**Student Timetable Creator**

Ron Wexler 307877340, Sagi Arieli 203637251

Supervisor: Naomi Unkelos-Shpigel

**ABSTRACT**  
*many students, like us, spend a lot of time on planning a time table for their next semester. In our project we utilize Genetic Algorithm based solution to help the students in Ort Braude College of Engineering generate a timetable. Automatic scheduling comes under the set of NP Complete problems, which makes it impossible to generate and validate quickly, we decided to use Genetic Algorithm to find an optimization to Ort Braude College of Engineering student's timetable problem. trying to satisfy the students requirements by creating a tailored timetable guided by their input. This will be the main part of a toolset we will offer to the Students in our web-based system, including course suggestions, and course and lecturers’ reviews. The system will also include some information for faculty members regarding students’ courses and enrollments.*

**Key words:** timetable generation, evolutionary algorithm, genetic algorithm, constrained optimization.

**INTRODUCTION**

In this paper we will be discussing both our genetic algorithm implementation for the timetable problem and our set of solutions as a whole. Fist we will introduce the genetic algorithm, expanding on this, the related work will include some options for each decision we had to make regarding the algorithm. Next we will explain why we chose each option for our implementation and how we overcame some of the challenges that are unique to this problem. The rest of the paper will overview other functionalities of our system and the software engineering documents.

The Main goal is to offer a set of solutions for the students of Ort Braude College of Engineering that want to pick courses and create a timetable for their semester. Each coming semester we the students of Braude encounter the same issue, choosing what courses to take and create a timetable that includes them as well as satisfy our needs such as specific days off, windows preferences, picking one out of many optional courses and more. We will offer a web-based solution that will include:

* For the student:
  + Timetable generation based on user input
  + Review for courses and lecturers
  + Semester courses suggestion
* For faculty members:
  + Enrollment data

We chose to use Genetic Algorithm to solve the timetable problem because:

* it was successfully used before to quickly solve slightly similar scheduling problems [[1](#ref1)], [[2](#ref2)], [[3](#ref3)]
* the optimization nature benefits us (unique optimized solution will result in diverse timetables between students, in contrast to optimal solution that might cause many students having the same timetable - which will cause stress on specific classes)

Genetic Algorithm is an algorithm that mimics nature’s evolution process. In nature each individual has a set of genes, making him better or worse than other in different fields. Many individuals form a generation, and individuals in this generation can reproduce (usually the fittest individuals in the population get to reproduce) to create an offspring, taking genes from (usually both) parents, and sometimes having some unique genes (mutation). Reproduction creates a new generation, taking the best from the last generation while trying to improve. In genetic algorithm individuals are possible solutions, solution is a set of genes that are a combination of chromosomes. Chromosomes are the most basic building block of the solution; each solution has a fitness level rating based on its genes and constraints for the optimal solution. A set of solutions makes a generation, and a generation can reproduce, utilizing crossover (2 parents create a child, or 1 parent copying itself to the next generation) and mutation (changing one chromosome in the solution). Choosing which solutions will reproduce is called selection and is based on the fitness (survival of the fittest). The timetable construction will be tailored to the student’s needs, taking as input his preferences such as:

* Courses to take.
* Courses groups (only 1 of the courses will be chosen by the algorithm based on the other preferences - this can be applied to general courses, elective courses, and picked by hand courses group).
* Number of free days.
* Specific free days.
* Minimum windows or maximum windows.
* Specific windows
* Ending and starting time for each day
* Preferred Lecturers.
* Whether classroom proximity matters (for some students having to go from L to P in 10 minutes is not feasible).
* Lessons that the student has no intention to attend.

**BACKGROUND**

Many work has been done in the field of scheduling problems, before analyzing it, let’s define the basics of genetic algorithm: [[1](#one)], [[2](#ref2)], [[3](#ref3)] describes similar way of presenting the basic concepts of the GA

Chromosomes - basic building block, some property of a solution.

Solution - individual solution, a set of chromosomes.

Generation - set of Solutions.

Fitness - value that evaluate the solution, based on its chromosomes and the desired optimal solution.

Selection - choosing candidate solutions from a generation to be part of the next generation and have a chance for mutation and crossover.

Crossover - creating a new solution (set of chromosomes) from an existing couple (number of parents might differ) using their chromosome pool.

Mutation - changing a solution’s chromosome.

**Genetic algorithm flow**

Common steps for GA flow can be seen at paper [[8](#ref8)]. First step is the initialization of the first generation. This step is done once. The size of the initial population is varied from problem to problem. Usually the first generation is randomly generated. The next steps can be done until we find a fit enough solution or for a specific amount of generations. Selection is the step where a section of the population is chosen for breeding, the selection is fitness based where fitter solutions (evaluated by the fitness functions) is more likely to be selected. Reproduction contains crossover and mutation operators. These are the operators that populate a new generation, often crossover is done in pairs, like in nature. The goal is to create a new generation that is different from the initial one, when generally we aim for higher fitness for every generation. Maintaining diversity is an important part of reproduction, because a local maximum might take over an entire generation and hold the algorithm from other possibly better directions. The main sources of diversity are the mutations and the crossover between high and low fitness solutions. The output can be the last generation or the fittest solutions in the entire last generation, since the latter is part of former, we will choose the last generation to be the output.

Pseudo code:

Genetic Algorithm (size\_of\_population,)

Population = generate\_first\_generation()

While (stop\_condition())

Population\_fitness = fitness(Population)

Parents = selection(Population ,Population\_fitness)

New\_generation = crossover(Parents )

New\_generation = mutation (New\_generation)

Population = New\_generation

Return population

**RELATED WORK**

**Representation**

According to [[8](#ref8)] there are 2 types of genetic algorithm: binary coded (every chromosome is coded to a binary string) and real coded, this are more natural to the problem representations and include for example:

Vectors of floating point.

Vectors of integers.

Ordered lists.

Tow dimensional matrix.

**Constraint management**

In addition to optimizing the solution based on the preferences, some problems may occur when a set of constraints could make a solution infeasible. The difference is that an un-optimized solution may not be the best, yet be acceptable. While the infeasible solution is not valid. Document [[5](#ref5)] offers classification of the constraints to a direct handling (i.e. directly search for them and handle them differently than optimization problems) and indirect handling (i.e. dealing with them like an optimization problem). Direct approach means that the objective function (fitness) will not include them, thus the solutions will not become more feasible over time, because the genetic operation will be “blind” to these constraints. Indirect approach means that we create a penalty function and include it in the objective function (merging the fitness and the penalty). The main advantages of indirect approach are: Generality, reduction of the problem to a “simple” optimization, and the possibility of embedding user preferences by means of weight. The disadvantages are: Loss of information (by packing everything into a single number), and it requires choosing a method to merge fitness function with the penalty function. An in-depth research on how to deal with constraints penalties in GA was can be found here [[4](#ref4)]. The article notes that constraints problem is challenging to handle in GA, and proceeds to explain different methods proposed from other researches. The most common and effective way of dealing with constraints that the paper discusses about are: Death Penalty: a method that rejects infeasible solutions, if a solution is deemed infeasible, it will no longer be part of this generation’s population. The advantage of this method is that we have to deal only with feasible solution, and thus more likely to populate the next generation with more feasible solutions. This however can also cause problems, if we choose to not replace those who die we might end up with a small population or even an empty generation. And if we choose to replace infeasible solution with feasible ones, it might waste a lot of time finding them, and reduce our population variety, limiting the search range, and preventing the algorithm from finding a better solution. The latter is more likely to happen in a problem where the feasible solution number is noticeably smaller than the entire solution range. Penalty function: this method transformed a constrained problem to unconstrained, there are mainly two ways to do so: the additive way and the multiplicative way. The additive way function is as follows:

Where is a solution in the population, is the feasible region of the problem, is the fitness function a solution, is the penalty function and is the total evaluation score of a solution in a population. If we use a minimization problem, the function should be a positive number that we want to minimize to get the fittest solutions. The penalty function should be constructed in a way that promise that if the constraint is held, the penalty is 0, otherwise the penalty should be high enough that the solution is pushed away from the feasible region. Choosing a big weight to the penalty can cause issues such as converging to a feasible solution very quickly even if it is far from the optimal. Similarly giving low weight can be a problem too, and will cause to spend too much time in searching an unfeasible region. Thus, GA would converge to an unfeasible solution. Finding the right weight must be a product of testing.

**Selection**

The selection operator in the genetic algorithm defines which individuals will proceed to populate the next generation and be candidates for reproduction and mutation, there are a few commonly known methods, each with its advantages, disadvantages, different time complexity and may suit better or worse for different problems. A more thorough examination of different kinds be can found in this paper [[6](#ref6)]. Tournament selection: this is a selection process that includes picking a winner from a set of candidates to be a part of the breeding pool (future parents), we take a small set of candidates from the population into a method, this method takes an number of random individuals (usually =2) from the population, the best individual is chosen in the group and is assigned with a variable . In case of a binary tournament. The other individual will be assigned a variable q. . The winner will be chosen according to the chance of being picked by and q respectively. This operation can be repeated as much as desired (usually until the breeding pool is full).

Proportionate Reproduction:

A group of selection methods where individuals are chosen to reproduction based on their objective function. For example, the most common Proportionate Reproduction method is: Roulette Wheel Selection: in this method, the selection is being made based on the proportion of the individual’s fitness against the generation’s fitness. The probability of choosing individual p as a parent with an objective value (fitness) and on the *t*th generation is:

Where k is the number of unique fitness levels, is the number of individuals with the same fitness value *i* on the same generation *t*.

**Crossover**

Crossover occurs by probability in each individual that was selected to survive. In many studies crossover is referred to as two individuals changing themselves, as if the input is two solutions and the output is two solutions that are a mix of the input solutions gene pool. Another approach is more nature like and defines crossover as two parents (solution) creating one new solution using their genes. Paper [[8](#ref8)] Suggests a few common crossover methods for real coded genetic algorithm: Discrete crossover: each child’s gene is chosen uniformly from {parent1, pareent2}. Simple crossover: chose uniformly an index *i* in the genes list, and produce an offspring such that the first *i* genes are from one parent and the next genes are from the other parent. Another important crossover method is shown for binary encoding: N point crossover: n crossover points are randomly selected and the segments of the parents, defined by them, are exchanged for generating the offspring.

**Mutation**

The purpose of mutation is to explore more possible solutions to the problem. Like crossover, mutation has 2 approaches as well, one is that mutation get as input a gene and changes it, allowing 1 solution to be mutated more than once in a generation (each gene has a probability to be mutated). The second approach is that the mutation gets a solution as an input and chooses what gene to change, this means that the probability to mutate is applied to each solution and not each gene. A few Common methods for mutation as described in paper [[8](#ref8)] are: Random mutation (uniform mutation) - A random gene within a chromosome is chosen and its value changed within the domain it is belonged to. Non-uniform mutation-  
Same principle as uniform mutation but for each succeeding generation, the domain of the possible mutation value is decreased in order to promote the solutions to stabilize towards a local optimum as generations continue on. These methods are for the second approach (solution as an input).

**Local optimum debate**

Since we want to make the GA as optimized as possible, it is important that the algorithm converge to an optimum with the least amount of generations possible. If we are not careful enough, the algorithm may converge at a local optimum, to prevent this a probability on the operators of cross-over and mutation is introduced as and . With the help of this values, the region of the algorithm is expanded. Defining a too low value could make these operators irrelevant and for a possibility of premature convergence, on the other hand, defining a too high value for mutation chance could lead to a purely random search algorithm or, for crossover it would lead to a minimal diversity in the feasible solutions region. Choosing the right value for these operators can be a very difficult task itself. Usually these probabilities are constants, and we apply them to each solution that was selected in the selection in order to choose if he gets to be part of crossover, mutated, or left as he is. Paper [[7](#ref7)] propose an idea for an adaptive GA so that the chances of mutation and crossover to occur will not be constants but, whether be dynamic according to the fitness of the individuals. There is a known consistent pattern in GA that says: the closer the maximum fitness from a generation to the average fitness in that generation means that: the algorithm is converging and it is getting closer to some sort of optimal solution (global or local). With that knowledge, the values of mutation and crossover could be adjusted according to the relation of maximum fitness and average fitness in that generation. A function is required to increase the values of the operators when the algorithm is about to converge or decrease the values otherwise. Many of this function requires finding the max fitness and the average for example:

Where f’ is the maximum fitness level of the solutions in the generation, f is the fitness of the solution, is the maximum fitness possible, and k1 is some constant between [0,1].

