University Course Scheduling System

Tsu-Hao Kuo and Yang Zeng, Carnegie Mellon University

Course Scheduling is an important issue for every university. How to efficiently allocate classrooms to each class, to prevent an instructor teach two courses at the same time, and to satisfy all instructors' and students' preference are always the challenging parts of this problem. In this paper, we proposed a course scheduling system to address this problem by formulating it as a Constraint Satisfaction Problem (CSP) and solved it by using the backtracking search. The system not only guarantees to match the capacity of a course and the assigned classroom and make the most efficient usage of classrooms, but also takes the instructors' preference of time and location and students' preference of location into account. We adopted the most constraint variable (MVC) heuristic to speed up the searching process and used a scoring system to improve the performance of searching process. Furthermore, we applied a genetic algorithm method to perform replanning when adding a course or changing the capacity of a course in order to keep the majority schedule unchanged while to make the schedule as satisfiable as possible. Simulation results show that the proposed system outperforms the system which does not use this heuristic in terms of the number of expanded node and the processing time.

Categories and Subject Descriptors: []: —

Additional Key Words and Phrases: Course Schedule, Constraint Satisfaction Problem, Genetic Algorithm

ACM Reference Format:

Tsu-Hao Kuo and Yang Zeng, 2013. University Course Scheduling System. *ACM Trans. Appl. Percept.* 2, 3, Article 1 (May 2010), 11 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Course scheduling problem is closely related to our life in school. It requires the assignment of courses into different time slots at different classrooms. The problem can be categorized as a classical scheduling problem, which is NP-complete. There are many classical algorithms solving this kind of problem. The most popular ones are genetic algorithm, linear programming, tabu search, and simulated annealing [Corne et al. 1994][Hertz 1992][de Werra 1985]. There are also researches using fancy algorithms such as artificial bee colony algorithm [Oner et al. 2011]. As we can treat the conditions for matching a course and a classroom as constraints, we can also formulate this as a constraint satisfaction problem (CSP) [Russel and Nrovig 2010]. In our approach, the formulation of the CSP problem is divided into two parts: (i) the hard constraints, which are the physical conditions we have to meet, and (ii) the soft constraints, which are the preferences of instructors and students. For the first part, we use the backtracking search to solve the basic scheduling problem, with the most constraint variable (MVC) heuristic and a scoring approach inspired from genetic algorithm. For the second part, we also incorporated genetic algorithm to solve the replanning part of the system.

The remainder of this paper is organized as follows. In section 2, we discussed the constraints and denoted all the variables. Section 3 describes the input and output of the problem. Section 4 describes the details of the formulation of the constraint satisfaction problem. The formulation of the genetic algorithm for replanning is given in Section 5. Finally, Section 6 shows the simulation result and gives the conclusion.

2. CONSTRAINTS FOR COURSE SCHEDULING

As mentioned in the previous section, the constraints for the course scheduling problem are categorized into two kinds: (i) hard constraints and (ii) soft constraints. We will give the definition of these two kinds of constraints separately in the following subsections.

2.1 Hard Constraints

These are the kind of constraints that defined by nature, which must be strictly followed. The hard constraints we have here are:

—No instructor can give two or more classes at the same time
—No two or more courses will take the same classroom in the same time
—The classroom should have a capacity larger than the class required

2.2 Soft Constraints

These are the constraints that defined by the instructors' preference and students' preference, mainly for the requirement on time and location assignment of a course. The soft constraints are listed below:

—The time preference of the instructor: when the instructor wants to lecture.
—The location preference of the instructor: in which building the course will be given.
—The location preference of the students: Students prefer to take classes in there own department building. This constraint dominates the location preference of the instructor.
—Classroom efficiency preference: larger classrooms should hold larger courses.
—Instructor tends not to teach classes consecutively: If the instructor is teaching more than one course, she/he tends to have a time gap in between.

We denote all these constraints from C_1 to C_8. For the time preference of the instructor, C_4, it is further divided into two smaller constraints: which day the instructor tends to give lectures (Monday, Tuesday, etc), we denote this constraint as C_{4a}, and what time period during a day the instructor would like to give a lecture (morning, afternoon or evening), which is denoted as C_{4b}.

