Fast A* Heuristics for Solving the Travelling Salesman Problem

Team 3

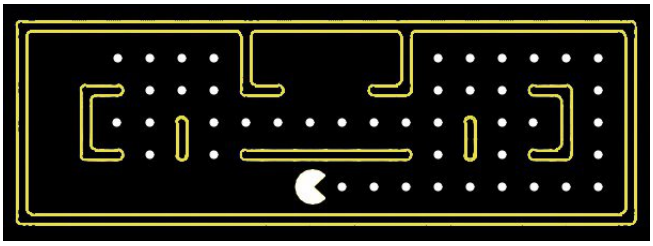

Fig. 1. Example Pac-man grid with no ghosts.

I. Problem

Within the Artificial Intelligence community, there is a great need for fast and accurate graph traversal algorithms, specifically those that find a path from a start to goal with minimum cost. Real world applications include streamlining logistics for package delivery services and optimizing the layout of computer chips. Games such as chess and sudoku, and the Google AI Challenge are popular testbeds for new AI algorithms and the development of specialized heuristics. Some of these games are represented as graphs, with the optimal solution being the lowest cost path to a winning node. Finding the optimal solution to a game represented in a graph, such as Pac-Man, can be modeled as an instance of the Euclidean Traveling Salesman Problem because the protagonist must traverse every cell in a maze at least once. The Traveling Salesman based Pac-Man problem is very useful in real life for cities like New York where the entire city is made of rectangular blocks much like the world of Pac-Man. The Traveling Salesman Problem is NP-Complete and the best brute force solution is unrealistic in practice taking O(n!) time. We will attempt to improve upon the state of the art approximation algorithms to produce a near-optimal solution while taking only polynomial time.

II. Related Work

The Traveling Salesman Problem is a well-known NP-Complete graph traversal problem. The A* search algorithm was first proposed in 1968 by Hart et. al. [1] as an algorithm to find minimum cost paths in graphs. Using a clever heuristic, A* is capable of very closely approximating the true solution to the Traveling Salesman Problem [2]. The customization of the heuristic allows A* to function well in many different situations [3]. The continuing work dedicated to improving A* attempts to improve heuristics for specific applications. The Berkeley Pac-Man framework was created as an academic testbed for teaching and algorithm development [4] It provides variants of the traditional Pac-Man arcade game to test and easily compare solutions created by different search algorithms. State of the art heuristics approximate the solution to Pac-Man without Ghosts within 1% of the true value in O(nlog(n)) [5]. While this is very close to exact, graphs often comprise millions of nodes, so even a quarter percent improvement in solution approximation or a fractional reduction in computational complexity is significant.

III. Approach

A. Framework

We used the University of California Berkeley's Pac-Man framework implemented in Python to test our approach. The Pac-Man projects were developed for UC Berkeley's introductory artificial intelligence course. They allow an array of AI techniques to be tested using a common framework. However, these projects do not focus on building AI for video games. Instead, this platform serves as an excellent test-bed for graph search algorithms. The Pac-Man framework has mazes of three different sizes. The small maze is a 20 by 8 grid, the medium maze is a 34 by 16 grid and the big maze is a 35 by 35 grid. In the problem where Pac-man has to find just one dot, Pac-man and the dot are usually placed on opposite corners. Other variations have a combinations of dots and open spaces to elicit specific movements through the maze dependent on the search techniques. It must be noted that our work involves only non-adversarial search, so the grids we use do not have ghosts to actively avoid.

B. Single target search

We started off by implementing deterministic graph search techniques on the Pac-Man framework. The methods we wrote were depth-first search, breadth-first search and uniform-cost search. Next, we implemented A* with a null-heuristic (Dijkstra's algorithm) to use this as a baseline for the other heuristics we develop. The run time of the A* algorithm depends entirely on the tightness of the heuristic, so the main chunk of our work was developing a heuristic that improves on the current state of the art with respect to approximation and compute cycles required. This is accomplished mainly by minimizing the number of nodes examined during the search. Our implementation minimizes the time taken to generate the path cost from the start to the current node by using dynamic programming with memoization to keep a table of previously computed paths to visited nodes. When expanding new nodes, we only have to add the parent-child cost to the existing lowest-cost parent path. For A*, the heuristics that we worked

on were the Manhattan distance, the Euclidean distance, and Chebyshev distance. The results produced by these heuristics are benchmarked against the null heuristic and the other search algorithms we implemented.