**GA FOR BRAUDE TIMETABLE CREATION PROBLEM**

As mentioned before, Braude timetabling problem is a scheduling problem that belongs to the family of NP-complete problems, GA is known for its advantages regarding this kind of problems. Every GA consists of its basic operators: Selection, Mutation, Crossover, etc’. We will discuss the specific implementation and the reason behind every decision we made for the Braude timetabling problem. Solution representation (what’s considered a gene?): We will be discussing the differences between the representations of solutions, but, the key difference between the representations is the definition of genes. There were two kinds of representation that were on the table for this problem. The first is an array of lessons, the 2nd is a timetable. Array of lessons representation: this is similar to vector of integer representation in a way. Each solution is represented as an array of lessons, each lesson contains course id, lesson type, starting hour, ending hour, lecturer, and classroom. This representation can be encoded as an integer vector where each value is the encoding of a lesson (with all of its attributes). Encoding to vector of integers, although possible, does not benefit the algorithm in our problem in any way, usually real genetic algorithm encoding takes advantage of numerical vectors in order to calculate the fitness. This is not possible in this scenario. Hence, we will use the array of lessons without encoding it to a numerical value, but is it still possible to use crossovers and mutations methods designed for vectors with numerical values. In this representation our genes are active lessons and not empty periods. This has a few drawbacks when using crossover, crossing genes between 2 good parents could cause overlapping lessons on the same period and creating an infeasible child. For example:

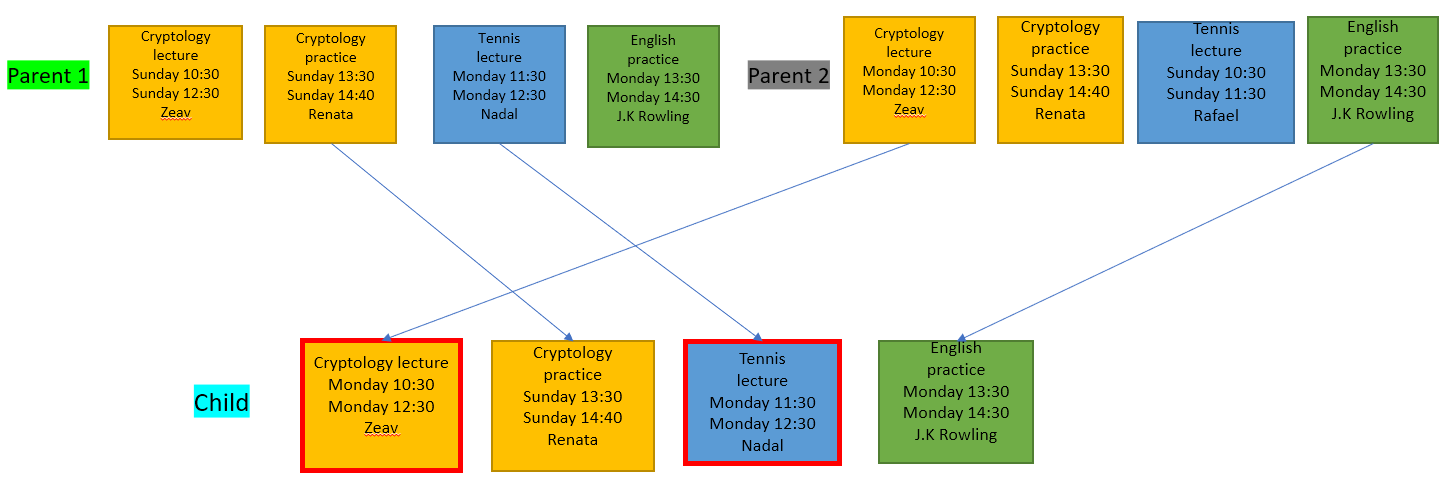


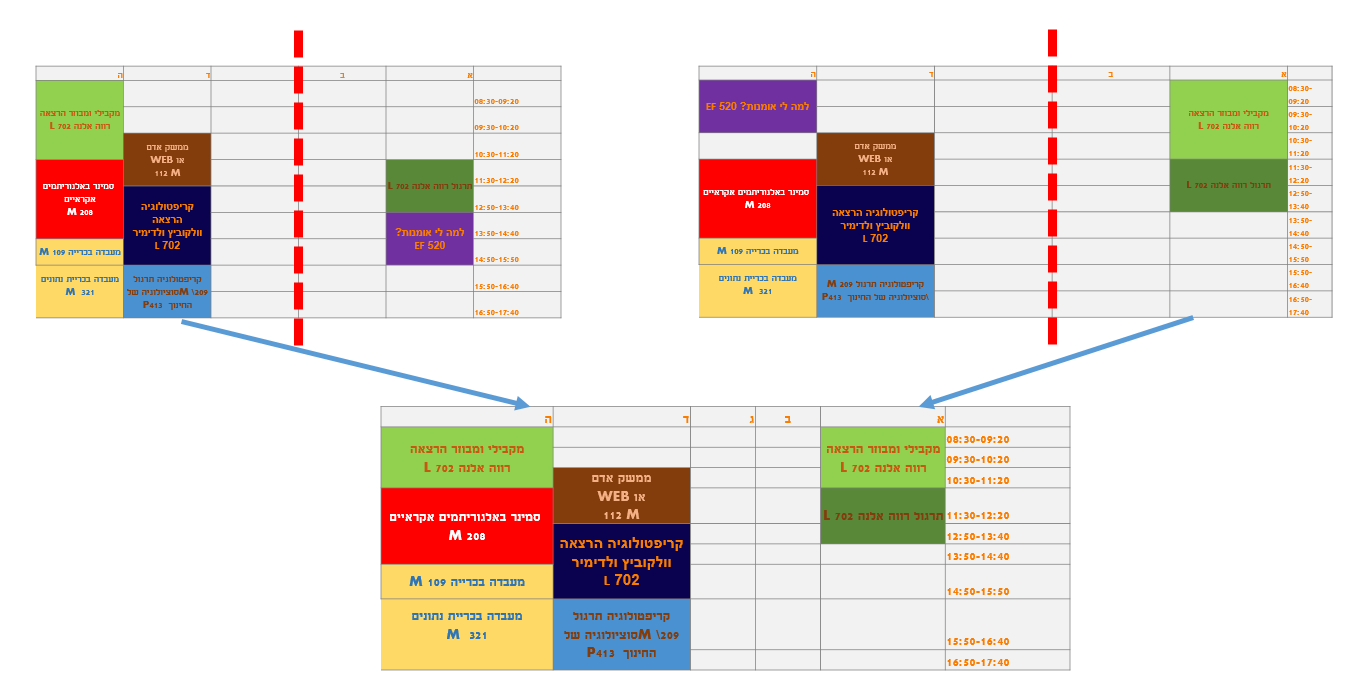
Fig. 1: Crossover overlapping lessons

In [Fig 1](#figure1), we can see that the child array has overlapping lectures on Monday 11:30-12:30.

Timetable representation: Time table representation is a 2-dimension matrix, 1 dimension is for the days and the other is for the period. In this representation we hold all the periods (including empty ones) for each solution, and in each period, we save lessons. Hence the genes are the periods. This representation means that in cross over we take periods from each parent and not only lessons. Crossover methods for this representation can cause issues as well, 2 good parents with all the required classes can create an infeasible solution with 2 of the same course’s lesson type (easy to maintain by deleting one) or a missing one (a bit harder to fix).

For example, [Fig 2](#figure2) shows 2 valid parents and a child with 2 green lectures and no purple lesson at all. Fixing both drawbacks: Both of the representation problems are caused by constraints that make a solution infeasible. In the lesson array representation, the constraint is “no overlapping lessons in the same period”, and the constraint in the timetable representation is “no missing lessons or more than one occurrence of the same lesson” (same lesson = 2 different lessons from the same course and the same type). As discussed in the related work section, there are different methods for dealing with constraints.  
In this case, if we want to eliminate the infeasible solutions from the population (direct handling) the best method is gene editing after crossover (changing a few genes in an infeasible solution to make it feasible). And, if we choose indirect handling approach, we can use some sort of a penalty function. Allowing solutions that break the “no overlapping lessons in the same period” constraint into our population can be helpful for the algorithm to find a better solution because we are searching for the best schedule (the best timing for the input – a set of courses). While not allowing them, might leave us with a small gene pool.  
On the other hand, allowing solution with less or more than the required courses to the population will not help us provide better schedule, we are not searching for the best courses set, but for the best timing of lessons, hence our question is when and not what. (We do allow for students to insert a course set and let the algorithm choose 1 from the set but this is different than letting the algorithm choose the best course set of all the required courses). These are the key factors, for the decision of the solution representation:   
1. We concluded that the constraint of “overlapping lessons” violation can help the algorithm as a penalty parameter. Unlike the time table constraint, as mentioned before, using the indirect approach does not benefit and may even harm the population.

Figure 2 Timetable Crossover



2. In case we want to prevent infeasible solutions (direct approach), both options will require the same amount of work. For example, we can use Death Penalty method for infeasible solutions for both cases.

3. In addition to the constraints that each representation has to take care of, timetable array also has the constraint of “overlapping lessons”, when mutating and when constructing the 1st generation.

This makes two constraints for the timetable representation. According to the above factors, we concluded that the array of lessons best suits for our representation debate. Testing for the better representation: In order to test the two and further validate our opinion we created a prototype for the algorithm and tested both representation options. The prototype was written in Python, the input included various permutations of seven courses that each has lecture, practice, lab type and with different periods and lecturers.  
We compared the results by the fitness level of the last generation and the amount of infeasible solutions. We tested on various population sizes and number of generations per run. The theoretical issue between the two representations focused heavily on the crossover operation. We used the crossover method of Simple crossover for both representations, where in the timetable representation we chose period number as an index, while array of lessons used the lesson’s index. For mutation we used Random mutation for both, in timetable we chose an empty period by random and tried to fit in a lesson to add to it (if found and it fits, we switch its current period to the new one), while, for array of lessons we pick a random lesson from the array and switched it with another possible lesson that possess the same type and course. The first generation was created by randomly filling lessons from the input courses. Fitness was evaluated by minimum number of windows and maximum free days. As well as the added penalty for violating the constraint (each violation added a big value to the fitness level). From the results we saw that the timetable did not reach good fitness levels, the final generations was filled with a lot of infeasible solutions, because most of the crossovers negatively impacted the fitness as witnessed for each generation. On the other hand, array of lessons did a better job and generated the best possible solutions in most of the tests.

**Course groups and optional courses**

In Ort Braude college course structure, there are 2 important points that has to be sorted if we want to truly generate useful time tables.

1.    Courses groups – every course has at least one course-group, a course-group is a subset of the appropriate course-lessons in that specific group, acting as sort of a mini course within the course, making one lecture compatible with specific practices and labs according to the timetable restrictions.

2.    Optional courses – every student has to choose between a set of courses during his semesters in Ort Braude college, including general courses, sport courses, and department specific optional courses (divided into clusters by subject).