In the next section, we describe the details of the formulation of course scheduling problem in this paper.

3. INPUT AND OUTPUT OF THE PROBLEM

In this section, we formulate the scheduling problem and define the input and output of the problem. The input data are three tables, one is the table of capacity of each classroom, another is the table of courses. The other one is the table of instructors. And the output is a scheduled time table. We first describe the details of the output data, and then explain the details of each input data in the following subsections.

3.1 Output: Solution of the Time Schedule

The output of this problem is a scheduled time table, which is a 2-D array. The dimensions of the schedule is the (# classrooms) $\times$ (# time slots). The elements of the matrix represent the ID number of each class, which is denoted as classID. The data structure of the 2-D matrix is shown in Figure 1.

3.2 Input 1: Classrooms

The classrooms array stores the capacity of each classroom. The dimension of it is (# classroom) $\times$ 1. The classroom IDs are arranged such that the classrooms in the same college are put together. The

	Timeslot1	Timeslot2	Timeslot3	Timeslot4	…
Classroom1	CourseID4	0			
Classroom2	CourseID3	CourseID5			
Classroom3	CourseID2	0			
….					

Fig. 1. Data Structure for Resulted 2D Schedule

distribution of the number of the classrooms in a college follows the distribution of college in CMU. There are three kinds of capacities for a classroom: 50, 100 and 150. This formulation reflects the capacities of classrooms in most universities. Figure 2 shows the data structure of the classroom array.

Classroom1	Capacity
Classroom2	….
Classroom3	….
….	….

Fig. 2. Data Structure for Classroom Capacity

3.3 Input 2: Courses

The courses matrix stores three kinds of information:

(1) how many students the course can take (capacity of the course)
(2) which instructor is teaching the course (instructor ID)
(3) what is the location preference of the students taking the course (given by a start classroom ID and an end classroom ID).

Figure 3 shows the data structure of the course matrix.

CoursesID1	Capacity	InstructorID1	LocStart1	LocEnd1
CoursesID2	Capacity	InstructorID2	LocStart2	LocEnd2
CoursesID3	Capacity	InstructorID3	LocStart3	LocEnd3
….	….	…..		

Fig. 3. Data Structure for Classroom Matrix

3.4 Input 3: Instructors

The instructors matrix stores the time and location preferences of the instructors. The time preference is composed of the weekday preference and timeslot preference during the day. The location preference is specified in the same way as the location preference of the course. A sample of the table of instructors is shown in Figure 4.

InstructorID1	WeekDay1	Time1	LocStart1	LocEnd1
InstructorID2	WeekDay2	Time2	LocStart2	LocEnd2
InstructorID3	WeekDay3	Time3	LocStart3	LocEnd3
….	….	….	….	….

Fig. 4. Data Structure for Instructors Matrix

3.5 Time Slots

As mentioned in the subsection 2.2, we divided the constraint of instructors' time preference into preference of weekday and preference of time period. We consider all possible time slots within a week and divide them into 15 parts, based on the usual time slots we have in every school. Usually, a course will be conducted twice during a week. So we set the week day time of time slot to be Monday+Wednesday, Tuesday+Thursday, and Friday. Note that courses conducted on Friday will take 2 time periods. For Monday to Thursday, we have 6 time periods each day, with 1 and 2 in the morning, 3-5 in the afternoon and 6 in the evening. For Friday, we only have 3 time periods, each corresponds to the morning, afternoon and evening periods. The distribution of the time periods are shown in Figure 5

Mon+Wed	Tue+Thur	Friday		
1	7	13	8:00-10:00	Morning
2	8		10:00-12:00	
3	9	14	12:00-14:00	Afternoon
4	10		14:00-16:00	
5	11		16:00-18:00	
6	12	15	18:00-20:00	Evening

