AI FINAL PROJECT

Nicholas Tierney

The problem domain that my final project covers is time scheduling for a sporting tournament. This specifically includes creating a booking schedule for times within a tournament that takes place across multiple venues. The goal is to find the solution that results in the least travel distance totaled across all teams participating in the tournament.

The tournament consists of a number of teams that play games in one of several venues. Teams must play each other team once and there are only a limited amount of slots that booking time can be given out. Each arena has a list of distances that it takes to get to each other arena.

I decided to choose this problem because I have played in a hockey league for a long time and for as long as I can remember the schedule has been made by hand and is a very lengthy process. Applying an AI technique to this problem can help create a better schedule in less time and is applicable to many similar problems.

Since there isn't a specific goal that has to be reached and the problem is NP-hard, iterating better schedules from previous tries seemed to be the best choice; this is why I decided to solve the problem by using a genetic algorithm.

Normally when using a genetic algorithm you perform a set of operations on a bit string that represents your state space, but this method did not seem to suit my problem very well. In this problem every team must play each other once so there are only a limited amount of games that can be played. Each game with a unique set of teams was assigned an integer value and stored in a list. Each bit in the state space string represents a time slot in the tournament; by doing this we can assign a team pairing to each bit and perform genetic algorithm operations on it. For example a tournament with 4 teams would require 3+2+1= 6 games to be played and the string could look like:

(243510)

When using a string of integers representing the games played, we cannot use the default genetic algorithm operators that are meant for bit string. This is due to the fact that if you were to have multiple occurences of the same game in a string, the tournament would be missing another game that was required to be played. The first operation used was *order crossover*. Order crossover "builds offspring by choosing a subsequence of [games] within the path of one parent... [and] also preserves the relative ordering of cities from the other parent."[1] The second operation used was mutation, but seeing as individual numbers could not be changed I did this by taking a small substring and inverting it.

The operations that I used worked well on the problem at hand, but were limited in what they could do; they would have to be modified so that the genetic algorithm could be more flexible in what it takes in as input. The main limitation that this imposes is that you must submit a list of time slots that matches the exact amount of games that are to be played. In an ideal situation the best solution would allow you to submit any number of time slots, so long as it is greater than the amount of games that are to be played, and still come up with the best solution possible. The next major enhancement that could be made to the program is making it so that the teams in the tournament can be split up into groups, so that they do not have to play each other team; this is more likely how a tournament would actually happen.

The main starting point for my program was from the book *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* The book gives an example of how to use a genetic algorithm for solving the traveling salesman problem, which I was able to apply to my problem. In the book each integer in the string represents a city that the salesman must travel to, I replaced this with a game that must be played.

The next step was to get information on the general life cycle of genetic algorithms. I found what I needed from the Wikipedia article on genetic algorithms .[2] This is from where I learned what my algorithm should generally be doing in each iteration of its lifetime. The final step was to find out about small details such as what population size, mutation percentage, number of iterations and death rate to use. I found my answer by looking at an article about solving class scheduling with a genetic algorithm.[3]

To run the program you must have 3 text files as input. The first text file is a list of

teams with the format of one team name per line. The second is a list of time slots, this must be equal to the number of games that must be played. The file must have one time slot per line that has the following format:

venueName - MM/dd/yyyy - hh:mm

The third file is a list of distances between each venue specified in the time slot file. The file must have one distance per line with the following format:

venueName1 - venueName2 - distance

The program can be run from the command line with no arguments at all, in which case the default files will be used, or by using the following command:

java TournamentScheduler -t [teamsFile] -s [timesFile] -d [distancesFile]

Once the program is started you can choose to use the default settings for the genetic algorithm or enter a custom configuration. The settings that can be modified are initial population size, survival percentage, mutation percentage and the number of generations to populate before you stop.

References

Books:

[1]Luger, George F, 2009, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, p. 514-515.

Internet:

[2] Wikipedia, *Genetic Algorithms*, http://en.wikipedia.org/wiki/Genetic_algorithm.

[3] Mladen Jankovic, *Making a Class Schedule Using a Genetic Algorithm*, http://www.codeproject.com/KB/recipes/GaClassSchedule.aspx.