

University Timetabling

Curriculum-based Course Timetabling

March 6, 2014

Course: 42137 – Optimization using Metaheuristics – Spring 2014

Michael Lindahl (ML)

Niels-Christian Fink Bagger (NCFB)

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

| | | |
|----------|--|-----------|
| 1 | Outline | 2 |
| 2 | Introduction | 2 |
| 3 | Problem Formulation | 3 |
| 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 | 6 |
| 3.3.4 | Constraints and Functions | 6 |
| 3.3.5 | Objective | 7 |
| 3.3.6 | The complete model | 7 |
| 4 | Data Instances and File Formats | 8 |
| 4.1 | Input Format | 8 |
| 4.2 | Output Format | 11 |
| 5 | Competition Setting | 12 |
| 5.1 | Rules of the Competition | 12 |
| 5.2 | The CodeJudge System | 12 |
| 5.2.1 | Login | 12 |
| 5.2.2 | Assignment Overview | 14 |
| 5.2.3 | Group Assignment | 15 |
| 5.2.4 | Submitting Code | 16 |
| 5.2.5 | Infeasible Solutions | 19 |
| 5.2.6 | Feasible Solutions | 22 |
| 5.2.7 | Special Note for C# Developers using Visual Studio | 24 |
| 5.3 | Benchmark tool | 26 |

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 the online tool CodeJudge, 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 CodeJudge. 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 an online tool which calculates the objective and checks for feasibility.
- A good variety of datasets are provided.

If you have questions regarding this project Niels-Christian Fink Bagger and Michael Lindahl have office hours and can be contacted by mail.

Office hours in building 426, room 040:

| | |
|------------|-------|
| Wednesdays | 13-15 |
| Thursdays | 10-12 |

Sometimes during the semester we will not have office hours. We will notify you on CampusNet whenever this occurs. If you have urgent matters you can contact us by mail:

| | |
|-----------------------------|---------------|
| 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 such that any pair of courses in the group have 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

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

$$v_{t,r} = \text{The number of students the capacity of room } r \in R \text{ is exceeded by in time slot } t \in T$$

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

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

Calculate how much the capacity of a room is exceeded

$$\sum_{c \in C} S_c \cdot x_{c,t,r} - v_{t,r} \leq C_r \quad \forall t \in T, r \in R \quad (5)$$

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:

$$\begin{aligned} & \sum_{c \in C} 10 \cdot U_c(x) + \sum_{c \in C} 5 \cdot W_c(x) + \sum_{q \in Q, t \in T} 2 \cdot A_{q,t}(x) \\ & + \sum_{c \in C} 1 \cdot P_c(x) + \sum_{t \in T, r \in R} 1 \cdot v_{t,r} \end{aligned} \tag{6}$$

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.

$$\begin{aligned}
\min \quad & \sum_{c \in C} 10 \cdot U_c(x) + \sum_{c \in C} 5 \cdot W_c(x) + \sum_{q \in Q, t \in T} 2 \cdot A_{q,t}(x) \\
& + \sum_{c \in C} 1 \cdot P_c(x) + \sum_{t \in T, r \in R} 1 \cdot v_{t,r} \tag{1a} \\
\text{s. t.} \quad & \sum_{r \in R} x_{c,t,r} \leq F_{c,t} \quad \forall c \in C, t \in T \tag{1b} \\
& \sum_{c \in C} x_{c,t,r} \leq 1 \quad \forall t \in T, r \in R \tag{1c} \\
& \sum_{t \in T, r \in R} x_{c,t,r} \leq L_c \quad \forall c \in C \tag{1d} \\
& \sum_{r \in R} x_{c_1,t,r} + \sum_{r \in R} x_{c_2,t,r} \leq 1 \quad \forall c_1, c_2 \in C, t \in T : \chi(c_1, c_2) = 1 \tag{1e} \\
& \sum_{c \in C} S_c \cdot x_{c,t,r} - v_{t,r} \leq C_r \quad \forall t \in T, r \in R \tag{1f} \\
& x_{c,t,r} \in \mathbb{B} \quad \forall c \in C, t \in T, r \in R \tag{1g} \\
& v_{t,r} \in \mathbb{Z}_+ \quad \forall t \in T, r \in R \tag{1h}
\end{aligned}$$

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
- (1f) – Calculate how much the capacity of a room is exceeded

