Beyond Simple Heuristics: Unlocking the Power of Heuristic Combinations for Efficient A* and AWA* Search in the Pancake Problem

Jan Zaatra
Information Systems Engineering
Ben-Gurion University
Be'er-Sheva, Israel 85104
janz@post.bgu.ac.il

Abstract

The Pancake Problem presents a challenging search space for AI algorithms due to its large branching factor and complex goal state. Traditional approaches often rely on individual heuristics, which may not provide sufficient guidance for efficient search. This paper investigates the potential of combining multiple heuristics to enhance the performance of A* and Adaptive Weighted A* (AWA*) search algorithms in solving the Pancake Problem. We implement and evaluate various heuristic combination techniques, including simple averaging, weighted averaging, and taking the maximum value. Our experimental results demonstrate that sophisticated heuristic combinations significantly improve the search efficiency of both A* and AWA*, with AWA* showing particular promise in adapting to different problem instances. These findings highlight the importance of heuristic combination in solving complex search problems and suggest that AWA* can be a powerful tool for navigating large search spaces effectively.

Introduction

The Pancake Problem, a classic combinatorial puzzle first introduced by Jacob E. Goodman in 1975 [1], involves sorting a stack of pancakes, each with a distinct diameter, using a spatula to flip the top portion of the stack. The objective is to arrange the pancakes in ascending order of size, with the smallest on top, employing the fewest possible flips. Each flip reverses the order of a prefix of the stack, resulting in a search space that grows factorially with the number of pancakes, rendering it NP-hard. Figure 1 illustrates an instance of the Pancake Problem with 6 pancakes. Beyond being a fundamental search problem in artificial intelligence (AI), the Pancake Problem finds applications in diverse domains such as computational biology and robotics , where sorting or reordering processes are essential.

Heuristic search algorithms offer a powerful approach to tackle complex problems like the Pancake Problem. A* search, devised by Peter Hart, Nils Nilsson, and Bertram Raphael in 1968 [2], is an informed search algorithm that efficiently explores the state space using an evaluation function:

$f(n) = g(n) + h(n)$, where $g(n)$ represents the cost from the start node to the current node n, and $h(n)$ is a heuristic estimate of the cost from n to the goal. A* maintains an *open list* of nodes awaiting exploration and a closed list of already expanded nodes. At each step, it expands the node with the lowest $f(n)$ value. The selection of an appropriate heuristic function, h(n), is crucial for guiding A* search effectively, impacting both search efficiency and the optimality of the solution. A well-crafted heuristic can drastically reduce the number of nodes expanded, thus improving runtime and memory usage.

Adaptive Weighted A* (AWA*), an extension of A* proposed by Ira Pohl in 1970 [3], incorporates a dynamic weighting scheme for the heuristic function. The core idea behind AWA* is to adjust the weight of the heuristic during the search, enabling a more effective balance between exploration and exploitation of the search space. This adaptive mechanism allows AWA* to prioritize promising regions when necessary, potentially outperforming A* in terms of runtime and memory usage, especially for large or complex problem instances. This flexibility makes AWA* an attractive alternative to standard A*, particularly for tackling problems with high complexity, such as the Pancake Problem.

While individual heuristics can be valuable, they often have limitations in capturing all the pertinent information required for optimal problem-solving. Combining multiple heuristics can offer a more robust approach by integrating diverse perspectives on the state space and potentially enhancing search efficiency. Various techniques exist for combining heuristics, including simple averaging, maximization, and learning-based approaches that adaptively select the most suitable heuristic during the search. These combination strategies hold the potential to overcome the limitations of individual heuristics, particularly in complex search problems like the Pancake Problem.

Despite the potential benefits of heuristic combinations, their exploration within the context of AWA* remains limited, especially for the Pancake

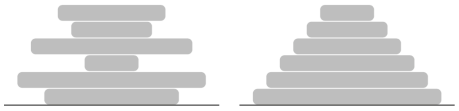

Figure 1: A 6-pancake instance (s = {3, 2, 5, 1, 6, 4, 7})

Problem. This study aims to bridge this gap by investigating the impact of heuristic combinations on the performance of A* and AWA* in solving the Pancake Problem. Our primary objective is to assess whether combining multiple heuristics can improve search efficiency, and whether AWA*, due to its adaptive nature, can achieve comparable or even superior performance with simpler heuristics.

