Data Collected from Experimentation of Pacman AI using Differing Budgets and Propagation Types

ALECY - ALEC YU 993433

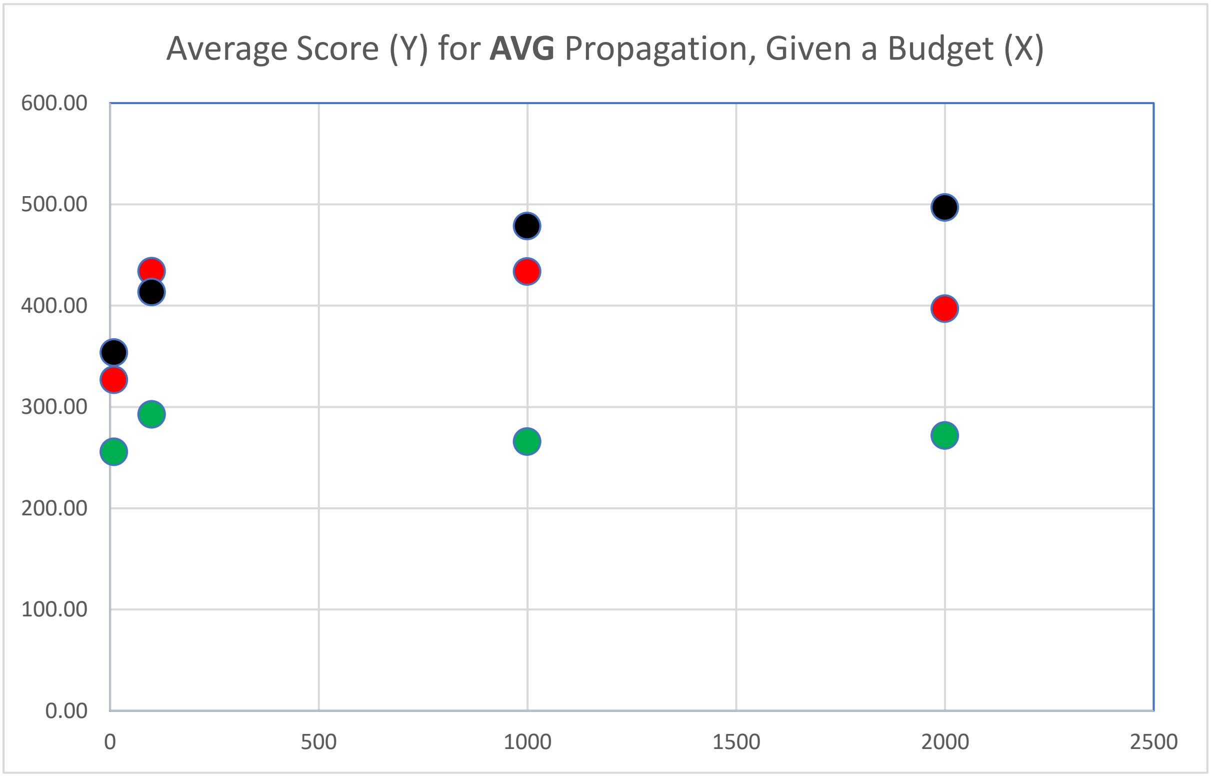
Propagation Type	Budget	Level	Mean Total Expanded	Mean Time	Mean Expanded/Second	Std Deviation Expanded/Second	Mean Score	Std Deviation Score
max	10	1	7034.00	75.86	93.44	15.37	312.00	70.47
max	100	1	49513.00	50.18	985.67	10.87	325.67	52.49
max	1000	1	356035.67	35.55	10016.00	22.69	305.67	18.86
max	2000	1	856132.33	92.40	9297.67	414.23	284.67	23.61
avg	10	1	6020.00	66.67	90.00	0.00	255.33	44.43
avg	100	1	42747.67	43.43	988.00	16.68	292.33	41.10
avg	1000	1	378589.33	37.63	10061.67	85.95	265.67	9.42
avg	2000	1	932608.67	100.18	9316.00	89.83	271.67	34.65
max	10	2	6330.67	70.13	90.00	0.00	361.67	37.25
max	100	2	45494.00	43.92	997.67	6.13	573.33	194.82
max	1000	2	465096.33	46.48	100007.00	43.61	493.33	67.99
max	2000	2	815712.00	89.66	9100.00	52.47	353.33	37.70
avg	10	2	6126.67	75.49	90.33	0.47	326.67	38.77
avg	100	2	64192.67	68.34	996.00	13.15	433.67	31.29
avg	1000	2	622041.33	62.97	9887.00	41.87	433.33	57.17
avg	2000	2	853673.33	93.21	9158.67	29.74	396.67	61.82
max	10	3	6681.33	88.35	90.00	0.00	371.67	37.19
max	100	3	70494.00	70.83	991.33	10.47	513.33	28.14
max	1000	3	546315.00	55.18	9848.67	52.73	566.67	23.97
max	2000	3	1096251.67	109.89	9976.33	91.54	353.33	37.68
avg	10	3	7963.33	88.12	90.00	0.00	413.33	26.58
avg	100	3	74065.00	74.73	993.00	4.79	478.33	37.51
avg	1000	3	589761.67	59.58	9889.67	48.27	496.67	22.73
avg	2000	3	1251963.00	128.62	9799.33	46.15	460.00	34.18

