ECSE 526 - A1 - DynamicConnect4 Report

Author : Angus McLean - 260529744

Question 1 - Number of States Explored MiniMax-vs-AlphaBeta

> For each of the configurations given in the assignment specifications, graph the total number of states explored by your program when using depth cutoffs of 3, 4, 5 and 6, both with minimax and alpha-beta. Assume it is white's turn to play.

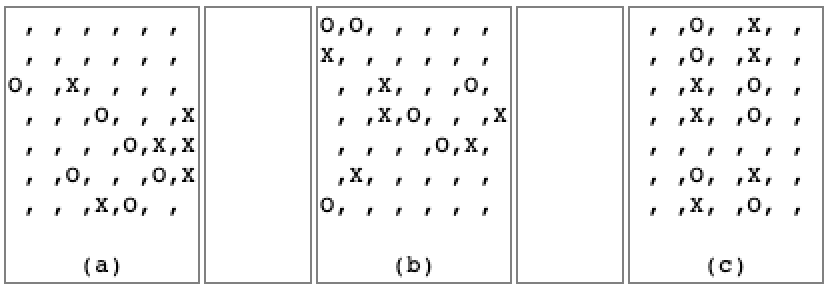
Answer Process :

- Create function to run a game with a given algo, depth, and initialState.
- Iterate all combination and track states visited and execution time
- Plot results

Conclusion :

- AlphaBeta pruning greatly reduces the number of states visited.
- Number of states searched with MiniMax very quickly exponentiates with increasing depth

Test MiniMax-vs-AlphaBeta in Starting States a,b,c



```
 , , , , , ,
 , , , , , ,
O, ,X, , , ,
 , , ,O, , ,X
 , , , ,O,X,X
 , ,O, , ,O,X
 , , ,X,O, ,

(a)
```

```
O,O, , , , ,
X, , , , , ,
 , ,X, , ,O,
 , ,X,O, , ,X
 , , , ,O,X,
 ,X, , , , ,
O, , , , , ,

(b)
```

```
 , ,O, ,X, ,
 , ,O, ,X, ,
 , ,X, ,O, ,
 , ,X, ,O, ,
 , , , , , ,
 , ,O, ,X, ,
 , ,X, ,O, ,

(c)
```

MiniMax-vs-AlphaBeta Testing - DataFrame

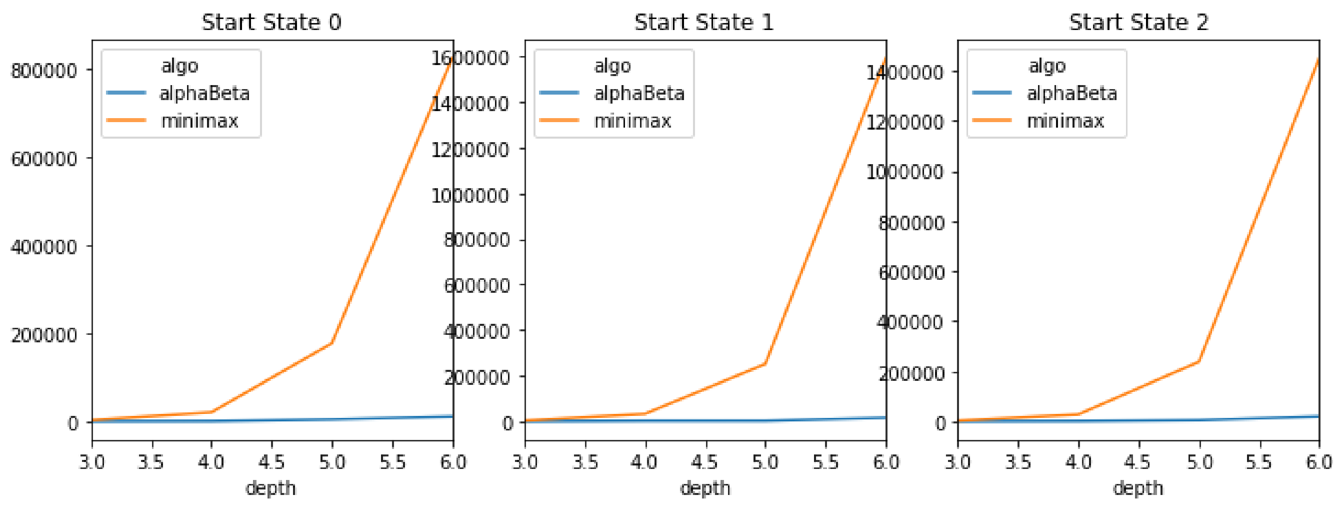
Out[7]:

	algo	depth	statesVisited	initState	time	score
0	minimax	3	3307	0	0.455527	4
1	minimax	3	3522	1	0.471711	3
2	minimax	3	3237	2	0.524340	4
3	minimax	4	21056	0	2.854944	4
4	minimax	4	33090	1	4.248054	-3
5	minimax	4	28180	2	3.737220	4
6	minimax	5	177053	0	21.735395	4
7	minimax	5	251240	1	30.519922	3
8	minimax	5	237568	2	28.832614	4
9	minimax	6	823238	0	95.942616	4
10	minimax	6	1592943	1	194.975436	-3
11	minimax	6	1450266	2	173.448145	4
12	alphaBeta	3	507	0	0.075721	4
13	alphaBeta	3	267	1	0.041557	3
14	alphaBeta	3	514	2	0.076527	4
15	alphaBeta	4	921	0	0.165416	4
16	alphaBeta	4	2434	1	0.326073	-3
17	alphaBeta	4	1548	2	0.220567	4
18	alphaBeta	5	4866	0	0.664091	4
19	alphaBeta	5	2526	1	0.376431	3
20	alphaBeta	5	5200	2	0.769133	4
21	alphaBeta	6	11317	0	1.690097	4
22	alphaBeta	6	15770	1	2.049586	-3
23	alphaBeta	6	19759	2	2.855761	4

Description :

By iterating (algos, depths, startStates) and tracking the number of nodes visited and execution time for each run, we are well equiped to comment on any relationships within.

MiniMax-vs-AlphaBeta on states a,b,c



Description :

By plotting the depth of search against number of states visited for both MiniMax and AlphaBeta pruning algos we can evaluate the performance increase that pruning brings.

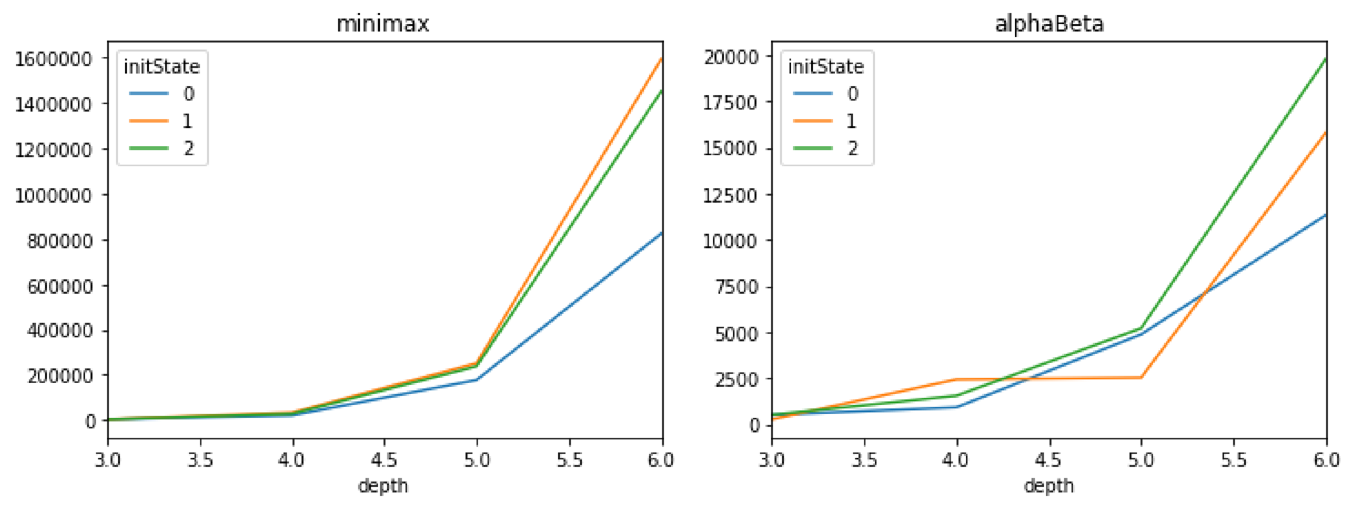
Graph Info :

- xAxis : Depth
- yAxis : Number of nodes visited during graph search
- TraceColors : The different algorithms used for each algo

Observations :

- AlphaBeta searched far fewer states for all board configurations
- Number of states visited by MiniMax appears to exponentiate (to be validated next question)

Efficiency at Various Board States



Description :

By plotting the depth-vs-nodes for both MiniMax and AlphaBeta pruning algorithms across the various starting states we can investigate how these 2 algorithms interact with different board configurations

Graph Info :

- xAxis : Depth
- yAxis : Number of nodes visited during graph search
- TraceColors : The different starting board formations (0,1,2)

Observations :

- Again, we see AlphaBeta visits considerably (orders of magnitude) less states than MiniMax
- Interestingly, AlphaBeta searches oddly very few states at depth 5 for board configuration 1. This is likely because in 4 turns total (ie 2 turns for O), the player can score very highly and therefore many states can be pruned.