We hypothesize that sophisticated heuristic combination techniques will lead to improved A* search performance, characterized by reduced node expansions and faster runtime, compared to using individual heuristics or simple averaging. However, we also anticipate that AWA*, with its inherent ability to dynamically adjust weights during the search, may exhibit less dependence on complex heuristic combinations and demonstrate comparable or even superior performance against A*.

The remainder of this paper is structured as follows. Section 2 provides a comprehensive background on A* and AWA* search algorithms, along with an overview of the Pancake Problem and the specific heuristics employed in this study. Section 3 details our experimental setup, encompassing the heuristic combination techniques and the evaluation metrics used. In Section 4, we present and analyze the results of our experiments, comparing the performance of A* and AWA* under different heuristic combinations. Finally, Section 5 concludes the paper with a discussion of our findings, their implications for future research in heuristic search algorithms, and potential avenues for further exploration.

Background and Related Work

While dynamic heuristics have shown promise in enhancing pathfinding algorithms, current approaches face challenges in scalability, real-world applicability, and integration with other methods. Existing techniques like D* [Stentz, 1995] [4] and D* Lite [Koenig & Likhachev, 2002] [5] often struggle with large or complex environments. Furthermore, the focus on isolated dynamic heuristics leaves opportunities for exploring hybrid approaches that leverage their strengths alongside other techniques, such as local search or planning under uncertainty. Our project aims to address these limitations by investigating the impact of sophisticated heuristic combinations on A* and AWA* search performance in the Pancake Problem, a challenging domain that requires efficient and adaptive search strategies.

Problem Definition

The Pancake Problem is a state-space search problem in which a stack of pancakes of different sizes must be sorted by size, with the largest at the bottom and the smallest at the top. Each pancake stack is represented by a permutation of numbers, where each number denotes a pancake's size. The problem allows one action: flipping the top k pancakes, reversing the order of the first k elements in the stack. The objective is to find a sequence of flips that transforms a given initial stack into the sorted goal state, where the pancakes are arranged in increasing order.

Formally, the Pancake Problem is represented by a tuple (S, A, T, C), where:

- S is the set of all possible states (permutations of pancakes).
- A(s) represents the set of actions (flips) applicable to a state s.
- T(s, a) is the transition function, which produces a new state by applying the flip action a to state s.
- C(s, a) is the cost of applying action a to state s, typically defined as a unit cost (i.e., C(s, a) = 1 for each flip).

The goal state is a fully sorted stack, represented as [1, 2, 3, ..., n] for a stack of n pancakes. For the experimental setup, initial states are generated by randomly permuting the sorted stack. To ensure reproducibility, a fixed random seed is used for generating the random start states. These randomly generated initial states are used across multiple test runs for comparative analysis of different heuristic approaches in solving the Pancake Problem.

Algorithm 1: A*

```
1.  function A*(startNode, goalNode):
2.    OPEN = Priority Queue containing startNode
3.    CLOSED = Empty Set
4.    startNode.g = 0
5.    startNode.h = heuristic(startNode, goalNode)
6.    startNode.f = startNode.g + startNode.h
7.    while OPEN is not empty:
8.      current = ExtractMin(OPEN) // Node with lowest f value
9.      add current to CLOSED
10.     if GoalTest(current): // Check if goal reached
11.       return reconstruct_path(current)
12.     for each neighbor in neighbors(current):
13.       if neighbor in CLOSED:
14.         continue // Ignore already expanded nodes
15.       tentative_g = current.g + cost(current, neighbor)
16.       if neighbor not in OPEN or tentative_g < neighbor.g:
17.         neighbor.parent = current
18.         neighbor.g = tentative_g
19.         neighbor.h = heuristic(neighbor, goalNode)
20.         neighbor.f = neighbor.g + neighbor.h
21.         if neighbor not in OPEN:
22.           add neighbor to OPEN
23.   return failure   // No path found
```