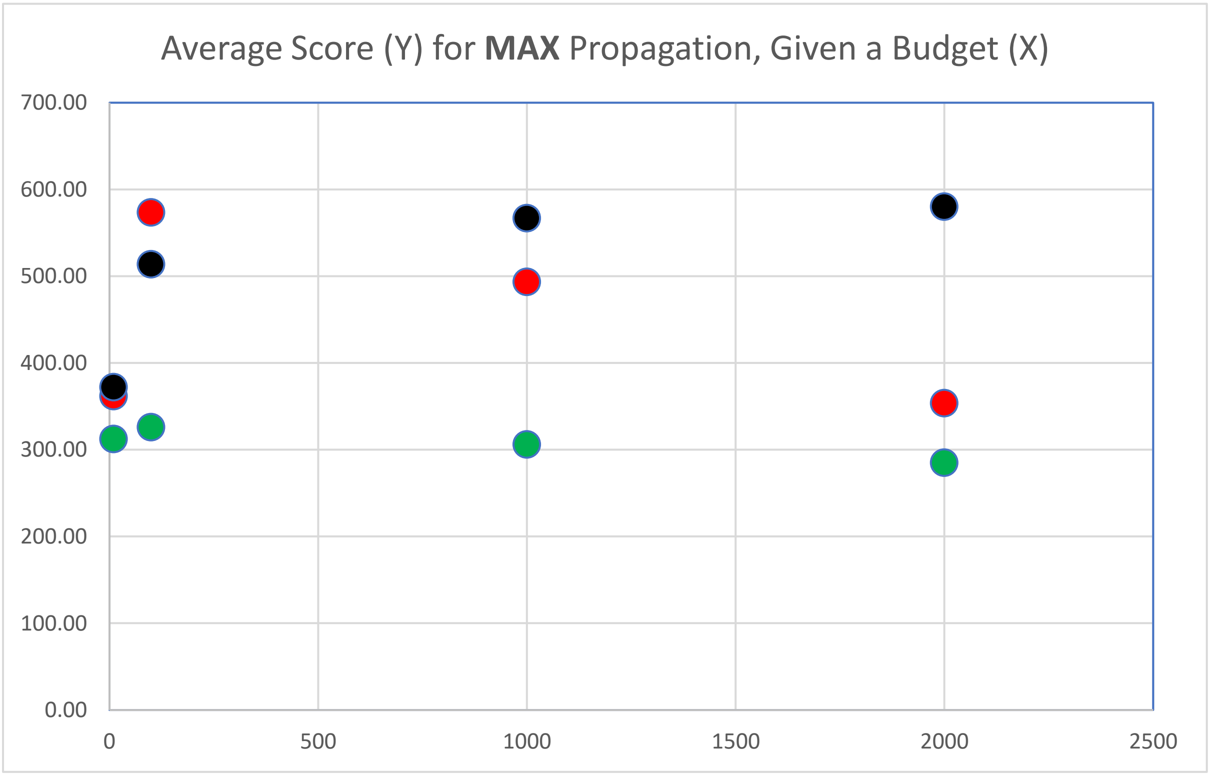
KEY:

GREEN = LEVEL 1

RED = LEVEL 2

BLACK = LEVEL 3





General Performance of the Algorithm Based on results

The results above were obtained by running the Pacman AI through numerous different combinations of settings, specifically Propagation Type, The budget, and the level that the AI was run on, which each mean statistic being obtained from a series of 3 runs.

