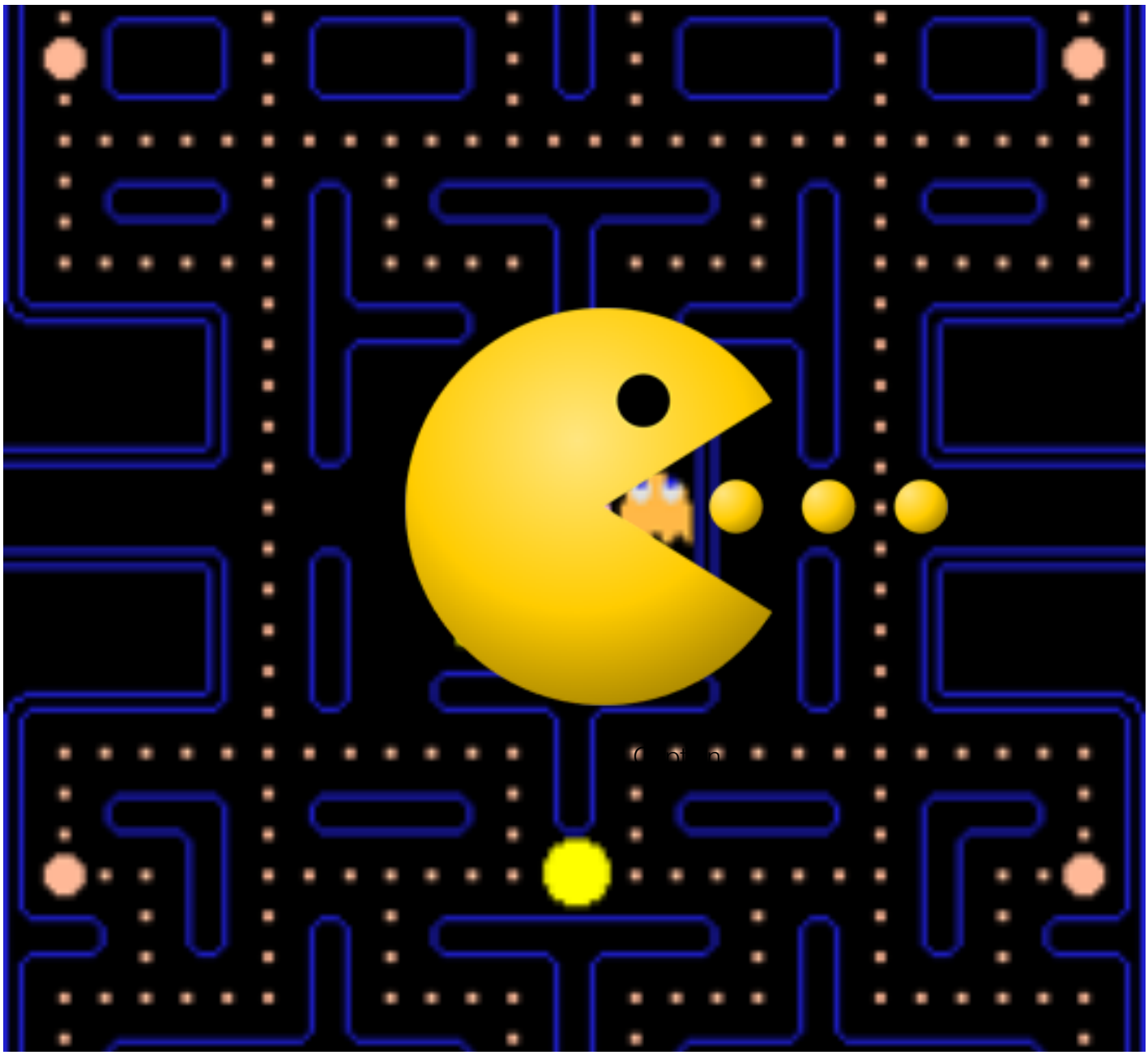
AI final project

Solving Pacman With Multi Heuristic A* Search



Aryeh Nailand, Maya Swisa, Yaakov Korman, Miriam Goldstein

Summer 2021

Solving Pacman With Multi Heuristic A* Search

Aryeh Nailand, 342473642

Maya Swisa, 209028760

Yaakov Korman, 329692966

Miriam Goldstein, 206221509

Abstract

Heuristics in general are hard to create since proving their consistency can be exceedingly difficult, and therefore they cannot be used for searches such as A*. Furthermore, heuristics can have depression regions in a search space in which they do not perform particularly well. In this paper two variations of the Multi Heuristic A* algorithm (IMHA* and SMHA*) are proposed to address these problems (and a few others), and their efficacy is tested against the A* and WA* search algorithms in the Pacman game search space.

I. Problem Description

The search problem being used for analysis is the **Pacman Multiple Food Search problem**

Input: A Pacman maze of size $m \times n$, the starting position of the Pacman agent (a tuple of x, y coordinates in the maze) and a list of all of food coordinates which the Pacman must eat to win.

Output: A legal path from the starting position through the maze where the Pacman agent eats all the food in the provided coordinates.

The problems with A* and WA* attempting to be solved:

1. These algorithms are not fast enough when given problems with large search spaces.
2. For use with these algorithms, heuristics must be consistent and therefore admissible to guarantee optimality and completeness. When creating heuristics, proving admissibility can be onerous and could even be impossible. The freedom to think of heuristics without the trouble of these proofs could be a major asset, without sacrificing completeness and optimality.
3. Heuristics may have depression regions in which they slow the process of the search. A heuristic function's calculation may guide the search to remain in areas that seem better in the short term but are worse for finding a good solution fast. Have the choice between multiple heuristics that have different depression regions (or synthesizing them together) can significantly decrease the search time.

II. Solution Concept

In the course A* search was used to solve various search problems. While A* search with an admissible, consistent heuristic is complete and guarantees an optimal solution (if a solution exists), constructing and developing a fast, admissible and consistent heuristic to find solutions that are memory efficient is very difficult. One of the reasons for this is that by using one heuristic, it is difficult to cover all regions of the search space. Two variations of Multiple Heuristic A* search were used: IMHA* (Independent Multiple Heuristic A* Search) and SMHA* (Shared Multiple Heuristic A* Search). Their performance is compared against regular A* and weighted A* (WA*) which use admissible and consistent heuristics.