We want to give the students the freedom to create the most suitable timetable for them. That’s why we consider this problem as a constraint in our solution and our way of dealing with it by not allowing infeasible (combining different groups or not including requested optional courses) solution from generating at all. Our solution will allow students to create their own set of courses as well making the algorithm choose the best suitable one. After tackling the problem of defining the course’s groups representation, solving the representation problem of optional courses can be treated as the same solution, because a course group is actually a valid representation of a course, therefore having a set of this groups can be defined as a set of courses. We came up with a data structure for the courses and lessons that will solve both of the problems simultaneously. Course is defined as either a set of courses or a set of groups. Every course group includes the set of this group valid available lessons. ([Fig 3](#figure3)). This representation help solves the problem because:

1. A course with more than one group will include a set of groups, each group will only include the set of appropriate lessons in that group.
2. A set of optional courses will be represented as one course, while having a set of courses (each sub course is an optional course)

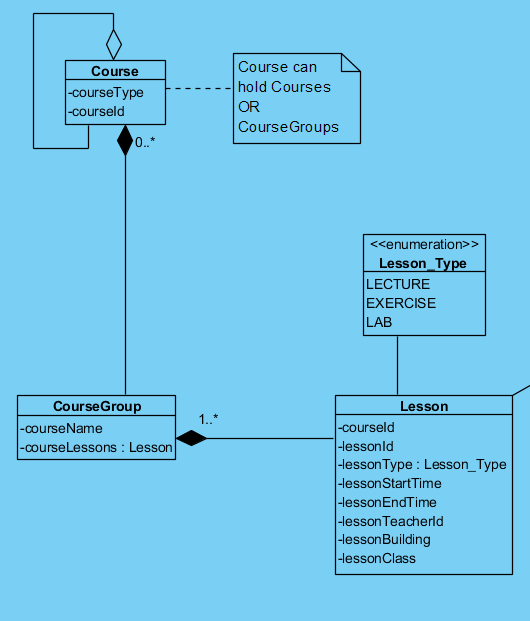


Figure 3 course groups class diagram

Course groups description with example:

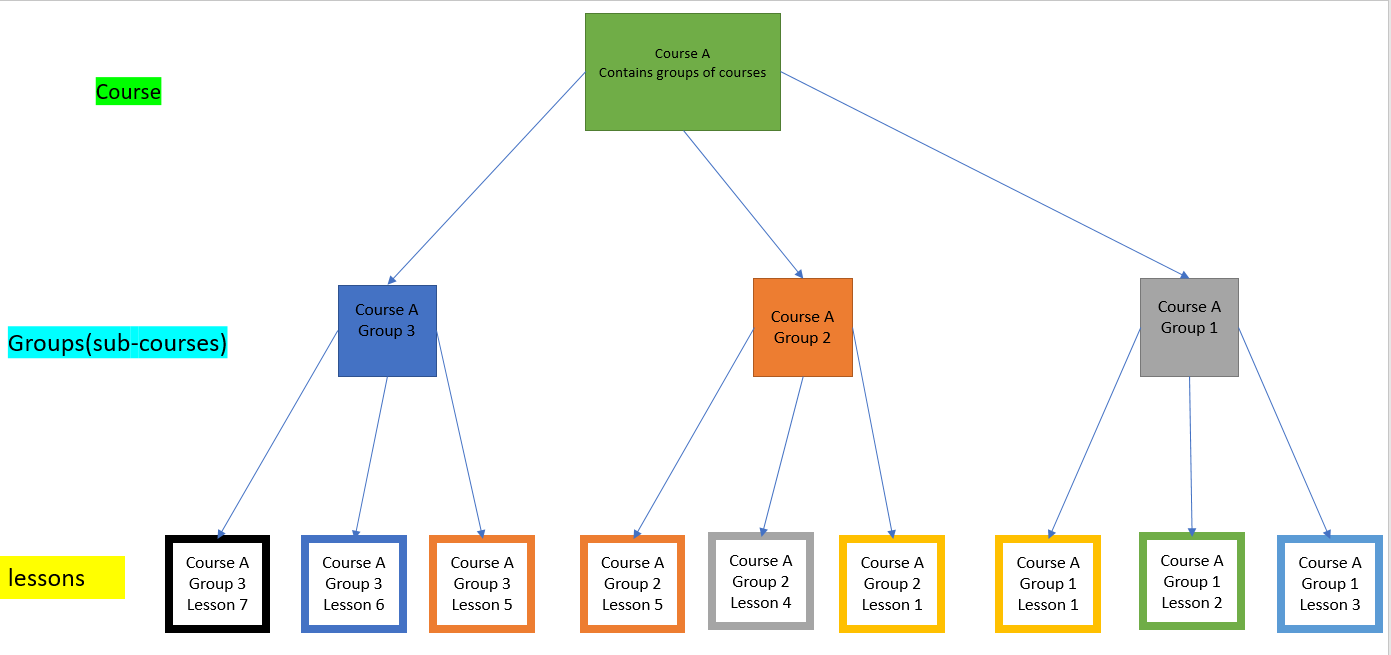


Figure 4 course groups example

In [fig 4](#figure4) we have course A that has three possible groups to form up a valid course and its lessons in the solution. With our representation, we can refer each course group for course A like an individual course of itself, this means that each group has its own pool of valid lessons to choose from. our solution, a lesson with whatever type that belongs to a certain group, can only exist with lessons of different types and in the same group. for example: Course A can be built from three different groups, for each group Course A can be built from only the lessons of the chosen group of course A. possible solution: Course A, Lessons 5,6,7. These lessons are of different types and are in the same group. Not possible solution: Course A, Lessons 2,4,5. Lesson 2 belongs to group 1, while lessons 4,5 belongs to group 2.  This solution is not valid and cannot happen in our representation.

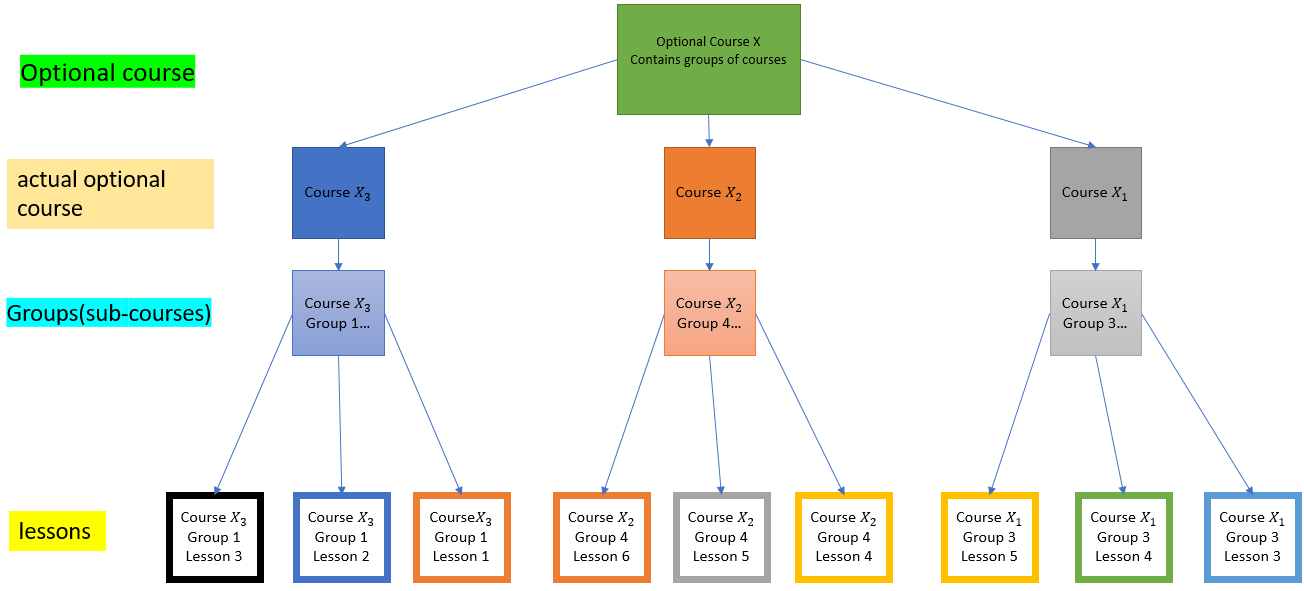


Figure 5 optional course example

Optional course description with example:

[Fig 5](#figure5) describes the structure of the representation of optional courses. The only difference from course-group representation is that, the algorithm refers to an optional course as a set of courses, while every course is a different from the others, but all of them belongs to a certain set of optional courses (X1, X2, X3, € X). Every actual optional course functions the same as a course group, and could have multiple groups.

The effect on genes:

Genes will have to be dynamic now, in some cases we will still consider lessons as the gene but in other cases the group or the course will be the gene. In mutation when we choose a lesson to change, we want to switch it with another lesson in the same group to not keep incompatible lessons in the same timetable. in order to create diversity and avoid local optima, mutation should have the chance of treating course and groups and the gene, in this case we will switch all the lessons in the course (because switching course or group means changing all the lessons). In crossover, there is the risk of 2 incompatibles (different courses groups) parents creating a child. to solve this, if in any course there is no match between the 2 parents, we treat the entire group as a gene and pick all the lessons of this course from 1 parent, otherwise the lessons are the genes as before.

**Evaluation**

Our fitness is based on a minimization function, with optimization factors and a penalty function that are merged together to get the evaluation score for each individual solution. Some of the optimization factors may appear or not depend on the user’s input. In order to maintain an evaluation score that we are able to work with.

Objective function:

Each violation from a constraint (optimization factor) adds a positive value to the fitness score according to that constraint weight.

Penalty function:

The objective of the penalty function is to have a more significant weight to a constraint that we define as making the solution is infeasible and thus, these solutions evaluation score will be heavily reduced by considering a penalty function. The penalty function consists from the number of overlapping courses in the solution. We decided to choose this factor for the penalty function, because the impact it has is much more significant than all the other factors. Every consecutive hour of overlap has an exponential growth bonus to the penalty function. this way solution that have a significant number of overlapping courses will have their fitness score very poor and will be less likely to be selected for reproduction in the selection phase. A pseudo code for this can be looked like the following:

For each period

If there are more than one lessons on this period

Add the amount of overlapping lessons -1 to the penalty

Total evaluated fitness score: After we calculated the objective function score and the penalty function score, the total fitness is calculated by adding the penalty function score to the objective function score.

These are the constraints and their pseudo code in the current build of the algorithm:

Windows: generally, students dislikes windows in their timetable (unless stated otherwise). this constraint checks the number of windows (hours), and adds a value for each hour of window in the timetable.

Windows value pseudo:

For each day, search first course

Then search next course in that day

Calculate gap between courses

Add the value of the gap

Specific windows: the user have the option to insert windows in specific preferred periods. The algorithm checks these specified periods and adds a value if these periods are not considered a window in the timetable.

Specific windows value pseudo:

For each chosen specific window

If this period is not empty

Then add value

Amount of free days: generally, students like to have as much free days as possible, the algorithm adds a value for every day of study.

Amount of free days pseudo:

For each day

If user has a course

Then add value

Specific free days: the user has the option to specify a day or days to have a study free day, the algorithm adds a value if these days are not free days.

Specific free days pseudo:

For each chosen specific free day

If user has any course in that period

Then add value

Preferred lecturer: the user has an option to choose a preferred lecturer for a course. The algorithm checks every course which has that input and adds a value if lecturer is different from input.

Preferred lecturer pseudo:

For each course in the courses array with the given preferred lecturer

compare current solution lecturer to preferred lecturer

If lecturers are different

Then add value

Start time/end time: the user can insert a specific time for the preferred earliest/latest studying hour period, the algorithm adds a value for each day the constraint is violated.

Start time/end time pseudo:

For each day

Get current solution start time to user preferred start time

If solution’s first class is before preferred start time

Then add preferred start time - first class time

If last course is later than preferred end time

Then add last course time - preferred end time

Lessons classroom: the algorithm adds a value for each adjacent lesson that are far from each other’s location (long walking distance).

Lessons classroom pseudo:

For each lesson

Get lesson’s classroom location

If solution’s classroom is physically far from next adjacent lesson’s classroom location

Then add value

**Selection method**

For selection method we choose the Roulette Wheel Selection method. The reason behind our choice is that other non-Proportionate Reproduction methods were too random for our problem. For example, the tournament selection had an equal chance to choose two low fitness candidates vs two high fitness candidates. In our problem crossover can easily create an infeasible solution, and because of that we want to minimize the crossovers of infeasible solutions (the probability of an infeasible solution being created is higher when one of the parents is infeasible). Proportionate Reproduction methods give a lot more emphasis on fitness, hence, it will suit this problem better. Our problem for creating a new population by crossover alone is not optimal (as discussed crossover can easily create infeasible solutions) and we need to copy some of the old generation’s population to the new generation. We can also enable elitist approach where we reserve a few solutions that have the best fitness for certain reproduction without even doing crossover and mutation operations on them choose.