Algorithm 2: AWA*

```
1. function AWA*(startNode, goalNode, w):
2.    OPEN = Priority Queue containing startNode
3.    CLOSED = Empty Set
4.    startNode.g = 0
5.    startNode.h = heuristic(startNode, goalNode)
6.    startNode.f = startNode.g + w * startNode.h
7.    while OPEN is not empty:
8.       current = ExtractMin(OPEN) // Node with lowest f value
9.       add current to CLOSED
10.      if GoalTest(current): // Check if goal reached
11.            return reconstruct_path(current)
12.      for each neighbor in neighbors(current):
13.         if neighbor in CLOSED:
14.               continue // Ignore already expanded nodes
15.         tentative_g = current.g + cost(current, neighbor)
16.         if neighbor not in OPEN:
17.               neighbor.parent = current
18.               neighbor.g = tentative_g
19.               neighbor.h = heuristic(neighbor, goalNode)
20.               neighbor.f = neighbor.g + w * neighbor.h
21.               add neighbor to OPEN
22.         else if tentative_g < neighbor.g:
23.               neighbor.parent = current
24.               neighbor.g = tentative_g
25.               if neighbor in OPEN:
26.                  update neighbor's priority in OPEN
27.   return failure     // No path found
```

Search Algorithms

In this study, we explore two search algorithms, A* and Adaptive Weighted A* (AWA*), both of which are commonly used in solving state-space search problems like the Pancake Problem.

A* is a best-first search algorithm that optimizes for both path cost and estimated cost to the goal. It uses a priority queue (OPEN list) to manage nodes based on their f value, where $f(n) = g(n) + h(n)$, with g(n) representing the path cost from the start node to node n, and h(n) denoting the estimated cost from n to the goal (the heuristic). The algorithm expands the node with the lowest f value from the OPEN list and adds it to a CLOSED list to prevent redundant expansions. The search continues until the goal state is reached or the OPEN list is exhausted. A* guarantees optimality if the heuristic used is admissible (i.e., it never overestimates the true cost to reach the goal).

The pseudocode for A* is detailed in Algorithm 1. The OPEN list is implemented as a priority queue, and the CLOSED list is a set that stores expanded nodes. The key operations include updating the g, h, and f values for each neighbor of the current node, checking if they need to be re-added to the OPEN list with a new priority.

AWA* builds on the A* algorithm by introducing an adaptive weighting mechanism that adjusts the weight of the heuristic dynamically. The algorithm prioritizes nodes based on a weighted f value, where $f(n) = g(n) + w * h(n)$. The weight parameter w allows for a trade-off between exploration and exploitation. A larger w emphasizes the heuristic, potentially reducing search time at the cost of sub-optimality. Unlike A*, AWA* can adaptively adjust w during the search based on certain criteria, enabling it to balance the trade-off between solution quality and runtime efficiency. In the experimental implementation used here, we set an initial w value, which influences the weighting of the heuristic at the start of the search.

The pseudocode for AWA* is outlined in Algorithm 2. Similar to A*, it uses a priority queue for the OPEN list and a set for the CLOSED list. However, in AWA*, the priority of nodes is updated based on the weighted heuristic, and the OPEN list is dynamically re-prioritized when a better path is found.

The primary difference between A* and AWA* lies in the weighting mechanism applied to the heuristic function in AWA*. While A* uses a strict $f = g + h$ calculation, AWA* introduces a weighting factor w, allowing the algorithm to behave more aggressively in favor of heuristic-driven exploration. This adaptive approach can potentially reduce the number of node expansions and improve runtime performance, albeit with the possibility of returning suboptimal solutions depending on the value of w.

The intuition behind weighting the heuristic in AWA* is to control how much emphasis is placed on estimated distance to the goal versus actual cost incurred.

A higher weight w biases the search towards faster exploration of nodes believed to be closer to the goal, which can speed up the search but may increase the risk of suboptimal paths. Conversely, a lower weight shifts focus back towards ensuring optimality, making the search more thorough but potentially slower.

Heuristic Functions

In the context of solving the Pancake Problem using A* and AWA* search algorithms, the choice of heuristic functions is paramount in guiding the search efficiently toward the goal state. Below, I provide a mathematical breakdown of each heuristic used and discuss their admissibility, calculation methods.