Question 2 - Estimate Depth->States Formula

Estimate a formula that relates the depth cutoff to the number of states visited for each of minimax and alpha-beta algorithms.

Answer Process :

- Define structure of fitting function
- Using DataFrame from previous test, fit function to Number of Nodes
- Plot resulting function and original data

Conclusion :

- AlphaBeta's branching factor is roughly half the size of MiniMax resulting in significant performance increase.
- The theoretical formula of states-visited aligns very closely with the test results in practice.

Define Formula & Compute Parameters

The number of nodes explored by the minimax algorithm is defined theoretically as :

$$N_{mm} = O(b^d)$$

- N_{mm} is the number of nodes visited
- b is the branching factor
- d is the search depth

Obviously the branching factor is an approximation because the number of possible moves for each piece is typically 4 but not always (edge of board, proximity with other pieces). By using the same board states for both AlphaBeta and MiniMax we can guarantee that the actual branching factor will be the same between trials because game-treee (and hence the actual branching factor) is a result of game state rather than the game agent.

Based on the theoretical forumlation of depth to nodes we will define the parameterized function as follows :

$$y = ae^{bx}$$

- y is the estimated number of nodes visited (ie N_{mm})
- a is the tunable parameter representing branching factor
- b is the tunable parameter scaling search depth
- x is the inputted search depth

After fitting the function $y = ae^{bx}$ to the measured test results for MiniMax and AlphaBeta the parameters were as follows :

MiniMax :

Out[13]:

$$N_{mm} = O(b^d) = 30.53e^{1.78d} = 30.53(5.9)^d$$

AlphaBeta :

Out[14]:

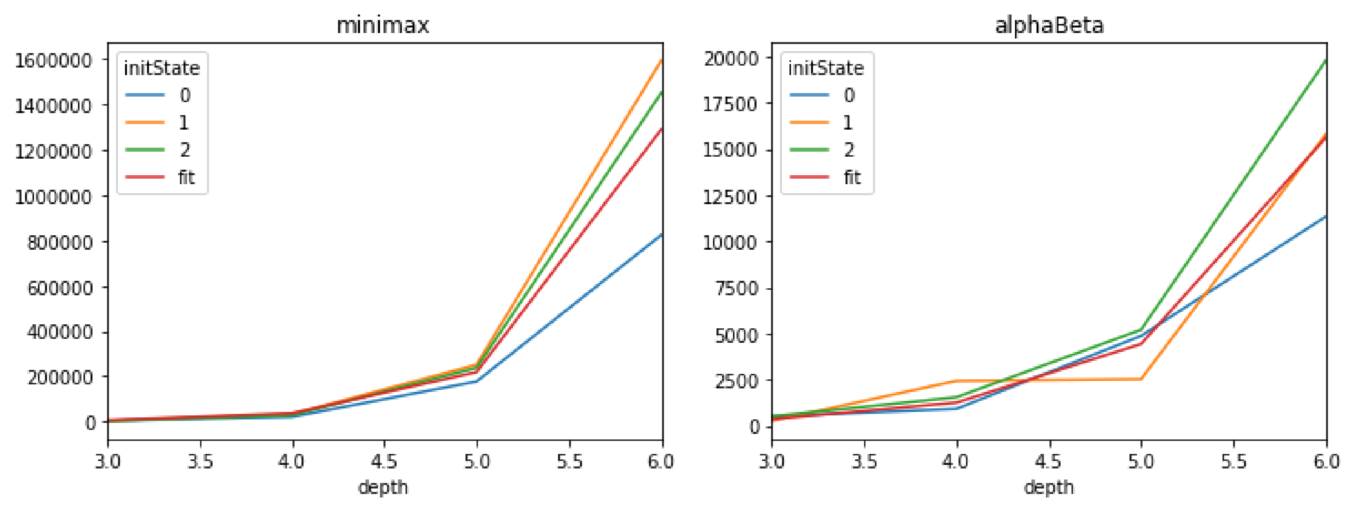
$$N_{\alpha\beta} = O(b^d) = 8.21e^{1.26d} = 8.21(3.52)^d$$

Comparing Branching Factors :

By algebraically simplifying the parameterized fitting function we can see that the branching factor for AlphaBeta is considerable lower than that of MiniMax.

Graph Best Fit Line

Fitting MiniMax-AlphaBeta Complexity



Description :

Similar to above, by plotting the depth-vs-nodesVisited for both MiniMax and AlphaBeta algorithms across the starting states we can investigate how accurately our fitted complexity function models the observed behaviour of each algorithm.