WA* search is identical to A* search apart from the calculation of the value of a state. When adding states to the priority queue, heuristic value is multiplied by a weight factor. This lowers solution optimality in exchange for speed. So instead of adding the state to the priority queue with the following priority:

$$(cost\ of\ path\ until\ this\ state) + heuristic(state)$$

we add the state to the priority queue with this priority:

$$(cost\ of\ path\ until\ this\ state) + w * heuristic(state)$$

(For some $w > 1$)

Therefore, WA* is considered to be a $w-$approximation algorithm of A*, such that w is the weight factor. This is faster than A* since the algorithm gives priority to states which are closer to the goal in order to minimize the cost to the goal. WA* is also complete, just suboptimal depending on the weight factor.

Multiple Heuristic Search Algorithms:

(Pseudo Code can be found in articles in references at end of report)

IMHA*:

Input:

- A search problem
- 1 consistent admissible heuristic
- n potentially inadmissible heuristics,
- weights which together bound the optimality w_1, w_2

Output: The path from the start node to the goal node.

How it works:

The algorithm generates $n + 1$ priority queues (one for each heurisitic participating) and runs n independent searches, one for each heuristic supplied. The algorithm proceeds to run each search, expanding node by node in a 'round robin' fashion between the searches. Every time a node is expanded in each search the heuristic value is calculated for this node while ensuring the value is less than the suboptimality bound determined by w_1, w_2 and the admissible, consistent heuristic. If the suboptimality bound is breached by one of the searches, instead of expanding a state from this search, a state from the admissible heuristics' priority queue is expanded in its place. This process continues until a path is found to the goal. Note that state expansion works identically to that of A* search.

In the research paper several important theorems were proven with regard to IMHA*:

Theorem: The cost of the solution returned by IMHA* is bounded by the $w_1 * w_2$ suboptimality factor.

Theorem: In IMHA* no state is expanded more than $n + 1$ times.

SMHA*

SMHA* has the same inputs and output as IMHA* however there are a few differences in the actions taken in the algorithm.

There are still $n + 1$ priority queues, however when a state is expanded its successors are updated in all of the priority queues. This effectively causes the paths to be shared between all the heuristics searches, therefore a path to the goal is found using a combination of heuristics. There is an internal mechanism which ensures that a single node is expanded at most twice – once by any of the potentially inadmissible heuristics, and once by the consistent heuristic.

In the research paper the following theorems were proven with regard to SMHA*:

Theorem: In SMHA* no state is expanded more than twice

Theorem: The cost of the solution returned by SMHA* is bounded by a $w_1 * w_2$ sub optimality factor.

We also note that SMHA* and IMHA* are complete since the default search is an admissible, consistent heuristic search which is complete (and ends up just being WA*).

Advantages of SMHA* and IMHA* over WA* and A*:

1. Since in SMHA* and IMHA* multiple heuristics are used, there is a higher chance of avoiding depression regions in the search. For IMHA* this means that there is a higher probability that at least one of the heuristics will be able to avoid depression regions in the search. For SMHA* this means that multiple heuristics are combined to hopefully avoid all the depression regions in the search. For example, if heuristic 1 (h_1) has depression region 1 (d_1) and heuristic 2 (h_2) has depression region 2 (d_2), then if d_1 and d_2 do not overlap, then the combination of using h_1 and h_2 can avoid both depression regions. This increases speed of the search.
2. SMHA* and IMHA* both can use inadmissible heuristics which speeds up the search.
3. There is a much higher degree of freedom when creating heuristics since heuristics don't need to be provenly admissible or consistent.

IMHA* vs SMHA* comparison

1. SMHA* expands each state maximum twice whereas IMHA* expands each state up to $n + 1$ times. However, the expansion operation in SMHA* is a lot more expensive in terms of run time and memory which is an important trade-off to consider (see page 25 in the research paper).
2. Every heuristic may have some form of depression region in the search space. IMHA* returns a path based on the calculations of one heuristic which could have a depression region – slowing down the search. If the heuristics complement each other well, then SMHA* could use a combination of heuristics to avoid this region.

Heuristics

Initially, all algorithms used the following admissible, consistent heuristic – **Max Maze Distance.** This heuristic returns the actual distance from the Pacman agent's location to the furthest uneaten food. Even though this heuristic opens less nodes than the **Max Manhattan Distance** described below since it more accurately estimates the cost to the goal, when calculating the heuristic value of a particular state, Breadth-First-Search must be run from the current position to all of the remaining food left in the Pacman maze. This is inefficient in terms of run time. Because of this inefficiency, the following admissible, consistent heuristic was used for the algorithms instead:

Max Manhattan Distance - This heuristic calculates the Manhattan distance from the current location of the Pacman agent to each of the remaining uneaten food locations in the maze and returns the maximum. Since this is simply reducing the problem (of eating all the remaining food) and Pacman must travel to the furthest food in the optimal path, this heuristic is both consistent and admissible.

Mathematically:

$$\max_{food \in uneaten} \{ManhattanDistance(\text{Pacman}, \text{food})\}$$

The following inadmissible heuristics were used for the IMHA* and SMHA* algorithms as well:

Max Two Food Distance – calculates the maximum distance between the coordinates of any two uneaten foods and then multiplies this distance by two.

Mathematically:

$$2 \times \max_{f_1, f_2 \in uneaten} ManhattanDistance(f_1, f_2)\}$$

Amount of Food Left Heuristic – calculates the number of squares with uneaten food in the Pacman maze and multiplies this number by two.

Mathematically:

$$2 \times \#uneaten\ food$$

Four Quarters Heuristic – divides the maze into four equal quarters (or as equal as possible if one of the dimensions is odd) and returns the number of quarters which still have food inside of them multiplied by the board width added to the board height and finally multiplied by 0.2.

Mathematically:

$$0.2 \times (board\ width + board\ height) \times \sum_{i \in [4]} quarter_i$$

Such that $quarter_i = 1$ if and only if there is uneaten food in $quarter_i$.

Note that the reason why some of these heuristics are multiplied by a certain constant to ensure the heuristic values were all the same order of size and appropriate for each individual size of the Pacman maze.

III. Results

A*, WA*, IMHA*, and SMHA* were run with the above heuristics (A*, WA* with the consistent ones only) on 7 different board sizes and with 7 different weight combinations. Note that all of the following data and graphs are from using IMHA* and SMHA* with four heuristics, 1 admissible and consistent, the other 3 not necessarily. Further research can be done with different amounts of heuristics, like with 2,3,5 or some other larger number.