1. Gap Heuristic

The Gap Heuristic calculates the number of gaps between consecutive pancakes that are not in their goal positions. Specifically, a gap occurs between two pancakes if their difference is greater than one in terms of their value.

Mathematical Definition:

Let $S = [s_1, s_2, \ldots, s_n]$ represent the current state of the pancake stack, where s_i denotes the value of the pancake at position i. Define the goal state $G = [1, 2, \ldots, n]$. The gap heuristic $h_{gap}(S)$ is given by:

$$h_{\text{gap}}(S) = \sum_{i=1}^{n-1} \text{gap}(s_i, s_{i+1}),$$

Where

$$\text{gap}(s_i, s_{i+1}) = \begin{cases} 1, & \text{if } |s_i - s_{i+1}| > 1 \\ 0, & \text{otherwise} \end{cases}$$

This heuristic is admissible because it never overestimates the number of flips needed to bring the pancakes into consecutive order. It counts only gaps between consecutive pancakes, and each gap requires at least one flip to resolve.

2. Breakpoint Heuristic

The Breakpoint Heuristic counts the number of breakpoints in the pancake stack. A breakpoint occurs when two consecutive pancakes are not in consecutive order with respect to the goal state.

Mathematical Definition:

Let $S = [s_1, s_2, \ldots, s_n]$. A breakpoint occurs at position i if $|s_i - s_{i+1}| \neq 1$.

The breakpoint heuristic $h_{\text{breakpoint}}(S)$ is given by:

$$h_{\text{breakpoint}}(S) = \sum_{i=1}^{n-1} \text{bp}(s_i, s_{i+1}),$$

Where

$$\text{bp}(s_i, s_{i+1}) = \begin{cases} 1, & \text{if } |s_i - s_{i+1}| \neq 1 \\ 0, & \text{otherwise} \end{cases}$$

This heuristic is admissible because it underestimates or correctly estimates the number of flips needed to resolve all breakpoints, as every breakpoint requires at least one flip to resolve.

3. **Position-Based Heuristic**

The Position-Based Heuristic penalizes pancakes based on their distance from their correct positions in the goal state. This heuristic is non-admissible as it may overestimate the true number of flips required.

Mathematical Definition:

Let $S = [s_1, s_2, \ldots, s_n]$ *and* $G = [1, 2, \ldots, n]$. For each pancake s_i calculate the distance from its goal position:

$$h_{\text{position}}(S) = \sum_{i=1}^{n} |i - s_i| .$$

This heuristic is non-admissible because the sum of absolute distances may overestimate the number of flips required to reach the goal.

Implementation Details: Despite its non-admissibility, this heuristic can be used effectively in combination with admissible heuristics like the gap and breakpoint heuristics to guide search more aggressively toward the goal. Its non-admissible nature allows it to explore more diverse paths by encouraging larger jumps in state space exploration.

Heuristic Combination Techniques

In this section, we explore the heuristic combination techniques employed in this study, which include Simple Averaging (SAH), Weighted Averaging (WAH), and Max Heuristic (MAH). These techniques are designed to combine the strengths of multiple heuristic functions in guiding the A* and AWA* search algorithms. Below, we describe each technique in detail, provide the respective formulas, and discuss parameter choices or tuning involved.

1. Simple Averaging Heuristic (SAH)

SAH combines the values of multiple heuristics by taking their arithmetic mean. This method assumes that each heuristic contributes equally to the overall guidance of the search. Formally, if we have n heuristics $h_1, h_2, \ldots, h_n$, the combined heuristic is calculated as:

$$h_{\text{SAH}}(s) = \frac{1}{n} \sum_{i=1}^{n} h_i(s)$$

where s is the current state. SAH is simple to implement and does not require any tuning, making it a straightforward way to leverage multiple heuristics.

