Cachex - $A*$ Search Algorithm

Sean Conlon 129886 sconlon@student.unimelb.edu
Chow Wai Kin Wilkin 1068161 cwilkin@student.unimelb.edu

The first section of the project involving the fictitious 'Cachex' game is the implementation of the famed A Search Algorithm, to find an optimal traversal throughout a given board. For now, the state of the board is constant.*

Section 1. Implementation & Analysis of $A*$

The implementation of the $A*$ search algorithm closely follows the pseudo-code described in the algorithms wikipedia page. There are two main data structure considerations when implementing the algorithm

1. Storing and accessing node n's path value $g(n)$, and total distance value estimate $f(n)$.
2. The priority queue, for popping the node with the smallest current distance estimate.

The first of the two problems can be solved easily and efficiently with a hash map, or a Python `dict`. In our implementation, we store these values in the dictionaries `g_score` and `f_score` respectively. The use of a dictionary results in efficient $O(1)$ insertion, update and access for when we require these values for a particular node throughout the algorithm. On a slightly technical note, the algorithm requires an initialisation of $f(v) = g(v) = \infty \quad \forall v \in T$, where T is the search tree. We ensure this by instancing the `g_score` and `f_score` dictionaries as `defaultdict`'s with Pythons equivalent of infinity, `float('inf')`.

To address the second consideration, a Min Heap data structure was chosen to maintain the priority queue of node $f(\cdot)$ values. Python provides a robust implementation of a Min Heap with the `heapq` library. As we require the queue to be maintained on priority of smallest $f(\cdot)$ value[1], we add Tuples of `(f(n), n)` to the heap. In terms of complexity, on each iteration of the algorithm, we pop and potentially insert to the heap, each operation requires $O(\log k)$ complexity, where k is the amount of elements in the heap.

In general, the time complexity of A^* search is dependant on the choice of heuristic, and how well it prunes away branches of the search tree. In a worst-scenario, the heuristic provides no insight and results in $O(b^d)$ time complexity, with b and d representing branching factor of the search space and depth of goal node respectively. As we store all discovered nodes in a Min Heap data structure, this means that the space complexity is similar to that of depth-first search, which is $O(b^m)$.

Section 2. Choice of Heuristic Function

As input to the algorithm, $A*$ search requires an admissible heuristic function. Recall that a heuristic function h is admissible if $h(v) \leq h^*(v) \quad \forall v$ where h^* is the true distance. For our implementation, we chose the L_1 metric, also known as *Manhattan distance*. The heuristic estimates distance between a potential node v and goal node G as

$$\begin{aligned} h(v) &= \sum_{i=1}^{2} |v_i - G_i| \\ &= |v_1 - G_1| + |v_2 - G_2| \end{aligned}$$

As the L_1 function performs a simple and bounded computation of a given node, this adds an additional $O(1)$ work each time it is computed. The L_1 metric is clearly admissible, as the optimal distance

[1] That is, the node n that produces smallest `f(n)` is placed at the front of the queue.

between any two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is given by the path of traversing along the y axis and then x axis resulting in a path length of

$$|x_1 - x_2| + |y_1 - y_2| = L_1(p_1, p_2)$$

This is never forms an overestimate between p_1, p_2 as there may be cells occupied along the way, thereby resulting the *true*[2] distance between larger.

Section 3. Extending the Problem

We now consider an extension of the problem, in which we can include already colored board pieces to our path. We can extend our algorithm by making the cost to reach any compatibly colored node (from an adjacent position) 0. This would encourage $A*$ search, or any other Best-First Search algorithm to explore these nodes and thereby likely to incorporate them into a solution[3].

Unfortunately, extending the problem in such a way renders the L_1 distance metric as a no-longer admissible heuristic. To illustrate why L_1 may result in an overestimate of the distance, consider the case that we wish to perform a path find between node $(0, 0)$ and goal node $G = (0, 3)$. Under the L_1 heuristic, we have the distance estimate for $(0, 1)$

$$h(n) = |0 - 0| + |1 - 3| = 2$$

However, suppose that $(0, 2)$ is one such colored cell that can be included into the solution with cost 0, then this would make the *true* shortest distance between $(0, 1)$ and $(0, 3)$ as 1; thus the heuristic overestimated the true cost and is therefore not admissible.

[2] As in true, optimal distance between p_1 and p_2

[3] In the case of $A*$ search, they will definitely be included into the solution when it is optimal to do so