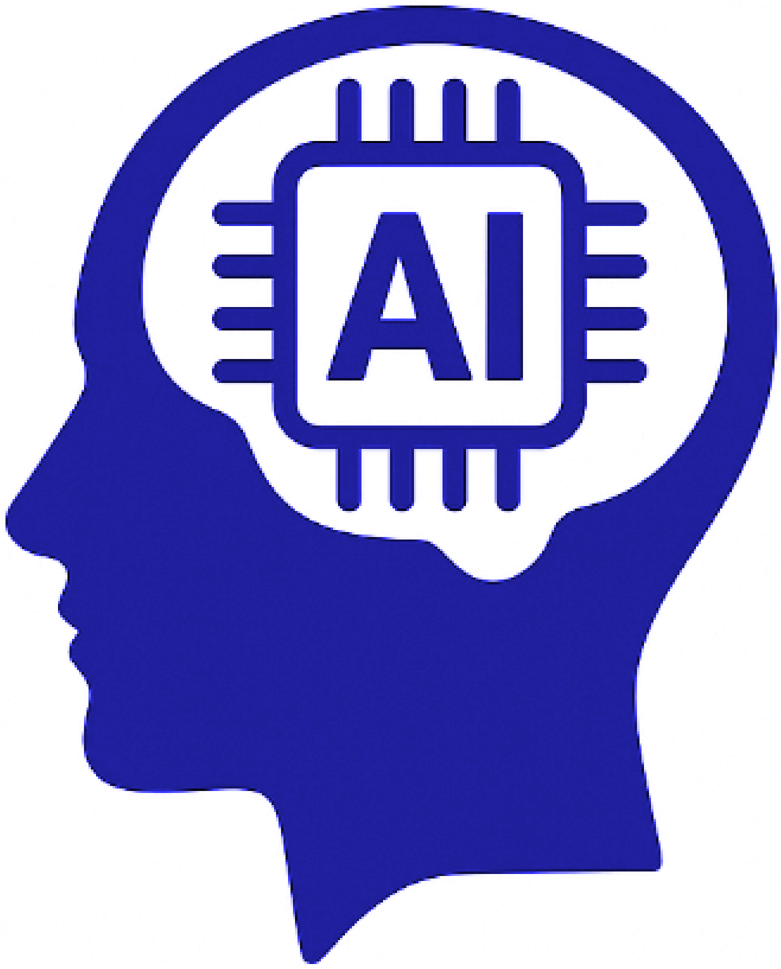
<u>Practice 2 - Artificial Intelligence</u>

Problem solving and searching: Informed search.



Author: Fernando Pastor Peralta (897113)

Delivery date: 21/10/2025

Group: Tuesday B, 4:00 PM - 7:00 PM

Computer Engineering - University of Zaragoza, Academic Year 2025-2026

Index

Introduction:

The objective of this lab is to become familiar with Java code for solving informed search problems and to distinguish their main characteristics. Specifically, we continue to focus on the 8-puzzle problem.

Blind search algorithms (BFS – Breadth-First Search and IDS – Iterative Deepening Search) were compared with the informed search algorithm A*, using two heuristics:

- **Misplaced tiles heuristic (h1):** Counts the number of tiles that are not in their goal position.

- **Manhattan heuristic (h2):** Sums the Manhattan distances of each tile to its goal position; the blank space is ignored.

Efficiency is measured using the number of generated nodes and the effective branching factor b*, computed by solving the equation N=b*(b*d-1)/(b*-1) using the bisection method. A total of 100 experiments were carried out with different depths, specifically with depths ddd from 2 to 24, generating 100 random initial states for each depth. It should also be noted that IDS is limited to depths greater than 10 due to its high time and space cost.

Methodology:

Implementation:

Our implementation is based on the Java AIMA code provided, which has been adapted and extended with new classes to solve the problem at hand.