Graph Info :

- xAxis : Depth
- yAxis : Number of nodes visited during graph search
- TraceColors : The different starting board formations (0,1,2)
- Note : Red line is the approximated complexity function

Observations :

- The parameterized function is able very closely model the search algorithms' efficency. Therefore, the above comparison of fitted parameters is an apt quantitative comparison of these 2 algorithms

Question 3 - Impact of Exploration Order on Efficiency

> Explain whether the number of states explored depends on the order in which you generate new states during the search. Justify your response using results from your program.

Answer Process :

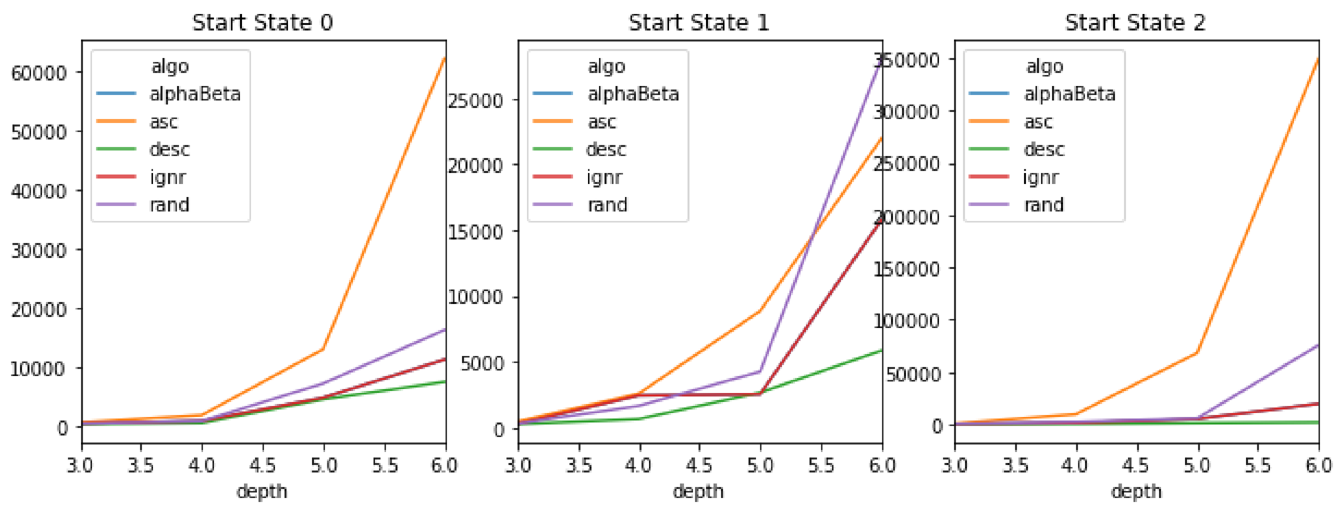
- Modify AlphaBeta function to accept optional sorting parameter
- Track performance of AlphaBeta with different sorting strategies
- Plot performance

Conclusion :

- For AlphaBeta pruning the order in which states are generated does impact the number of states explored during the search.
- Order (worst to best) : ascending, random, no-sorting, descending

---

Sorting Strategies with AlphaBeta



Description :

By plotting the depth-vs-nodes for AlphaBeta Search Strategies across the starting states we can investigate how sorting by score changes amount of pruning taking place

Graph Info :

- xAxis : Depth
- yAxis : Number of nodes visited during graph search
- TraceColors : The different sorting strategies

Observations :

- Ascending (expanding worst moves first) is always worse than the others
- Descending is always best
- Interestingly, in states 0 and 2 (where a win is quickly achievable) we see the greatest decrease in performance using ascending sort. This makes intuitive sense because picking 'strategic' moves first has less impact when there is less obvious advantage to be gained in the near term.

Intuition :

- AlphaBeta pruning works by not further exploring moves that are proved to be worse than a previously explored state. By sorting in ascending order we never encounter an opportunity to prune and therefore revert to the same performance as minimax.

