**AI Search Assignment**  
Maxi Clayton Clowes   
17/01/2015

Intro

Use the word heuristic

**Using the code**

**Note:** Code is in Python, version 3.4.2

To test the code, run *main.py*. This file will prompt the user to select one of 4 implemented algorithms:

* *“brute”* – a true Brute Force algorithm (included purely for academic discussion purposes)
* *“modified”* – a modified Brute Force algorithm, which returns the best found tour after *n* seconds
* *“nearest”* – a Nearest Neighbour algorithm
* *“genetic”* – a Genetic algorithm

Once an algorithm has been chosen, the user will then by asked to select a valid input file, which the program will then parse to the desired search function. Once the algorithm has completed running, it will produce at output file containing the best tour it discovered and that tour’s length.

*Describe implementations -> overview*

*Focus on issues  
Specific details -> data structure choice, etc*

*Demonstrate understanding of each algorithm*

*ADD MST STUFF IN -> MST = Minimum(graph)*

**General Points**

One of the main problems with these algorithms is knowing when to stop; that is, the optimum solution may be the first tour found, but still they will keep running

**Modified Brute Force Search**

The first algorithm I chose to implement, due to both it’s simplicity of implementation (appropriate for getting to grips with the assignment) and the quality of it’s results was a Brute Force search algorithm (contained in *brute\_force.py*, found in the folder *pvxf29rest*).

Explain brute force, tour results in folder xxxx

O(n!)

n!/(n-r)!

For example, there are 40,320 possible tours of a graph with 8 nodes, 479,001,600 for 12, and 355,687,428,096,000 for 17; hence, the impracticality of a pure brute force implementation is clear.

To overcome this limitation, I then modified the brute force algorithm so it includes a time-based break point.

It was also necessary to change the way the program passed permutations. When the algorithm was run without an adapted permutation generator, there was little variation to be found between the tours. Any amount of variation will not

randomizing the order to as to increase variation across generated tours.

Without this randomization, tours generated were all too similar, xxxxx

n seconds pass, therefore O(n). However, as is evidenced in the tour file outputs

**Nearest Neighbour Search**

I next implemented a Nearest Neighbour Search algorithm

I then improved upon this by changing it into a *repetitive* nearest neighbour implementation by running the nearest neighbour method on all possible starting nodes.

Why recursion?

**Genetic Algorithm Search**

The most interesting solution I implemented xxxxx

**Results**

The first algorithm I chose to implement, due to both its simplicity of implementation (appropriate for getting

*Tabulated description of results*

* *lengths of best tours*
* *analyse quality*
* *run time*
* *better tours = better marks*

*Details of experiences running the implementations on different inputs*

*Comparative analysis*

*Fine tuning and experimentation -> improving performance*

|  |  |
| --- | --- |
| Algorithm | Time Complexity |
| Brute Force | O(n!) |
| Modified Brute Force | O(n) |
| Nearest Neighbour | O(n3) |
| Genetic Algorithm | ? |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Algorithm run time (s) | | |
| Input file | Modified brute force | Nearest neighbour | Genetic |
| AISearchfile012.txt | 12 |  |  |
| AISearchfile017.txt | 17 |  |  |
| AISearchfile021.txt | 21 |  |  |
| AISearchfile026.txt | 26 |  |  |
| AISearchfile042.txt | 42 |  |  |
| AISearchfile048.txt | 48 |  |  |
| AISearchfile058.txt | 58 |  |  |
| AISearchfile175.txt | 175 |  |  |
| AISearchfile180.txt | 180 |  |  |
| AISearchfile535.txt | 535 | ~3900 |  |

**Discussion**

The first algorithm I chose to implement, due to both it’s simplicity of implementation (appropriate for getting