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: | | | | | | | | | | | | | | |
| | <table> <tr> <td>Courses</td><td>The total number of courses</td></tr> <tr> <td>Rooms</td><td>The total number of rooms</td></tr> <tr> <td>Days</td><td>The total number of days</td></tr> <tr> <td>Periods_per_day</td><td>The number of periods per day</td></tr> <tr> <td>Curricula</td><td>The total number of curricula</td></tr> <tr> <td>Constraints</td><td>The total number of unavailable time slots for all the courses</td></tr> <tr> <td>Lecturers</td><td>The total number of lecturers</td></tr> </table> | 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 | 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: | | | | | | | | | | | | | | |
| | <table> <tr> <td>Course</td><td>The ID of the course</td></tr> <tr> <td>Lecturer</td><td>The ID of the lecturer teaching the course</td></tr> <tr> <td>Number_of_lectures</td><td>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 Unscheduled</td></tr> <tr> <td>Minimum_working_days</td><td>The preferred minimum amount of working days for the course according to the objective MinimumWorkingDays</td></tr> <tr> <td>Number_of_students</td><td>The number of students attending the course</td></tr> </table> | 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 Unscheduled | Minimum_working_days | The preferred minimum amount of working days for the course according to the objective MinimumWorkingDays | Number_of_students | The number of students attending the course | | | | |
| 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 Unscheduled | | | | | | | | | | | | | | |
| Minimum_working_days | The preferred minimum amount of working days for the course according to the objective MinimumWorkingDays | | | | | | | | | | | | | | |
| Number_of_students | The number of students attending the course | | | | | | | | | | | | | | |
| lecturers.utt | A list of all lecturers. The file only contains one column with the ID of the lecturers | | | | | | | | | | | | | | |
| rooms.utt | A list of all rooms and their capacity where each line has the format <RoomID> <Capacity> | | | | | | | | | | | | | | |
| curricula.utt | 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 | | | | | | | | | | | | | | |
| relation.utt | Each line in this file specifies a relation between a curriculum and a course. Each line has the format <CurriculumID> <CourseID> | | | | | | | | | | | | | | |
| unavailability.utt | 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)

In the CodeJudge system (which will be described in Section 5.2 and is used for the competition) your program will be given as input the location of the files and the time limit of 60 seconds, i.e. your code gets the following arguments:

```
basic.utt courses.utt lecturers.utt rooms.utt curricula.utt relation.utt unavailability.utt 60
```

The reason for giving the time limit as input even though it is fixed at 60 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) have period 1 and time slot (2,2) have 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 should be able to run on CodeJudge and only one core on the machine will be assigned. This mean that your code will not benefit from a parallel implementation and only one of the following programming languages should be used: C (gcc 4.7.1), C++ (g++ 4.7.1), C# (Mono 2.10, .NET 4.0), Java (OpenJDK 7) or Python (2.7)
2. A benchmark program is provided to ensure that you tune your parameters for the same amount of iterations as you get on CodeJudge
3. The rank in the competition will be determined based on some hidden datasets.
4. The code is allowed to run on CodeJudge for 60 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
5. 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
6. 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 The CodeJudge System

The CodeJudge is a tool provided to check whether your code is able to compile on the server where the competition is held and whether it returns feasible solutions and respects the provided running time. To use the CodeJudge go to <http://codejudge.compute.dtu.dk>.

5.2.1 Login

The First time you go to this page you will be met by the following screen:



The image shows the login page for CodeJudge. At the top is a blue header with the CodeJudge logo and the text "Automated Code Judging". Below the header is a dark grey bar with the word "Login". The main content area has a light grey background with a diagonal line pattern. It features a "Login" heading, followed by "DTU Users" and a red-bordered button labeled "CAS Login". Below this is another "Login" heading, followed by input fields for "Username / E-mail:" and "Password:", and a blue "Login" button. At the bottom right, there is a footer with the text "CodeJudge.net - Automated Code Judging", "Technical Support: support@codejudge.net", and a timestamp "28-02-2014 12:48".

CodeJudge
Automated Code Judging

Login

Login

DTU Users

CAS Login

Login

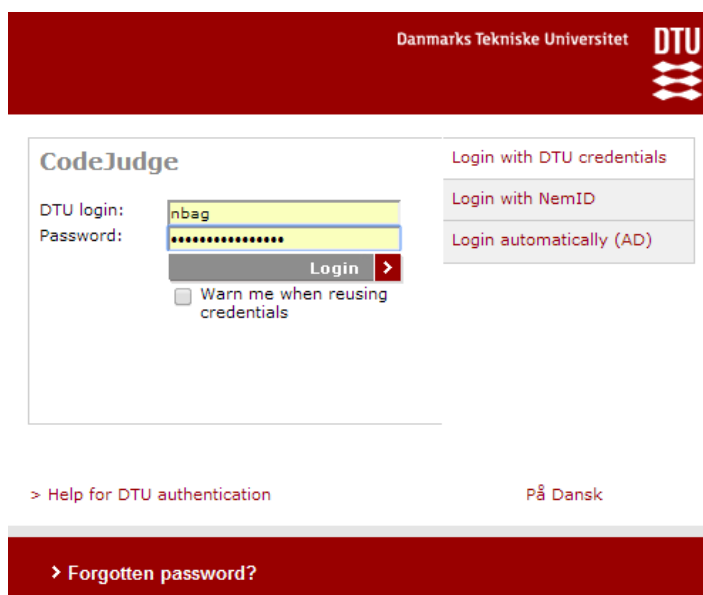
Username / E-mail:

Password:

Login

CodeJudge.net - Automated Code Judging
Technical Support: support@codejudge.net
28-02-2014 12:48

Here you should click on the *CAS Login* button as illustrated in the picture. This will redirect you to the usual DTU login interface:



The image shows the DTU login interface. At the top is a red header with the text "Danmarks Tekniske Universitet" and the DTU logo. Below the header is a white box with the CodeJudge logo and the text "DTU login:". The box contains input fields for "DTU login:" (with the text "nbag") and "Password:" (with masked characters). There is a "Login" button with a red arrow. Below the password field is a checkbox labeled "Warn me when reusing credentials". To the right of the login box is a vertical list of three buttons: "Login with DTU credentials", "Login with NemID", and "Login automatically (AD)". At the bottom of the page, there is a red bar with the text "> Help for DTU authentication" and "På Dansk". Below this is another red bar with the text "> Forgotten password?".

Danmarks Tekniske Universitet DTU

CodeJudge

DTU login: nbag

Password:

Login

☐ Warn me when reusing credentials

Login with DTU credentials

Login with NemID

Login automatically (AD)

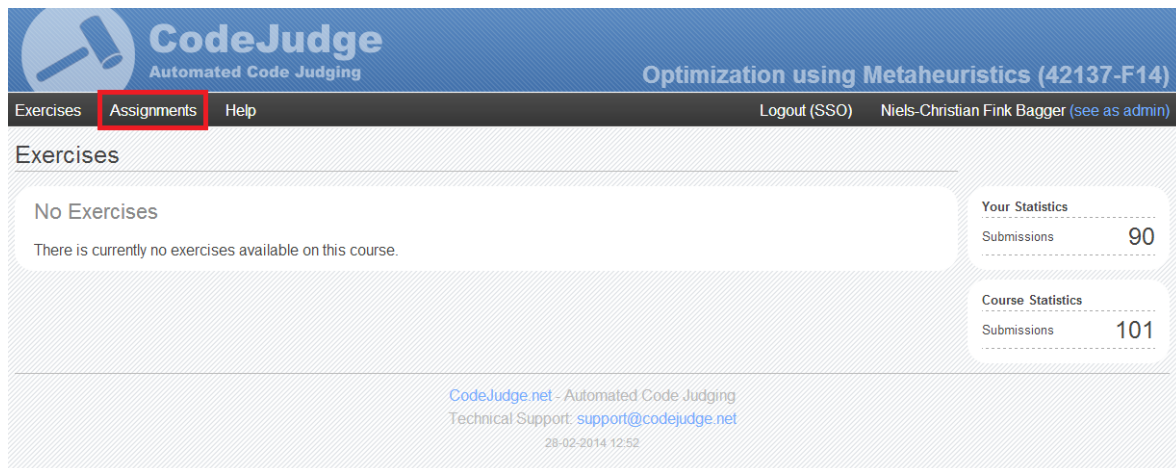
> Help for DTU authentication På Dansk

> Forgotten password?

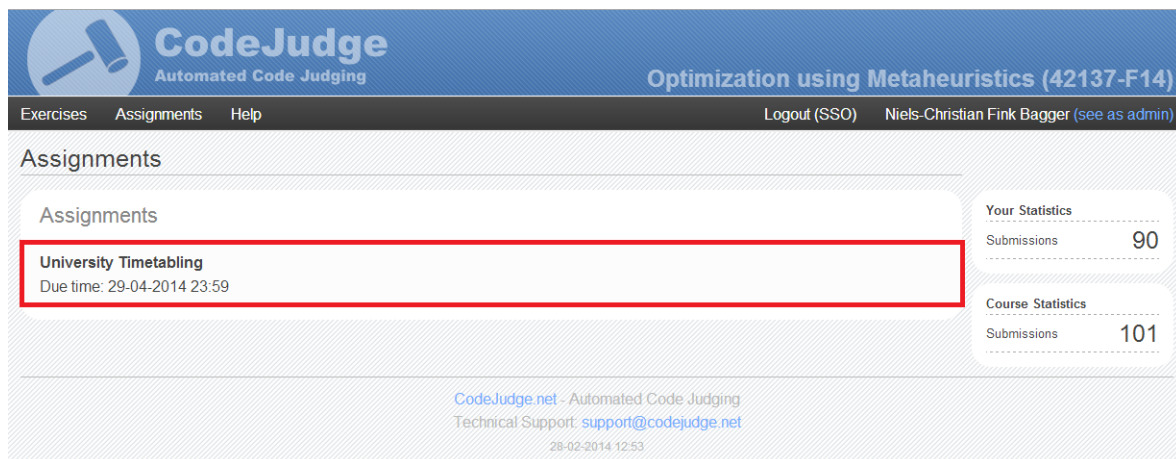
Your login information is the same as you use when you login to CampusNet.