The 7 different board sizes are: $8 \times 4,\ 9 \times 5, 10 \times 6,$ $11 \times 7, 12 \times 8, 13 \times 9,\ 14 \times 10.$

For each board size, 3 Pacman mazes were randomly generated, randomizing the food locations, initial Pacman agent position and walls of the maze.

Weight combinations for IMHA* and SMHA*:

1. $w_1 = \sqrt{1.5}, w_2 = \sqrt{1.5}$
2. $w_1 = \sqrt{1.5}, w_2 = \sqrt{2}$
3. $w_1 = \sqrt{2}, w_2 = \sqrt{1.5}$
4. $w_1 = \sqrt{2}, w_2 = \sqrt{2}$
5. $w_1 = 10, w_2 = 10$
6. $w_1 = 1, w_2 = 10$
7. $w_1 = 3, w_2 = 3$
8. $w_1 = 1, w_2 = 1$

From now on these weight combinations will be referred to by using tuple notation: (w1, w2).

For WA* the following weights were used so that it would have the same upper bound on optimality of solution costs as the MHA* algorithms, so $w=w_1*w_2$:

1. $w = 1.5$
2. $w = \sqrt{3}$
3. $w = \sqrt{3}$
4. $w = 2$
5. $w = 100$
6. $w = 10$
7. $w = 9$
8. $w = 1$

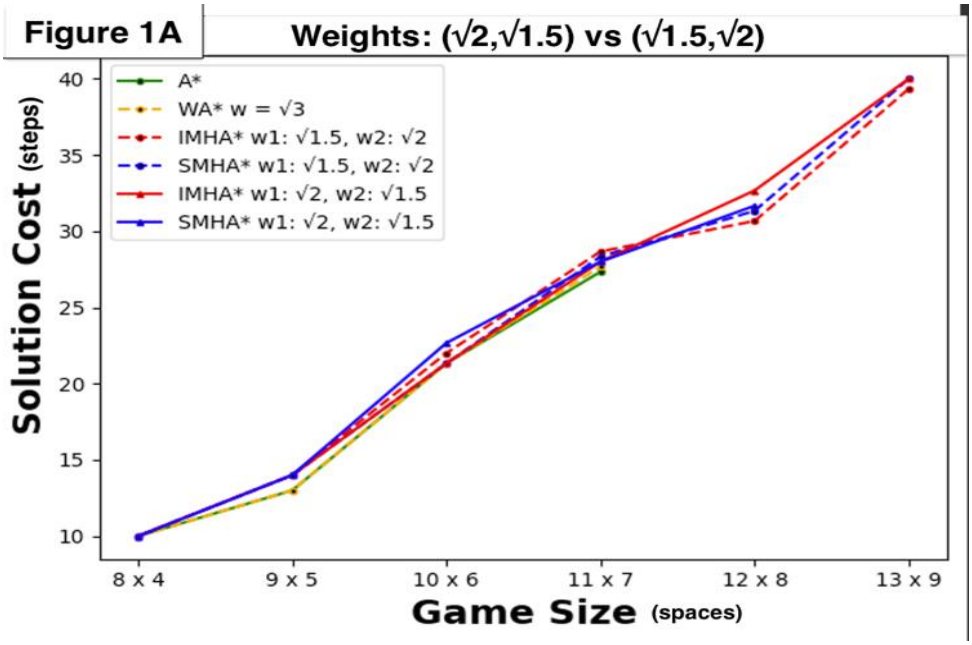
Note however that w =100 and w=1 were not run, because w=1 is exactly A* and w=100 takes too long because in a research paper in the references section at the end, apparently increasing weights slows down WA* significantly in some search domains, including this one as found when attempting.

The performance of the 4 algorithms was judged based on the following 3 parameters:

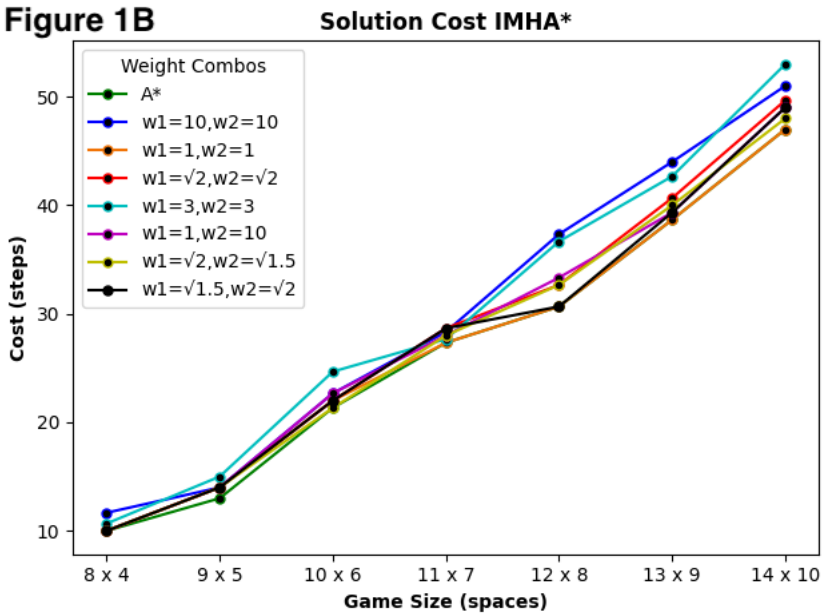
1. **Running Time**
2. **Expanded Nodes**
3. **Solution Cost**

For each category the average values were taken for the 3 generated Pacman mazes for each board size. Pacman agents had 90 seconds to find a solution and if none was found the algorithm was terminated. If a particular agent did not solve a problem in under 90 seconds for a specific board size, then the average for that size wasn't calculated. Note that all the data shown in the graphs below are ***averages*** of the three times each algorithm ran on a different board of a specific size.