2. Weighted Averaging Heuristic (WAH)

WAH extends the simple averaging approach by assigning different weights to each heuristic, allowing for more flexible tuning based on the perceived importance of each heuristic. The weighted combination is defined as:

$$h_{\text{WAH}}(s) = \sum_{i=1}^{n} w_i \cdot h_i(s)$$

where w_i is the weight associated with heuristic h_i, and $\sum_{i=1}^{n} w_i = 1$ to ensure that the weights normalize the combined heuristic. The choice of weights w_i depends on experimental tuning, and different configurations may perform better depending on the characteristics of the problem domain.

3. Max Heuristic (MAH)

MAH takes the maximum value of the heuristics at each state, ensuring that the most pessimistic (highest) estimate of the remaining cost is used to guide the search. The formula is:

$$h_{\text{MAH}}(s) = \max_{i=1}^{n} h_i(s)$$

While MAH ensures that the search is guided by the most informative heuristic at each state, the admissibility of MAH depends on the individual heuristics being combined. If all heuristics used in MAH are admissible, then MAH will also be admissible. However, if a non-admissible heuristic, such as the position-based heuristic, is included, the overall MAH will not guarantee admissibility, potentially sacrificing optimality for improved speed in some cases.

Parameter Choices and Tuning

For SAH, no parameters need to be tuned, as all heuristics are equally weighted. In contrast, WAH requires careful selection of the w_i based on empirical evaluation. These weights were tuned during initial

testing to strike a balance between the competing heuristics. The specific weight configurations were also applied consistently throughout the experiments.
MAH does not require parameter tuning but benefits from a strong set of heuristics since it chooses the maximum heuristic value at each state, ensuring robustness across different scenarios.
This combination of techniques allows us to explore the potential of leveraging multiple heuristics, either by balancing their contributions or focusing on their most informative estimates at each stage of the search.

Experimental Setup and Evaluation

To evaluate the performance of the various heuristic functions and combination techniques, we conducted experiments across a range of pancake sizes and test instances. Specifically, we considered the following configurations for pancake size and number of test instances:

(100, 5), (200, 6), (200, 7), (100, 8), (30, 9), (10, 10), (1, 11)

Such as the left number is the number of test size and the right number is the number of pancakes. The choice of test sizes was driven by the need to balance computational feasibility with statistical robustness. Larger pancake sizes naturally increase the complexity and computation time for each test, necessitating fewer test instances for these configurations. This approach ensures that we can gather sufficient data while managing computational resources effectively. For the three heuristic combination methods (**SAH**, **WAH**, and **MAH**), all three basic heuristics (gap, breakpoint, and position-based) were utilized.
For each experiment, the test instances were randomly generated to ensure diversity and coverage of different possible scenarios. This randomness in initial states helps in providing a comprehensive assessment of the heuristic functions across various starting conditions.
The experiments were conducted on a MacBook Pro 2019 with an Intel Core i7 processor, 16GB RAM. The software environment included Python 3.10, and the following libraries: pandas, matplotlib, seaborn, heapq, math, and tracemalloc. The operating system used was macOS.
The performance evaluation metrics include **Average Time** (mean time to find a solution), **Average Solution Cost** (mean cost of the solution path), **Average Nodes Expanded** (mean number of nodes explored), and **Average Memory Usage** (mean memory consumption). These metrics provide a comprehensive assessment of the search algorithm's efficiency and effectiveness.

Results

In this section, we present the performance outcomes of the A* and Adaptive Weighted A* (AWA*) search algorithms on the Pancake Problem, focusing on the impact of various heuristic combinations. The results are illustrated in **Figure 2** for A* and **Figure 3** for AWA*, which depict four key metrics: average time, nodes expanded, solution cost, and memory usage. These metrics were measured across different heuristic functions and their combinations, including gap, breakpoint, position-based heuristics, SAH, WAH, MAH.