5.2.2 Assignment Overview

When you login you will be met by the following screen:



There are three tabs in the top *Exercises*, *Assignments* and *Help*. The tab that you should use is the *Assignments* tab as illustrated in the picture which will bring you to the *Assignments* overview as depicted in the following picture:



In this overview you can see all assignments of the course (currently only the assignment for the competition is supported by the CodeJudge). Click on the assignment that you wish to implement. This will bring you to the overview of the assignment:

CodeJudge
Automated Code Judging

Optimization using Metaheuristics (42137-F14)

Exercises Assignments Help Logout (SSO) Niels-Christian Fink Bagger (see as admin)

[Assignments](#) » University Timetabling

About the Assignment
Due time: 29-04-2014 23:59

Group Assignment
It is allowed to submit the assignment in groups. When you join a group, all your earlier and future submissions to the assignment will be shared with the group.
Invite User to Group
User:

Comment
This comment will be visible to the instructors together with the submitted code.
Comment

Exercises
For each exercise the last submitted solution before the due time, will count as the solution handed in.

Curriculum Timetabling 28-02-2014 11:47 (submission #106155) Succeeded

Your Statistics
Submissions 90

Course Statistics
Submissions 101

As you can see in the picture there are four areas in this overview; *About the Assignment*, *Group Assignment*, *Comment* and *Exercises*.

If you do NOT want to participate in the competition you should write it in the *Comment* text box. If the *Comment* text box is left empty then we will assume that you wish to participate.

Remember that the outcome of the competition does NOT affect your grade. Your grade is solely based on the report you hand in.

5.2.3 Group Assignment

If you are handing in the assignment as a group you must also hand in the code as the same group. This is done by choosing a group representative. This person should login and go to the assignment. Here there is a box named *Group Assignment* as seen in the last picture of Section 5.2.2. In this box there is a selection box named *User*. Select another member of the group and press the *invite* button. The *Group Assignment* box will appear as the picture below showing that the user has been invited.

Group Assignment

It is allowed to submit the assignment in groups. When you join a group, all your earlier and future submissions to the assignment will be shared with the group.

- Niels-Christian Fink Bagger (group leader)
- Michael Lindahl (has **not** accepted) [remove](#)

Invite User to Group

User:

When the invited user logs in and go to assignment the *Group Assignment* box will appear as below showing that the user has received a group invitation.

Group Assignment

It is allowed to submit the assignment in groups. When you join a group, all your earlier and future submissions to the assignment will be shared with the group.

Group Invitations

- Niels-Christian Fink Bagger

Invite User to Group

User:


Click the *Accept* button to accept the invitation and be added to the group. Click the *Decline* button to decline the invitation for the group.

5.2.4 Submitting Code


In the last picture of Section 5.2.2 it can be seen that there is a box named *Exercise*. This is where the actual exercise(s) of the assignment can be found. Here you get an overview of the state of your last submission. Click on the exercise to go to that specific exercise:

Exercises

For each exercise the last submitted solution before the due time, will count as the solution handed in.

| | | | |
|------------------------|---------------------------------------|-----------|---|
| Curriculum Timetabling | 28-02-2014 11:47 (submission #106155) | Succeeded |  |
|------------------------|---------------------------------------|-----------|---|

When you click on an exercise you get to the overview of the exercise as illustrated in the following picture:


CodeJudge
Automated Code Judging

Optimization using Metaheuristics (42137-F14)

Exercises
Assignments
Help

Logout (SSO)
Niels-Christian Fink Bagger (see as admin)

Assignments » University Timetabling » Curriculum Timetabling

Exercise

Look at the attached PDF for a description of the assignment
[Sample test data](#) - click here.

Attached Files

universitytimetabling.pdf

Your Statistics

Submissions 90

Course Statistics

Submissions 101

Submit Solution

Language: C++ (g++ 4.7.1)

Comment:

File Source Code

Files:

CLICK TO SELECT FILES OR DROP FILES HERE

Submit

Your Submissions

| | | | |
|---------|--------------------|------------------|---|
| #106155 | Succeeded | 28-02-2014 11:47 | 😊 |
| #106149 | Succeeded | 28-02-2014 10:57 | 😊 |
| #106148 | Succeeded | 28-02-2014 10:52 | 😊 |
| #106147 | Compilation failed | 28-02-2014 10:51 | 😞 |
| #106146 | Compilation failed | 28-02-2014 10:49 | 😞 |
| #106120 | Succeeded | 27-02-2014 14:33 | 😊 |
| #106119 | Succeeded | 27-02-2014 14:30 | 😊 |

There are three boxed areas; *Exercise*, *Submit Solution* and *Your Submissions*. The *Exercise* box gives a short description of the exercise, an example of how the test data which will be given to your code looks like and possibly some attached files.