Fig. 5. Data Structure for Time Slot

4. PROBLEM FORMULATION

Since the course scheduling problem is formulated as a matching problem in this project, the basic idea is to match a $< classroom, timeslot >$ pair to one of the courses, which not only satisfies all the hard constraints but also maximizes the fitness value of the soft constraints. The hard constraints are set to be the constraint of CSP since these constraints must be satisfied. As for the soft constrains, they are satisfied by using a scoring approach, which is explained in subsection 4.2. In addition to the back backtracking search, we use some heuristics here to speed up the searching process while trying to meet as many soft constraints as possible. The heuristics used here are:

4.1 Assigning the most constraint courses first

There are courses that have more requirements than other courses. For example, if a course has a larger capacity, it might have smaller number of rooms available for it. This is accurately working as a Most Constraint Variable (MCV) heuristic for choosing a variable to be assigned. Therefore, we processed the class which has a larger maximum enrollment first.

4.2 Assign the classroom that meets most soft constraints first

As soft constraints need not to be strictly followed, we can not formulate them as constraints for CSP. In order to take the soft constraints into consideration and try to maximize the fitness value of the soft constraints, we will treat them as weights when assigning values for them. The calculation of the weight is based on how well the soft constraints are satisfied by a classroom. Each of the soft constraints (C_3 to C_8) count as 20 points. We will add up the score and rank the available classrooms using the score. When assigned classrooms to a course, we will start with the one that has the highest score. Algorithm 1 shows the pseudo code for the backtracking algorithm.

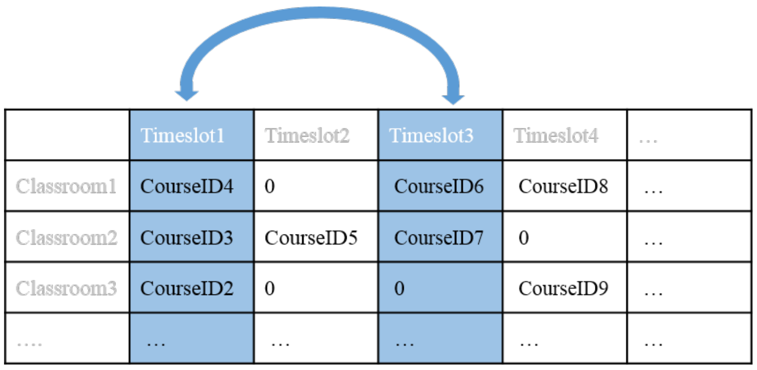
ALGORITHM 1: Constraint Satisfaction Back Tracking Algorithm

```
function BACKTRACKING (Assignment, Courses, Classrooms) returns a solution or failure
if Assignment is complete then
    return Assignment
end
var ← most constraint variable (the class has the largest capacity)
compute score based on soft constraints, and sort Classrooms based on their scores
for each classroom (from the highest score to lowest) do
    if $< ClassroomID, CourseID >$ consistent then
        insert $< ClassroomID, CourseID >$ to Assignment
        delete ClassroomID from Classrooms
        delete CourseID from Courses
        result← BACKTRACKING (Assignment, Courses, Classrooms)
        if result$\neq$ failure then
            return result
        end
        remove $< ClassroomID, CourseID >$ from Assignment
        recover Classrooms and Courses
    end
end
return failure
```

5. FORMULATION OF THE GENETIC ALGORITHM

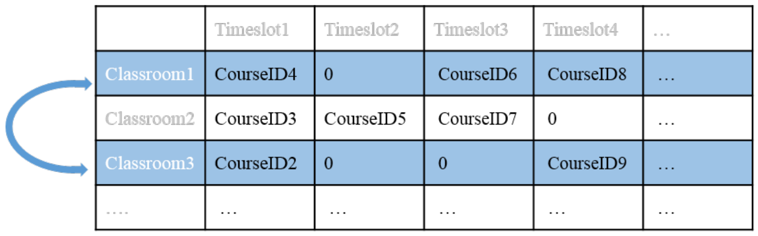
The genetic algorithm is used for solving the replanning part. When we add or change a course, we would like to keep all the other courses unchanged. If we use the backtracking search to run the scheduling again, it will change a lot of the assignments, because the new course will change the priority queue we use when choosing a variable. Instead, we can use genetic algorithm to solve the problem. Because genetic algorithm can change only one element pair at a time, we can keep most of the courses unchanged. To address the problem, we first try to insert the new element directly. We will first check the open spots within the preferred location of the course. If inserting in such an open spot gives a satisfactory fitness value, we call this a valid insertion and keep the schedule. If there are no open position that can give fitness value high enough, we would start with an open position in the desired locations and call genetic algorithm to generate a better schedule.