A* Algorithm Performance

Figure 2 demonstrates that the combination heuristics, particularly the SAH and WAH, significantly enhance the performance of the A* algorithm. When comparing these combinations against basic heuristics like gap and breakpoint, a clear reduction in average time, nodes expanded, and memory usage is observed. This outcome aligns with our hypothesis, indicating that sophisticated heuristic combinations can indeed optimize the search process.
For instance, the SAH consistently achieves a balanced trade-off by reducing both average time and nodes expanded, while also closely maintaining the solution cost relative to the gap and breakpoint heuristics. Although the average solution cost for this combination is slightly higher than the basic admissible heuristics, it remains close enough to validate its effectiveness. Similarly, the WAH demonstrates even more pronounced improvements in performance, achieving reductions in all three key metrics—time, nodes, and memory—while maintaining a competitive solution cost.
These results confirm that sophisticated heuristic combinations, particularly SAH and WAH, are capable of striking an optimal balance between computational efficiency and solution quality. The A* algorithm benefits substantially from these combinations, reducing the computational burden without compromising the integrity of the solutions.

AWA* Algorithm Performance

Moving on to the AWA* results depicted in **Figure 3**, the data reveal that AWA* also benefits from the use of complex heuristic combinations. Contrary to initial assumptions, the combination heuristics significantly improve the algorithm's performance by providing solutions with marked reductions in time, nodes expanded, and memory usage. These improvements come with a good trade-off in solution cost, which remains close to that of the basic heuristics like gap and breakpoint.

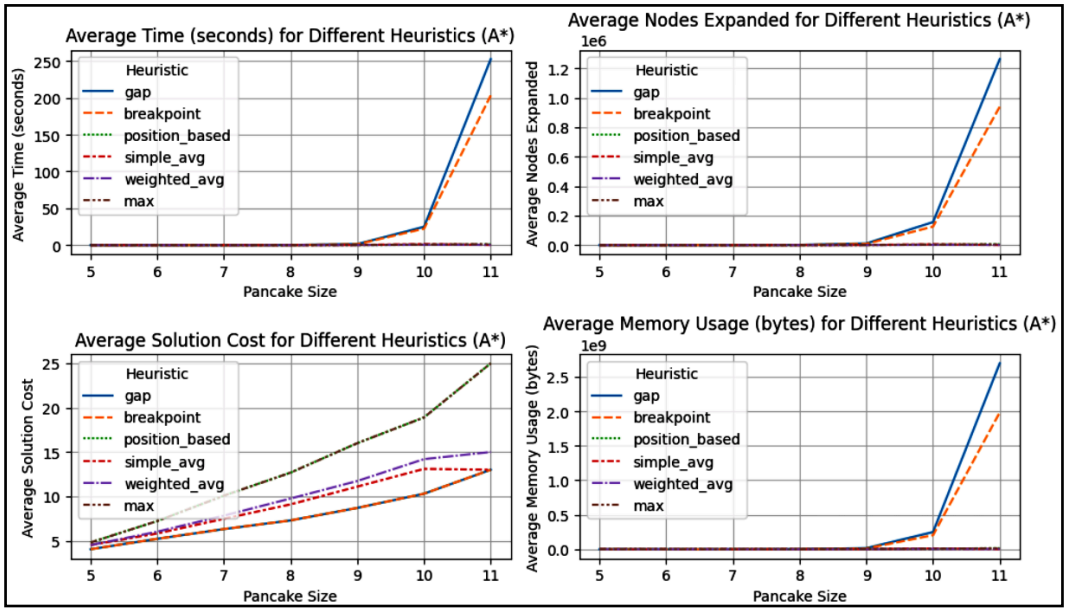

Figure 2: Performance Metrics for A* Algorithm Using Various Heuristic Combinations

The WAH, for example, exhibits a noteworthy reduction in average time and nodes expanded compared to basic heuristics. The memory usage, while slightly higher than that of A*[1], is still efficient, and the trade-off is justified by the significant improvements in other metrics. Importantly, the average solution cost remains well within an acceptable range, demonstrating that AWA* maintains a strong balance between speed and solution quality, even when leveraging more sophisticated heuristic combinations.

The results thus indicate that, rather than showing less reliance on heuristic combinations, AWA* effectively utilizes these combinations to enhance its performance. The dynamic adjustment feature of AWA* complements these combinations, allowing the algorithm to achieve superior results across multiple metrics.

Comparison Between A* and AWA*

When comparing A* and AWA*, the results underscore the strengths of AWA* in leveraging its dynamic weighting feature in conjunction with heuristic combinations. As shown in **Figure 3**, AWA* consistently outperforms A* in terms of time efficiency and nodes expanded, while maintaining a comparable solution cost. This reinforces our hypothesis that AWA*, with its ability to adapt during the search process, can achieve a more effective balance between computational efficiency and solution quality.