Submitting code can be done in the *Submit Solution* box either by writing the source code directly in the interface or by uploading the files containing the source code. To write the source code directly click on the *Source Code* button and write the code in the text field below the button:

Submit Solution

Language: C++ (g++ 4.7.1) ▼

Comment: ?

File Source Code

Source Code:

```
#include <iostream>

int main(int argc, char* argv[])
{
    std::cout << "No solution";
}
|
```

Submit

To upload source code files click on the *File* button and a box appears which you can either drag files into or click on to select the files you wish to upload:

Submit Solution

Language: C# (Mono 2.10, .NET 4.0) ▼

Comment: ?

File Source Code

Files:

CLICK TO SELECT FILES
OR DROP FILES HERE

DataObjects.cs [\[X\]](#)

Program.cs [\[X\]](#)

Submit

When some source code is submitted it will show in the *Your Submissions* box that the code is being processed (compiled and run on multiple data sets):

| Your Submissions | | |
|------------------|--------------------|--|
| #106172 | Processing... | 28-02-2014 14:25  |
| #106171 | System error | 28-02-2014 14:22  |
| #106170 | Compilation failed | 28-02-2014 14:20  |
| #106169 | Compilation failed | 28-02-2014 14:18  |

When submitting code please ensure that the correct language is selected corresponding to the source code being submitted. For instance selecting the C++ language but uploading C# source code results in the following error:

Your Submissions

#106169 Compilation failed28-02-2014 14:18

Source Code

TestData
Graded: 28-02-2014 14:18

Compilation error:

```
DataObjects.cs: file not recognized: File format not recognized
collect2: error: ld returned 1 exit status
```

Download test data

#106168 Succeeded28-02-2014 14:14

#106167 Compilation failed28-02-2014 14:13

If the code can compile then multiple tests are run to see if the solutions produced by the code are infeasible or feasible.

5.2.5 Infeasible Solutions

In the case where the submitted code produces an infeasible solution the result of the submission will be *Test failed* along with an overview of which of the tests that failed:

Your Submissions

| | | | |
|---------|--------------|------------------|---|
| #106173 | Tests failed | 28-02-2014 14:27 | ☹ |
|---------|--------------|------------------|---|

Source Code

TestData

Graded: 28-02-2014 14:27

| | | | |
|--------|---|---------------|-----------|
| Test01 | Wrong answer (The solution is infeasible. The number of violations is 1) | RUNTIME 0.09s | Test Data |
|--------|---|---------------|-----------|

Download test data

| | | | |
|---------|--------------------|------------------|---|
| #106172 | Tests failed | 28-02-2014 14:25 | ☹ |
| #106171 | System error | 28-02-2014 14:22 | ☹ |
| #106170 | Compilation failed | 28-02-2014 14:20 | ☹ |
| #106169 | Compilation failed | 28-02-2014 14:18 | ☹ |

When clicking on the test that produced the result *Wrong answer* a more detailed description of the test will be shown. Here all the files that were given as input is shown as well as a file called *violations.log* which specifies what the violations are in the solution produced by the submitted code:

Test01 - violations.log

Close

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

violations.log Arguments Output (stdout) Error (stderr) basic.utt courses.utt curricula.utt lecturers.utt relation.utt rooms.utt unavailability.utt

On the example here the violation is that lecturer L0002 is scheduled for two lectures at the same time; course C0002 and C0027. The file *courses.utt* that was given as input can be inspected by clicking on it and here it can be seen that those two courses indeed are being taught by lecturer L0002:

```

Test01 - courses.utt
Course Lecturer Number_of_lectures Minimum_working_days Number_of_students
C0000 L0000 6 4 130
C0001 L0001 6 4 75
C0002 L0002 7 3 117
C0003 L0003 3 3 75
C0004 L0004 1 1 65
C0005 L0005 8 3 65
C0006 L0006 7 3 65
C0007 L0007 2 2 65
C0008 L0008 4 3 55
C0009 L0009 8 3 55
C0010 L0010 5 4 55
C0011 L0011 5 4 20
C0012 L0012 5 4 11
C0013 L0013 1 1 31
C0014 L0014 6 4 31
C0015 L0015 5 4 2
C0016 L0016 5 4 2
C0017 L0017 6 4 7
C0018 L0018 6 4 6
C0019 L0019 5 4 10
C0020 L0020 6 4 8
C0021 L0020 6 4 6
C0022 L0021 6 4 5
C0023 L0008 6 4 14
C0024 L0022 5 4 7
C0025 L0023 6 4 9
C0026 L0007 6 4 7
C0027 L0002 6 4 4
C0028 L0001 6 4 10
C0029 L0003 6 4 9

```

violations.log Arguments Output (stdout) Error (stderr) basic.utt courses.utt curricula.utt lecturers.utt relation.utt rooms.utt unavailability.utt