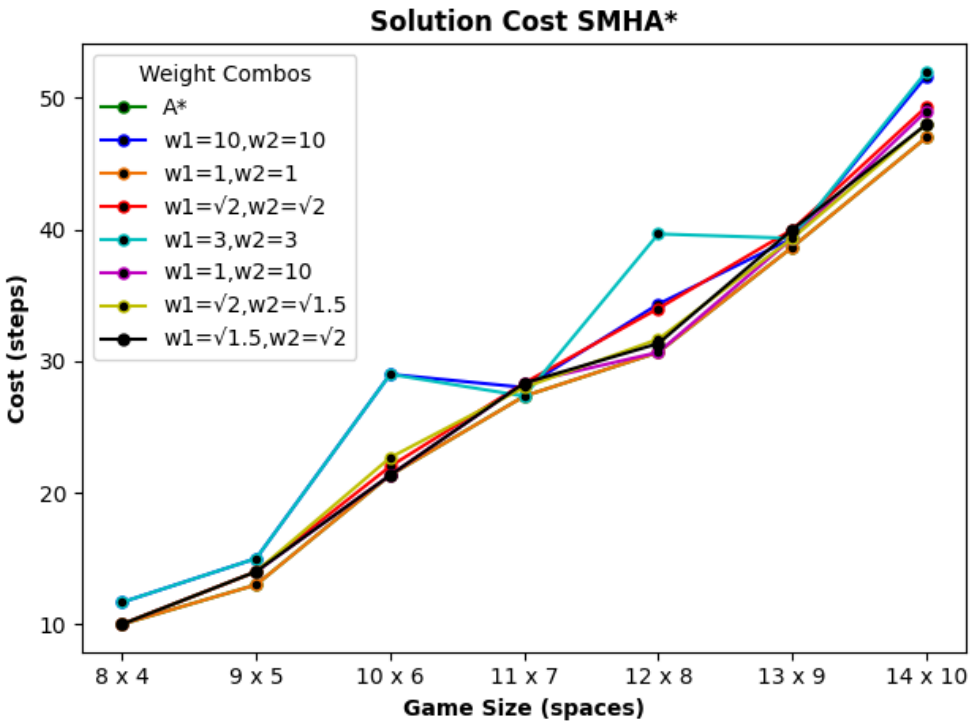
Solution Cost:



- **Description:** Figures 1.A shows the average solution cost each algorithm returned using specific weights of (√2, √1.5) and (√1.5, √2) for each Pacman maze size. Note that the weight for WA* is equal to w1*w2. If an algo did not finish solving one of its boards in under 90seconds it was omitted from the data. Therefore A* and WA* couldn't go past size 12x8.
- The data shows that solution cost is very similar for all algorithms.
- These are good results since the weighted algorithms are well within their sub optimality bounds (up until the size 11×7). The graph shows that MHA* algorithms have similar costs to A*, and that switching w1 with w2 has minimal effects on optimality of solution

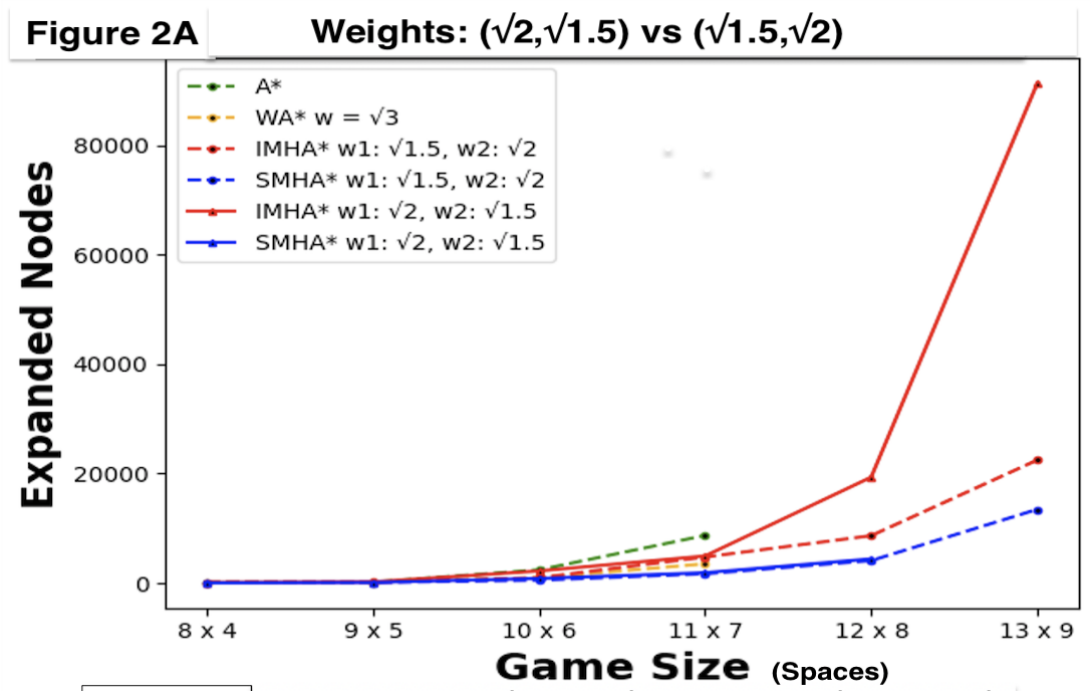


- **Description:** This graph shows the solution costs of IMHA* with all the tested weight combinations against A* which is in green, and does so for each board size.
  In this graph algorithms were not filtered out if they could not finish in 90 seconds.
- Most of the solution costs are very similar to A* except for (3,3) and (10,10) which return highly suboptimal solutions.
- A* is obscured by the orange (1,1) line for larger boards, where IMHA* returned the optimal solution as well
- The larger the board size the more variation from A*'s optimal path cost result.