The encoding of the chromosome of the genetic algorithm uses a 2D structure, which is shown in Figure 1. This configuration is naturally induced from the configuration of the time schedule [Ellingsen and Penaloza 2003]. In order to make sure that each courseID will only appear once in the schedule, we will do the mutation and crossover within the schedule. So the population size is kept as 1, which is a whole schedule. The crossover will be performed within two rows or columns. Figure 6 and Figure 7 show the two crossover methods. The mutation action is taken with two elements in the schedule.



	Timeslot1	Timeslot2	Timeslot3	Timeslot4	…
Classroom1	CourseID4	0	CourseID6	CourseID8	…
Classroom2	CourseID3	CourseID5	CourseID7	0	…
Classroom3	CourseID2	0	0	CourseID9	…
….	…	…	…	…	…

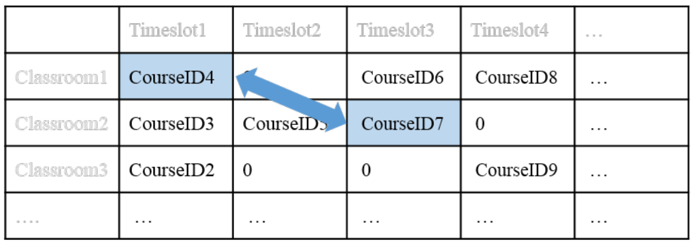
Fig. 6. Crossover within Time Slots



	Timeslot1	Timeslot2	Timeslot3	Timeslot4	…
Classroom1	CourseID4	0	CourseID6	CourseID8	…
Classroom2	CourseID3	CourseID5	CourseID7	0	…
Classroom3	CourseID2	0	0	CourseID9	…
….	…	…	…	…	…

Fig. 7. Crossover within Rooms

Figure 8 shows how the action is performed.



	Timeslot1	Timeslot2	Timeslot3	Timeslot4	…
Classroom1	CourseID4		CourseID6	CourseID8	…
Classroom2	CourseID3	CourseID5	CourseID7	0	…
Classroom3	CourseID2	0	0	CourseID9	…
….	…	…	…	…	…

Fig. 8. Mutation within Elements

The pseudo code of the replanning part for inserting a new course is shown in Algorithm 2. The algorithm for changing a requirement for a course is similar.

ALGORITHM 2: Replanning for adding a new course

```
function Replanning (OldAssignment, NewCourse) returns a new schedule
Calculate the fitness1 of OldAssignment.
for all the spare $< ClassroomID, CourseID >$ pairs within the location preference of the new course do
    Insert the new course
    if fitness value large enough then
        NewAssignment = DirectInsert (OldAssignment, NewCourse,$< ClassroomID, CourseID >$)
        return
    end
end
$fitness1 \leftarrow fitness(OldAssignment)$
$NewAssignment \leftarrow OldAssignment$
repeat
    $fitness2 \leftarrow fitness(NewAssignment)$
    if fitness2 large enough then
        return
    end
    $OffSpring \leftarrow GenerateOffSpring(OldAssignment, NewCourse)$
    $fitness2 \leftarrow fitness(OffSpring)$
    if $fitness2 > fitness1$ then
        $fitness1 = fitness2$
        $OldAssignment \leftarrow OffSpring$
    end
until fitness2 large enough and number of iterations exceeded;
```

6. RESULTS AND DISCUSSION

In this section, we show the performance of the proposed system for solving the course scheduling problem. We discuss the influence of different parameters by tweaking one of the parameters in different test instances. Furthermore, we compare the efficiency of the algorithm with heuristics to the one without any heuristic.