Generally, my AI was able to “solve” each level, as it could eventually find its way to eat all the food pellets. There were a few cases where if the budget was low enough (10, 100), Pacman could not detect food sources that were on the other side of the map, and would require a lucky streak of random movements to get close enough, or to have a ghost to usher it towards the other side of the map. Because of its inefficient movements, eventually, Pacman would get itself trapped between two ghosts and eventually die. However, on higher budgets (1000, 2000), Pacman’s movements would be a lot more efficient and sensible, and avoid situations such as these. Pacman seemed to perform similarly on average and max propagations across all runs. However, Pacman would choose to take the safer route (Avoid ghosts) when using average propagation, and tend to get itself tangled in ghosts, but still narrowly avoid them, when using max propagation.

When looking at the results, the average score is a misleading statistic on the performance of the algorithm. A lot of the deviation in results I obtained were due to RNG, as eating ghosts made up a large portion of the score. Sometimes, Pacman would be extremely lucky and be able to eat multiple ghosts in a row, as the points obtained stacks multiplicatively. And of course, sometimes there would be runs where Pacman did not eat any ghosts at all. Still, it is worth mentioning that the average score of Levels 2/3 are much higher than Level 1, probably due to the fact that these levels just have more pellets to eat. It’s also evident the average scores for budget 10 is much lower than the other budgets. This suggests that Pacman dies a lot more often with a budget of 10, that often times it would not be able to complete the level and die to ghosts. This is probably caused by it not being able to detect food far away due to its small budget, and would waste a lot of movements flailing around randomly, eventually to be trapped by ghosts.

A good indicator for the performance of the AI is the time taken to “beat” the level. While certain AI’s may want to maximise the number of points obtained in a level, my AI places a focus on eating and finding all the pellets. As explained earlier, with a budget of 10, often times Pacman would not know where his optimal route is, and would randomly flail around. This would obviously affect the time taken to solve the level, as the random movements would mostly be inefficient when trying to solve the level. This trend is present in all the runs using a budget of 10, with the average time taken to be much longer than other budgets (with the exception of 2000). It seems that budgets of 100, 1000 and 2000 are large enough for Pacman to detect food from very far away. This means that even if food is scattered across the whole map, Pacman would know exactly where the food is and would plan an optimal route towards it. We might expect that a budget of 2000 should solve the map the quickest, however this is not the case. When doing runs with budget 2000, Pacman solves the map the most efficient, however the program starts slowing down when using higher budgets (Lots of nodes generated, from the results!). From our results, budgets of 100 and 1000 seem to work the best, solving the level in a relatively short amount of time.

Good and Bad behaviour of my Algorithm

Generally the algorithm, with an appropriate budget, performs as you'd expect. If you can see a path that leads towards more food, Pacman would usually take that route. If you see ghosts coming towards you, Pacman would narrowly squeeze in an escape route, while also prioritising finding food pellets. However, there are still some sub-optimal behaviours that Pacman often does.

Sometimes, Pacman will "stutter step" while walking in a line of food. This is possibly due to the heuristic function and the rewards associated with each outcome. One possible reason I can think of is that the heuristic function doesn't punish Pacman hard enough for dying (Only -10). A possible scenario is this: Pacman could go one path which has much less food, but is much safer, or a path with MUCH more food and ghosts present. Due to the heuristic function, sometimes Pacman would tend towards the safer route, however on the next move, a ghost might start walking away, which would make Pacman now tend towards the route with more food but is more dangerous. While this change in decision might be good for choosing the move with the most points, ultimately it would be inefficient in solving the level as we are sometimes stutter stepping.

