COMP30024 – Artificial Intelligence

Project Part A - Searching

Team: ELDEN_KING
Jiahao Chen, Zhiquan Lai

I. Implementation of A* Search

A* search used an evaluation function so that if the function is monotonic, the solution must have lower costs than all unexpanded nodes. The evaluation function (f(n)) consists of two parts, the cost so far to reach the current node (g(n)) plus the estimated cost to goal from the current node (h(n)). The latter one is an admissible heuristic function, which will be detailedly discussed in the next part. The algorithm expands the node with lowest f(n) each time. To help performing node expansion, a priority queue is applied so that the node with lowest f(n) is guaranteed to be placed at the top. Figure 1 explains our implementation details.

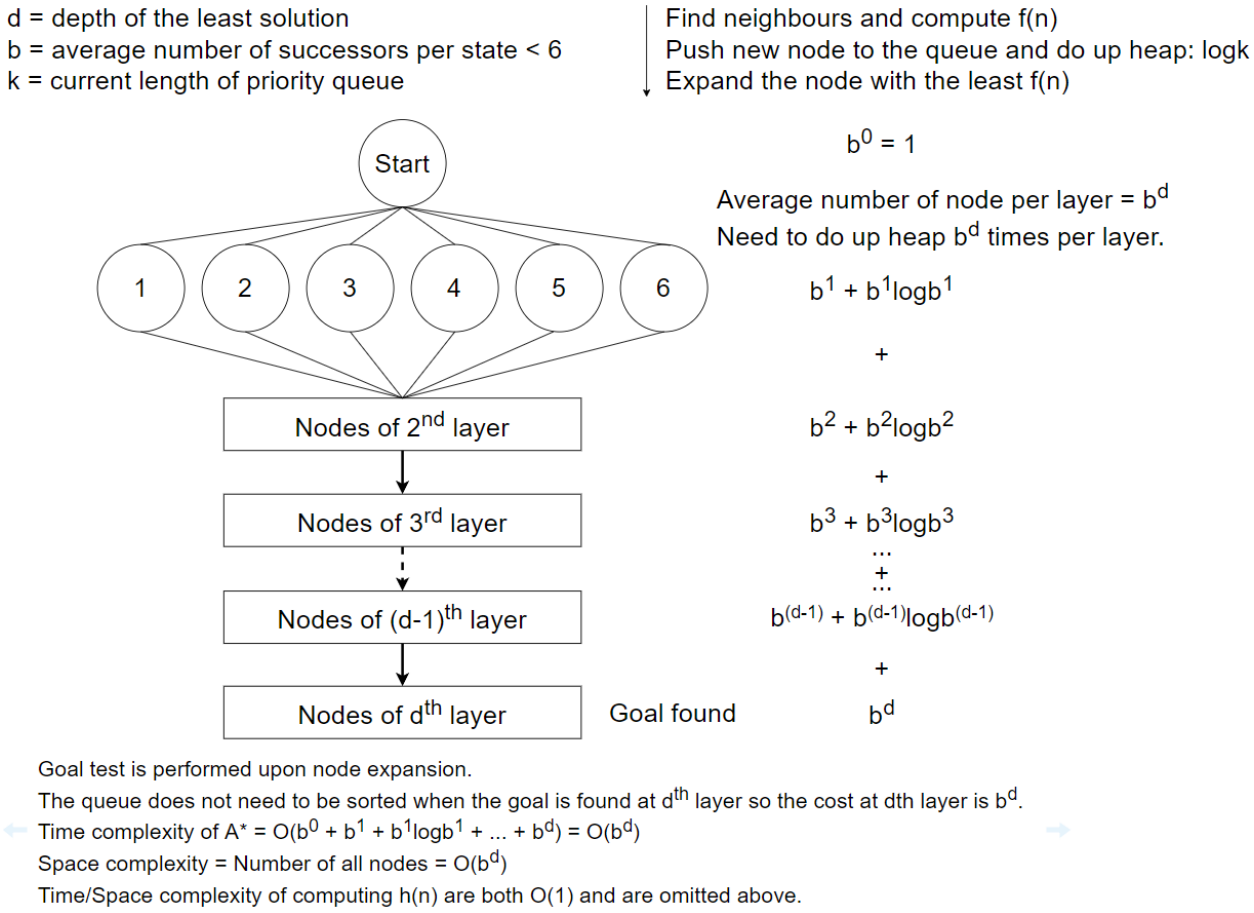

Figure 1: Implementation & Analysis of A*

II. Heuristic Function

Manhattan distance is applied as our heuristic function. By definition, a heuristic function h(n) is admissable if h(n) is not greater than the true cost from n. Suppose the shape of the board is quadrilateral, the length of the nearest path between two cells equals their Manhattan distance as it computes the sum of lengths of the projections of the line segment between the cells onto the coordinate axes.

In the case of a hexagonal board, cells in diagonal directions also have distances of 1. We improved the algorithm so that the nearest path between two cells on a hexagonal board equals their Manhattan distance as well. Therefore, h(n) never overestimates the cost since some cells are occupied, which proves it is not greater than the true cost, namely admissable.

Euclidean distance is also considered, but it is less than Manhattan distance as figure 2 shows. Since the larger the admissible function is, the better it performs, Manhattan distance becomes the optimal choice. Time and space complexity of h(n) are both O(1). The function only consists of basic math operations and stores coordinates of two cells.

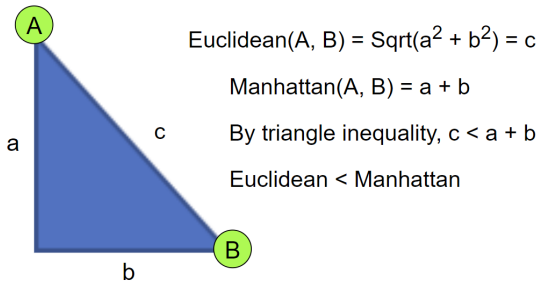

Figure 2: Euclidean < Manhattan

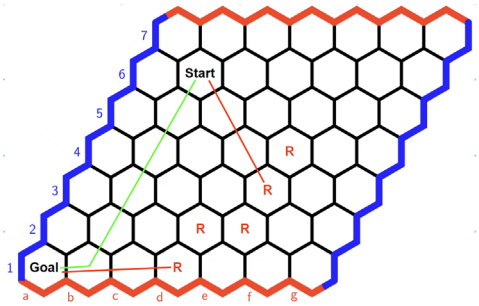

Figure 3: Example of suboptimal path

III. Challenge

In the case that existing pieces can be made use of, Manhattan distance overestimates the true cost because some moves cost 0 so that it is no longer admissible. At first, we intuitively set cost of moving to existing pieces as 0. However, as figure 3 depicts, this can result to a suboptimal path, namely the green one. The existing pieces are not captured so that the algorithm cannot find the optimal path which is the red one.

To solve this problem, we designed a new heuristic function h(n) = Manhattan distance – number of existing pieces and it is forced to be bounded below by 0. Since it takes the existing pieces into consideration, it cannot be greater than the true value so that it is admissible. When there are no pieces can be captured, it works the same as the original one. If all the nodes have h(n) equals 0, it will become a uniform-cost search, and the cost of capturing an existing piece is 0. As a result, the algorithm will make fully use of the pieces to find the optimal path.