6.1 Test on CSP

We tested our CSP algorithm using difference test instances. We tweak the parameter of the test instances to change the difficulty of the problem and also discuss how these parameter affect the performance of the algorithm. The following sections show how the fitness score (100 for full credit) change with these parameters. The fitness score is calculated as follows:

$$Fitness = \sum_{c_1}^{c_5} \frac{\text{Number of Courses with } c_i \text{ Met}}{\text{Number of Courses}} \tag{1}$$

We use a method of controlling variable, which means changing one parameter at a time. The parameters we have here are:

(1) number of classrooms

(2) number courses/(number of classrooms $\times$time slots) (filling percentage)

(3) percentage of instructors who have time preference (preference percentage)

Unless specified, the values we use for these parameters are: 200 classrooms, 80% filling percentage, 70% instructor percentage, 15% preference percentage. In the following subsections, we discuss the influence of different parameter.

6.1.1 *Number of Classrooms.* Figure 9 shows the result we get when changing the number of classrooms.

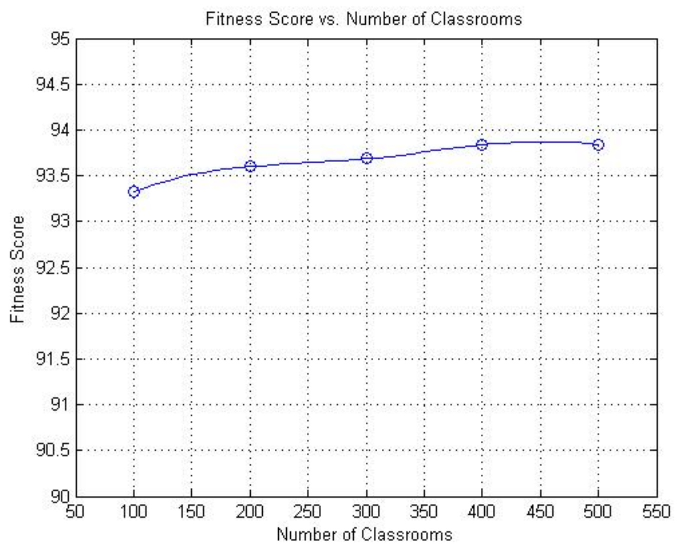


Fig. 9. Fitness Score vs. Number of Classrooms

We can see that the fitness score are around 95, which means that 95% of the requirements are met by our algorithm. It is interesting to note that although increasing the number of classrooms increase the difficulty in searching for a solution, it actually help to increase the fitness value of the whole schedule. This is because that, if the number of classroom is increased, the number of classroom in a certain college will also increase. This will give us more flexibility when trying to arrange the course with the location preference in mind. The number of classrooms we use here covers most of the university scales (CMU is kind of on the 100 classroom scale. A university with 500 classrooms can hold about 400,000 students!)

6.1.2 *Filling Percentage.* Figure 10 shows the result we get when changing how full the schedule is (filling percentage). We can see that the filling percentage have a huge impact on the performance of the algorithm ($92\% \sim 94\%$). This is because that, when the filling percentage increases, the number of spare spots in the schedule decreases, the flexibility of finding a good fit for a course is lowered down. However, the algorithm still gives good enough result (92%) even when we push the filling percentage to 100%.

6.1.3 *Instructor Percentage.* Figure 11 shows the result we get when changing how many instructors are teaching two courses (instructor percentage). We can see that when increasing the number of instructors, the fitness value will increase significantly. This is because that, when the number of instructors is increased, the number of instructors that have to teach two courses will decrease. This will relax the hard constraint that an instructor can not teach two courses at the same time, thus adding flexibility on choosing a time slot. We can see that, when all the instructors teach only one course, the fitness value can reach as high as 99.72.