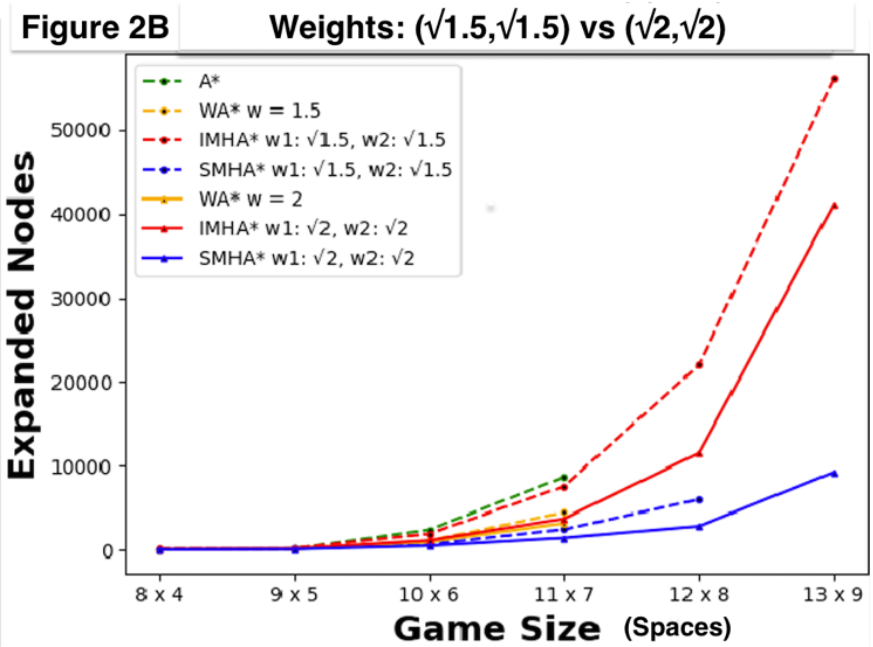


- **Description:** This graph shows the solution costs of SMHA* with all the tested weight combinations against A* which is in green, and does so for each board size.
  In this graph algorithms were not filtered out if they could not finish in 90 seconds.
- A* continued all the way, but it is obscured by the orange (1,1) line for ALL boards this time, so SMHA* with weight (1,1) returned the optimal solution on average for each board size
- SMHA* with weight (3,3) produced extremely suboptimal results, as much or moreso than (10,10), which is unexpected

Expanded Nodes:

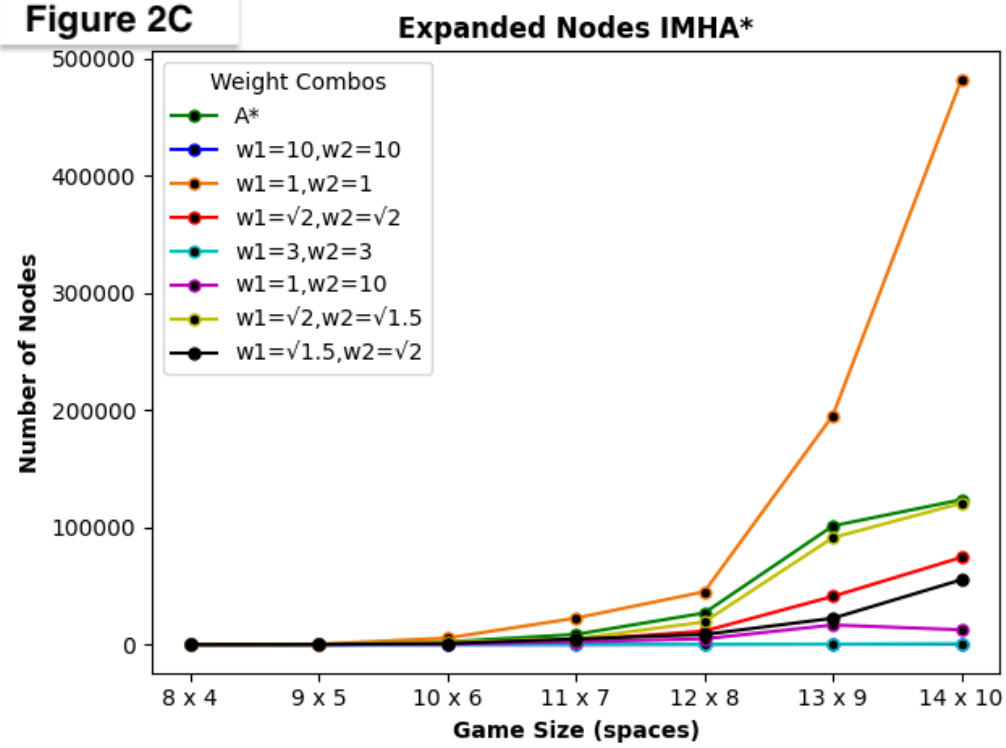


- **Description:** Figure 2A shows the average expanded nodes for each board size of all the algorithms using weights (√2, √1.5) and (√1.5, √2). If an algo did not finish in under 90s it was filtered out
- Clear that (√2, √1.5) expands more nodes when using IMHA* and SMHA* than when using (√1.5, √2).
- While the two blue SMHA* plotlines almost overlap, (√2, √1.5) comes out slightly ahead, and when zooming in this is significant, because the range the two lines is in is from 0 to 20000, so they look almost the same but they are not.



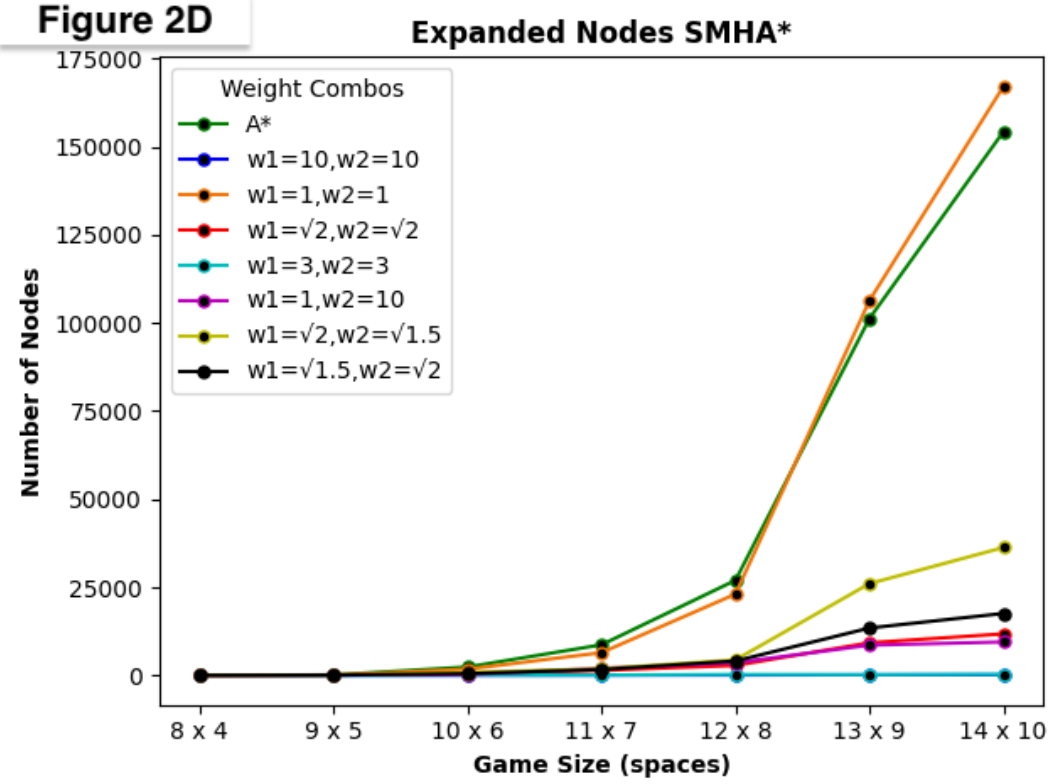
- **Description:** Figures 2B shows the average expanded nodes for each board size of all the algorithms using weights (√1.5, √1.5) and (√2, √2). If an algo did not finish in under 90s it was filtered out
- From the data it follows that for both algorithms more nodes were expanded for (√1.5, √1.5) than for (√2, √2).
- WA* expanded less nodes than IMHA* surprisingly. Addressed in analysis

