Algorithm

The problem of course scheduling is known to be NP-hard. The course scheduling problem can be approached by using search algorithms. As a matter of fact this works only for simple cases. Course scheduling, especially at universites and similiar big facilities, is faced with complex constraints. With the rise of input and more constraints finding an optimal solution cannot be computed in a sufficiently fast time. Based on several papers the approach of using a genetic algorithm is famous, seemed to be promising and was chosen therefore.
Genetic algorithm is a subset of metaheuristic optimization algorithm. Metaheuristics, being part of stochastic optimization, are algorithms using to some degree randomness in order to find optimal or as optimal as possible solutions to hard problems. Metaheuristics are applied when there is little to know about how the optimal solution looks like, there are too little heuristics to search on and brute-force is not questionable as the space of possible solution is too big. However when a candidate solution is given it can be scored in order to evaluate how good it is.
Typically a population of λ number candidate solution is generated randomly, named *setup*. The next step is an iteration containing the following process:

1. scoring the candidate solution by means of their *fitness*
2. keeping only the μ best candidate solutions
3. generating new candidate solutions
    a) selecting a candidate solution
    b) *cross* it *over* with another candidate solution
    c) *mutate* the candidate solution

To understand the the scheduler algorithm the genetic algorithm operations, *setup*, *fitness function*, *crossover*, *mutate* and *selection* are best understood in means of core components of the scheduler algorithm. These operations are described below.

Setup generates the initial population of candidate solutions in a random or semi-random way. A candidate solution is created when every course is allocated in a room at a given time. The *setup* process is finished when λ candidate solutions were created.

Fitness function evaluates the candidate solution by scoring it. The score is mapped to the number of constraints satisfied. The more constraints are satisfied the higher the candidate solution is scored. The score reaches from 0.0 to 1.0.

Crossover creates a new candidate solution by mixing and matching parts of two given candidate solution. How the mixing and matching is done relies on the representation of a candidate solution. As our representation is a mapping from courses to allocated room and time, these allocations are mixed and matched.

Mutation creates a new candidate solution by taking a given candidate solution and changing a specified amount of course allocations to new, randomly chosen, course allocations.

Selection iterates the given candidate solutions and keeps only the μ best solutions. The solutions are selected, according to the score given by the *fitness function*, through dropping the rest of the candidate solutions.

The problem of course scheduling is defined by the given constraints which have to be satisfied. In order to meet the MyCourses specification requirements of high configurability we distinguish constraints by *hard* and *soft constraints*.

Hard constraints are constraints which have to be satisfied. If these constraints are not satisfied the lecturing of courses is not possible. This includes the following constraints:

- not more than 1 course in the same room at the same time
- the course lecturer has no other course at the same time
- courses belonging to the same year do not overlap with each other in time in order to guarantee studiability
- every constraint defined by the user with the priority of 100%, for instance preferred time, preferred room, etc.

Soft constraints are constraints which do not have to be ultimately satisfied. This includes only constraints defined by the user with a priority less than 100%.

An *optimal solution* is a scheduling which asserts the satisfaction of all *hard* and *soft* constraints. Respectively the schedules score for *hard* and *soft fitness* is both 1.0.
In the implementation phase it turned out using a *setup* allocating the course completely random works for little input very well. With increased input, however, the results by the classical *setup* lead to quiet worse results. These results are supposed to be optimized by the phase of applying *crossover* and *mutate*. As a matter of fact this happens to a certain degree but starts to converging to a certain score.
Therefore an alternative *setup* was implemented. *Greedy setup* combines the approach of a genetic algorithm using metaheuristic with the Greedy algorithm being a subset of the combinatorial algorithm.

Greedy Setup generates the initial population of candidate solutions by using a Greedy algorithm. Every course is allocated by placing it to the room which fits its constraints the best, for instance the requirement for a specified amount of seats. In order to avoid choosing rooms which exceed the required constraints every room is scored and sorted. In order to avoid overlapping in space and time the courses are place one after another in the timetable.

Figure 1 illustrates the components interaction. At the start of each scheduling a initial population containing λ possible schedule solutions is generated by *Greedy Setup*. Randomly chosen courses are placed into the room which fits the course constraints the best.

In this step the time slots are chosen at the earliest possible time slot and are distributed along all days at the week. If, however, there is a constraint for a preferred room or a preferred time slot entered by the user this spot is chosen directly.

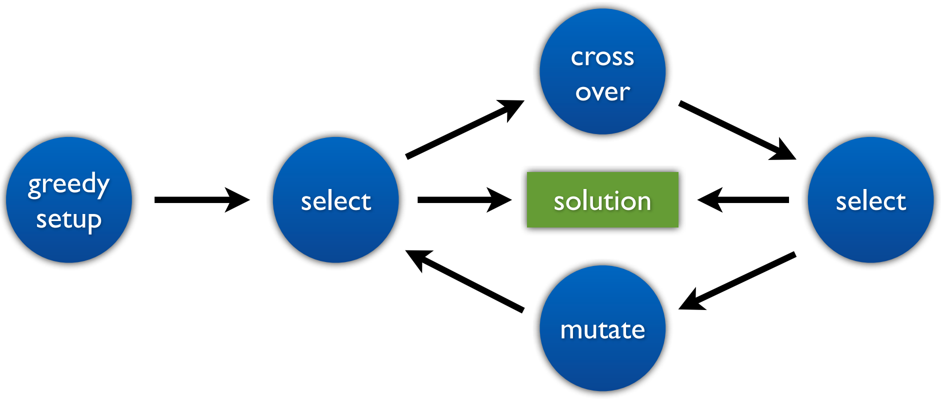

Figure 1: Routine of the scheduler algorithm. Greedy Setup generates a initial population of candidate solutions. Every candidate solution is scored. If non-solvable conflicts appear the scheduling has to be terminated by means of not being scheduleable. Otherwise the Setup phase is followed by optimizing the candidate solutions by applying crossover and mutation operation until an optimum schedule is found.

There is the possibility of finding an optimal solution in the phase of *Greedy setup*. Therefore the *fitness function* is applied on every generated candidate solution. If a optimal schedule was found the algorithm has terminated.
It is also possible the wished scheduling is not scheduleable at all. For instance when two course have the hard constraint to be placed at the same room with overlapping time. In this case the scheduling is stopped and the user has to resolve the constraint conflict by himself.
Otherwise the *Greedy setup* will complete and the iteration of applying *crossover*, *mutation* and *select* operation is started. After each *crossover* and *mutation* only the best μ candidate solutions are kept and are tested for reaching the score 1.0 which would lead to the algorithm termination.