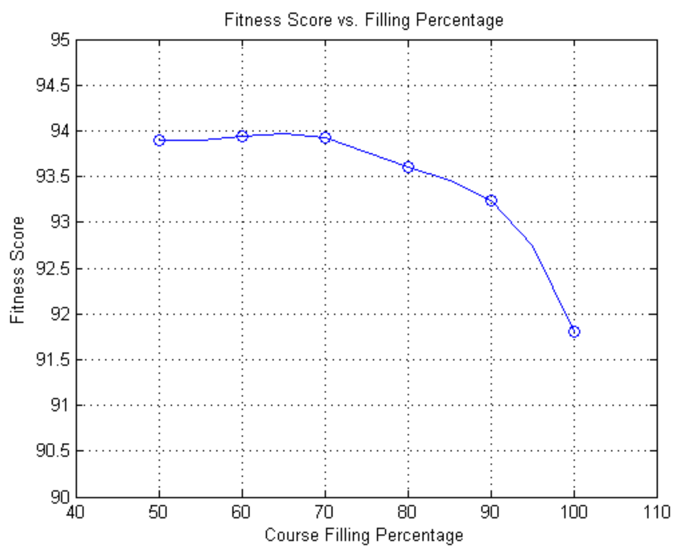


Fig. 10. Fitness Score vs. Filling Percentage.

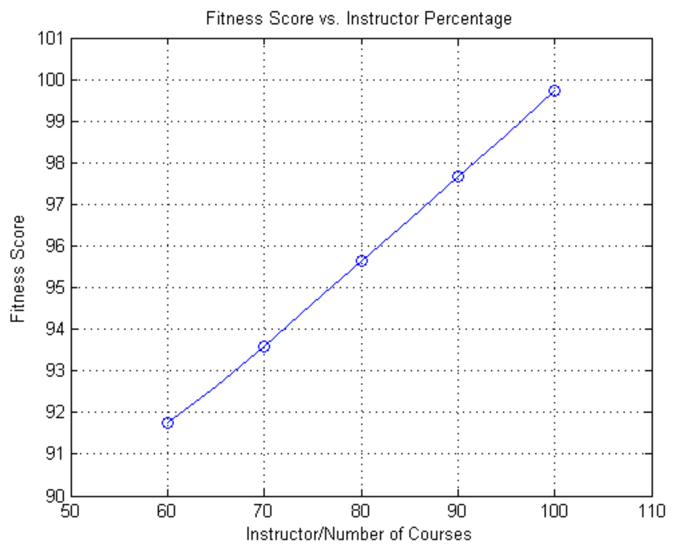


Fig. 11. Fitness Score vs. Instructor Percentage.

6.1.4 *Preference Percentage.* Figure 12 shows the result we get when changing how many instructors have time preferences (preference percentage).

We can see that when more instructors have time preferences, the fitness value tend to go down. This is straightforward since we need to satisfy the need of more people while holding the same amount of resource, it's getting harder and harder when the requirement increase. Despite this fact, our algorithm can still reach to 92 even when 90% of the instructors have time preferences.

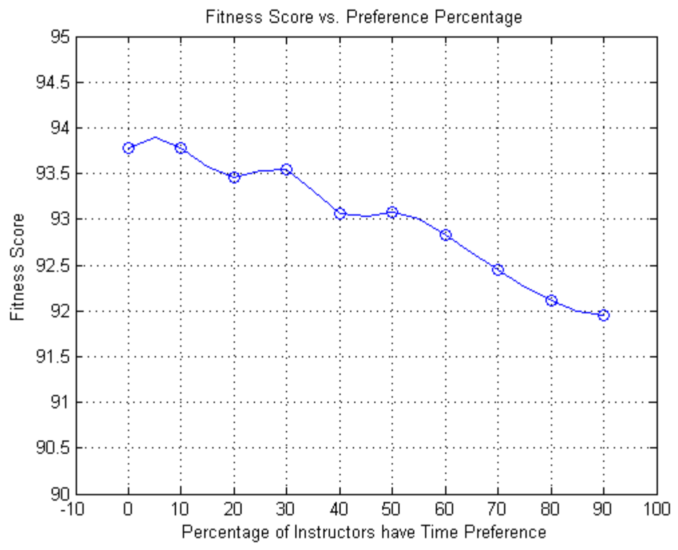


Fig. 12. Fitness Score vs. Preference Percentage.

6.2 Efficiency of using the Heuristic

In addition to show the capability of solving the course scheduling problem for the proposed system, we also compare the efficiency of using heuristic by comparing the proposed system to the system that does not use the most constraint variable constraint and the scoring approach. Since this system does not assign the most constraint variable first, it is more likely that this system has to perform more backtracking during its assignment process.

