1. Introduction & Related Work

Heuristic search is a powerful problem-solving technique that utilizes educated guesses or heuristics to navigate a problem space and find a solution. This approach is widely used in the field of artificial intelligence and other related domains to tackle complex problems efficiently. Heuristic search algorithms such as beam search and A* are frequently utilized in pathfinding, planning, and scheduling tasks. Heuristics play a key role in these algorithms as they aid in estimating the cost of reaching a goal state and identifying promising directions to explore in the search space. Through my studies, I have gained knowledge about beam search, which is a variant of A* that operates at a limited depth and maintains a fixed-size set of promising states. I have also focused on A* search, which is a greedy algorithm that employs a heuristic function to estimate the cost of reaching a goal state. Although A* is often considered the most efficient heuristic search algorithm, it may perform slowly for problems with large search spaces. In my research, I delved into an extension of the A* algorithm, known as Anytime Weight A*(AWA*). In A*, the evaluation function is defined as $f(n) = h(n) + g(n)$, where $h(n)$ is the heuristic function that estimates the cost of the cheapest path from node n to the goal node, and $g(n)$ is the cost of the path from the start node to node n. In AWA*, on the other hand, the evaluation function is defined as $f'(n) = w*h(n) + g(n)$, where w is a weight parameter that determines the importance of the heuristic function $h(n)$ relative to the actual cost of the path $g(n)$.
Additionally, I drew upon other important concepts, such as graph theory, pathfinding, and search algorithms, to enrich my work.

2. Description of Work

I implemented the Anytime Weight A* algorithm on Korf's 100 puzzles to gain a better understanding of AWA* and gain a broader perspective. I tested the algorithm on a batch of 100 puzzles with different weights, including 3, 5, and 7. I chose not to run the algorithm with a weight of 1 as it is equivalent to running Weight A*. The code was developed using C++ for the AWA* algorithm and Python for visualization. I focused on how to store and display the results obtained from each run.

The AWA* algorithm initializes internal variables such as the closed list and the pool of nodes. The search method starts the search by initializing the start node and adding it to the open list. The algorithm then enters a loop that pops the next node from the open list, expands it, and adds its child nodes to the open list. The loop continues until either the goal node is found or the search time limit is reached.

Regarding computing resources, I used specifically algorithm AWA* for this project, but I did set memory and time limits on resources. Finally, I used the Python code to generate three scatter figures: solution quality, which is the quality of the solutions found by the algorithms, coverage is the fraction of instances that the algorithms were able to solve; and the cost is the amount of time or resources that the algorithms used to solve the instances, all three scatter cross-referenced with time.

3. Results

In my work, I tested the Anytime Weighted A* (AWA*) algorithm with cost units in Korf's puzzles domain. AWA* has the important feature of being able to provide a solution at any time and can iteratively find the optimal solution. My focus was initially on the cost type being units. Through experimentation, I found that for the average quality, a lower weight can have better performance than a heavier weight.

Based on Fig 3, it was observed that W=10 found some solutions for all problems in the shortest time (10^{-2}) but did not improve much in terms of quality (Fig 1). Conversely, W=3 found some solutions for all problems in the longest time ($\sim 10^{-1}$) but had the best quality in the shortest time. Additionally, the first solution found by W=3 was better than the other weights based on solution cost (Fig 2).

Further research indicated that if the weight is higher, it pays less attention to g (cost) and favors lowering h(x) (heuristic function). This results in a higher cost but less time for search, giving more solutions with quicker coverage. On the other hand, a lower weight pays more attention to g, resulting in a lower cost but a longer time to find the solution and lower coverage with higher quality. If the weight becomes 0, it is the same as the search with A*.

In summary, the strengths of AWA* include its ability to find solutions at any time and iteratively find the optimal solution. The limitations observed include the need for careful selection of the weight parameter and the tradeoff between solution quality and search time. Figures and specific details about problem instances were included to support the observations and conclusions.