We defined our test program to carry out the lab in the main method of the EightPuzzlePract2 class. This class has a structure similar to EightPuzzlePract1 from the first lab, especially in the use of the eightPuzzleSearch() and tipoSearch() methods to execute the different algorithms starting from a given initial state. It is also similar in how results are displayed on screen, following the same approach of printing a header and then filling each row using a dedicated function. Within the main method, several tasks are performed to solve the problem:

1. We initialize the goal state, defined in a new class EightPuzzleGoalTest2, which is a small modification of the original EightPuzzleGoalTest. This allows us both to modify the goal state (via the setGoalState() method) and to ensure that the heuristics take the defined goal state into account (via the getGoalState() method).

2. We configure the heuristics to work with a specific goal state. To do this, the original definitions of both heuristics were modified, and the classes ManhattanHeuristicFunction2 and MisplacedTileHeuristicFunction2 were created.

3. We generate a total of 100 random initial states for each depth:

    - We generate initial states using the `GenerateInitialEightPuzzleBoard` class, provided by the course instructors. Specifically, it contains the `EightPuzzleBoard random(int depth)` method, which generates an initial state that can be reached in at least `d` moves. This method has been modified because it only accepts moves if the hole can be moved to that position or if the resulting state has not yet been visited. This is good; however, if the current board reaches a state where all possible moves are invalid, the code gets stuck in a loop. For small depths (2-10), this is rare, but for greater depths, it is more likely to happen, hence the modification.

    - With these initial states, we call eightPuzzleSearch() with each algorithm; the result of the call represents the nodes generated by each of them. In each iteration, we will store the results of each algorithm using an array that functions as a summation.

4. We compute the average number of generated nodes.

5. We compute b* in each case using the method calcularBestrella(), which relies on the class Biseccion provided by the instructors. This class implements the metodoDeBiseccion() method, which solves the equation N=b*(b*d-1)/(b*-1) using the interval [1.0001, 10] with a tolerance of 0.001.

6. Finally, each row is printed with its corresponding format, and the process continues to the next iteration (if there are more depths to explore).

Analysis of results:

After running the algorithm multiple times, several conclusions can be drawn:

- The number of generated nodes grows exponentially with d, as expected in search trees of complexity O(b^d), where b is the branching factor and d is the depth.

- BFS and IDS scale worse than A*, confirming the efficiency of informed heuristics compared to uninformed algorithms.

- BFS reduces its b* from 2.37 to 1.54, showing a decrease in effective branching as depth increases, thanks to the pruning of invalid and duplicate states (graph search).

- The explosion in the number of nodes generated by IDS as depth increases is particularly notable (which is why it is a good idea to stop the algorithm early to avoid excessively long execution times). Additionally, its b* remains constant, which is consistent with expectations.

- A* is significantly more efficient than BFS and IDS. With the misplaced tiles heuristic (h1), better results are obtained both in terms of generated nodes and b*. On the other hand, the Manhattan heuristic (h2) performs even better, generating approximately ten times fewer nodes than h1 and achieving an even lower b*.

- The time required for the algorithms to finish increases considerably as depth increases.

- The number of generated nodes in our experiments is smaller because the random(d) method generates d steps without ensuring that the goal state is exactly at depth d. In contrast, the PDF problem statement guarantees that the goal state is at depth d.

The appendix includes an image showing the results after executing the program.