To show the efficiency of using the heuristic, we compare the number of expanded node when performing backtracking and the processing time between the system with heuristic, i.e., the proposed system and the one without any heuristic. The results are shown in the table below:

Table I. Comparison of the performance of the efficiency of using heuristic

	With Heuristic		Without Heuristic	
# Classes	# expanded node	Processing time (sec)	# expanded node	Processing time (sec)
15	15	0.000051	52353	0.091681
16	16	0.000109	273	0.000975
17	17	0.000059	110799	0.17132
18	19	0.000107	1388542	2.35754
19	19	0.0001	59060	0.152854
20	20	0.000102	23571	0.059238

As we can see from this table, the proposed system guarantees to find a solution more efficiently than the one without any heuristic. Note that the number of variables, i.e., classes, shown in this table is below 20 since the processing time will be dramatically increase if as the number of class increases. In these test cases, the number of departments is 10. Assume that each department has only one classroom. The instructor percentage is 60%, and the preference percentage is 15%. In our experiment, it will take the system without heuristic more then 20 minutes to find a solution for assigning 30 classes while the proposed system is able to solve the same problem within 0.001 seconds.

6.3 Conclusion

The course scheduling problem is a challenging problem since it has to satisfy multiple constraints. In this project, we implemented a university course scheduling system to address this problem by formulating it as a constraint satisfaction problem and solving it with the backtracking search, combining the most constraint variable heuristic and a scoring approach. We divided multiple constraints into two categories: (i) hard constraint and (ii) soft constraints. The hard constraints are solved by using backtracking search, and we satisfy the soft constraints as much as we can by using a scoring approach to weight each classroom according to the time and location preference of the students and instructors, as well as the efficiency preference of classroom capacity. Simulation results show that this system is able to find a solution in a low processing time, helping satisfy the requirements from instructors and students and is suitable for helping schedule courses for school authority

REFERENCES

Dave Corne, Peter Ross, and Hsiao lan Fang. 1994. Evolutionary Timetabling: Practice, Prospects and Work in Progress. In *In Proceedings of the UK Planning and Scheduling SIG Workshop, Strathclyde*.

D. de Werra. 1985. An introduction to timetabling. *European Journal of Operational Research* 19, 2 (1985), 151 – 162. DOI:http://dx.doi.org/10.1016/0377-2217(85)90167-5

K. Ellingsen and M. Penaloza. 2003. *A Genetic Algorithm approach for finding a good course schedule*. Technical Report. South Dakota School of Mines and Technology, USA.

Alain Hertz. 1992. Finding a feasible course schedule using Tabu search. *Discrete Applied Mathematics* 35, 3 (1992), 255 – 270. DOI:http://dx.doi.org/10.1016/0166-218X(92)90248-9

A. Oner, S. Ozcan, and D. Dengi. 2011. Optimization of university course scheduling problem with a hybrid artificial bee colony algorithm. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*. 339–346.

Stuart Russel and Peter Nrovig (Eds.). 2010. *Artificial Intelligence: A Modern Approach* (3rd. ed.). Prentice Hall, Chapter 6, 202–232.

Appendix A: Test Instance Generator

The test instance we use here are generated using a test instance generator. The generator uses the idea of reverse engineering. We first randomly generate a course schedule with the desired number of classrooms, number of timeslots, number of courses, number of instructors and percentage of instructor who have time preferences. This is then worked as a reference to the assignment of the input matrices (section 2). Take the classroom matrix for example. For a certain course with courseID, we will first check in which classroom and which time slot it is in, let's say classroom i and time slot j. The capacity of classroom i is n. Then we will assign the capacity of the course to be a number smaller or equal to n, with more probability to be greater than n-50 (which is the second largest capacity of a classroom). We will assign the location preference of the course corresponding to the college where the classroom is located. The weekday preference is set the same as the day the timeslot is in and so to the time period preference.

This test generating method will make sure that the test instance we use will have a solution. To make the test instance harder and more real, we introduce noises to the test instance. When assigning each of the matrices, we choose a certain part of the variables to be randomly assigned (we use 1.5% here).