**Crossover**

The one solution (one child) output approach is more commonly used and that’s why we chose it. We considered either Simple crossover or N gram crossover as the crossover method for the timetable representation because of the way we can implement them using periods as indexes. But since we chose the array of lessons as our representation the most appropriate crossover we figured, was Discrete crossover. Because of our dynamic gene’s, it is difficult to adapt anything different, Discrete crossover chooses uniformly for each gene the parent that the child will inherit the gene from. This allows us to do lesson by lesson and checking for compatibility of the course group with both parents. If they are incompatible then the gene is the entire course and all its lessons, otherwise the lesson is the gene. If one parent was chosen for crossover, the other parent chosen is its neighbor in the array, this is to simplify the pairing operation and does not damage the process in any way because these solutions were inserted by random to the array from the first place.

**Mutation**

Our mutation will take a solution as an input. We considered both Random mutation and non-uniform mutation for our implementation. Random mutation matched our problem pretty nicely for the same reason we chose the crossover method. The dynamic aspect of our genes fitting a method that separate genes and takes care of each one independently. Random mutation allows us to randomly pick a lesson and then decide what to consider as a gene. We will choose what is a gene by random (the probability for each one is not already determined, because we must test in order to make the correct decision), as said before a gene can be the lesson, the course group, or the course (if it’s an optional course). Non-uniform mutation expands on this idea by reducing the domain of genes we can switch to after each generation. This can be useful in cases where mutations are making the solutions too scattered. A way to implement this in our problem is to discard lessons that has constantly reduced the fitness of solution that possess them. But this introduce two issues:

* 1. How will we know if the gene is in charge of decreasing the solution fitness?
  2. This gene might be bad for many solutions but it might still be a part of the optimal solution, and removing it might hinder the progress of the algorithm.

The first can be solved easily by changing the gene and comparing the fitness levels before and after the change. The second is harder and require testing, if we encounter a problem where mutation is scattering our solution too much, then we will use this method in an effort to improve the impact of mutation on the population and test for the best way to reduce the domain of each gene.

**Local optimum debate**

In conclusion to what was discussed in the relevant work section, a critical part of GA is choosing the best values for and . This is a task that requires heavy testing to achieve good optimization. Overall it seems that crossover probability should be around 0.5. The mutation probability is usually smaller, and around 0.1. We will also try to exploit the adaptive GA method, calculating for each generation the probability of crossover and probability of mutation. Comparing some of the functions mentioned in [[7](#ref7)]. It is also important to note that on our implementation reaching a pre-defined number of generations is our termination condition for the algorithm. From our tests the algorithm should be able to reach a good enough valid solution in relatively few generations, so we decided that a constant number of generations will suit our algorithm fine.

**Time complexity**

Time complexity can vary a lot between implementations. We left a lot of place for testing various methods and thus, we have a lot of possibilities for time complexity. Here we will examine the time complexity of one of the many options we can do.

Variables:

* Number of courses: C
* Number of lessons in course i:
* Number of lessons in the course with maximum lessons:
* Number of all the lessons (representing the maximum number of lessons a solution can have): L
* Population for each generation: P
* Number of generations: G

Note that L can change if there are optional courses, and some of them have more lessons then other, in this case L will be the maximum lessons possible for the course set. and will be 1.0 for this example, because they fall somewhere between 0 and 1 and change each generation, to simplify things we will say there are P mutations and P crossovers each generation (this is in fact the worst case and is theoretically possible). The flow can be divided like so:

First generation: this is where we construct the first generation, for this example we will generate it randomly. The time complexity will be for each solution in the population for each course pick all the lessons by random: = O(PL).

Evaluation: here we use fitness function and penalty function, we can divide the fitness and the penalty functions to three categories based on their time complexity:

* + 1. O(C): go through all the courses for example: Preferred lecturer
    2. O(L): go through all the lessons for example: Lessons classroom
    3. O(1): go through all the days or all the periods, this is a constant number, for example: Penalty function of overlapping lessons

Hence the time complexity will be:

Selection: using the Roulette Wheel Selection, we have a probability for each solution for further reproduction with a function: .

We can compute the denominator once, and insert for all the solutions in a generation, was already computed in the evaluation phase. Each candidate for reproduction will receive “tickets” with the amount from the Roulette Wheel Selection times 100. Now that each candidate has tickets, we build an array where each candidate will have the number of cells in that array. A random index will be generated between [0, number of tickets] to pick the chosen solution for reproduction.  
Time complexity will be:

Crossover: using Discrete crossover, here in the worst case we will go through every lesson choosing a dominant gene from one of the parents (picking a course or a group as a gene means that we will pick more than one lesson at a time, hence the worst case is that the parents are compatible in all the courses).

Creating the child will take:

Mutation: here we will use Random mutation, the worst case is changing an entire course, and picking the course with the most lessons. The time complexity for this is:

.

Overall time complexity:

= O(PL) +

= O(PL) +

= O(PL) +   
(note that P can join the two PL’s because L)= O(PL) +

(PL because all L (the lessons) include)=O(PL) +

()== O(PL) + =

**FURTHER EXPLANATION ON OTHER FUNCTIONALITIES OF THE SYSTEM:**

**Database**we asked for permission to gain access to Braude’s database, or its schema, in order to create an implementation with a potential to serve Ort Braude college for real life scenarios. In case we do not gain any permission, we face two problems:

* 1. Courses information for the timetable generation and reviews, as well as for the recommendation of courses.
  2. Student information for faculty member functions as well as, also real student login option is not possible.

To solve the Courses information issue, we can get the required information from Barude’s ‘info’ website in advance, or during the students semester, using a web API to query what courses are available, and their lessons time, class, lectures, groups, credits, and prior required courses. To tackle the student information issue, we can create a scheme that will represent the real scheme, and fill it with testing data.

**User types**

**Braude faculty member**

This user is intended for the secretaries of Braude in general and for specific faculties. In order to better understand the faculty members needs we conducted an interview. The Interviewee was the assistant department head of SW department. She described for us a list of issues, mainly regarding data that is difficult to gather or just no possible. To let the staff members have easier way of getting data, we proposed a tool that a built in query system to generate some reports. These reports should help the faculty staff in theory by acquiring data that cost a lot of effort or data that was gathered by guessing or from experience. One report is about showing all the students who completed certain courses. Previously, this report was generated purely by guessing and from the experience over the years of these courses grades and number of students registered per semester. This is going to help for the following reasons;

* The relevant population group will receive Emails based on the courses it hasn’t finished yet and have completed the prior courses.
* With this information available, the secretary could better estimate enrollment numbers.

Another report is for students that enrolled to less than required credits this semester, right now there is no easy way to perform this basic function.

**Student:**

The students will have access to more functions other than the timetable creation:

1. **Review of lecturers and courses:** the students will be able to write and browse reviews from other students. Lecturer reviews include score for knowledge in the field, teaching skills, over hours accessibility, and overall score. Course reviews will include score like course difficulty and interest. Course reviews will also include questions like: are there weekly submissions, are there assignments, and is attendance mandatory.
2. **Courses recommendation:** once we have the student’s completed courses via an input by the student himself or from Braude’s database, the student will be able to use the function of “courses recommendation”. The student has the option to choose courses that must be on the recommendation list and courses that he doesn’t want to have on the recommendation list, also the student will choose the difficulty level and credits amount for the group of courses he want to generate. (to make it easier for them we could build presets based on semesters and they could edit the courses list), or using simple GA can be used here with a solution as a set of courses and gene as courses where the goal is to maximize the credits and still have the desired difficulty level, it will allow us to present the user with even more than one possible option (all of the solutions in the last generation).
3. **Editing a timetable:** there will also be a drag and drop utility to edit the timetable that was generated by the GA. When clicking on a lesson, all the available option will pop on the timetable, and the user can use them freely. The student can delete courses from the table as well as add, if a student chooses to add, he will have to pick the course and then pick a time for each of its lessons.

**Technologies**Genetic algorithm: the algorithm code will be written in Python language because we have experience in Python, and it can be written relatively faster and simpler than other languages that we are familiar with. We are aware that Python will perform slower in compare to languages like C++, C#, or Java, but since we wrote our test using Python we expect it to run fast enough for the system’s requirements needs.   
Database: the database will be created in MySQL, because of our experience in MySQL from our school and work. We did consider a non-relational approach like MongoDB, BigTable or Redis but we preferred to stay with a more mature and familiar database.

Web: because we have no experience with web, we decided to use angular with bootstrap as it is very common and will allow us to create a more robust system in contrast of just using JavaScript with HTML. This will also allow us to create a responsive web application.

**EXPECTED RESULTS**

With our project, we expect the timetable generator, with all of the mentioned features for the algorithm, including the different user preferences to be fully implemented, and optimized (after testing the options we mentioned). Our goal is that the system will be in a useable state, allowing students to start generating timetables, write reviews and browse them. In regard to the other functions in our system, we might not be able to implement all the features for the student and faculty member users.