#106155 Succeeded 28-02-2014 11:47

Clicking on *Output (stdout)* the produced solution (output) of the submitted code can be inspected. In the example here it can be seen that course C0002 is scheduled in day 4 period 0:

```

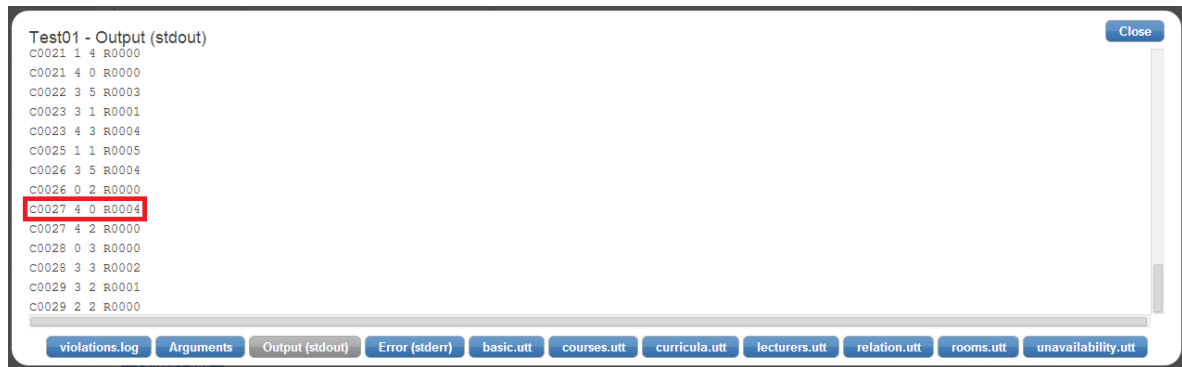
Test01 - Output (stdout)
C0001 3 2 R0002
C0002 4 0 R0001
C0002 2 3 R0000
C0002 3 5 R0000
C0003 0 4 R0003
C0004 1 1 R0001
C0005 0 1 R0004
C0005 4 0 R0002
C0005 3 2 R0003
C0006 1 2 R0004
C0006 4 2 R0004
C0006 3 1 R0000
C0006 1 5 R0005

```

violations.log Arguments Output (stdout) Error (stderr) basic.utt courses.utt curricula.utt lecturers.utt relation.utt rooms.utt unavailability.utt

#106155 Succeeded 28-02-2014 11:47

Scrolling further down it can be seen that course C0027 is also scheduled at day 4 period 0:




5.2.6 Feasible Solutions

When the solutions produced in all the tests are feasible the result of the test is *Succeeded*. An overview of all the tests is shown saying that the solution is feasible as well as the calculated objective value of the solution:

Your Submissions

#106178 Succeeded

28-02-2014 15:20 




Source Code

TestData

Graded: 28-02-2014 15:20

| | | | |
|--------|--|---------------|---------------------------|
| Test01 | Succeeded (The solution is feasible. The objective value is 2020) | RUNTIME 0.10s | Test Data |
| Test02 | Succeeded (The solution is feasible. The objective value is 4766) | RUNTIME 0.10s | Test Data |
| Test03 | Succeeded (The solution is feasible. The objective value is 4553) | RUNTIME 0.11s | Test Data |
| Test04 | Succeeded (The solution is feasible. The objective value is 4153) | RUNTIME 0.12s | Test Data |
| Test05 | Succeeded (The solution is feasible. The objective value is 4866) | RUNTIME 0.11s | Test Data |
| Test06 | Succeeded (The solution is feasible. The objective value is 5874) | RUNTIME 0.11s | Test Data |
| Test07 | Succeeded (The solution is feasible. The objective value is 5392) | RUNTIME 0.11s | Test Data |
| Test08 | Succeeded (The solution is feasible. The objective value is 4211) | RUNTIME 0.11s | Test Data |
| Test09 | Succeeded (The solution is feasible. The objective value is 3954) | RUNTIME 0.10s | Test Data |
| Test10 | Succeeded (The solution is feasible. The objective value is 5102) | RUNTIME 0.11s | Test Data |
| Test11 | Succeeded (The solution is feasible. The objective value is 1867) | RUNTIME 0.10s | Test Data |
| Test12 | Succeeded (The solution is feasible. The objective value is 2772) | RUNTIME 0.12s | Test Data |
| Test13 | Succeeded (The solution is feasible. The objective value is 4607) | RUNTIME 0.11s | Test Data |

[Download test data](#)

| | | | |
|---------|---------------------------|------------------|---|
| #106177 | Tests failed | 28-02-2014 14:41 |  |
| #106176 | Compilation failed | 28-02-2014 14:40 |  |
| #106175 | Tests failed | 28-02-2014 14:39 |  |

Clicking on one of the tests will show more detailed information of the specific test. All the files that were given as input can be inspected and a file named *penalties.log* is produced as well. 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 CodeJudge
- 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

Test01 - penalties.log

Close

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 |

penalties.log

Arguments

Output (stdout)