Question 4 - Explanation of Heuristic

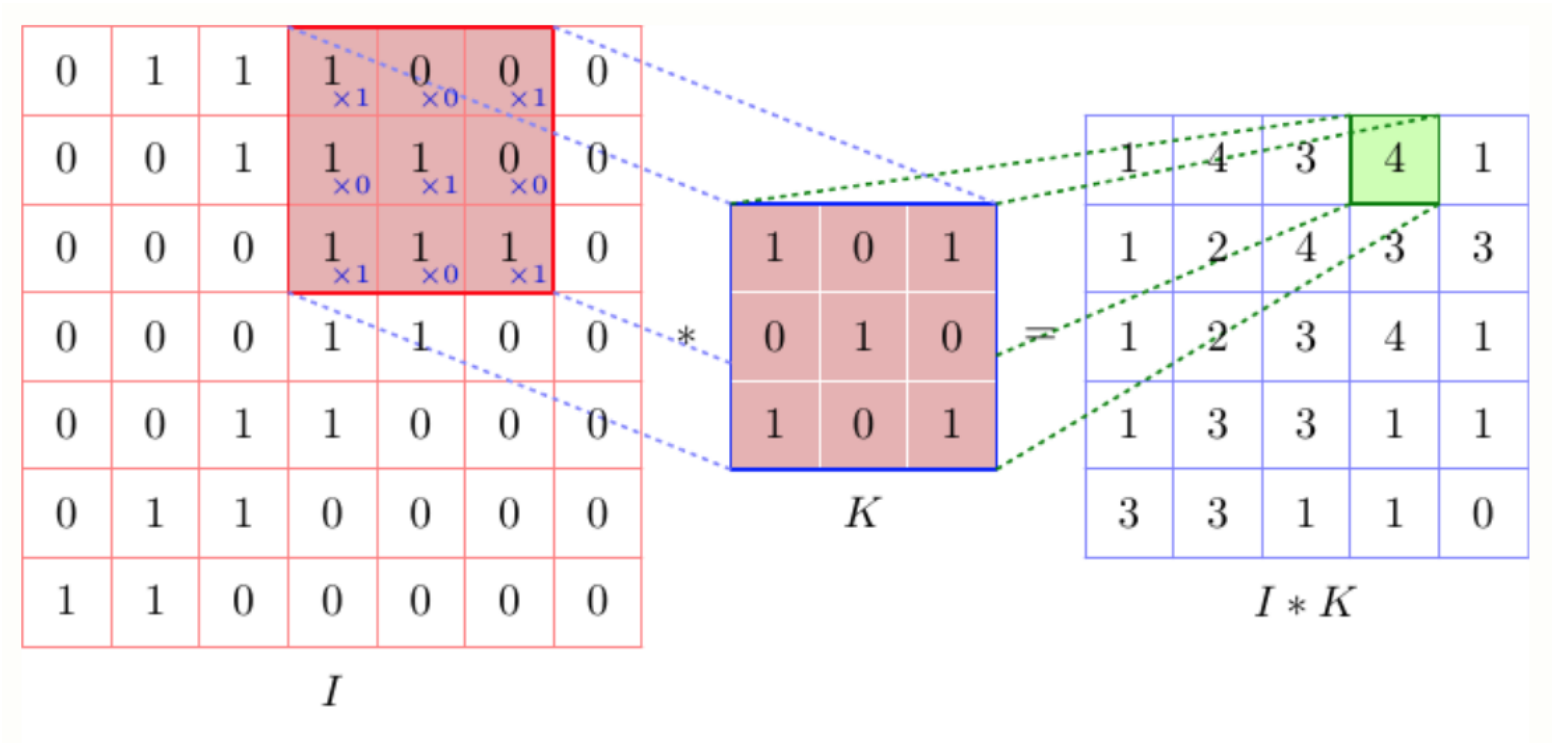
Explain the heuristic evaluation function you used and provide a clear rationale for the factors you included.

One advantage of the game state being represented as a matrix is that it is quites likely that your heuristic function can be designed as a matrix operation against the current state.

Heuristic as Convolution Operation :

In the case of Dynamic Connect 4 the objective is to achieve a particular spatial arrangement of your pieces (ie 4 pieces in a line). Similar to how CNNs operate we can apply a convolution operation to 'detect' or 'highlight' specific piece-configurations existing inside the 2d game board.

To detect this configuration of pieces on the board we can use a "filter"/"kernel" and convolution operation against the current game state matrix.



$$
I = \begin{pmatrix} 0&1&1&1&0&0&0\\ 0&0&1&1&1&0&0\\ 0&0&0&1&1&1&0\\ 0&0&0&1&1&0&0\\ 0&0&1&1&0&0&0\\ 0&1&1&0&0&0&0\\ 1&1&0&0&0&0&0 \end{pmatrix} * K = \begin{pmatrix} 1&0&1\\ 0&1&0\\ 1&0&1 \end{pmatrix} = I*K = \begin{pmatrix} 1&4&3&4&1\\ 1&2&4&3&3\\ 1&2&3&4&1\\ 1&3&3&1&1\\ 3&3&1&1&0 \end{pmatrix}
$$

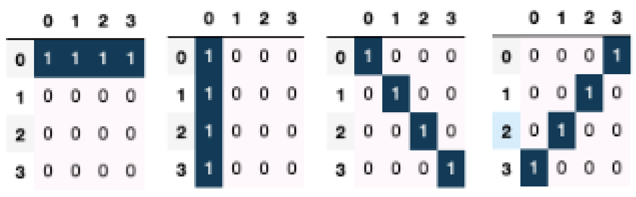
$score = max(conv_{K_{0-4}}(I_{state}))$

K : Filter Matrix (4 of them as defined above

I : Game State Matrix

Basic Approach :

The following filter matrices were used for counting pieces in a row:



	0	1	2	3
0	1	1	1	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

	0	1	2	3
0	1	0	0	0
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0

	0	1	2	3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

	0	1	2	3
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0

First take the game state to include only pieces of the given player and then convolve the 4 filter matrices, whatever the max value of the resultant matrices is the maximum number of connected pieces.

