Experimental report for the 2021 COM1005 Assignment: The 8-puzzle Problem*

Muhammad Kamaludin

April 24, 2022

I declare that the answers provided in this submission are entirely my own work. I have not discussed the contents of this assessment with anyone else (except for Heidi Christensen or COM1005 GTAs for the purpose of clarification), either in person or via electronic communication. I will not share the assignment sheet with anyone. I will not discuss the tasks with anyone until after the final module results are released.

1 Descriptions of my breadth-first and A* implementations

1.1 Breadth-first implementation

Breadth-first search is essentially a type of uninformed search algorithm which equally searches every nodes at each depth before proceeding to the subsequent depth until the goal is reached. This algorithm works by traversing the higher depth's nodes first and add those nodes to the open list.

Open list stores the unexplored nodes and involves in the node selection. In this part, the open list will be implemented as a queue as in Figure 1 which acts on the first-in-first-out basis to achieve the intended behaviour.

```
private void breadthFirst() {
    currentNode = (SearchNode) open.get(0); // first node on open
    open.remove(0);
}
```

Figure 1: Queue implementation of open list

*Link to personal GitHub *private* https://github.com/aimanarifi/COM1005.git

This experiment uses three fixed puzzle patterns as shown in Figure 3, and additional three randomly generated puzzle patterns of 2022 seed; with three different difficulties which are 6, 9, and 12 as shown in Figure 4. The efficiencies of this algorithm towards each puzzle patterns are recorded and the average efficiency is calculated.

1.2 A* implementation

A* is a search algorithm which includes heuristics to help determines the next nodes. This implementation requires the following variables to execute:

- Local cost
- Total global cost
- Estimated remaining cost
- Estimated total cost

In the 8-puzzle problem, the local cost is the number of tile movement at each node which is 1 since only 1 tile can move to the empty space at a time. The total global cost is the accumulated cost from the starting node to the current node. The estimated total cost is the sum of estimated remaining cost and total global cost.

In this implementation, the open list will not act like any elementary data structure and not heavily influences the node selection. However, the algorithm will select a node with the least estimated total costs of all nodes in the open list as in Figure 2.

```
  private void AStar(){

  Iterator i = open.iterator();
  SearchNode minCostNode=(SearchNode) i.next();
  for (;i.hasNext();){
    SearchNode n=(SearchNode) i.next();
    if (n.getestTotalCost()<minCostNode.getestTotalCost()){
      minCostNode=n;};
    }

  currentNode=minCostNode;
  open.remove(minCostNode);
}
```

Figure 2: Node selection by the least estimated total cost

In addition, the experiment will use two approaches of calculating the estimated remaining cost namely *Manhattan* and *Hamming*. From this, the

experiment can show whether or not a determination of heuristic affects the efficiency of an algorithm.

- *Manhattan* - Sum of distance of individual number to their correct place
- *Hamming* - Number of tiles that are out of place

This approach also uses the same puzzle patterns as the Breadth-first implementation which are shown in Figure 3 and Figure 4. The efficiencies of this algorithm towards each puzzle patterns are recorded and the average efficiency is calculated.

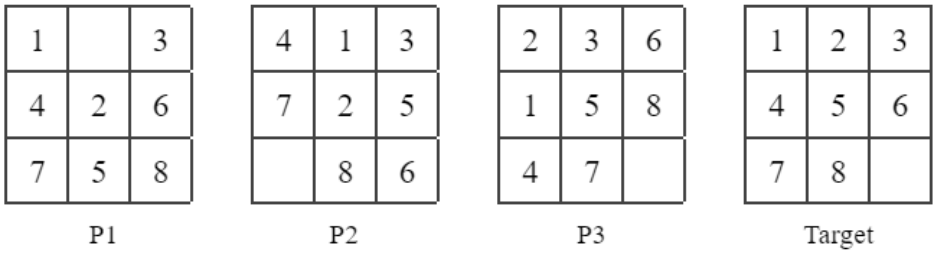

Figure 3: Fixed puzzle patterns as in assignment sheet

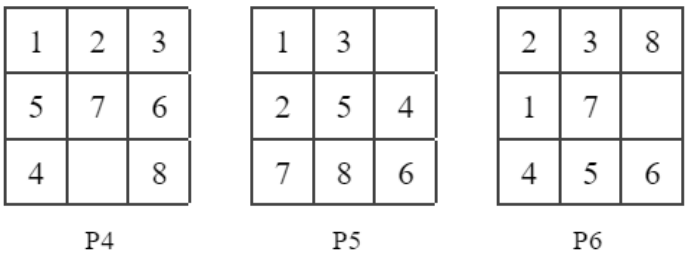

Figure 4: Randomly generated puzzle patterns of 2022 seed, and difficulty 6, 9, and 12

2 Results of assessing efficiency for the two search algorithms

Efficiency of the three algorithms on each puzzle patterns are recorded in percentage and tabulated in Table 1.Breadth-first < A*Manhattan <

A*Hamming in term of average efficiency. The efficiency ranging from 0.16% to 36.36% for Breadth-first, 4.87% to 100% for A*Manhattan, and 2.42% to 100% for A*Hamming. It is clear for that puzzle patterns that the efficiency of Breadth-first algorithm is lower than both A* algorithm. Thus, this finding supports the hypothesis, *A* is more efficient than breath-first*

Comparing A*Manhattan and A*Hamming for puzzle P4, P5, and P6 (puzzle with known increasing difficulty), the result suggests that A*Manhattan is more efficient than A*Hamming. However, the result shows that the efficiency of both A* approaches do not increase with the difficulty as the efficiency plummets from P4 to P5. Hence, it does not coherent with the hypothesis thus can't fully support the hypothesis, *the efficiency gain is greater the more difficult the problem and the closer the the estimates are to the true cost.* Likewise, there is no clear indication of which A* approaches is the best since A*Manhattan less efficient than A*Hamming for puzzle P1 to P3, but more efficient from P4 to P6.

Puzzle	Breadth-first	A* Manhattan	A* Hamming
P1	36.36%	100.00%	100.00%
P2	12.28%	87.50%	100.00%
P3	5.45%	75.00%	90.00%
P4	14.63%	75.00%	75.00%
P5	0.16%	4.87%	2.42%
P6	0.24%	10.81%	3.64%
Average	11.52%	58.86%	61.84%

Table 1: Efficiencies of algorithm on each puzzle patterns

3 Conclusions

Overall, the Breadth-first and A* approaches result in different efficiencies on the 8-puzzle problem. To be specific, the experiment supports one of the hypothesis, *A* is more efficient than Breadth-first.* This might be due to the heuristic which offers additional weights for a better decision making in an algorithm.

Apart from that, the findings failed to support the hypothesis, *the efficiency gain is greater the more difficult the problem and the closer the the estimates are to the true cost.* There are few suggestions that might be useful to overcome the incoherent result in the future experiment which are:

- Increase sample size by adding more puzzle patterns

Adding more puzzle patterns potentially increases the accuracy of the result and reduces any potential bias in the experiment.

- Use randomly-generated puzzle only

  Mixing fixed and randomly-generated puzzled could lead to inconsistency since the fixed puzzle's seed and difficulty remain unknown.

- Use puzzle with different seeds

  The sample size can increase by testing puzzle of other seeds. Besides, it might produce new insight such as *Does the hypothesis only true for specific seed?*, etc.