Error (stderr)

basic.utt

courses.utt

curricula.utt

lecturers.utt

relation.utt

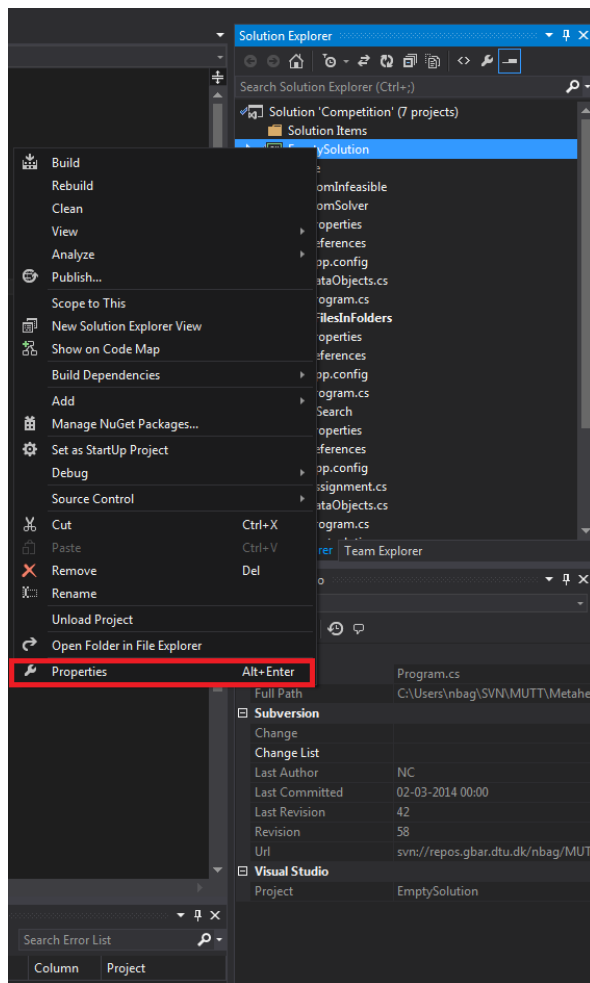
rooms.utt

unavailability.utt

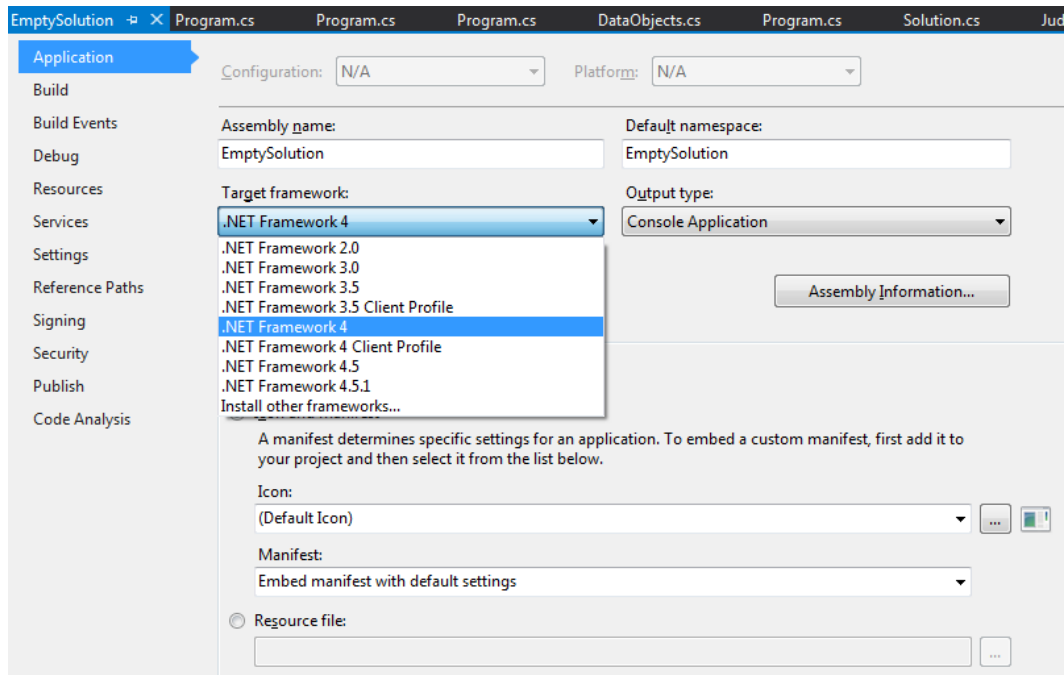
Note that even though the provided values of the objectives are incorrect in the last picture the test is still passed as the solution is feasible. This is to ensure that submissions do not get rejected in the competition even though the objective calculations are wrong since the result of the competition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

5.2.7 Special Note for C# Developers using Visual Studio

As mentioned in Section 5.1 it is possible to submit code written in C#. However it must be noted that only the .NET 4.0 framework is supported which means that special implementations exploiting the newer .NET 4.5 or 4.5.1 framework can fail to compile. If you are using Microsoft Visual Studio to develop your code then it can help you to ensure that you do not write code not supported by the .NET 4.0 framework. This is done by right-clicking your project in the solution explorer and selecting **Project properties** as illustrated in the following picture:



When the properties pane is opened go to **Application** and select **.NET Framework 4** or **.NET Framework 4 Client Profile** in the dropdown box under **Target framework** as illustrated in the following picture:



If your code can compile on your own machine with this setting it should also be possible for the CodeJudge to compile the code.

