COMP20003

PROJECT 2 EXPERIMENTATION REPORT

Sidakpreet Mann - 921322

Contents

Introduction and Purpose

The purpose of this report is to compare how the AI solver that we have designed (based on a variant of Dijkstra's pathfinding algorithm) in the classic game of Pacman.

The gameboard was conceptualised as a graph with the nodes being the valid moves, and by design Pacman should favour increasing the net score of the game. Two approaches were employed to propagate scores in order to pick the best action:

- **MAX -** Simulate a further given amount of game states (budget) and pick the option with the highest accumulated reward
- **AVG -** Simulate a further given amount of game states (budget) and pick the option with the highest average value of all the possible options within that path.

Each possible configuration of Pac-Man game is within a 29x28 grid, and is called a state (contains all auxiliary data).

Once the **best_action** has been identified by one of the strategies above, then pacman executes that action in game, and then resimulates the given number of valid actions to determine the next best move. This goes on till we are at lives < 0 (game over).

Brains of the AI - Heuristics

The Heuristic function was modified slightly in an attempt to alter the behaviour based on my hypothesis of survival above all else. I wanted to limit my downside risk, I did this in method of mitigating my current risk exposure, but keeping a call option for going for higher scores WHEN all movements are completely safe (when Pacman is invincible).

Invincibility modification -> Not only does Pacman prefer eating fruits, he also leans towards eating ghosts in a row. I added +10 to our heuristic score each time GhostsInARow changed. This account for both the initial invincibility, and also capitalising on eating ghosts in a row and seeking out further fruit in order to capitalise on that increasing multiplier for scores that we can receive by eating multiple ghosts in a row.

Losing Life modification -> I changed this heuristic to -100, as I starkly want to avoid losing a life in an attempt to survive the longest possible.

Game over modification -> Made Pacman extremely conservative by changing game over score to -1000, so on its last life, when played with large enough budget, Pacman will avoid game over to significant lengths.

LevelTest Cases

1. **t_buridans_ass.dat**
   As this test involves checking which path Pacman would take with food on equidistance, our algorithm ensures he will pick randomly if scores are equal either side. The problem is that both of them are quite out of reach, and the Max_depth when testing with a budget of 100 is around 7-8. Since the width of the row is larger than 16, this means that initially Pacman will move randomly left or right until he is able to account for one of the sides score. This also applies when he is travelling back after eating one of the pellets.

   A **potential optimisation** that this case suggest for the entire algorithm is having a slight negative impact (lower than losing a life) when backtracking occurs. For example if one of the graph paths is Left -> Right -> Left which is completely valid and expected, this just moves our player in place especially when there is no threat of ghosts nearby. If these moves are avoided, we can more efficiently expand nodes based on budget, and potentially perform better.

2. **t_consecutive_food.dat**
   This test is quite easily passed by just choosing a higher budget, as the farther we can simulate (the more sequence of moves we can simulate), the easier it will be to move in the correct direction.

3. **t_esacape.dat**
   Another simple test, which will work even with a budget of 2. As there is only one ghost and a simple looping level, based on our algorithm, Pacman will never be caught.

4. **t_escape_and_chase.dat**
   Another simple level, here with an advantage. While we are being provided fruit as well, normally our pacman would always chase the ghosts after being invincible as we get a higher score. My modification will perform the same as the normal implementation because of the lack of alternative pathways. In a normal level setting, while the standard expected heuristics implementation might actually prefer a path with food pellets if score is equal to eating a ghost, but my modification would pick the ghost to capitalise on the increased multiplier.

5. **t_escape_hard.dat**
   While my implementation avoids ghosts more aggressively, there is still a much larger luck component due to the random movement of the ghosts and limited movement options. In some cases, there is no way we can escape if we have been trapped. Here having a larger budget typically would correlate to a longer survival time over a big enough sample.

6. **t_far_food.dat**
   Much like the consecutive food level, the same arguments can be applied here about a larger budget performing better. Max approach seems to work better here as the accumulated score the the case of average can be divided by a large number of nodes and end up in an approximately 0 incentivising a random selection instead of the correct one.

Experimentation Statistics - Overall

Table dispalys the aggregated averages of the 3 sample runs ran over each [Budget, Level, PropagationType], performing a total of 72 tests.

Each row shows the average statistics for the 3 samples ran for that particular Budget, Level and propagation.

