will read in the data from the directory "Test01" and the solution from the file "test01.sol" and analyze the solution against the data instance. You have been asked to write your code such that the solution is written to the standard console output, so a way to get the printed solution written to a file is using the following command (assuming that your code is compiled into an executable file with the name "Solver"):

```
Solver basic.utt courses.utt lecturers.utt rooms.utt curricula.utt relation.utt  
unavailability.utt 300 > solution.sol
```

Notice the “> solution.sol” in the end of the command. This will take all the output written to the console/terminal by the program and put it into the file named “solution.sol” which can then be given to the Judge.

The output of the Judge will depend on whether or not the given solution is feasible or infeasible. If the solution is infeasible the following output will occur:

```
RESULT WRONG
The solution is infeasible. The number of violations is XXX
Score XXX
```

More details are given in the file ‘violations.log’

The XXX in the output is replaced by how many constraints that are violated in the solution. Furthermore a file named “violations.log” is written to the directory from which the Judge is called from. This file gives a detailed description of which of the constraints that are violated in the solution. The file is a text file and a small example of how this file could look like is given in the following picture:

#### Test01 - violations.log

```
Each lecturer can only teach a single course in each time slot

Lecturer L0002 has been scheduled in time slot (d4,p0) for courses C0002, C0027
```

In the example above a lecturer has been scheduled for two course in the same time slot which is infeasible.

If the provided solution is feasible then the output of the Judge is:

```
RESULT CORRECT
The solution is feasible. The Objective value is XXX
Score XXX
```

More details are given in the file ‘penalties.log’

The XXX in the output is replaced by the objective value of the solution. Furthermore a file named “penalties.log” is written to the directory from which the Judge is called from. This file is a text file which gives an overview of the different penalties of the solution. This file shows a table illustrating the values of each objective (soft constraint). There are four columns in the table;

Name: The name of the objective (soft constraint)

Calculated: The value of the objective calculated by the Judge

Given: The value of the objective specified in the output by the submitted code. If a value was not given then n/a is shown here

Difference: The absolute difference between the provided value of the objective and the calculated value

An example of such a file is illustrated below:

### Test01 - penalties.log

Table 1: Penalty values

Name		Calculated		Given		Difference
=====						
UNSCHEDULED		95		150		55
ROOMCAPACITY		742		n/a		742
MINIMUMWORKINGDAYS		46		130		84
CURRICULUMCOMPACTNESS		37		0		37
ROOMSTABILITY		24		99		75
OBJECTIVE		2020		5600		3580

Table 2: Description of penalty names

Name		Description
=====		
UNSCHEDULED		Unscheduled lectures
ROOMCAPACITY		Total violated room capacity
MINIMUMWORKINGDAYS		Total violation of minimum working days
CURRICULUMCOMPACTNESS		Number of secluded courses
ROOMSTABILITY		Number of room changes
OBJECTIVE		Total penalty of the solution

Note that even though the provided values of the objectives are incorrect in the last picture the test is still passed as the solution is feasible. This is to ensure that submissions do not get rejected in the competition even though the objective calculations are wrong since the result of the competition is based on the values calculated by the Judge.

It should be noted that the values reported in the column ''Calculated'' are **WITHOUT** the penalty multiplier, except for the value of **OBJECTIVE**. For instance in the example above the value of **UNSCHEDULED** is reported as **95**. This means that **95** lectures have not been scheduled which leads to a penalty of  $95 \cdot 10 = 950$ . The value of **OBJECTIVE** is reported as the sum of the other values multiplied by their respective penalty multipliers. For instance in the example above the value of **OBJECTIVE** is reported as **2020** since  $95 \cdot 10 + 742 \cdot 1 + 46 \cdot 5 + 37 \cdot 2 + 24 \cdot 1 = 2020$ . Referring back to the model in Section 3.3 this means that for some given solution  $x$ :

$$\begin{aligned}
\text{UNSCHEDULED} &= \sum_{c \in C} U_c(x) \\
\text{ROOMCAPACITY} &= \sum_{t \in T, r \in R} V_{t,r}(x) \\
\text{MINIMUMWORKINGDAYS} &= \sum_{c \in C} W_c(x) \\
\text{CURRICULUMCOMPACTNESS} &= \sum_{q \in Q, t \in T} A_{q,t}(x) \\
\text{ROOMSTABILITY} &= \sum_{c \in C} P_c(x) \\
\text{OBJECTIVE} &= 10 \cdot \sum_{c \in C} U_c(x) + 5 \cdot \sum_{c \in C} W_c(x) + 2 \cdot \sum_{q \in Q, t \in T} A_{q,t}(x) \\
&\quad + 1 \cdot \sum_{c \in C} P_c(x) + 1 \cdot \sum_{t \in T, r \in R} V_{t,r}(x) \\
&= 10 \cdot \text{UNSCHEDULED} + 5 \cdot \text{MINIMUMWORKINGDAYS} \\
&\quad + 2 \cdot \text{CURRICULUMCOMPACTNESS} \\
&\quad + 1 \cdot \text{ROOMSTABILITY} + 1 \cdot \text{ROOMCAPACITY}
\end{aligned}$$

## 5.4 Benchmark tool

Because the optimal parameters for a heuristic often depends on the amount of iterations it is able to have in the given time limit, a benchmark tool is provided. This will help to make sure that your algorithm perform similar on your computer and on the competition machine. By running the benchmark program it will output the amount of time you should use to tune the parameters on your own computer for getting the same amount of iterations as on the server.

