COMP 4106 – Artificial Intelligence

Final Project

Timetabling through A* Heuristics and Simulated Annealing

By: Adam Batson – 100949746

April 20th, 2017

Motivation

For many people scheduling their personal timetable in a way that allows them to efficiently find the time for all their tasks can be seen as extremely difficult. For example, many university students struggle to find the time to complete all of their out of class assignments due to the hecticness of their daily schedules.

The motivation for this project is to build an artificial intelligence to help people organize their daily lives. Users will be able to input their known weekly schedules (for example, a student's class times), as well as a list of tasks they need to accomplish and their estimated completion times (for example an AI assignment that they estimate will take them about five hours to complete). The agent will then work to find the time inside the users schedule for them to accomplish these tasks.

A variety of methods were explored to find a solution to this problem. This report outlines and provides brief explanations of all methods used.

Basic Tree Searches

The first strategies explored for solving the problem were basic (blind) tree searches. The goal of these algorithms was not to build the ideal solutions, but rather to quickly implement a "proof of concept" that the problem could in fact be solved using AI techniques.

Depth First Search

In the Depth First Search the search tree is explored in a depth first manner. This means that the agent moves down the tree, finding the first path from the root node to a "solution". Or in the case of this timetabling problem, the agent will schedule the first task, then the second, etc.

Breadth First Search

In the Breadth First Search the search tree is explored in a breadth first manner. This means that the agent moves sideways then down the tree, exploring all possible permutations of the state space until it finds a solution.

A* Heuristic Searching

The A* algorithm uses heuristic functions to guide the search along the optimal path through the tree. The heuristic function f(n) is broken into two parts [1]:

$$f(n) = g(n) + h(n),$$

Where:

- g(n) = Cost so far to reach state n
- h(n) = Estimated cost from state n to the goal (The Heuristic)
- f(n) = Estimated total cost from root to goal through path n

Two heuristics where developed as part of this project. A quick overview of both follows.

Low Time Per Day Heuristic

The low time per day heuristic attempts to minimize the average amount of hours booked in a day. The goal here is to create a balanced schedule, so that the user doesn't feel overwhelmed on a particular day.

Low Tasks Per Day Heuristic

The low task per day heuristic attempts to minimize the average amount of tasks booked into a certain day. The goal again is to create a balanced schedule, so that the user doesn't feel overwhelmed at any time throughout the week.

Simulated Annealing

Simulated annealing is a probabilistic heuristic for finding an approximation of the global optimum of a given function in a large search space [2]. The name simulated annealing is inspired by the notion of annealing metal objects in metallurgy.

The basic steps to simulated annealing are as follows. The algorithm starts at a certain state in the system and then chooses a random neighboring state to move to. If the new state is considered worse than the old state, the agent probabilistically determines whether or not it will accept a worse state for now, or return to the old state. This probability is based on the energy levels at both states, as well as the system temperature. In the case of this project, the agent will repeat these steps 10000 times. After this the agent will return the optimal solution that it found.

It should also be noted that due to the very large search space involved with this problem that it is not feasible to explore all states in the state space. In order to make the search time reasonable the agent will perform a depth first search of the state space, selecting the first 5000 nodes. This ensures that there will be valid solutions in the state space that the annealer will search, while also keeping search times reasonable.

Probability of accepting a worse state

Should the agent move to a worse state, the agent must probabilistically determine if it should remain at this state, or return to the previous one. In the case of this project the probability is calculated as follows using an exponential distribution:

$$p = e^{-(E(prev)-E(curr))/T}$$

Where E(x) is the energy at a state x, and T is the temperature.

Energy at a given state

The energy at a given state is a measure of the optimality of a given state. In the case of this project the goal of the simulated annealed is to minimize dead time between tasks (time wasted waiting for a new task to start after the previous task has ended). Therefore, at each new state the agent counts the number of minutes "wasted" during the week and returns that number as the state's energy.

Temperature and Cooling Schedule

The system temperature is reduced at every step in the annealing process. This reduction means that as time goes on, the agent becomes less likely to accept worse states, helping to guide the agent to the optimal solution.

The Cooling Schedule is the rate at which the temperature is lowered. It is vital to find the right cooling schedule since cooling too slow will waste time, and cooling too fast will cause the algorithm to become stuck in local minimums [1].

In the case of this project the temperature is cooled at rate of Log_2(Initial Temp / Final Temp). The initial temperature was chosen experimentally as 2500, and the final temperature 1.

Results

Overall, all AI techniques implemented were capable of solving reasonably sized time table problems (six tasks paired with the author's class schedule). In terms of finding "optimal" solutions the simulated annealing approach produced schedules that were most balanced and contained low amounts of "dead" time. Performance wise all algorithms performed well with

the exception of breadth first search, which sees a performance hit when the number of tasks to schedule exceeds three.

Running the program

The program was built in C# using Microsoft Visual Studio 2015. The visual studio project can be downloaded from https://github.com/adambatson/TimeTabler . Then simply import the project into visual studio and run the application. It is recommended that you run the application without debugging (debug → start without debugging, or control + F5), since having debugging on causes significant performance reductions, especially with the simulated annealing algorithm.

To input your known or constant schedule into the application simply edit the "schedule.csv" file. The values in this file are all comma separated enter values based on:

$$day\ of\ week\ (0-6), title, start\ time, end\ time$$

The GitHub project contains a sample schedule that will be used by the application be default. When the application is started, the user will be prompted to enter details about the tasks they wish to schedule. When entering times be sure to use the format HH:MM (AM/PM).

References

[1] B. J. Oommen, "AICh04IntelSearch," [Online]. Available: http://people.scs.carleton.ca/~oommen/Courses/COMP4106Winter17/AICh04IntelSearch.pdf. [Accessed 19 April 2017].

[2] Concordia University, "Simulated annealing," [Online]. Available: http://users.encs.concordia.ca/~kharma/coen352/Project/SimulatedAnnealing.pdf. [Accessed 19 April 2017].