	Budget	Level	PropagationType	ExecutionTime	AverageScore	StDev(Score)	AverageExpandedNodes/sec	StDev(ExpandedNodes/sec)
0	10	1	max	63.63s	205.666667	58.976455	59.402703	0.328560
1	100	1	max	141.91s	505.000000	127.898397	812.419593	310.440298
2	1000	1	max	142.49s	332.666667	67.726574	5959.123742	21.247445
3	2000	1	max	92.96s	457.666667	260.256711	13557.616493	5125.261854
4	10	1	avg	186.63s	229.000000	34.185767	59.674429	0.033124
5	100	1	avg	185.09s	235.000000	8.041559	594.820110	5.012821
6	1000	1	avg	127.55s	384.333333	82.794256	5913.026147	18.665108
7	2000	1	avg	131.72s	464.333333	230.845307	17217.552443	10309.028813
8	10	2	max	130.28s	508.000000	284.313208	73.930619	20.764278
9	100	2	max	106.26s	369.000000	47.377913	590.902508	3.440907
10	1000	2	max	84.35s	625.666667	194.602729	8257.131792	3344.226795
11	2000	2	max	181.37s	894.666667	526.554418	14773.900224	6878.957821
12	10	2	avg	97.40s	391.333333	110.927404	59.559096	0.171459
13	100	2	avg	41.99s	670.000000	176.081421	1412.812993	878.176093
14	1000	2	avg	103.30s	485.666667	139.270321	5914.611763	37.708652
15	2000	2	avg	206.51s	391.000000	119.132979	9900.546750	66.840975
16	10	3	max	133.41s	332.000000	65.863495	59.771728	0.079278
17	100	3	max	206.05s	628.333333	165.449556	859.154585	374.303963
18	1000	3	max	85.24s	623.000000	92.361608	8533.246616	3678.823024
19	2000	3	max	184.19s	817.000000	29.709707	22043.209441	6777.419479
20	10	3	avg	133.58s	375.000000	4.898979	59.678614	0.070242
21	100	3	avg	100.46s	429.000000	61.973112	596.364354	0.494692
22	1000	3	avg	142.68s	475.000000	78.042723	5966.909726	3.335985
23	2000	3	avg	828.37s	619.000000	238.312120	10882.866789	1268.099562

Aggregated statistics

		total_execution_time	
		mean	std
budget	propagation		
10	avg	139.203333	44.880003
	max	109.106667	39.415031
100	avg	109.180000	71.947420
	max	151.406667	50.568281
1000	avg	124.510000	19.865228
	max	104.026667	33.313196
2000	avg	388.866667	382.453619
	max	152.840000	51.876766

Average and standard deviation of all 72 tests' execution times arranged by [budget, propagation]

		average_expanded_per_s	
		mean	std
budget	propagation		
10	avg	59.637380	0.067828
	max	64.368350	8.283223
100	avg	867.999152	471.823258
	max	754.158895	143.302219
1000	avg	5931.515879	30.662223
	max	7583.167383	1413.222615
2000	avg	12666.988661	3971.392856
	max	16791.575386	4588.527184

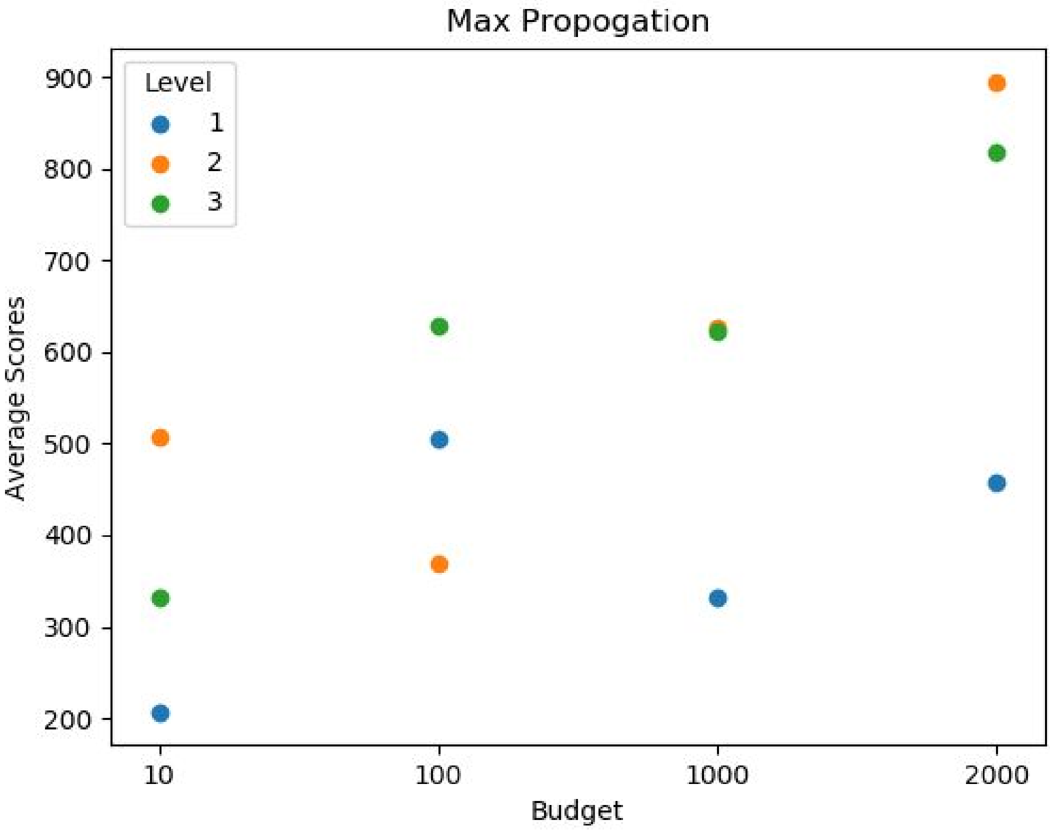
Average and standard deviation of all 72 tests' average expanded nodes per second arranged by [budget, propagation]