Figure 2C



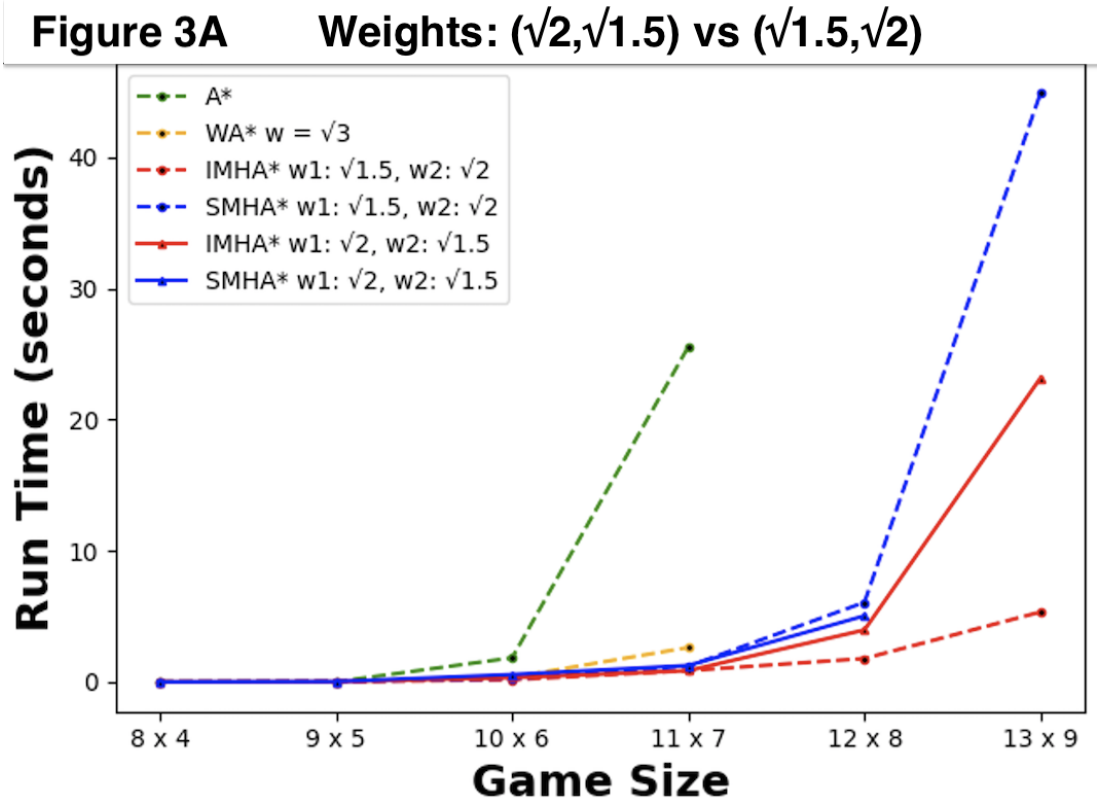
- **Description:** Figure 2C shows the number of nodes expanded by IMHA* for all its different weight combinations per each board size. A* is in green as usual. No 90second time limit imposed.
- Note how A* expands more nodes than all IMHA* weight combinations, except for IMHA* with (1,1), which in 13x9 expands double the amount A* does, and in 14x10 it more than triples the amount.
- Note that (3,3) and (10,10) overlap and on average expand effectively zero nodes. Will be addressed in analysis