**PRELIMINARY SOFTWARE ENGINEERING DOCUMENTS:**

**Requirements:**

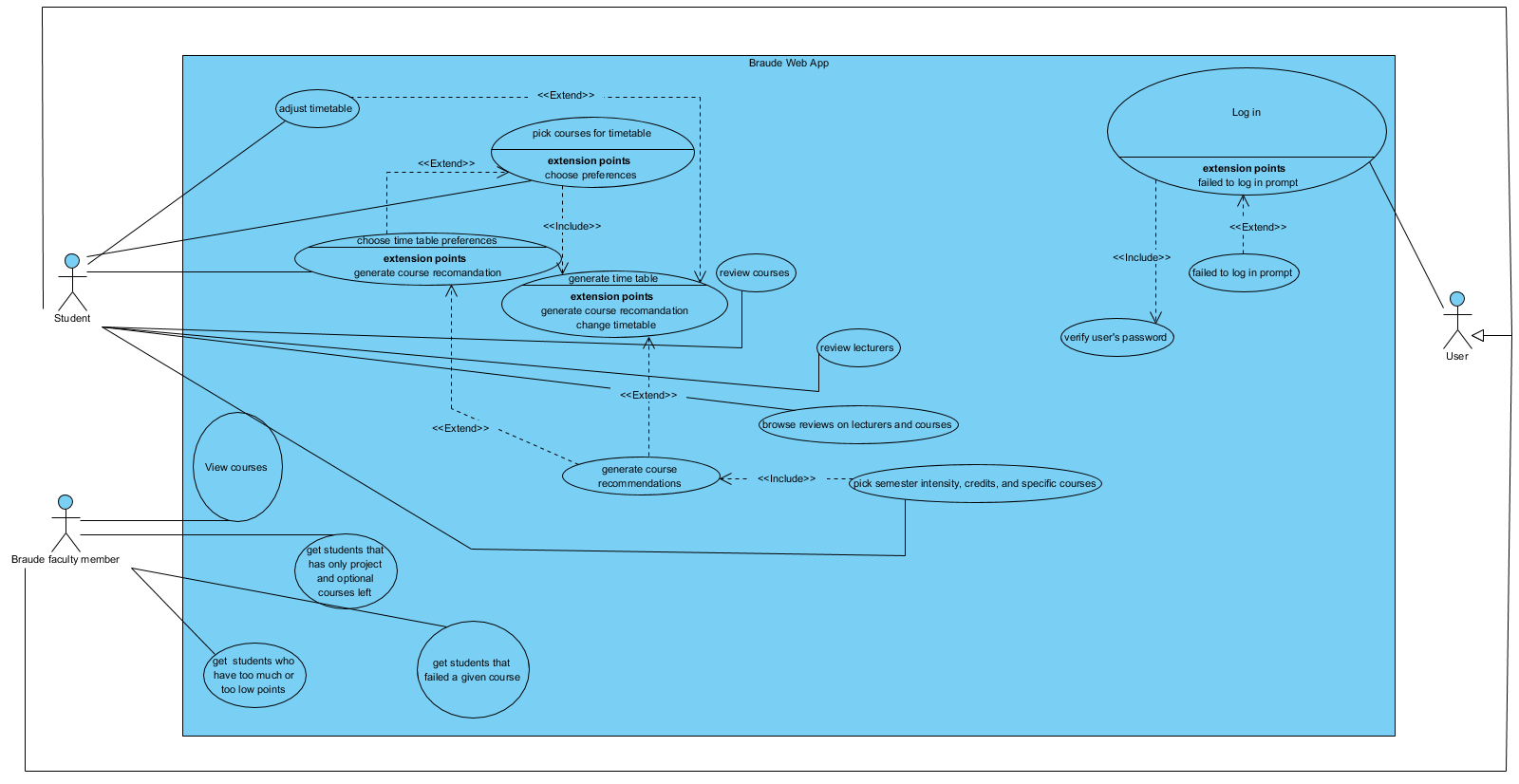


Figure 6 Use Case

**Class Diagram:**

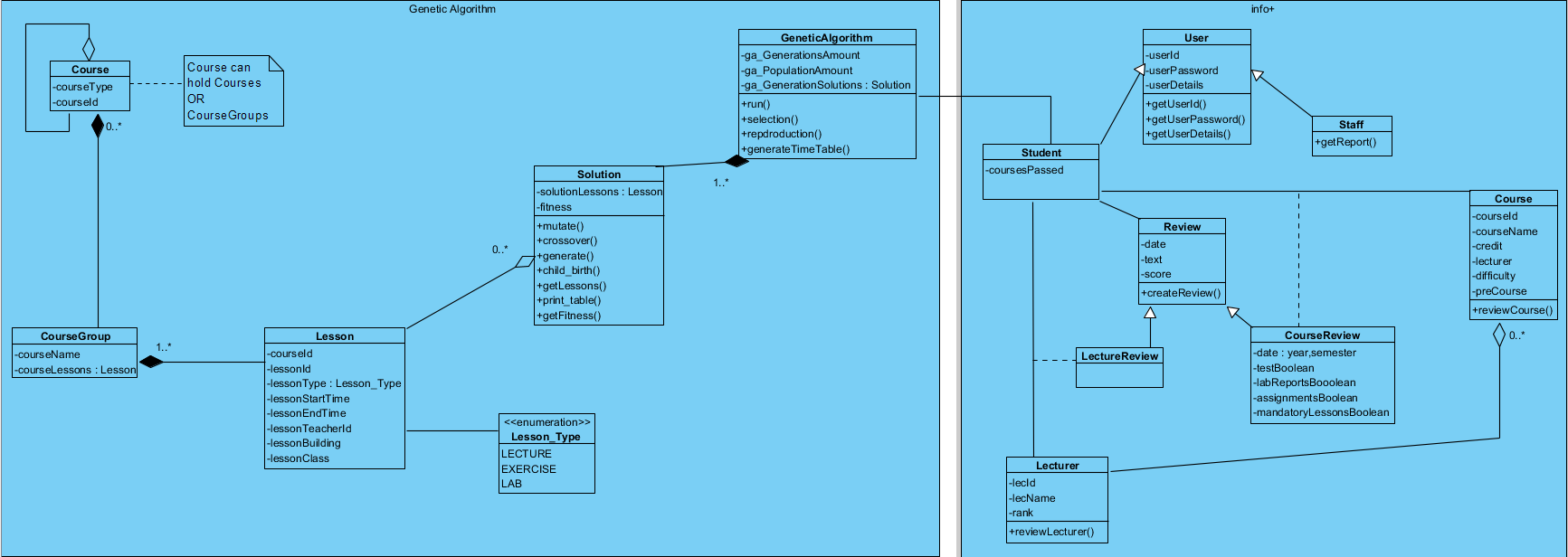


Figure 7 Class Diagram

**GUI:**

Our header will be consistent through all of the pages in the system, shown at the top with the different functions available, and just below it are the “stages” that are required by the user from left to right. These are essentially buttons that are clickable allowing to move back at any time, but moving forward is sometimes disabled (grey font) at early steps when a prior step is mandatory before the user can move to the next steps.

The purpose behind the division to steps is to reduce the short-term memory load for the user.

All of the buttons, tables, and text is color coded in our GUI for better mapping the controls to the actions.

“Login page”: the same login page is used for both students and faculty members, the system will refer the user to the correct page after the login.



Figure 8 Login Page

A student is directed to the system’s main functionality, time table creation. Allowing him to pick the courses and clusters, and to create his own course group that only one of them will be picked in the result timetable (like clusters).

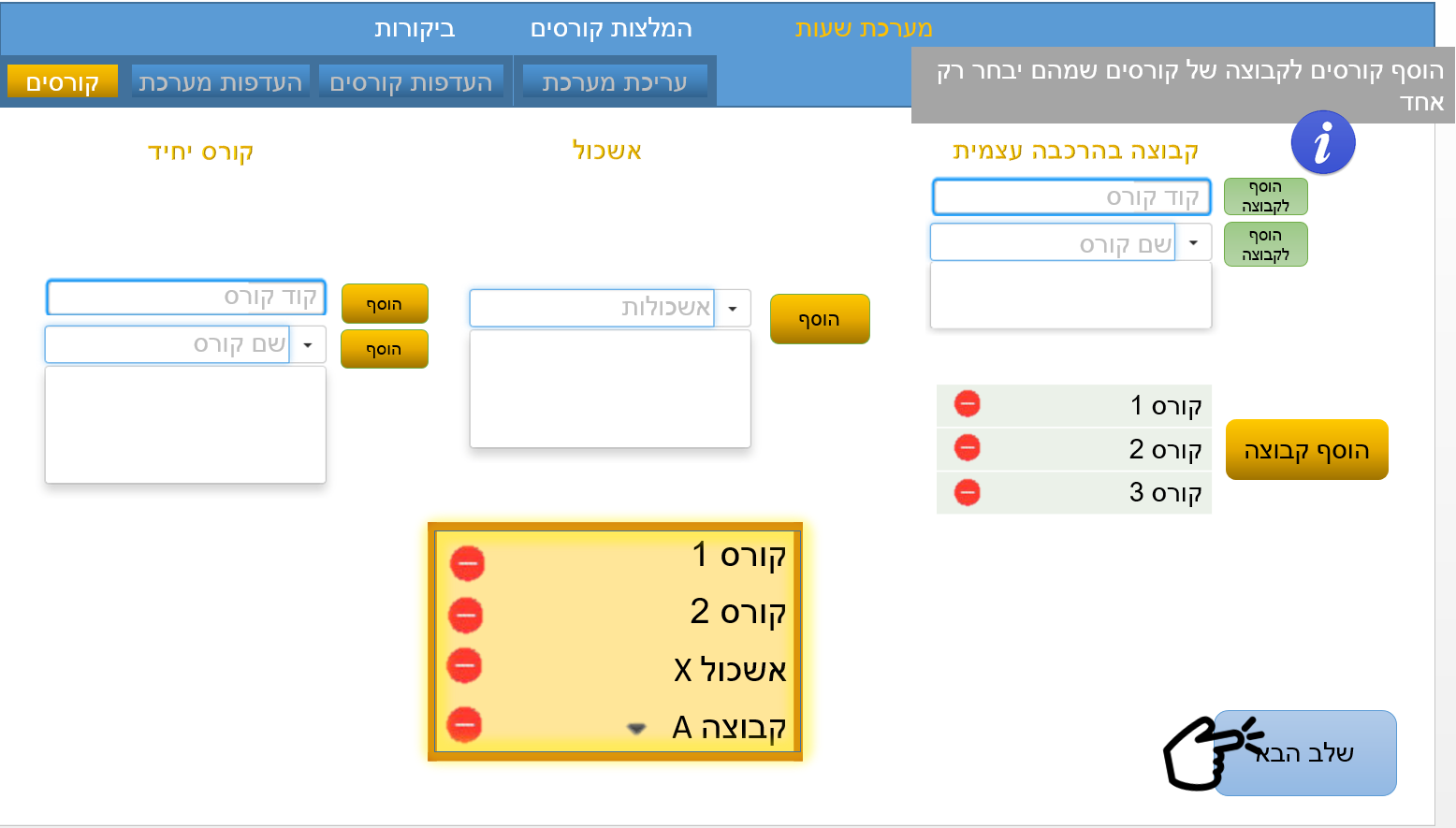


Figure 9 Timetable generation: step one

Next step is optional, the student can select empty periods for his preferences, he can select entire day by clicking on the day.



Figure 10 Timetable generation: step two

When picking course preferences, the student first pick a course, then a lesson type, and then the lesson specific preferences. Before picking a course the other preferences will be disabled (grayed).

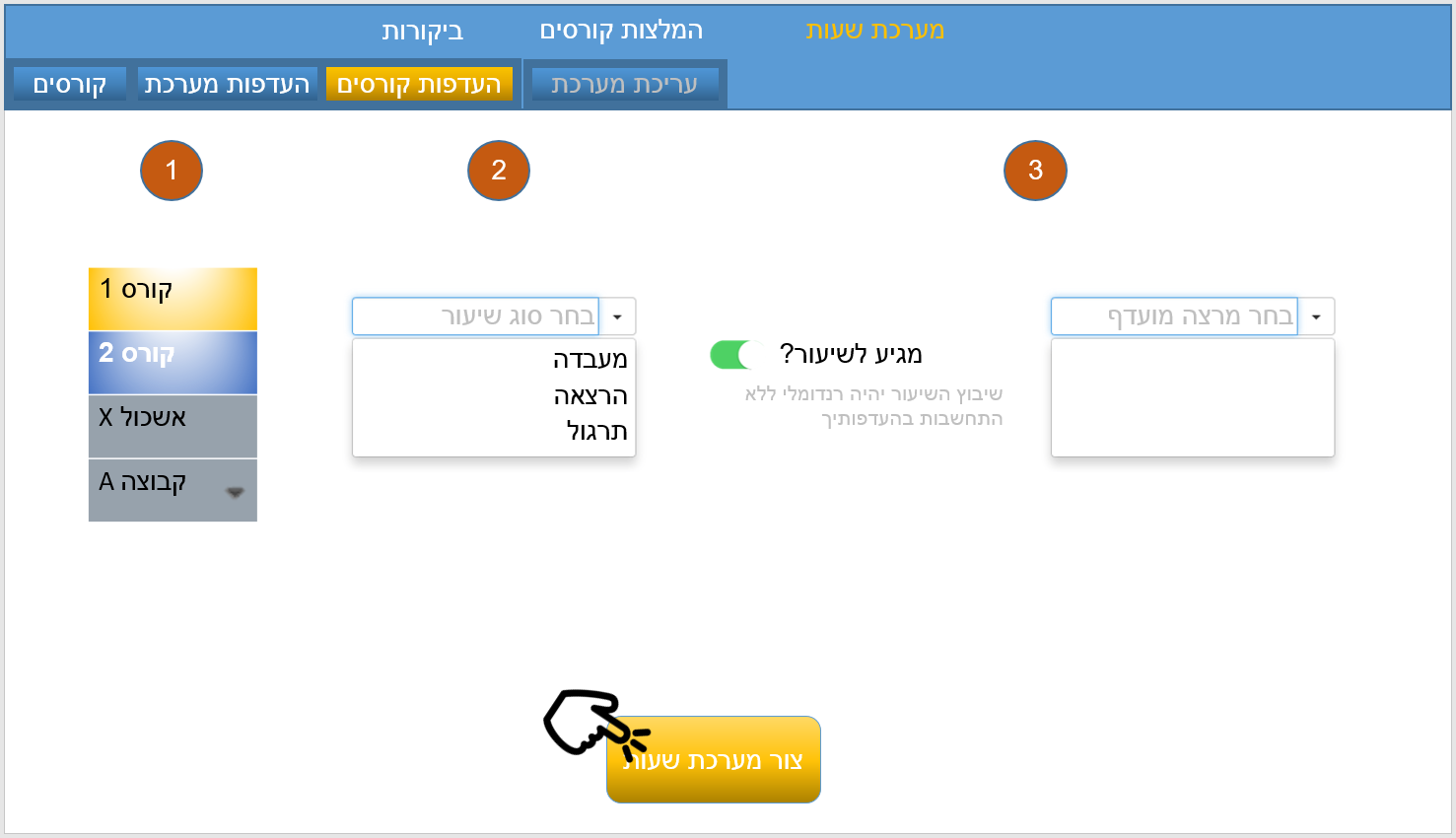


Figure 11 Timetable generation: step three

After the timetable is generated based on the student’s preferences, he can edit it by clicking on a lesson, all the available options for this lesson will be displayed on the timetable, clicking on one of the proposed options will pick it, and the other options disappear. The undo button is for undoing changes in the timetable and is disabled until the first change is made. Removing a course will delete it from the timetable. Adding a course (done by entering the course id) will start a set of steps, where the student will pick his choice for each lesson type in the same method as editing a lesson. Export will export the timetable to an Excel file.