The memory usage in AWA*, while slightly higher than in A*, is still within an acceptable range, and the trade-off is justified by the significant reductions in time and nodes expanded. These findings suggest that AWA* is particularly well-suited for scenarios where time and computational resources are critical, offering a more optimized search process.

Overall, the results support the hypothesis that sophisticated heuristic combinations improve the performance of both A* and AWA* significantly. AWA* shows that even with dynamic adjustments, the combination heuristics remain crucial in achieving the best performance. This balance between efficiency and solution quality positions AWA* as a powerful alternative to A* in solving complex search problems like the Pancake Problem.

Comparison of Heuristic Combinations

Among the three heuristic combinations tested—SAH, WAH, and MAH—the SAH emerges as the most effective. It consistently achieves results close to the optimal solutions provided by the gap and breakpoint heuristics. The WAH follows closely, demonstrating a strong balance between efficiency and accuracy, but not quite matching the performance of SAH. Lastly, the MAH heuristic proved to be the least effective, offering less promising results as a combination technique.

Sensitivity Analysis of Weight Adjustment Strategies in AWA*

In the sensitivity analysis, we evaluated three heuristic weight adjustment strategies in AWA*: **linear**, **exponential**, and **piecewise**. The **linear strategy** reduces the weight as $w = 1 - \frac{g}{g\max}$, maintaining a gradual balance between speed and accuracy. The **exponential strategy** decays the weight according to $w = e^{-0.1 \cdot \frac{g}{g\max}}$, prioritizing faster exploration early on. The **piecewise strategy** adjusts the weight in phases, holding w=1.0 initially, then reducing to w=0.2 as the search progresses. Among these, the **exponential strategy** consistently returned better

[1]Based on the numbers we have, which are not shown in this paper, the memory usage for AWA* with combination heuristics is slightly higher than A*, but the difference is small and remains within an acceptable range, supporting the trade-off for improved performance across other metrics.

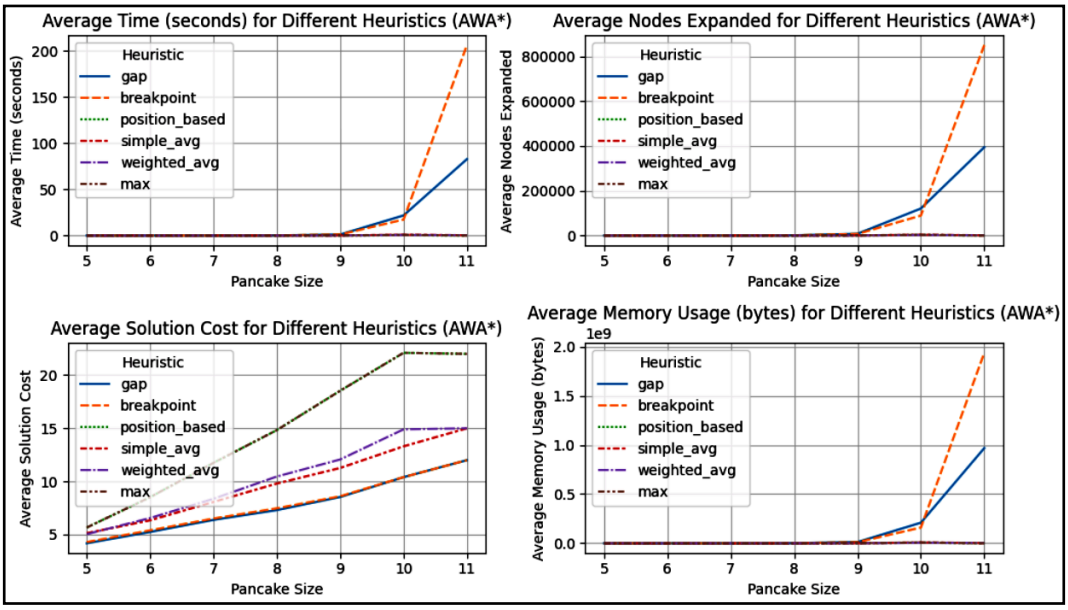

Figure 3: Performance Metrics for AWA* Algorithm Using Various Heuristic Combinations

results across key metrics (which were included in the results).