Demonstration of Scoring Game States (Basic Approach)

```
 ------- Filter Diagonal 1 -------
[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]]

 ------- State from Question 1a -------

 , , , , , ,
 , , , , , ,
O, ,X, , , ,
 , , ,O, , ,X
 , , , ,O,X,X
 , ,O, , ,O,X
 , , ,X,O, ,

 ------- Resultant Convolution -------
```

	0	1	2	3	4	5	6
0	-1	0	0	0	0	0	0
1	0	0	0	0	-1	0	0
2	1	0	1	-1	-1	-1	0
3	1	1	0	2	-2	-1	-1
4	0	0	1	0	3	-2	-1
5	0	0	0	1	0	2	-2
6	0	0	0	0	1	0	1

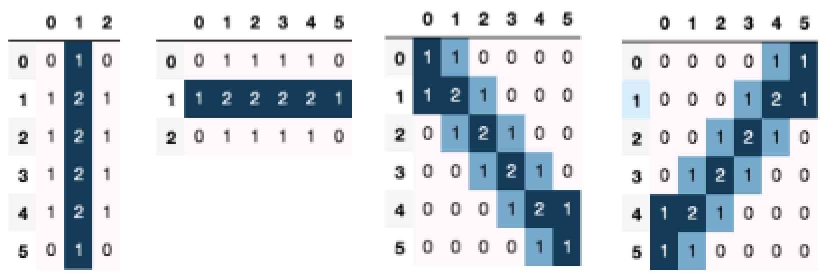
```
 Score for State (ie max convolution) :  3
```

Improving the Heuristic

Adding 'Awareness'

The basic approach works just fine but has some a couple short-comings.

- It has no 'periphery', ie it has no concept of pieces being 'almost' in a line
- No awareness of position of opponent pieces. Surely we want to penalize states where there are many opponent pieces nearby.



	0	1	2
0	0	1	0
1	1	2	1
2	1	2	1
3	1	2	1
4	1	2	1
5	0	1	0

	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	2	2	2	2	1
2	0	1	1	1	1	0

	0	1	2	3	4	5
0	1	1	0	0	0	0
1	1	2	1	0	0	0
2	0	1	2	1	0	0
3	0	0	1	2	1	0
4	0	0	0	1	2	1
5	0	0	0	0	1	1

	0	1	2	3	4	5
0	0	0	0	0	1	1
1	0	0	0	1	2	1
2	0	0	1	2	1	0
3	0	1	2	1	0	0
4	1	2	1	0	0	0
5	1	1	0	0	0	0

Adding 'fuzzyness' to the convolution allows for 'peripheral' abilities and including opposing pieces in the calculation incorporates the penalizing aspect.

Combining Heuristics :

By adding the scores of both the basic convolutions and fuzzy convolutions we can build a robust heuristic function for approximating the utility of a given state. The speed/accuracy trade-off of this combined heuristic is explored next..

Note about scaling : When adding both heuristics together I scaled the fuzzyHeuristic output such that it returns values no greater than one. In this way the search algorithms will always prioritize connecting additional pieces because increments of the basic heuristic are larger than total value of fuzzyHeuristic.

Question 5 - Complexity of Heuristic Function

> Discuss the computational trade-offs with respect to the use of a more complex evaluation function and the depth of the game tree that can be evaluated.

Answer Process :

- Modify AlphaBeta function to accept heuristic parameter
- Track performance of AlphaBeta with different basic and advanced heuristics
- Plot performance

Conclusion :

- More complex heuristic obviously will take longer to compute but also may change the number of nodes your GameAgent searches which will also impact execution time.
- Setting up a synthetic graph search problem is a good way to measure heuristic 'accuracy'.

Heuristic Speed

Speed Test :

As outlined above (and in the implementation docs), the combined heuristic uses encorporates both the basic heuristic as well as the fuzzy heuristic. This boils down to computing 8 convolutions against the game state for combined heuristic and 4 convolutions for basic heuristic.