To obtain the benchmark tool go to **File sharing** on the CampusNet group of the course and go to the folder **Project/BenchmarkMachineAndJudge**. There are three files to be aware of in this folder; **benchmark.machine.c**, **benchmark.my\_linux\_machine** and **benchmark.my\_windows\_machine.exe** as illustrated in the picture below:

## File sharing: BenchmarkMachineAndJudge

[Create folder](#) | [Upload file](#) | [Upload zip file](#) | [Storage usage](#) | [Trail](#)

Your browser supports drag and drop upload. Drag files onto the

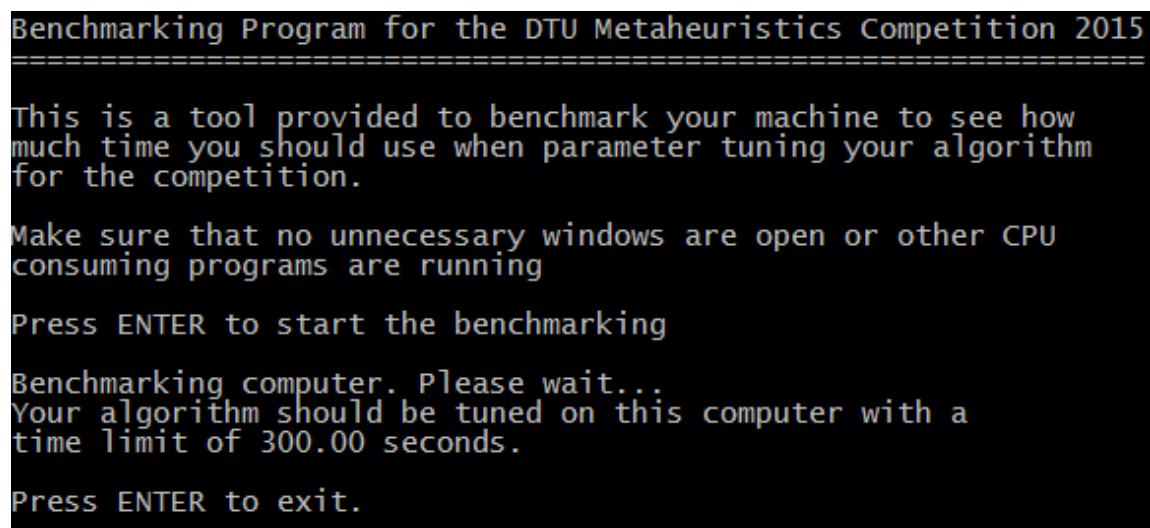
- ☐ **Name**
- ☐  Parent folder (Project)
- ☐  benchmark\_machine.c
- ☐  benchmark\_my\_linux\_machine
- ☐  benchmark\_my\_windows\_machine.exe
- ☐  Judge.exe

[Delete selected](#) | [Move selected](#) | [Download selected as zip](#)

The file `benchmark_machine.c` is the source code of the tool. The two other files are a result of compiling the source code. The file `benchmark_machine` has been compiled with the `gcc` compiler on the competition machine for the Windows executable and on Gbar at DTU for the Linux executable using one of the following commands:

```
gcc -Ofast benchmark_machine.c -o benchmark_my_windows_machine.exe
gcc -O3 benchmark_machine.c -o benchmark_my_linux_machine
```

The executable should be able to run under most Linux-based operating systems (only tested in the Gbar) and on most newer Windows operating systems (tested in Windows 7 & Windows 8.1). The output of the tool is similar to the following picture:



```
Benchmarking Program for the DTU Metaheuristics Competition 2015
=====
This is a tool provided to benchmark your machine to see how
much time you should use when parameter tuning your algorithm
for the competition.

Make sure that no unnecessary windows are open or other CPU
consuming programs are running

Press ENTER to start the benchmarking

Benchmarking computer. Please wait...
Your algorithm should be tuned on this computer with a
time limit of 300.00 seconds.

Press ENTER to exit.
```

In the example of the picture the computer that was benchmarked should be tuned using a time limit of 300.00 seconds.



The source code of the tool is included in the folder such that if none of the two executables can be used on your operating system then you can compile the source code yourself.

It should be noted that the benchmarking tool runs a very simple heuristic that are looking through some solutions in a mixed integer linear program. The number of iterations is fixed at a predefined integer which takes 300 seconds to run on the competition machine. The amount of time it takes to run the program on your computer is the amount of time you should use as time limit when tuning your heuristic for the competition.