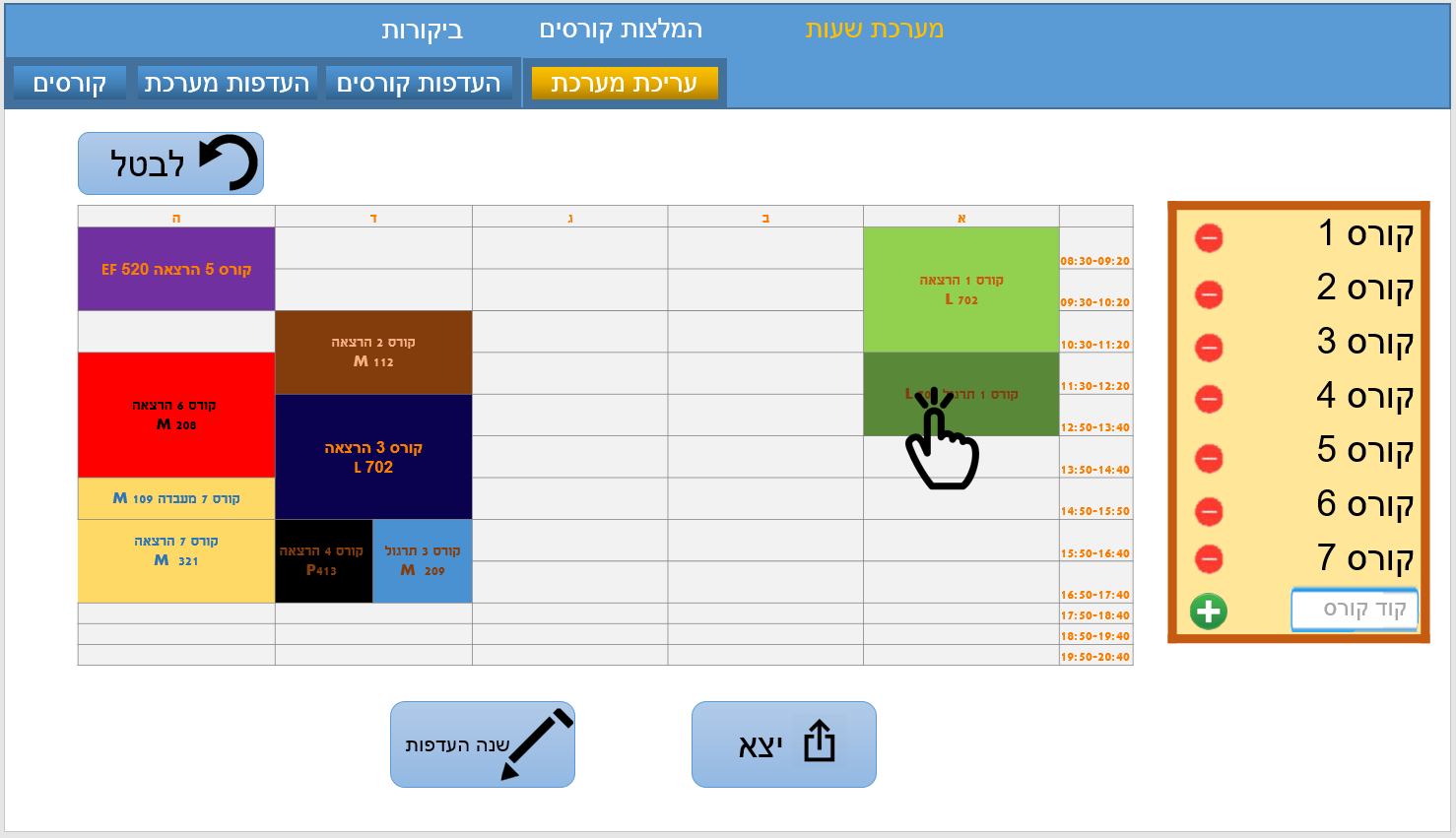


Figure 12 Timetable generation: step four

Course recommendation is another functionality for the student and shown on the student’s toolbar. The first step is preferences, where the student choose what courses he wants and doesn’t want to be in the result. The student has to choose the difficulty level of the proposed suggested courses, and total amount of credits which is optional.

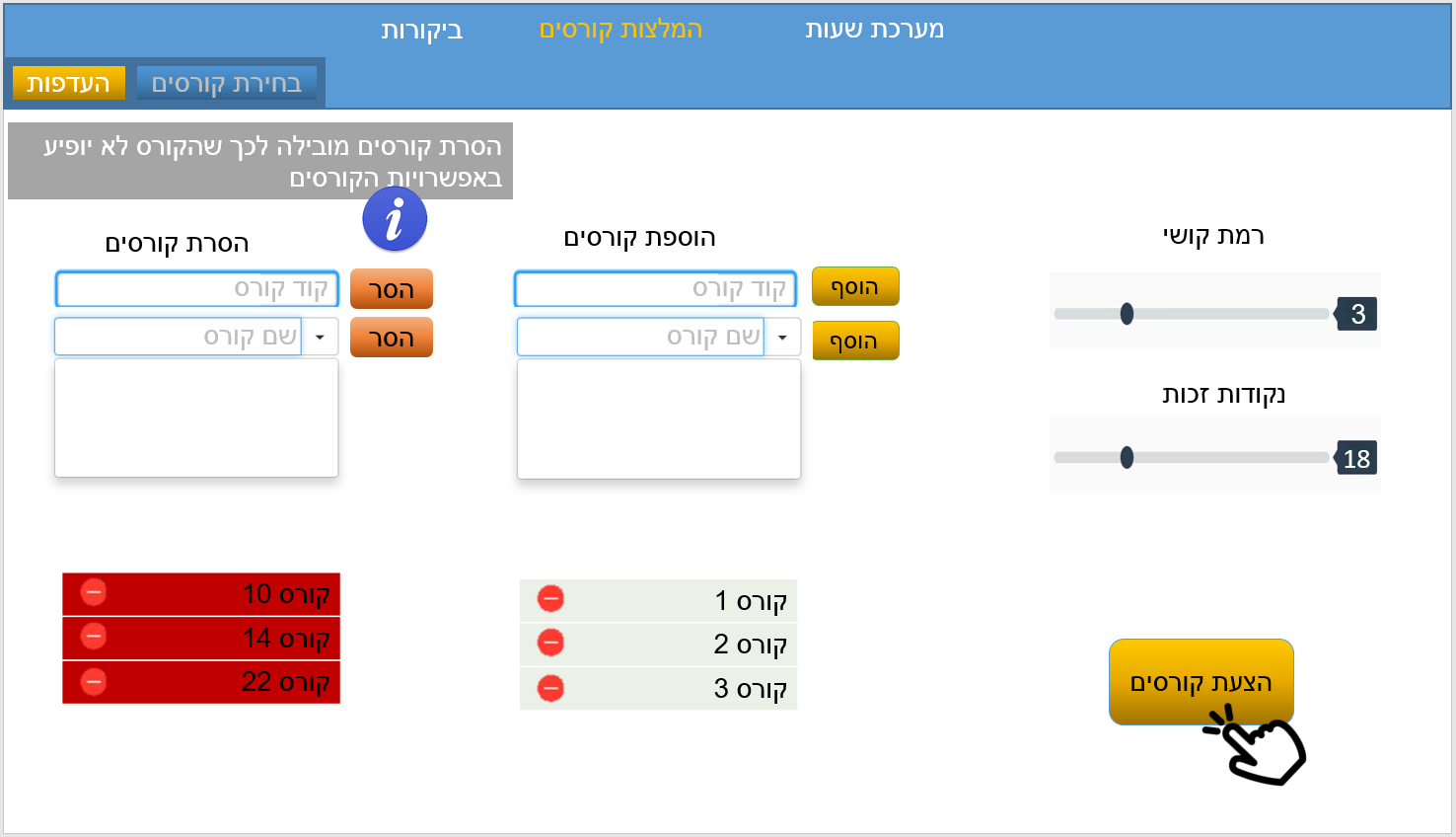


Figure 13 Courses recommendation: step one

The results are three options. The student can change his preferences by going back a step or clicking on the edit preferences button (same functionality). Another option is to generate a timetable that will direct the student to the preferences step in the timetable generator functionality, with the courses in the set he selected.

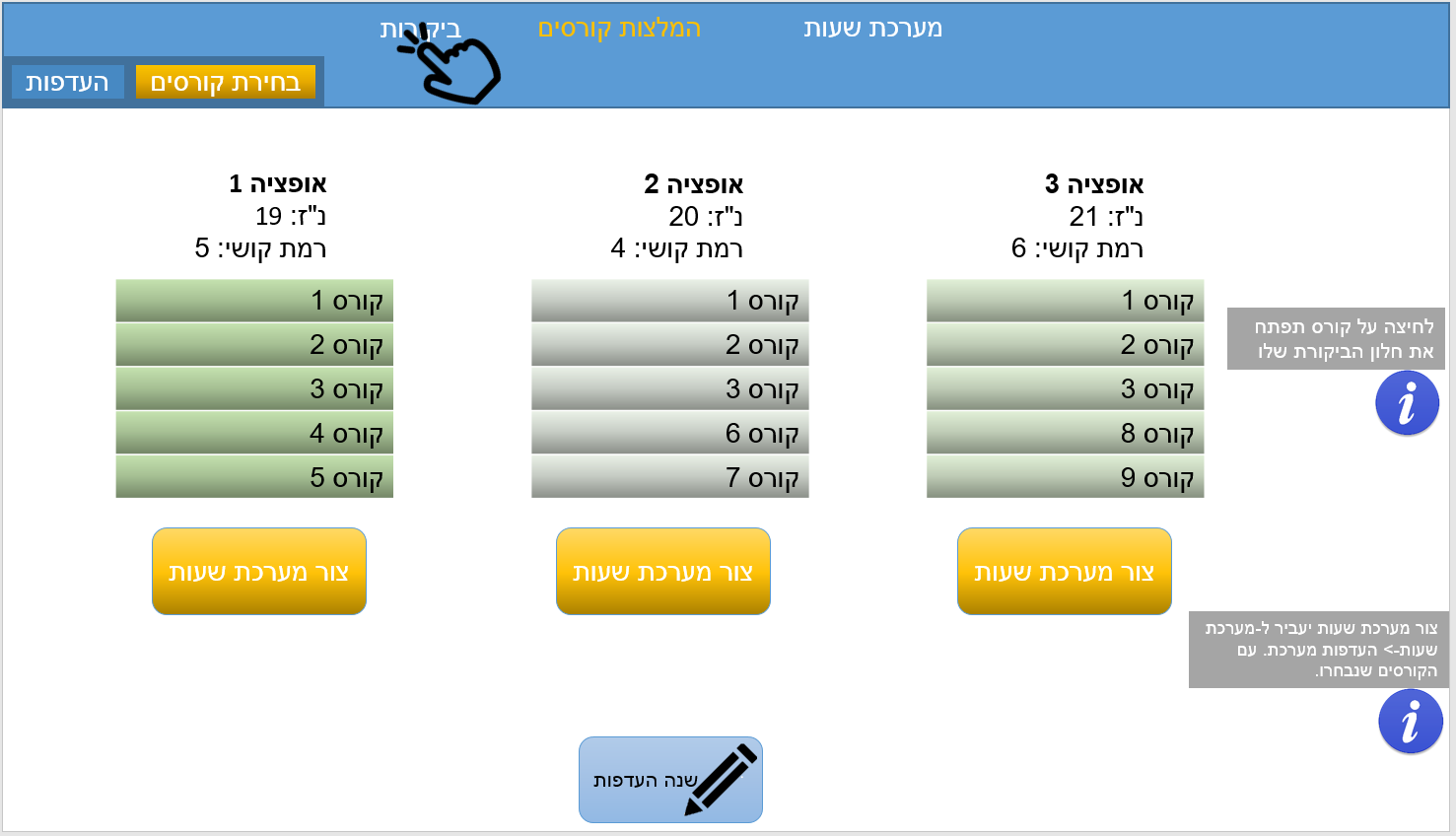


Figure 14 Courses recommendation: step two

The review functionality is on the student’s toolbar. It functions as both submitting and browsing reviews of courses and lecturers. The first step is to a pick course or a lecturer.