Both the basic heuristic and the combined heuristic are timed below :

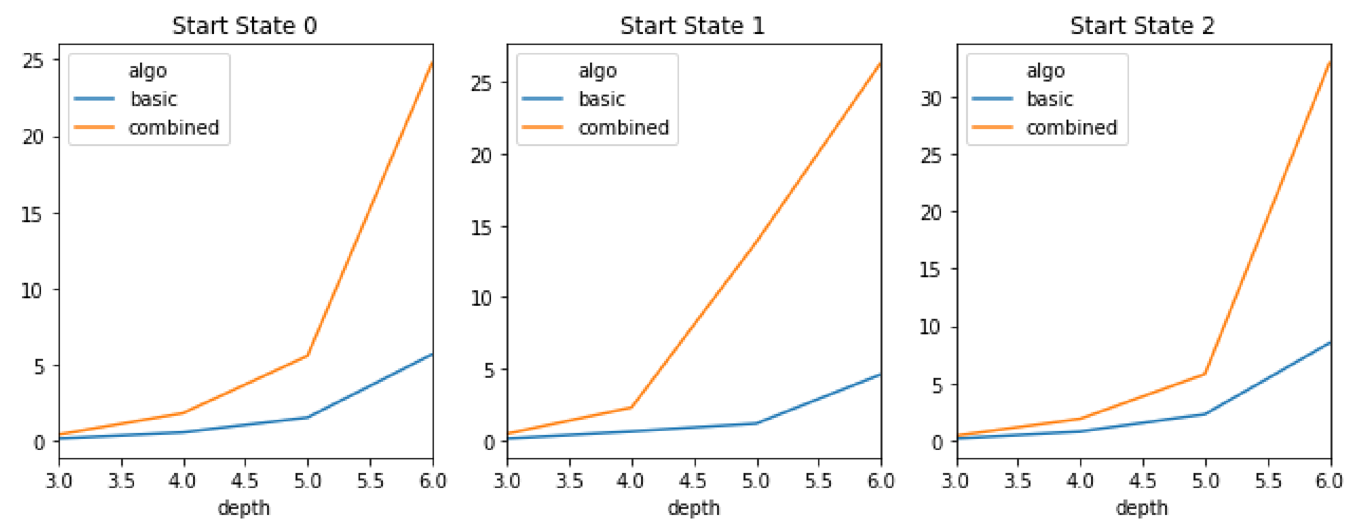
```
'%timeit -n 5000 basicHeuristic'

115 µs ± 3.69 µs per loop (mean ± std. dev. of 7 runs, 5000 loops each)

'%timeit -n 5000 combinedHeuristic'

237 µs ± 3.93 µs per loop (mean ± std. dev. of 7 runs, 5000 loops each)
```

Execution time Basic vs Combined Heuristic



Description :

By plotting the depth of search against the execution for both basic and combined heuristics we can measure the impact that additional heuristic complexity has on speed.

Graph Info :

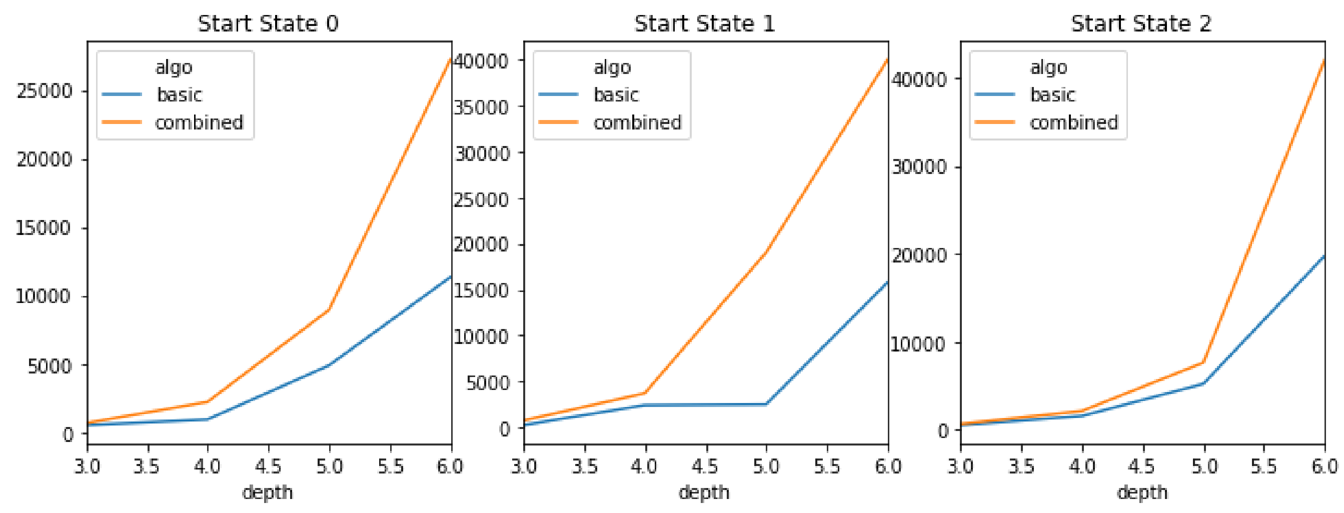
- xAxis : Depth
- yAxis : Execution time (seconds)
- TraceColors : The different heuristics used