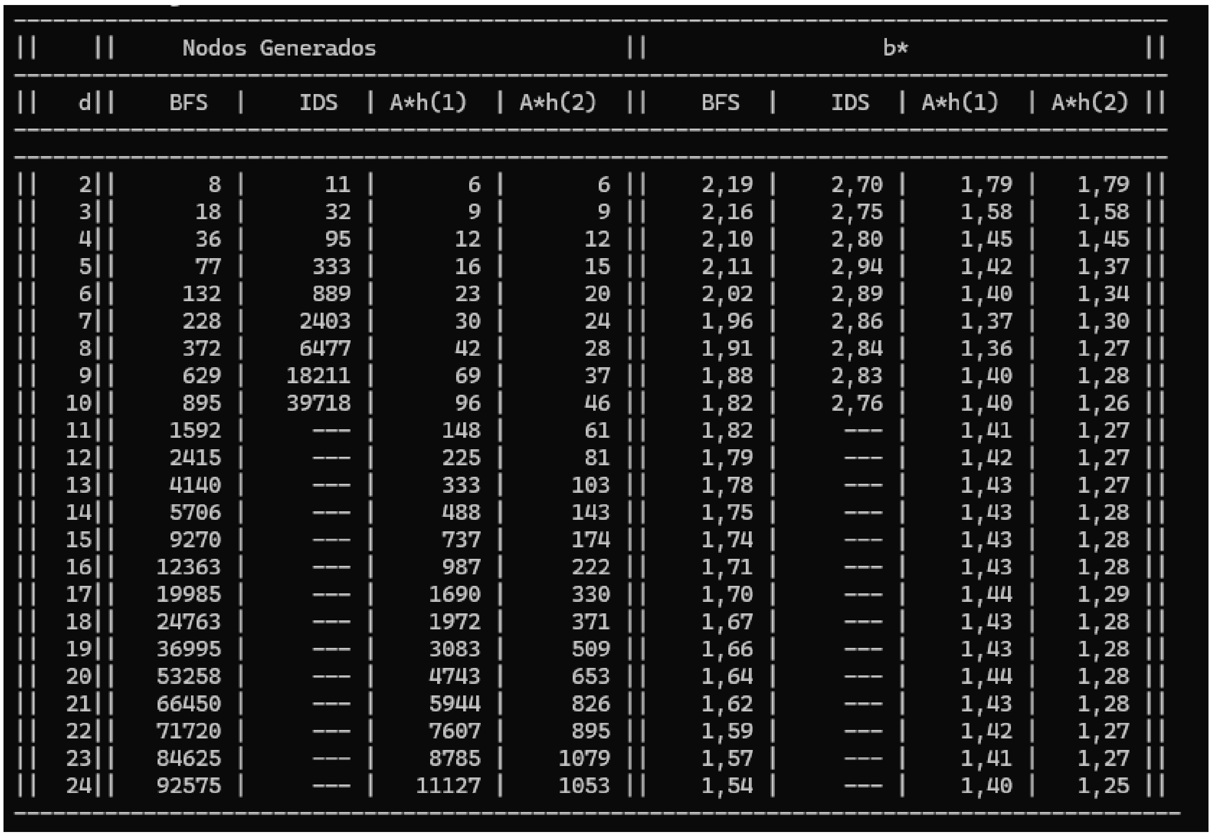
Conclusion:

We have gained an understanding of the efficiency of informed search algorithms compared to blind search algorithms in the 8-puzzle problem, confirming that the misplaced tiles (h1) and Manhattan (h2) heuristics drastically improve performance.

The results show exponential growth in the number of generated nodes with depth for BFS and IDS (the latter standing out due to its exponential explosion and constant b*). In contrast, A* is much more efficient, and specifically, h2 generates approximately ten times fewer nodes than h1, while also achieving a greater reduction in b*.