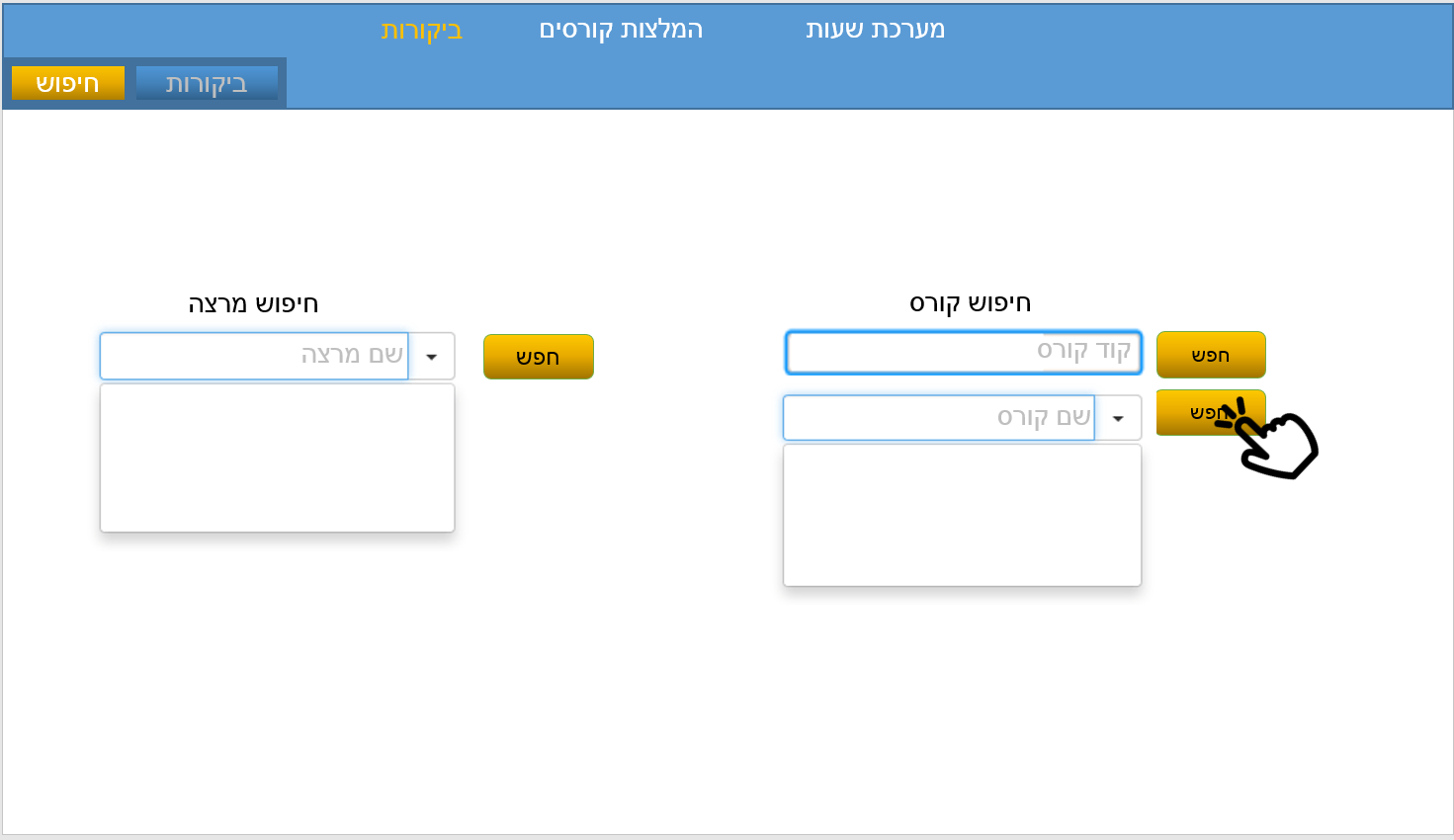


Figure 15 Review: step one

Next the overall score will be displayed next to the latest reviews, with the option to load more. Clicking on submit a review will pop up a review writing dialog where all the review parameters must be filled. (Binary option will have check box, sliders for ratings, and text boxes for free text)

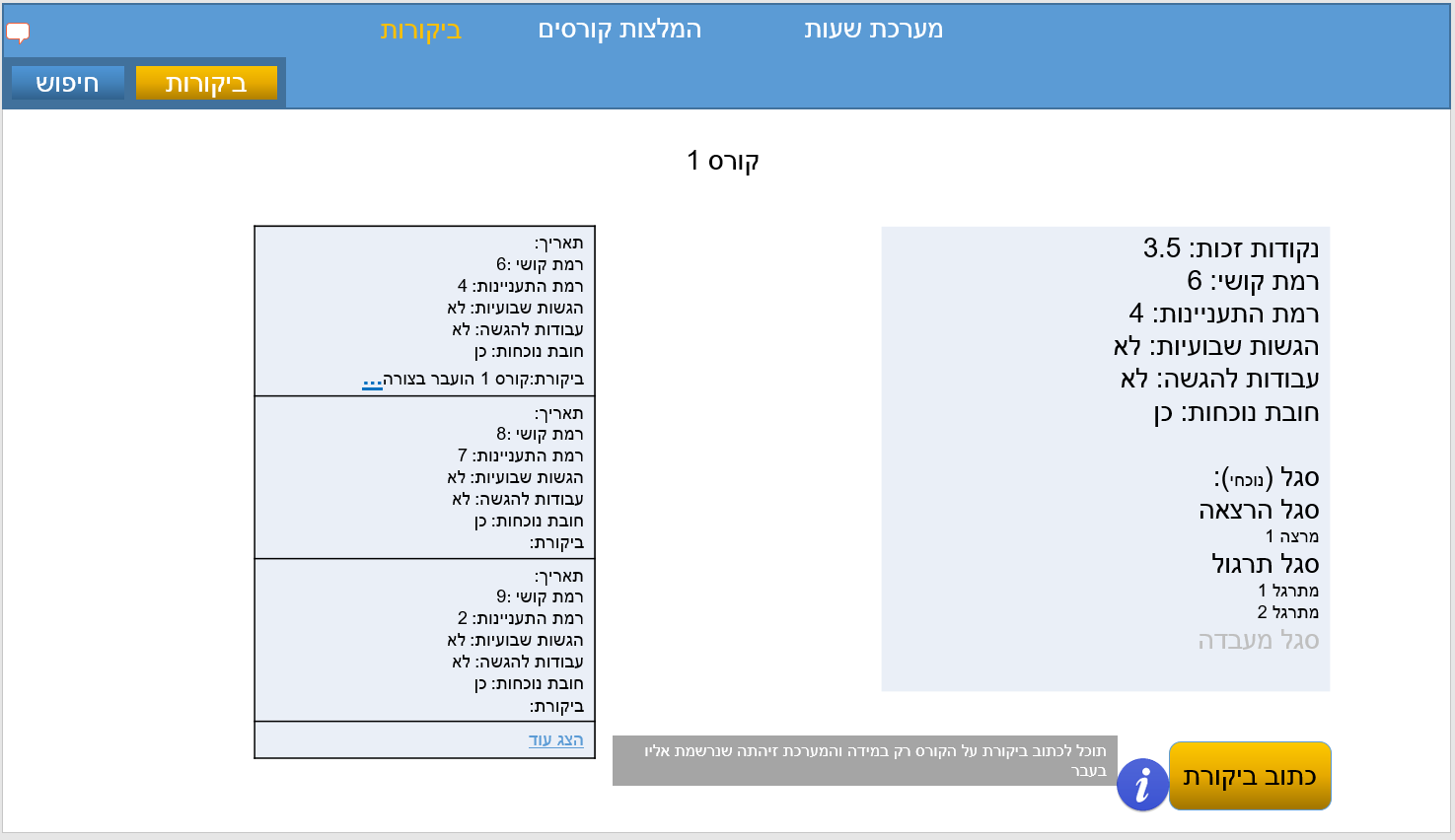


Figure 16 Review: step two

Faculty members will be directed to the courses enrollment functionality, where the first step is to pick completed courses, courses that was not completed and faculty of the students to show from.

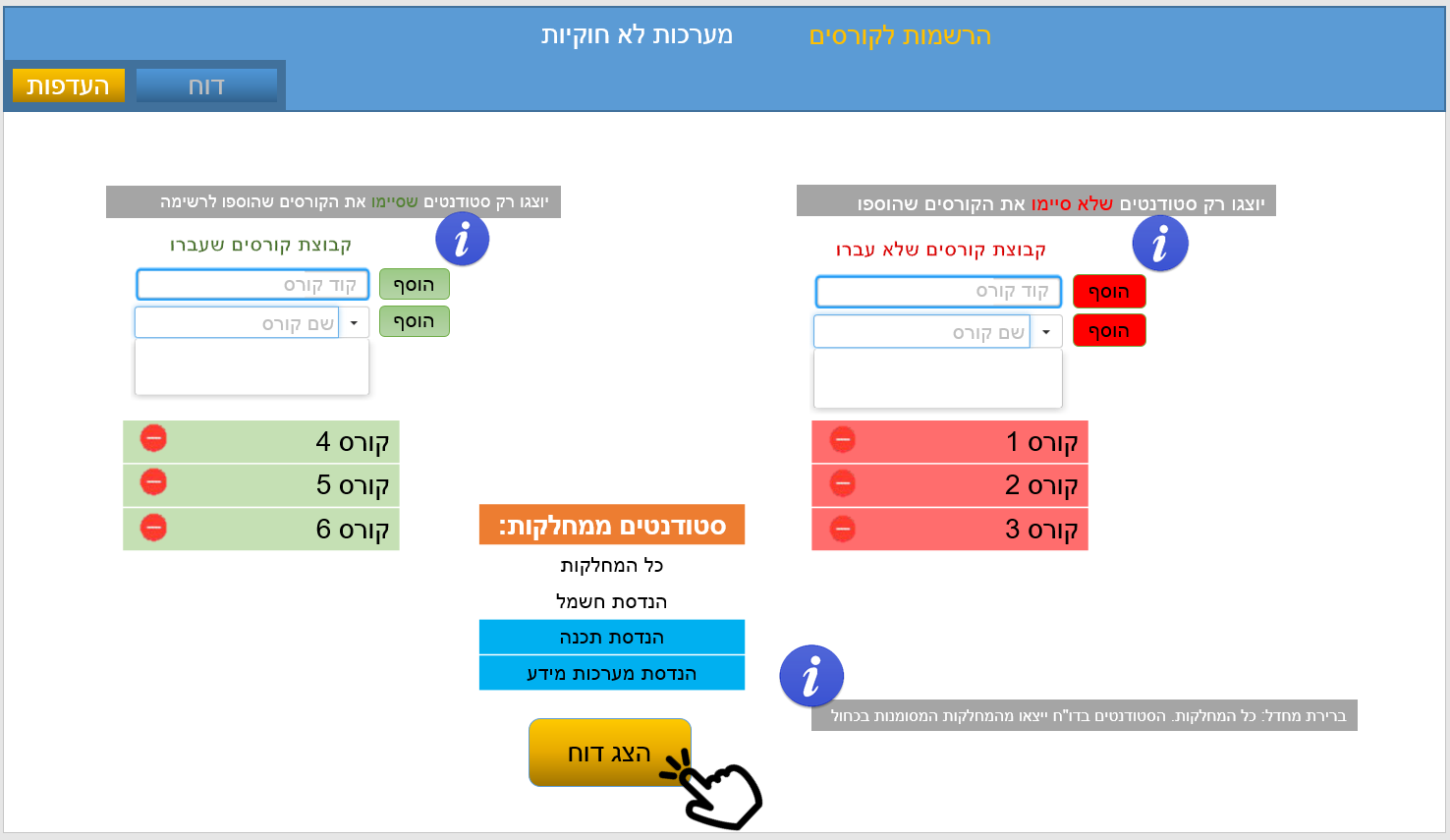


Figure 17 Courses enrollments: step one

This report will show the list of students from the faculties selected and from the input of completed courses and not completed courses, also it will show the courses that were picked. The report can be exported to an Excel file.