Figure 2D

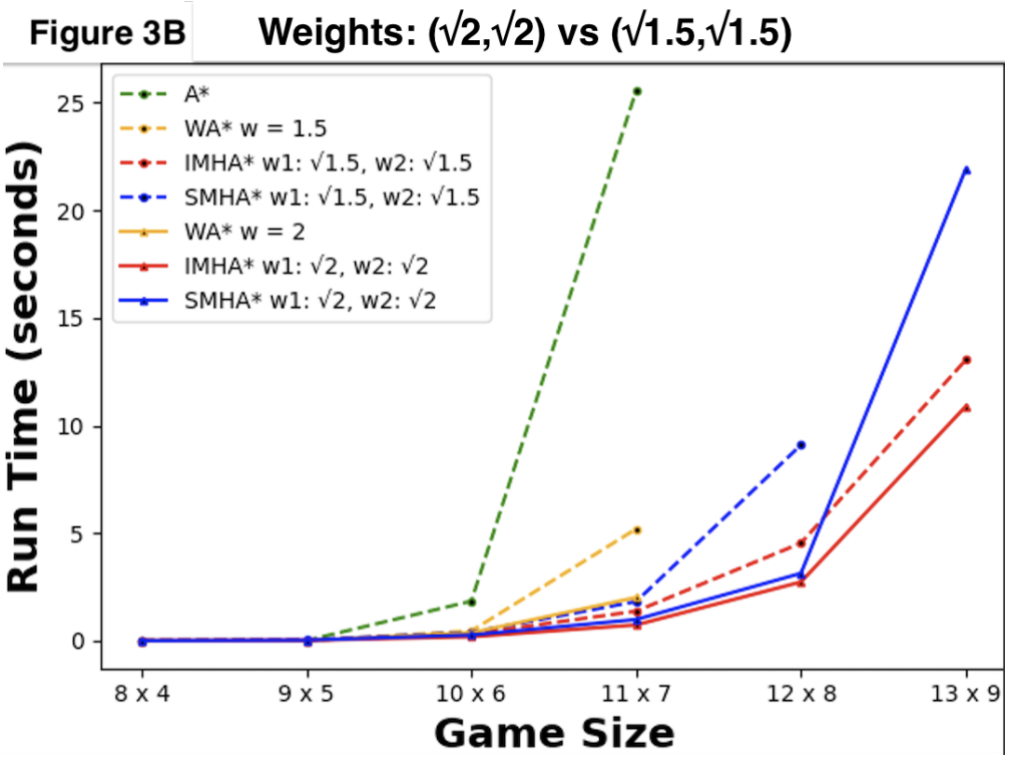


- **Description:** Figure 2D shows the number of nodes expanded per game size by different weight combos of SMHA* compared with A* in green.
- A* expands by far the most nodes, other than SMHA* (1,1) which passes it at 13x9 and 14x10
- Again (3,3) and (10,10) overlap, expanding effectively zero nodes. Note that (1,10) expands more than (3,3)

Run Time:

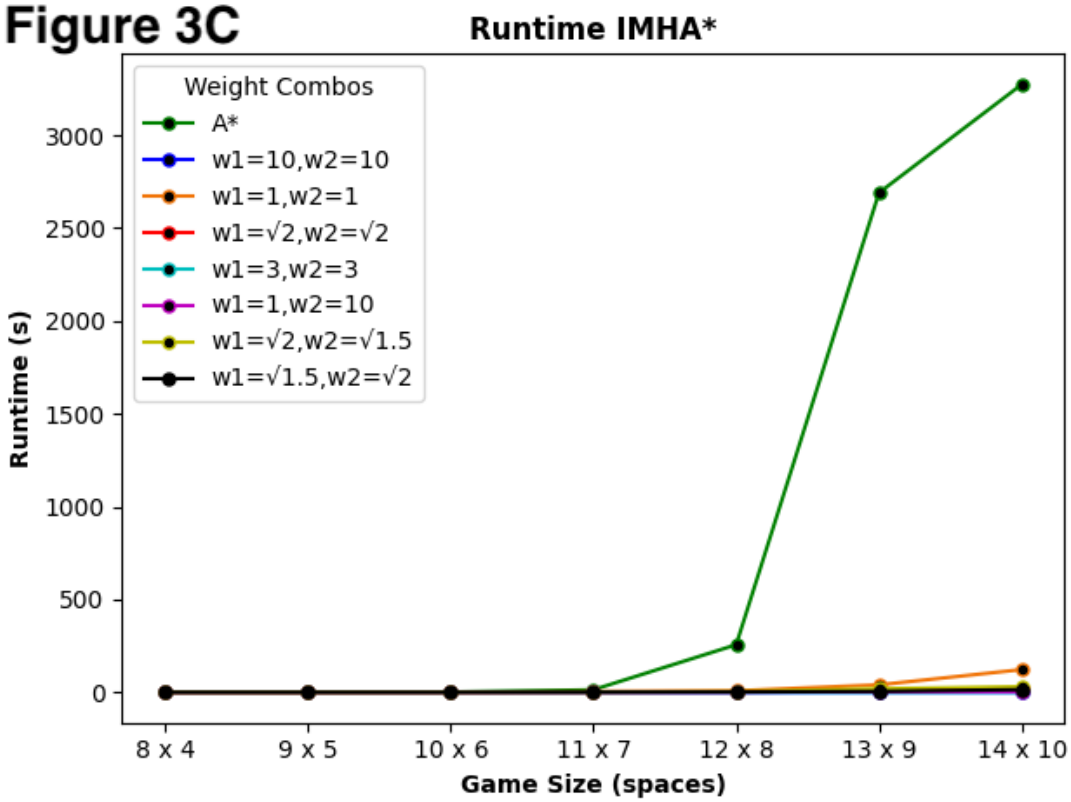


- **Description:** Figure 3A describes the runtime of each algorithm with weights (√2, √1.5) and (√1.5, √2), per board size. If algo did not finish in under 90seconds it was omitted.
- For IMHA*, (√2, √1.5) has longer runtime than (√1.5, √2). For SMHA*, until 12x8, (√2, √1.5) seems to run slightly longer, but at 12x8 they switch. Unclear from graph what happens at larger boards. See graph 3F for a better view.
- A* and WA* had much longer runtimes than all others, especially from 12x8 and onward, as shown above.



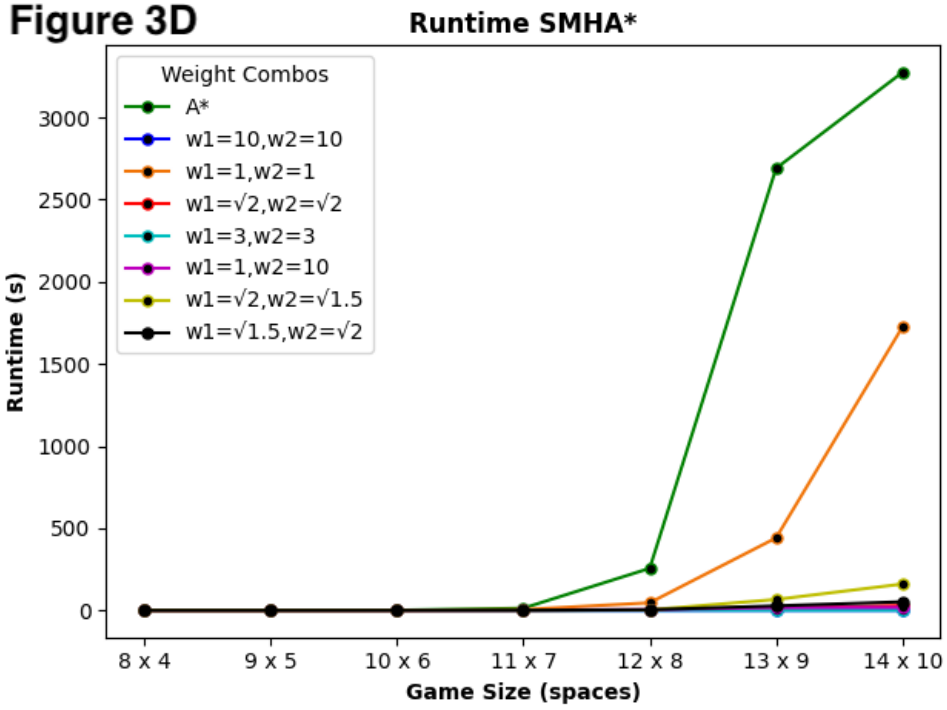
- **Description:** Figure 3B above describes the runtime of each algorithm with weights (√1.5, √1.5) and (√2, √2), per board size. If algo did not finish in under 90seconds it was omitted.
- For both IMHA* and SMHA*, (√1.5, √1.5) has longer runtimes than for when using (√2, √2) weight combo.
- A* and WA* have significantly worse runtimes than the others, A* is particularly bad.