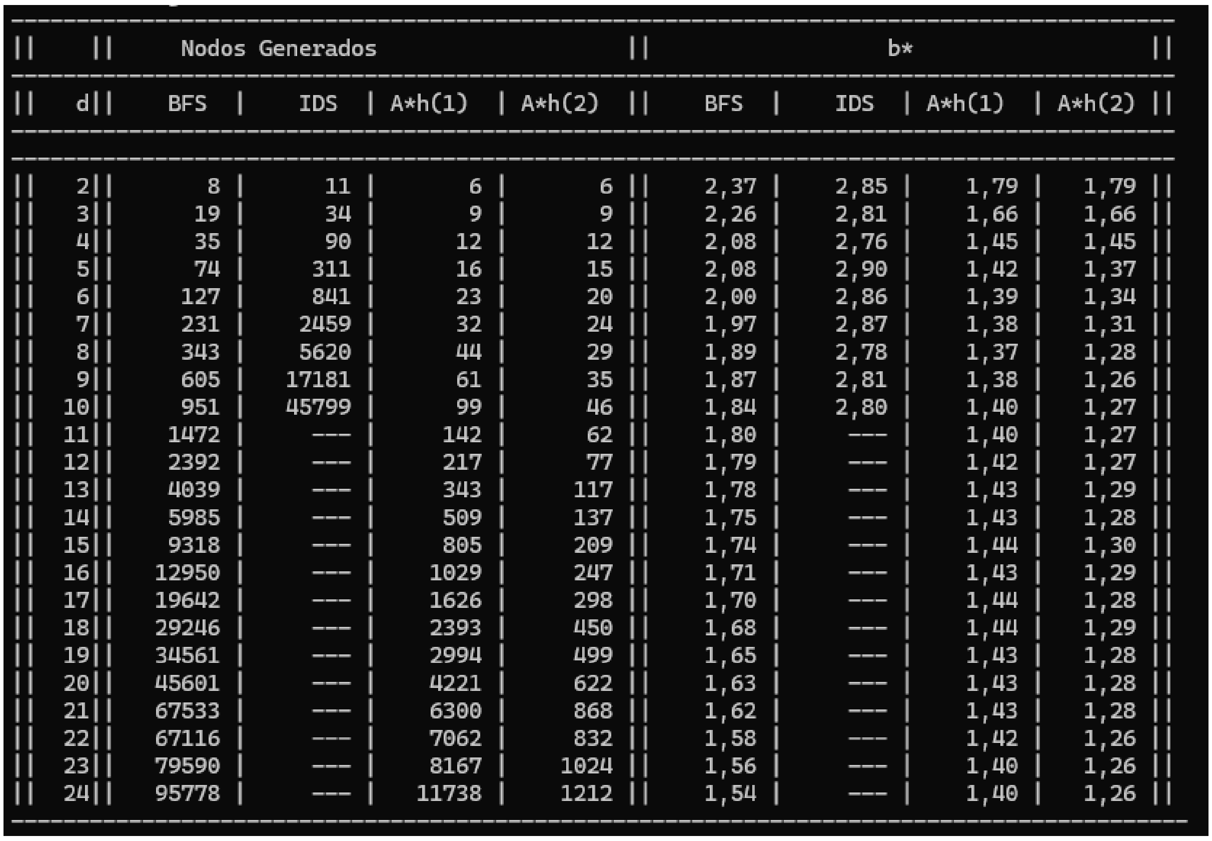
Appendix:

An example of the execution:



```
-----------------------------------------------------------------------------------------
||    ||     Nodos Generados                  ||               b*                     ||
-----------------------------------------------------------------------------------------
||   d||    BFS  |    IDS  | A*h(1) | A*h(2) ||    BFS  |    IDS  | A*h(1)  | A*h(2) ||
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
||   2||      8 |      11 |      6 |      6 ||    2,19 |    2,70 |    1,79 |   1,79 ||
||   3||     18 |      32 |      9 |      9 ||    2,16 |    2,75 |    1,58 |   1,58 ||
||   4||     36 |      95 |     12 |     12 ||    2,10 |    2,80 |    1,45 |   1,45 ||
||   5||     77 |     333 |     16 |     15 ||    2,11 |    2,94 |    1,42 |   1,37 ||
||   6||    132 |     889 |     23 |     20 ||    2,02 |    2,89 |    1,40 |   1,34 ||
||   7||    228 |    2403 |     30 |     24 ||    1,96 |    2,86 |    1,37 |   1,30 ||
||   8||    372 |    6477 |     42 |     28 ||    1,91 |    2,84 |    1,36 |   1,27 ||
||   9||    629 |   18211 |     69 |     37 ||    1,88 |    2,83 |    1,40 |   1,28 ||
||  10||    895 |   39718 |     96 |     46 ||    1,82 |    2,76 |    1,40 |   1,26 ||
||  11||   1592 |     --- |    148 |     61 ||    1,82 |     --- |    1,41 |   1,27 ||
||  12||   2415 |     --- |    225 |     81 ||    1,79 |     --- |    1,42 |   1,27 ||
||  13||   4140 |     --- |    333 |    103 ||    1,78 |     --- |    1,43 |   1,27 ||
||  14||   5706 |     --- |    488 |    143 ||    1,75 |     --- |    1,43 |   1,28 ||
||  15||   9270 |     --- |    737 |    174 ||    1,74 |     --- |    1,43 |   1,28 ||
||  16||  12363 |     --- |    987 |    222 ||    1,71 |     --- |    1,43 |   1,28 ||
||  17||  19985 |     --- |   1690 |    330 ||    1,70 |     --- |    1,44 |   1,29 ||
||  18||  24763 |     --- |   1972 |    371 ||    1,67 |     --- |    1,43 |   1,28 ||
||  19||  36995 |     --- |   3083 |    509 ||    1,66 |     --- |    1,43 |   1,28 ||
||  20||  53258 |     --- |   4743 |    653 ||    1,64 |     --- |    1,44 |   1,28 ||
||  21||  66450 |     --- |   5944 |    826 ||    1,62 |     --- |    1,43 |   1,28 ||
||  22||  71720 |     --- |   7607 |    895 ||    1,59 |     --- |    1,42 |   1,27 ||
||  23||  84625 |     --- |   8785 |   1079 ||    1,57 |     --- |    1,41 |   1,27 ||
||  24||  92575 |     --- |  11127 |   1053 ||    1,54 |     --- |    1,40 |   1,25 ||
-----------------------------------------------------------------------------------------
```