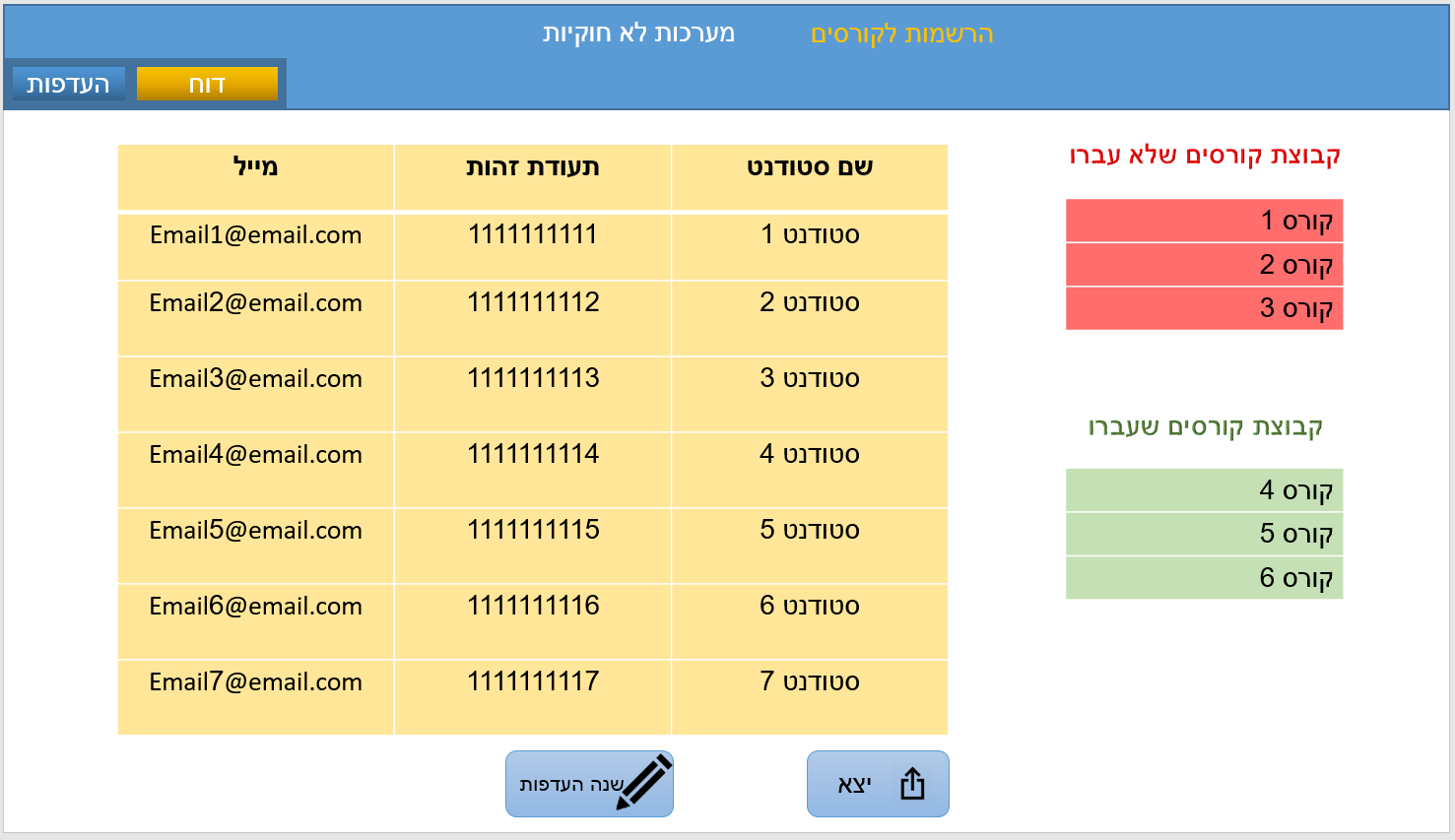


Figure 18 Courses enrollments: step two

The illegal timetables report work similarly, allowing the faculty staff to select between different faculties, and presenting the report with all the current students that have illegal timetables.

**TESTING PLAN**

**General user**

* **Login procedure**

|  |  |  |
| --- | --- | --- |
| **Test #** | **Test description** | **Expected results** |
| 1 | User leaves the “username” text field blank and press login | “Login” button is not clickable in that state, Prompt message appears that tells the user to fill the username text field |
| 2 | User leaves the “password” text field blank and press login | “Login” button is not clickable in that state, Prompt message appears that tells the user to fill the password text field |
| 3 | User fills username and password fields and press “Login” button | If user’s details matches database’s, then move user to the next window depends on that user-type. Else: prompt message appears that warns his user that his username or password is incorrect |

**Student**

* **Timetable generator**

|  |  |  |
| --- | --- | --- |
| **Test #** | **Test description** | **Expected results** |
| 1 | Student clicks on “Add” button but did not select any course from the combo box | A prompt message appears that tells the user to select a course before he can add it to the list |
| 2 | Student tries to type a non-digit character on the “Course ID” text field | The Gui allows only digits on that field, therefor non-digits character will be blocked and a popup will appear |
| 3 | Student tries to continue to step 2 without adding any course in step 1 | Button for next step is not clickable and greyed out until the user adds at least one course to the list |
| 4 | Student adds a course from “single courses” | The chosen course is added to the list of chosen courses |
| 5 | Student adds section from the “section courses” | Section course is added to the list of chosen courses |
| 6 | Student add multiple courses from the “create a set of courses” | The set of courses is added to the chosen courses list |
| 7 | Student presses on the button that removes a course from the set of courses | The chosen course is removed from the list |
| 8 | Student pressed on the button that removes a course from the list of chosen courses | The course is removed from the list of chosen courses |
| 9 | Student in step 1, presses on next step button after some courses were chosen | The student is moved to step 2 |
| 10 | The student selects a period in step 2 | The selected period turns to red |
| 11 | The student presses on next step button in step 2 | The student is moved to step 3 |
| 12 | The student selects a course from the list in step 3 | The student can now see the combo box to choose lesson type. The slider and proffered lecturer is greyed out until the student chooses a specific lesson |
| 13 | The student selects a proffered lecturer for a lesson in the chosen course | The chosen lecturer is coloured in the combobox |
| 14 | The student in step 3, presses on next step button | The student is moved to step 4 |
| 15 | The student presses on a lesson in the edit timetable screen | All the available periods for that lesson are shown in the timetable |
| 16 | The student presses on one of the available periods from the selected lesson | The original period is now removed and the new lesson period is on the timetable |
| 17 | The student types a “course ID” and press add | The new course is now added on the list and all its lessons-lectures type are shown in the timetable |
| 18 | The student selects one of the possible lectures from the new course in the timetable | The lecture is added to the list and the possible practices lessons are now shown in the timetable |

* **Course suggestions**

|  |  |  |
| --- | --- | --- |
| **Test #** | **Test description** | **Expected results** |
| 1 | The student presses on “Courses suggestions” | The student is moved to the “courses suggestion” window |
| 2 | Student clicks on “suggest courses” | Prompt message appears, it tell the student that he has to choose the difficulty level before proceeding |
| 3 | Student adjust “difficulty level” slider | The slider is adjusted and the number is changed according to the slider position |
| 4 | Student adjust “credits amount” slider | The slider is adjusted and the number is changed according to the slider position |
| 5 | The student adds a course to the suggestion list | The course is added to the suggested list |
| 6 | Student adds a course to the non-suggested list | The course is added to the non-suggested list |
| 7 | The student clicks on “suggest courses” | Student is moved to the proposed solution window considering his Preferences and other information taken from the database |
| 8 | The student clicks on “create timetable” on one of the proposed solution | The student is moved to the “timetable Preferences” window with the selected solution’s courses |

* **Reviews**

|  |  |  |
| --- | --- | --- |
| **Test #** | **Test description** | **Expected results** |
| 1 | The student clicks on “Reviews” button from the toolbar | The student is moved to the “Reviews” window |
| 2 | Student type a course ID and press on “search” button | The student is moved to the window with reviews on the selected course |
| 3 | In course review window, the student press on “view more” | Another batch of reviews for the course appear on the window |
| 4 | In course review, the student press on “…” | This button expands and show all of the content of the free text field from other reviewers |
| 5 | The student clicks on “add review” | “Add review” button is greyed out if student did not attend to that course in the last year. else: the student is moved to the course review form |
| 6 | The student clicks on “search” from the steps toolbar | The student is moved back to the “search review” window |
| 7 | Student types a lecturer name and search for it | If lecturer name fit a possible lecturer from the database, the student is moved to that lecturer review |
| 8 | The students clicks on “add review” | “Add review” button is greyed out if student did not attend any of the lessons this lecturer teach this year. Else: the student is moved to the “review lecturer form” |

**Faculty-Staff**

* **Reports**

|  |  |  |
| --- | --- | --- |
| **Test #** | **Test description** | **Expected results** |
| 1 | Staff member logins… | Staff member is moved to the “registered students for courses” report window |
| 2 | Staff member adds a course to the list of “only students who completed this courses” | Selected course is added to the green list |
| 3 | Staff member adds a course to the list of “only students who didn’t complete this courses” | Selected course is added to the red list |
| 4 | Staff member selects faculties for the group of students to show in the report | The selected faculties are marked in blue,  If the user didn’t select any faculty, then default is show students from all faculties. |
| 5 | Staff member presses on “show report” button | Button is greyed out and not clickable if the user did not add a course to any of the lists. Else: report is generated from the user input and the user is moved to the result window |
| 6 | Staff member presses on export in the results window | The results are exported to an excel type file |
| 7 | Staff member clicks on “illegal timetables review” on the toolbar | Staff member is moved to that review form window |
| 8 | Staff member clicks on “show report” button | The staff member is moved to the results window with the faculties selected |

**REFERENCES**

[1] A.H. Absa and Dr. S.W. Al-Sayegh, “*E-learning Timetable Generator Using Genetic Algorithms”,* University of Palestine.

[2] A. Colorni, M. Dorigo and V. Maniezzo, “*A GENETIC ALGORITHM TO SOLVE THE TIMETABLE PROBLEM”*.

[3] J.J. Moreira, “*A System of Automatic Construction of Exam Timetable Using Genetic Algorithms”*.

[4] O. Yeniay, *“PENALTY FUNCTION METHODS FOR CONSTRAINED        OPTIMIZATION WITH GENETIC ALGORITHMS”,* Department of Statistics, Beytepe, Ankara

[5] B.G.W. Craenen, A.E Eiben, E. Marchiori, “*How to Handle Constraints with Evolutionary Algorithm*”

[6] D.E. Goldberg and K. Deb, *“A Comparative Analysis of Selection Schemes Used in Genetic Algorithms”,* Department of General Engineering, University of Illinois at Urbana-Champaign.

[7] M. Srinivas, and L. M. Patnaik, Fellow, ZEEE, *“Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms”*, IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, VOL. 24, NO. 4, APRIL 1994

[8] F. HERRERA, M. LOZANO & J.L. VERDEGAY, “*Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis,* Department of Computer Science and Artificial Intelligence University of Granada, Avda. Andalucia 38, 18071 Granada, Spain