Another interesting mechanic is the way that the ghosts move. In the pacman.c file, it seems that the way we generate "simulations" of game states and the way that the game state is actually carried out is slightly different. More specifically, a portion of the ghost's movements are random, thus we can generate simulations of what could happen, but the actual outcome could be completely random due to the randomness in the ghost's movements. Basically, sometimes we are predicting the ghosts' movements incorrectly, which would lead us to incorrect decisions. This would sometimes cause Pacman to run head on into a ghost and die unexpectedly, however this is one of the rarer cases. Another result of this is that sometimes Pacman wouldn't chase after ghosts when he is invincible. If you placed Pacman in a map with only a ghost chasing it, it would run into the ghost eventually due to the way game states are generated and carried out.

Also, on lower budgets (10), if there are food pellets extremely far way, Pacman is unable to locate these food pellets. A consequence of this is if there are no ghosts nearby, Pacman would be indifferent between its moves and now move randomly. This causes Pacman to essentially be stuck in some corners of the map and not being able to progress, until a ghost starts chasing it out. This behaviour is expected however, as with lower budgets we are unable to extend our game tree to locate food from very far away.

Optimisation

There are a couple of optimisations that I implemented into the baseline ai of the assignment. These optimisations turned out to be extremely beneficial in the time it took solving the levels, and also reducing some unwanted behaviour.

The first optimisation was inspired by the fact that even on budgets of 1000+, sometimes Pacman was unable to locate food that was relatively close to it. In the baseline algorithm, the graph created considers all possible moves from every state of the game generated. However, a large majority of the nodes created (budget) would be spent on pointless, redundant moves. For example, the algorithm will contain paths such as: up -> down -> up -> down -> left -> right -> left -> right… etc. These moves would be pointless to consider, as this whole path would just take Pacman back to its original spot, and not help Pacman with the progression of the level. Thus, the budget given would be mostly wasted on these paths generated. It would also make the "range" of Pacman somewhere near Log of base 4 of the budget, as the algorithm's priority queue chooses the depth of the node to be the priority. Essentially, the search algorithm would be doing a breadth-first-search in determining which path would generate the most points as it would be considering the "shallowest" node to be the next node in the path considered, however the range of this search would be limited.

What I changed about this is to only consider nodes that haven't been "visited" before. Of course, it would not be of much use to consider if every game state is exactly the same as another, because there are many different organisations of the ghost's locations. Rather, I chose to compare nodes by the location of Pacman, and how many food pellets there are left. This would make it so that moves such as left->right would not be considered, as not only would Pacman be returning to its original location, but the number of food pellets would remain the same. This caused moves that did not help towards finding new pellets to not be considered, and to only make memory for nodes that actually would progress Pacman in solving the map. A result that suggests that our "range" of food detection has increased is the maximum depth of our graph. With this implementation, the maximum depth of our graph almost doubles for each budget compared to the baseline algorithm of the assignment, essentially allowing Pacman to "smell" food from twice as far away!

The second issue encountered was to do with running the program on average propagation. The problem was mentioned in the discussion forums and was proven that Pacman, while on average propagation, would often oscillate between decisions, when there was a clear decision to make, or if there were multiple paths with many food pellets down the road, but not immediately. The problem arises from the way the algorithm of average propagation, where the algorithm would first tell us to move left, then the next frame the algorithm would tell us to move right, causing an oscillation. This was due to the calculations that were involved in average propagation, as well as the weight placed on deeper moves down the graph. While in an oscillation, the only way to escape the oscillation is if a ghost comes along to scare us away.

A simple optimisation that helped solve this issue is to reward an extra value point to the move that next contains a food pellet. This persuades Pacman to take the route that directly has food in the next move. In the calculations involving oscillations, it seemed that the difference between the moves was very small (< 0.05), therefore if we increase the value of any move by 1, it would give Pacman a clear, decisive move to go next. Now, if Pacman is starting a trail of food pellets, it would follow through with that, as for every move, there is an adjacent square that would have a food pellet and thus that move would have an extra point in value. Hence, this would stop the oscillating behaviour, as it would mitigate the calculation "errors" inside of the average propagation algorithm. Of course, if we have a food trail, we want Pacman to eat all the food along that path, in most cases.