Another example:



```
-----------------------------------------------------------------------------------------
||    ||     Nodos Generados                  ||               b*                     ||
-----------------------------------------------------------------------------------------
||   d||    BFS  |    IDS  | A*h(1) | A*h(2) ||    BFS  |    IDS  | A*h(1)  | A*h(2) ||
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
||   2||      8 |      11 |      6 |      6 ||    2,37 |    2,85 |    1,79 |   1,79 ||
||   3||     19 |      34 |      9 |      9 ||    2,26 |    2,81 |    1,66 |   1,66 ||
||   4||     35 |      90 |     12 |     12 ||    2,08 |    2,76 |    1,45 |   1,45 ||
||   5||     74 |     311 |     16 |     15 ||    2,08 |    2,90 |    1,42 |   1,37 ||
||   6||    127 |     841 |     23 |     20 ||    2,00 |    2,86 |    1,39 |   1,34 ||
||   7||    231 |    2459 |     32 |     24 ||    1,97 |    2,87 |    1,38 |   1,31 ||
||   8||    343 |    5620 |     44 |     29 ||    1,89 |    2,78 |    1,37 |   1,28 ||
||   9||    605 |   17181 |     61 |     35 ||    1,87 |    2,81 |    1,38 |   1,26 ||
||  10||    951 |   45799 |     99 |     46 ||    1,84 |    2,80 |    1,40 |   1,27 ||
||  11||   1472 |     --- |    142 |     62 ||    1,80 |     --- |    1,40 |   1,27 ||
||  12||   2392 |     --- |    217 |     77 ||    1,79 |     --- |    1,42 |   1,27 ||
||  13||   4039 |     --- |    343 |    117 ||    1,78 |     --- |    1,43 |   1,29 ||
||  14||   5985 |     --- |    509 |    137 ||    1,75 |     --- |    1,43 |   1,28 ||
||  15||   9318 |     --- |    805 |    209 ||    1,74 |     --- |    1,44 |   1,30 ||
||  16||  12950 |     --- |   1029 |    247 ||    1,71 |     --- |    1,43 |   1,29 ||
||  17||  19642 |     --- |   1626 |    298 ||    1,70 |     --- |    1,44 |   1,28 ||
||  18||  29246 |     --- |   2393 |    450 ||    1,68 |     --- |    1,44 |   1,29 ||
||  19||  34561 |     --- |   2994 |    499 ||    1,65 |     --- |    1,43 |   1,28 ||
||  20||  45601 |     --- |   4221 |    622 ||    1,63 |     --- |    1,43 |   1,28 ||
||  21||  67533 |     --- |   6300 |    868 ||    1,62 |     --- |    1,43 |   1,28 ||
||  22||  67116 |     --- |   7062 |    832 ||    1,58 |     --- |    1,42 |   1,26 ||
||  23||  79590 |     --- |   8167 |   1024 ||    1,56 |     --- |    1,40 |   1,26 ||
||  24||  95778 |     --- |  11738 |   1212 ||    1,54 |     --- |    1,40 |   1,26 ||
-----------------------------------------------------------------------------------------
```