|  |
| --- |
| Homework 2 Report |
| *Search for Room Scheduling* |
|  |
| Artificial Intelligence  Jose Perez |
|  |

# Introduction

A problem all universities have at the beginning of every semester is putting together the semester schedule. To do that they need to consider all the courses, faculty, buildings, rooms, capacities, and time slots. This task is difficult to solve and quite time consuming. This is commonly known as the room scheduling problem or timetable problem.

In this assignment I implemented some algorithms that attempt to find the best possible schedule as quickly as possible using a variety of search techniques and a simplified version of the problem. The search techniques were simulated annealing and backtracking search for constraint satisfaction.

In my version of the problem we have a set of N rooms, a set of M courses that need to be scheduled, and a set of L buildings. Each building is given an (x,y) coordinate location. Each room has a building and maximum capacity. Each course has an enrollment number, a value for being scheduled, a list of values for each of 10 available time slots, and a preferred building. There are 10 possible time slots, and each room can have only one class scheduled in each time slot. In addition, courses can only be scheduled in rooms where the capacity is greater than the enrollment.

# Proposed Solution Design and Implementation

I was provided base code for a naïve schedule solver as well as implementation for evaluation of schedules.

I extended the base code to also include simulated annealing as one possible algorithm to solve the problem. I started the algorithm with a predefined temperature and cooling rate. Then I created an empty schedule and filled it in a naïve way to get a starting state. Afterwards I started a main loop which ran if there was time left for processing and in that main loop I ran the simulated annealing algorithm. My successor function got two random courses and swapped their rooms and time slots. My cooling function was simply the current temperature with the cooling rate subtracted. I kept track of the best schedule and once my time or temperature ran out I picked that one as my result.

# Experimental Results

The computer used to run these experiments is a Dell Inspiron 13 which has an Intel i7 Core and 12GB of memory.

These experiments were run in Eclipse 4.6.1 (Neon.1a) using Java 1.8.0\_111

Use seed 500 for random number generation to replicate the experiments in the program.

## Table of Results for Naïve

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Buildings** | **Rooms** | **Courses** | **Time Limit (s)** | **% Time Used** | **Score** |
| 5 | 5 | 5 | 2 | 0.10% | 74 |
| 8 | 8 | 8 | 2 | 0.05% | 316 |
| 10 | 10 | 10 | 2 | 0.05% | 203 |
| 12 | 12 | 12 | 2 | 0.05% | 301 |

## Table of Results for Simulated Annealing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Initial Temp** | **Cooling Rate** | **Buildings** | **Rooms** | **Courses** | **Time Limit (s)** | **% Time Used** | **Score** |
| 100 | 0.0001 | 5 | 5 | 5 | 2 | 33.85% | 137 |
| 100 | 0.001 | 5 | 5 | 5 | 2 | 6.15% | 137 |
| 100 | 0.01 | 5 | 5 | 5 | 2 | 2.10% | 126 |
| 100 | 0.1 | 5 | 5 | 5 | 2 | 0.60% | 114 |
| 100 | 1 | 5 | 5 | 5 | 2 | 0.35% | 110 |
| 100 | 10 | 5 | 5 | 5 | 2 | 0.40% | 74 |
| 100 | 0.0001 | 8 | 8 | 8 | 2 | 44.55% | 429 |
| 100 | 0.0001 | 10 | 10 | 10 | 2 | 51.90% | 367 |
| 100 | 0.0001 | 12 | 12 | 12 | 2 | 59.00% | 587 |

# Conclusion

Picking a starting temperature, cooling rate, and cooling function for simulated annealing is not a trivial task. My implementation is surely not the best way to do it. I’m sure there’s a way to change the temperature better based on the time that is left as well as how good our solutions are getting but I’m not sure how to implement that kind of adaptability into the temperature.

Simulated annealing does better than the naïve implementation as it gets higher scores. However, the time it takes is longer than that of the naïve method. Faster cooling rates lead to worse scores and faster times in my experimental data when everything else was constant.