		average_sample_score	
		mean	std
budget	propagation		
10	avg	331.777778	89.382035
	max	348.555556	151.845072
100	avg	444.666667	217.922769
	max	500.777778	129.718213
1000	avg	448.333333	55.681635
	max	527.111111	168.399107
2000	avg	491.444444	116.392694
	max	723.111111	233.138570

Average and standard deviation of all 72 tests' average game scores arranged by [budget, propagation]

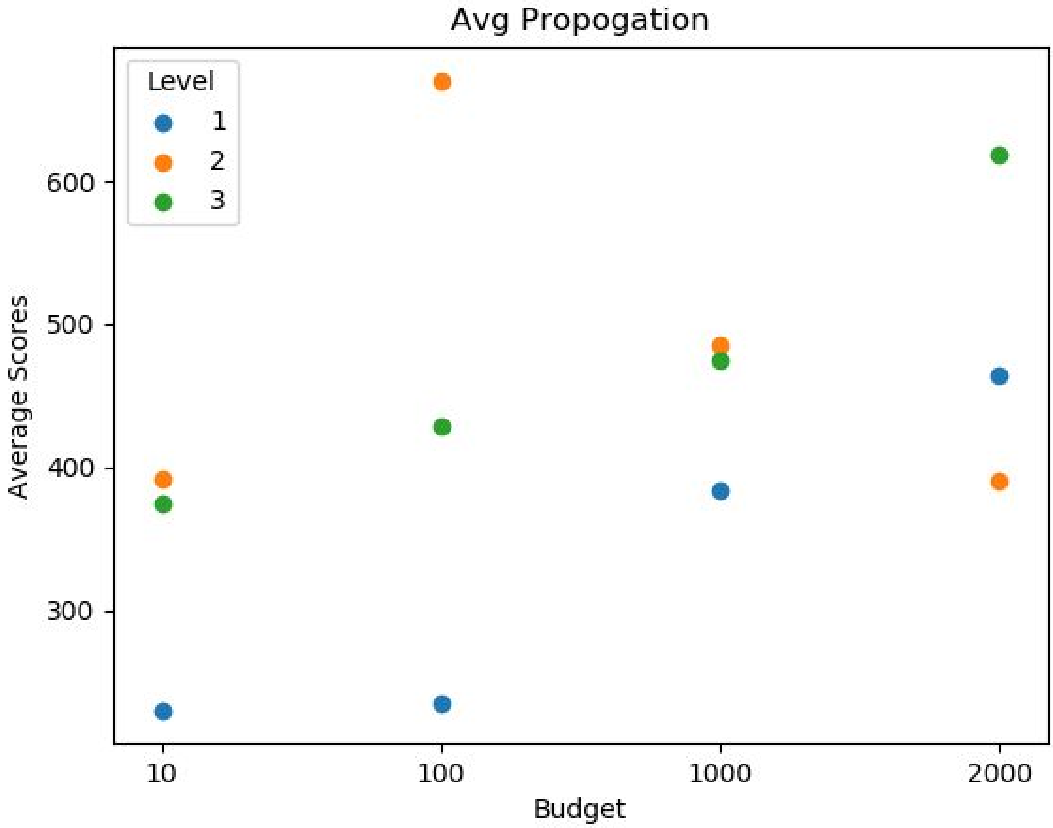
Discussion

From the above (aggregated) tables we can observe that naturally, as the budget increases, pacman is able to perform better on average overall. Looking at the average sample score table, we can see that the average scores display a linear positive trend between budget and average scores. Specifically, we can see major improvements in Max propagation, as this strategy tends to perform better with my modified heuristics focussing on survival, and capitalising on the ghost multipliers.



A general upward trend can be observed in average scores as the budget increases.

However, we can see that with Average propagation, we can get a more predictable / concise performance as it's standard deviation is much lower than Max propagation. This seems to be consistent across the board, with the deviation was approximately 50% of Max propagation. This means that if we wish to further optimise our algorithm, there is a lot less random noise within the average propagation, therefore the effect of the changes made within that propagation can be more clearly identified. Therefore this strategy still has its merits regardless of the lower average score.



Less pronounced upward trend, but the results are much closer together too.

Interestingly, as we observe the total execution time table, we observe that in terms of survival time, average strategy consistently outperforms the maximum strategy. Especially at the outlier level of performance at budget 2000. This could just be due to our small sample size and an outlier, as the standard deviation for these results is quite massive. Therefore, these results seem very biased. The results from my experiments are not statistically significant.