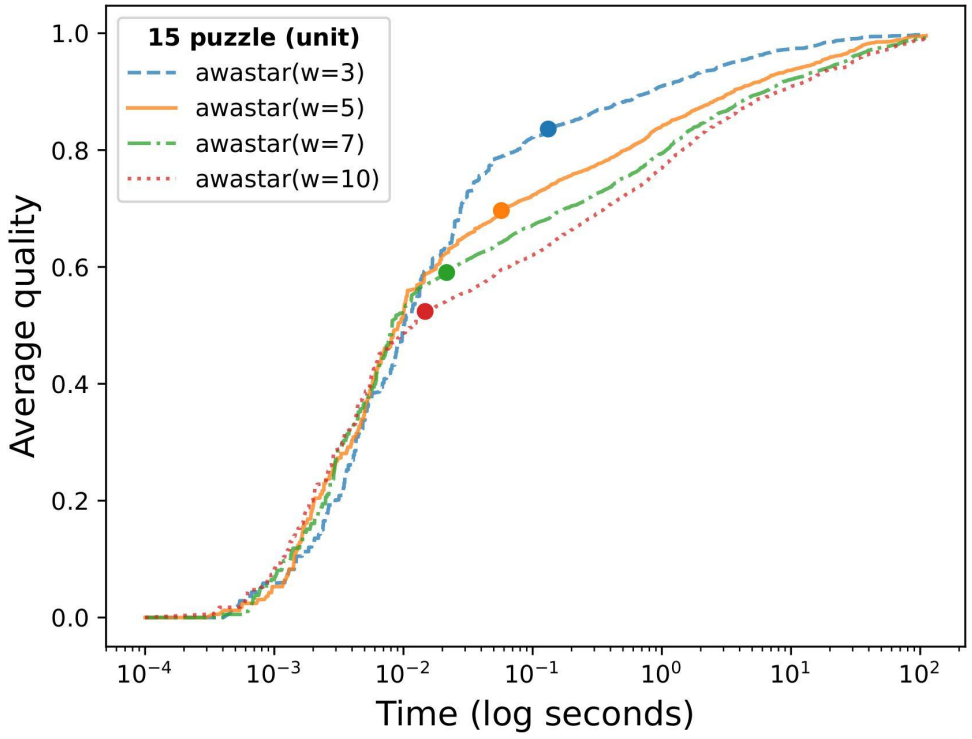
15 puzzle (unit)
awastar(w=3)
awastar(w=5)
awastar(w=7)
awastar(w=10)
Average quality
1.0
0.8
0.6
0.4
0.2
0.0
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Time (log seconds)

Fig 1

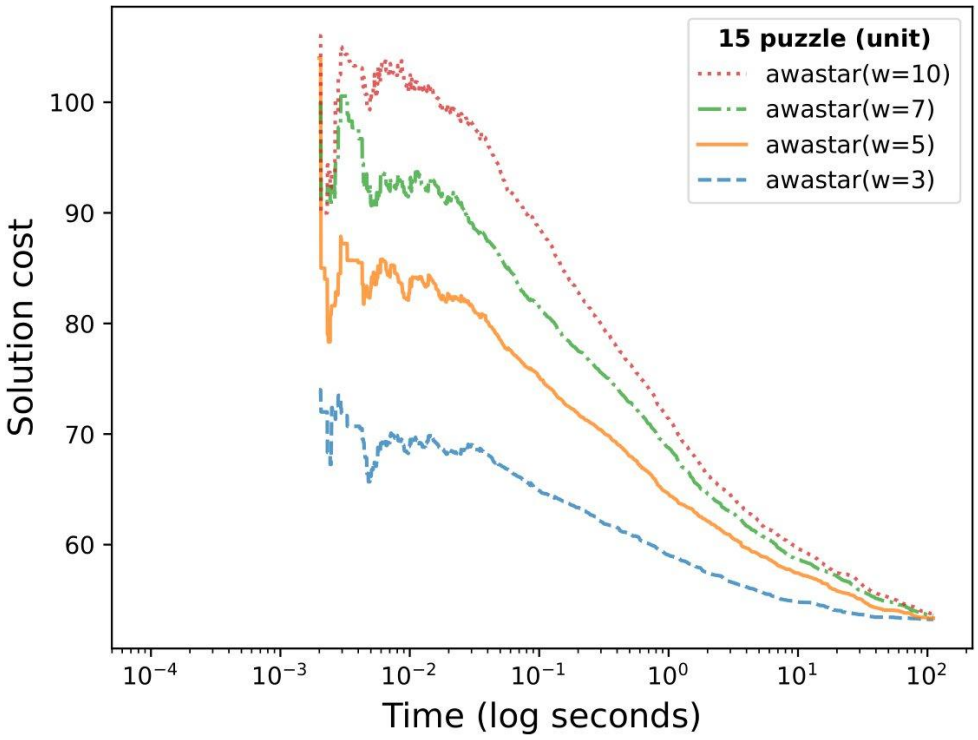
15 puzzle (unit)
awastar(w=10)
awastar(w=7)
awastar(w=5)
awastar(w=3)
Solution cost
100
90
80
70
60
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Time (log seconds)