C. Complete traversal

To solve the Traveling Salesman Problem efficiently we implemented graph search methods that would lead to Pac-Man grabbing every dot on the grid. Once again we used an A* based approach along with a variety of heuristics written in Python for use in the Pac-Man framework and benchmarked them against the results of the null heuristic. An important part of this task was to make sure that our heuristics were both admissible and monotonically increasing. Manhattan and Euclidean distances are known to be admissible. The Chebyshev's heuristic is calculated as the diagonal distance from the source cell to the target's row or column plus the distance from this point to the target cell. So, effectively the heuristic based on the assumption that the diagonal path to the target cell's row or column and then from that point to the target cell. The heuristic chooses the distance based on whichever of these two(the diagonal to the row or to the column) gives a higher value. This is admissible because the Chebyshev distance can be proved to be always smaller than the Manhattan distance using geometry. Since, the Manhattan distance is known to be an admissible heuristic, the Chebyshev distance is also an admissible heuristic. Once we ensured that this was true, we were able to successfully benchmark the various available heuristics to decide which is best. The next section discusses the results of our approach.

IV. Evaluation

We used the included Berkeley Pac-Man map configurations containing no ghosts to verify the execution time and evaluate performance comparing the various solutions. Map sizes used are the small, medium, and large grids with static wall configurations. We tested our heuristic against the other mentioned admissible A* Traveling Salesman Problem heuristics as well as the basic search algorithms we implemented. The final score of the game is used as a comparison metric for how close of an approximation our heuristic is to the true solution, determined by the breadth-first search in our tests. The final score is a combination of the total time taken for the method to reach completion and the number of dots on the grid (assuming each agent acquired every dot). This is a valid comparison because each heuristic tested is admissible, so the end score is just a function of how efficiently each heuristic can reach each piece of food in a given map. We also compare the number of nodes expanded during the search phase to compare the search efficiency because all of these searches are performed before the Pac-Man moves in the grid. Fewer nodes expanded over the course of the search translates to a solution that takes less time to compute. The maximum number of nodes in the largest grid is (35x35=1225)-walls. The framework provides a very useful tool to help visualize the search paths in the context of the game. We included some examples of this in Fig. 2-5 that include all 3 sizes of grids. The shading shows the nodes that were explored in order from lightest (earliest) to darkest (latest). The last node expanded is the one containing the white dot. Any locations that are still black or darker than the background of the white dot were not explored in the search.

	Maze Size					
	Small		Medium		Large	
Search Type	S	NE	S	NE	S	NE
BFS	491	92	442	273	300	1150
DFS	461	129	380	311	300	1000
UCS	481	53	358	166	300	435
A*-null	491	93	442	274	300	619
A*-Chebyshev	491	57	442	229	300	565
A*-Euclidean	491	57	442	229	300	553
A*-Manhattan	491	53	442	221	300	538

TABLE I
SCORE(S) AND NUMBER OF NODES EXPANDED(NE) FOR EACH SEARCH STRATEGY IN SMALL, MEDIUM, AND LAGE MAZES

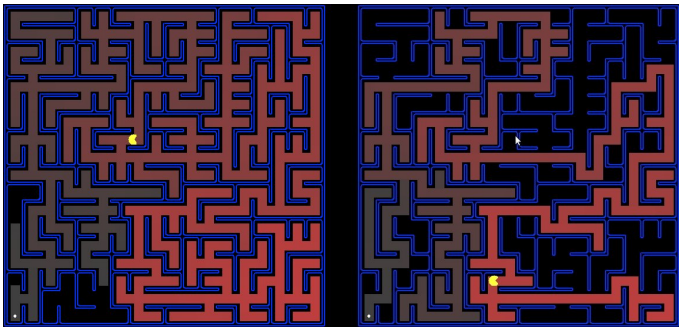

Fig. 2. Resultant path followed by Pac-Man using breadth-first search (left) and depth-first search (right). Nodes explored are colored lighter the closer they are to the start. Optimal path is a subset of the colored nodes.