Observations :

- Combined heuristic does indeed search slower than basic
- Interstingly the gap between basic and combined execution times widens at an increasing rate. This doesn't seem to match the 2x speed decrease we were seeing when running testing loop

---

Nodes Visited - Basic vs Combined Heuristic



Description :

By plotting the depth of search against the number of nodes visited for both basic and combined heuristics we can measure the impact that additional heuristic complexity has on states visited.

Graph Info :

- xAxis : Depth
- yAxis : Number of states visited
- TraceColors : The different heuristics used

Observations :

- Interestingly, with a compound heuristic the game agent ends up evaluating more states (seemingly a higher branching factor).
- Some of the speed decrease from more complex heuristic function can be attributed to evaluating more nodes.
- Comparing the execution speeds of different heuristics cannot be the only thing used when comparing different heuristics.

Heuristic Accuracy

Theoretically the true accuracy of a heuristic can only be evaluated by exploring the entire game tree and comparing the true utility of a game state vs the estimated value from the heuristic. Obviously exploring the entire game tree is not feasible as an approach to answering (ie quantifying) the issue of heuristic accuracy.

Conceptually, a more 'accurate' heuristic function should enable your agent to more effectively score game states and hence better naviagate the game tree in search of its goal.

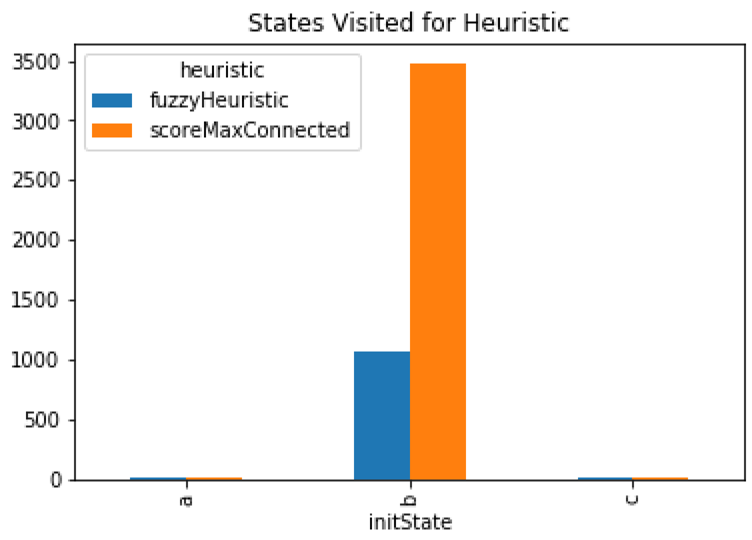
We can setup a simple synthetic A *search problem and see which heuristics find the solution in the least number of nodes explored. With a perfect heuristic the nodes searched by A* would be exactly the nodes needed to traverse to achieve the goal. Worst case scenario A* searches with BFS complexity

Various Heuristics and Test Performance - DataFrame

Out[33]:

	search_algo	heuristic	statesVisited	initState	time	cost
0	breadth_first	scoreMaxConnected	153	a	0.060142	2
1	breadth_first	scoreMaxConnected	212904	b	104.027003	5
2	breadth_first	scoreMaxConnected	71	c	0.030362	2
3	astar	scoreMaxConnected	9	a	0.029872	2
4	astar	scoreMaxConnected	3469	b	10.028145	5
5	astar	scoreMaxConnected	5	c	0.013476	2
6	breadth_first	fuzzyHeuristic	153	a	0.078964	2
7	breadth_first	fuzzyHeuristic	212904	b	118.795274	5
8	breadth_first	fuzzyHeuristic	71	c	0.039386	2
9	astar	fuzzyHeuristic	4	a	0.021523	2
10	astar	fuzzyHeuristic	1066	b	6.009320	5
11	astar	fuzzyHeuristic	3	c	0.014603	2

How Many States Before Finding Goal?



Description :

By plotting the number of states searched by A* before finding its goal we can roughly get a measure of accuracy of our heuristic function. Conceptually a heuristic that guides the algorithm perfectly to the goal state has the most accurate calculation of a game state's utility.

Graph Info :

- xAxis : Different Start States
- yAxis : Number of states visited
- TraceColors : The different heuristics used

Observations :

- The fuzzyHeuristic was able to find the solution to state b significantly faster than the basic (scoreMaxConnected) heuristic could.