Fig 2

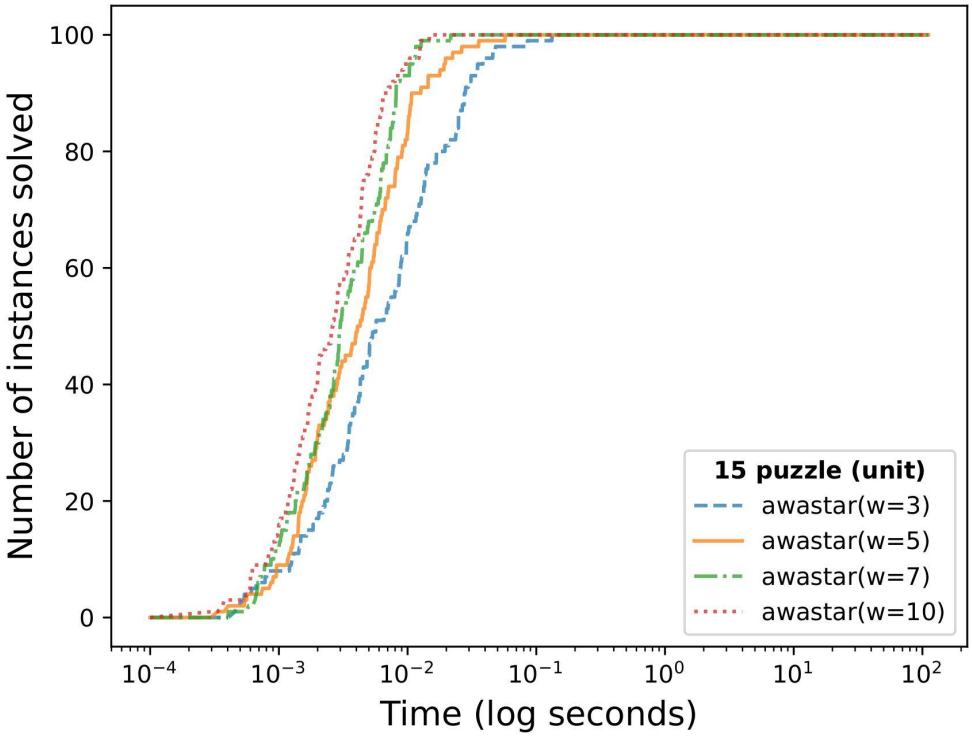

Fig 3

4. Questions & Future Work

Based on the work I have done, several questions and opportunities for improvement have emerged. One question is how the AWA* algorithm performs when compared to other cost types. Currently, I have focused on the cost type being unit, but it would be valuable to explore and compare the algorithm's performance with different cost types. This could involve implementing and evaluating AWA* with cost types such as weighted costs or heavy costs and comparing the results to assess the algorithm's effectiveness and efficiency.

Another potential avenue for improvement is to compare AWA* with other anytime search algorithms, such as Anytime Repair A* (ARA*). By conducting a comparative analysis, it would be possible to gain insights into the relative strengths and weaknesses of these algorithms in different problem domains. This comparison could involve designing experiments to evaluate the performance of AWA* and ARA* on a common set of benchmark problems, considering factors such as solution quality, search time, and resource usage.

Additionally, there is an interest in exploring an alternative approach for AWA* that does not involve using a weight parameter but rather utilizes distance estimation. This could involve investigating the use of distance-based heuristics, such as Manhattan distance or Euclidean distance, instead of relying on a weight parameter to balance the heuristic and path costs. By implementing and evaluating this modified version of AWA*, it would be possible to determine its performance and assess its advantages or disadvantages compared to the original weight-based AWA*.