Limitations

While the results of this study demonstrate significant improvements in performance through heuristic combinations in both A* and AWA*, several limitations should be acknowledged. First, the range of pancake sizes tested was constrained to relatively small instances (up to size 11) due to the exponential complexity of the Pancake Problem (n!), which significantly increased computational demands for larger stacks. This limitation prevents broader generalization to larger-scale instances. Additionally, the specific choice of weightings in the Weighted Averaging Heuristic (WAH) was manually tuned based on preliminary experiments, potentially introducing bias and limiting the adaptability of this technique across different problem instances. Another limitation involves the hardware used—a 2019 MacBook Pro with an Intel Core i7 processor. More powerful hardware could potentially yield faster results and permit larger test cases. Finally, the performance of AWA* is influenced by the specific choice of parameters in the dynamic weight adjustment, and further sensitivity analysis could reveal alternate settings that might further enhance its effectiveness.

Future Work

Future research directions could explore several avenues for enhancing the performance of search algorithms in the Pancake Problem. One promising direction is the application of optimization techniques, such as genetic algorithms or particle swarm optimization, to fine-tune the search process and further reduce computation time. Additionally, incorporating machine learning techniques to automatically learn and optimize parameters—such as weights in the WAH and adaptive_weighted_heuristic functions—could lead to more effective and adaptable search strategies tailored to different problem instances.
Other potential directions include exploring parallel and distributed computing approaches to handle larger pancake stacks more efficiently. Investigating the scalability of these methods across different combinatorial problems would also be valuable. Moreover, experimenting with alternative heuristic combination techniques, such as reinforcement learning or neural networks, could offer novel ways of improving heuristic guidance during the search.

Conclusions

In this paper, we investigated the effectiveness of sophisticated heuristic combinations on the Pancake Problem, applying both A* and Adaptive Weighted A* (AWA*) search algorithms. Our key findings demonstrate that heuristic combinations, particularly Simple Averaging Heuristic (SAH) and Weighted Averaging Heuristic (WAH), significantly improve the performance of both A* and AWA*. These combinations lead to reduced time, node expansions, and memory usage while maintaining a competitive solution cost. Notably, AWA* was found to outperform A* in terms of time efficiency, thanks to its dynamic weighting mechanism, further validating the importance of adaptive heuristics in complex search problems. **The results confirm that our hypothesis is correct**, as sophisticated heuristic combinations do indeed optimize the search process and improve algorithmic performance. Our work highlights the potential of heuristic combinations to strike a balance between computational efficiency and solution quality, offering new avenues for optimization in heuristic search algorithms.

Closing Remarks

This work highlights the potential of advanced heuristic combination techniques to significantly en-

hance the performance of search algorithms, particularly in complex problems like the Pancake Problem. By pushing the boundaries of traditional heuristics and introducing dynamic adaptations, we contribute valuable insights for future optimizations in search algorithms. The findings underscore the impact that heuristic combinations and adaptive strategies can have on computational efficiency and solution quality, opening doors for further exploration and innovation in this critical area of AI research.

Acknowledgments

I would like to extend my sincere gratitude to Professor **Ariel Felner** at Ben-Gurion University in Israel. His extensive research in heuristic search algorithms, particularly in the context of Artificial Intelligence, has been a significant source of inspiration and guidance throughout this work. His contributions to the field have profoundly influenced the direction and depth of this research.

References

[1] Goodman, J. E. (1975). A pancake problem. *Discrete Mathematics*, 11(2), 123-132.

[2] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107.

[3] Pohl, I. (1970). Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3-4), 193-204.

[4] Stentz, A. (1995). The focussed D* algorithm for real-time replanning. *Proceedings of the IEEE International Conference on Robotics and Automation*, 1652-1659.

[5] Koenig, S., & Likhachev, M. (2002). D* Lite. Proceedings of the National Conference on Artificial Intelligence (AAAI), 476-483.