5.3 Benchmark tool

Because the optimal parameters for a heuristic often depends on the amount of iterations it is able to have in the given timelimit, a benchmark tool is provided. This will help to make sure that your algorithm perform similar on your computer and on CodeJudge. 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 **Projects/UniversityTimetabling/BenchmarkMachine**. There are three files as illustrated in the picture below:

File sharing: BenchmarkMachine

Create folder | Upload file | Storage usage | Transfer file from Dropbox | Import | Fo
Your browser supports drag and drop upload. Drag files onto the page to upload them

| <input type="checkbox"/> | Name | Date | Aut |
|--------------------------|--|------------|------|
| <input type="checkbox"/> | Parent folder (UniversityTimetabling) | | |
| <input type="checkbox"/> | benchmark_machine <i>Linux executable</i> | 03/03-2014 | Niel |
| <input type="checkbox"/> | benchmark_machine.cpp <i>Benchmarking source code</i> | 03/03-2014 | Niel |
| <input type="checkbox"/> | benchmark_machine.exe <i>Windows executable</i> | 03/03-2014 | Niel |

The file `benchmark_machine.cpp` is the source code of the tool. The two other files are a

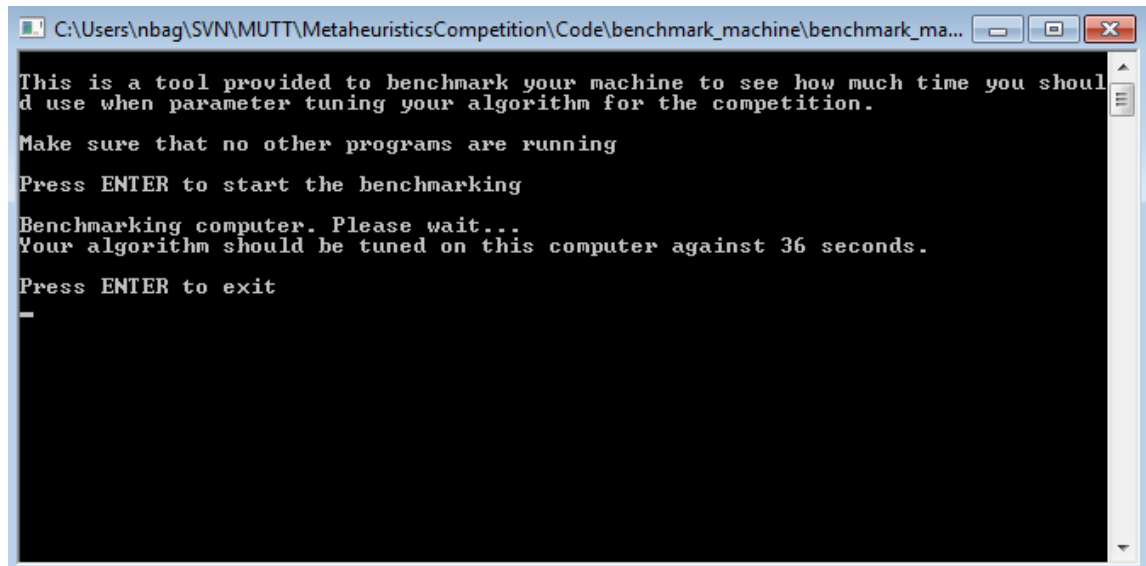
result of compiling the source code. The file `benchmark_machine` has been compiled with the `g++` compiler on Gbar at DTU using the following command:

```
g++ benchmark_machine.cpp -o benchmark_machine
```

It should be executable under most Linux-based operating systems. The file `benchmark_machine.exe` has been compiled with the `cl` compiler provided by Microsoft's Visual Studio using the following command:

```
cl.exe /EHsc benchmark_machine.cpp
```

It should be executable on most newer Windows operating systems (tested in Windows 7). The output of the tool is similar to the following picture:



```
C:\Users\nbag\SVN\MUTT\MetaheuristicsCompetition\Code\benchmark_machine\benchmark_ma...
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 other programs are running
Press ENTER to start the benchmarking
Benchmarking computer. Please wait...
Your algorithm should be tuned on this computer against 36 seconds.
Press ENTER to exit
-
```

In the example of the picture the computer that was benchmarked should be tuned using a time limit of 36 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.