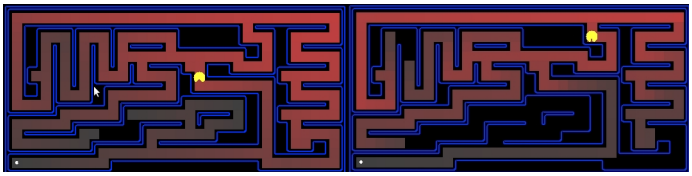

Fig. 3. Resultant path followed by Pac-Man using A* with null heuristic (left) and A* with Manhattan heuristic (right). Nodes explored are colored lighter the closer they are to the start. Both options generate the same optimal path, but the Manhattan heuristic expands fewer nodes.

Fig. 4. Resultant path followed by Pac-Man using the A* with null heuristic (left) and A* with Manhattan heuristic (right). Nodes explored are colored lighter the closer they are from the start.

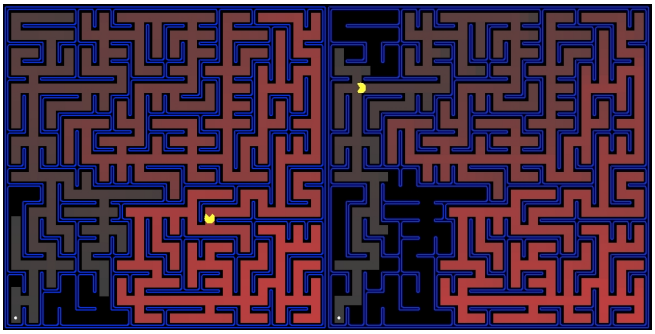

Fig. 5. Resultant path followed by Pac-Man using the A* with null heuristic (left) and A* with Manhattan heuristic (right). Nodes explored are colored lighter the closer they are from the start.

V. Discussion

Visualizing the search results in Fig. 2, breadth-first search obviously expands the most nodes, and depth-first search gets lucky on the big maze because the evaluation policy for children is based on their move direction ordered: South, West, North, East. Looking at Table 1, it is obvious that A* with any admissible heuristic produces a solution while expanding fewer nodes than any deterministic method tested. While the framework gives an intuitive visualization of the search strategies, a major limitation is the size of the largest grid. Ideally, there would be grids available with between 100,000 and 1,000,000 nodes, but visualization would be problematic and is not realistic for a game of Pac-Man. These larger grids would allow true comparisons to more established heuristics, and also better comparisons of the quality of the solutions generated as the search space grows. With so few nodes in our grids, all solutions can quickly generate the ideal solution as seen in the end-game scores being equal for all A* solutions in each map. The number of nodes expanded varies among heuristics chosen, but not significantly. Only testing on one static maze of each size limits the ability to discern heuristic quality versus how well the heuristic matched the selected layout. The baseline comparison for score is the breadth-first search, which guarantees an optimal path solution. All of the A* methods produce a score equal to breadth-first search. It is obvious that A* with any admissible heuristic generates a better solution than depth-first search, breadth-first-search, and uniform cost search, but we cannot quantitatively say how much better within the Pac-Man Framework. The improved solution quality to go from the other search solutions to A* does require more memory. The memory is used to generate and maintain the current path costs as well as the heuristic costs. Within A* heuristics, all require the same amount of memory within an order of magnitude around the number of nodes when implemented efficiently in addition to taking polynomial time. Based on our evaluations, the Manhattan distance is the best heuristic for use in applying the Traveling Salesman Problem to Pac-Man because it is able to find the optimal path while expanded the fewest nodes. Heuristics are, by definition, application specific, so the applicability of these findings is limited to Traveling Salesman Problems working on grids with constrained paths and simple movement sets.

References

[1] P. Hart, N. Nilsson, B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," Systems Science and Cybernetics, IEEE Transactions on , Vol.4, Vo.2, pp.100-107, July 1968

[2] S. Lin and B. W. Kernighan , "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," Operations Research, Vol. 21, No. 2, pp.498-516, March-April, 1973

[3] C. Nilsson. "Heuristics for the traveling salesman problem". Tech. Report, Linkping University, Sweden, 2003

[4] J. DeNero and D. Klein, Teaching Introductory Artificial Intelligence with Pacman, Symposium on Educational Advances in Artificial Intelligence (EAAI), 2010

[5] Sanjeev Arora. "Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and other Geometric Problems," Journal of the ACM, Vol.45, Issue 5, pp.753-782. September 1998