Figure 3C

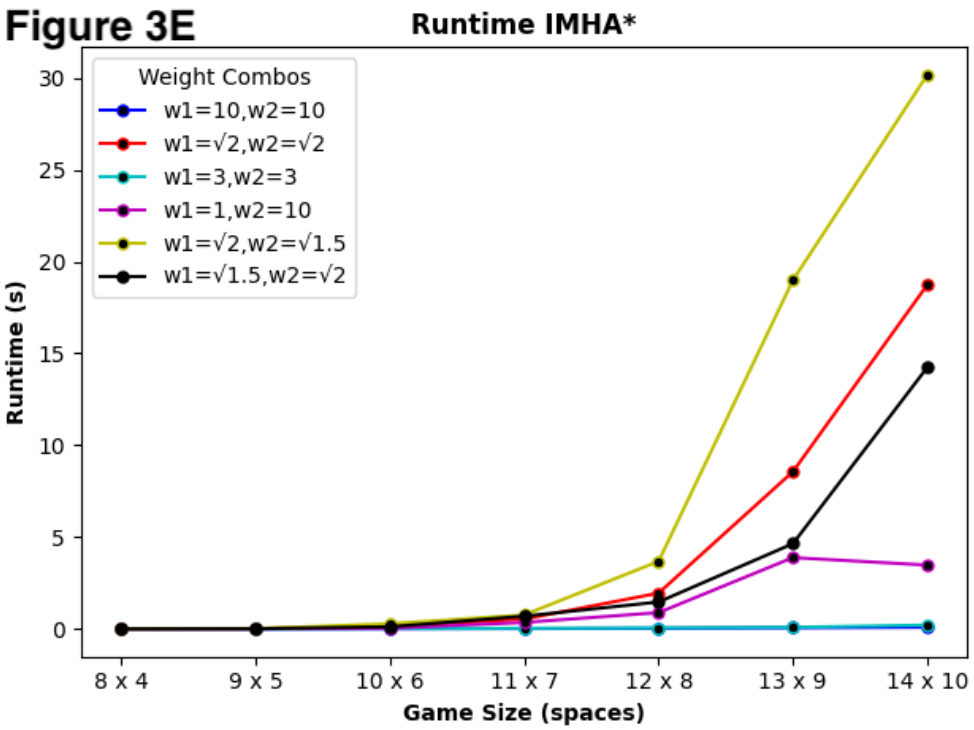


- **Description:** Figure 3C describes runtimes of IMHA* with different weights and against A* in green, all per board size. A* is so much slower that all the data for IMHA* is indiscernible.
- Note IMHA* (1,1) in orange expands 5x the nodes A* does at 14x10, yet the runtime is more than 6x less.
- A zoomed in version will be shown in Figure 3E below

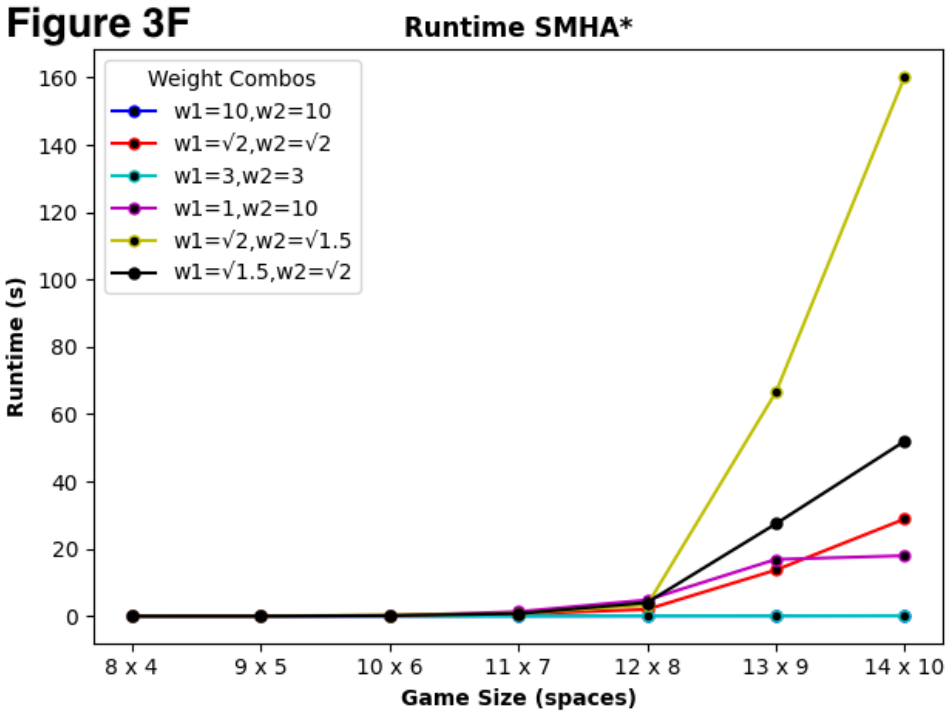
Figure 3D



- **Description:** Figure 3D above describes the runtimes of SMHA* algorithms of different weights against each other and against A* in green, all per board size. From here, SMHA* (1,1) is slower in runtime than the other weights, but still far from A*'s. The other SMHA* weight combos are bunched up together however.
- A zoomed in version will be shown in Figure 3F below



- **Description:** Figure 3E shows the zoomed in runtime of IMHA* with different weights per board size. A* and IMHA* with (1,1) are omitted for greater clarity.
- (3,3) and (10,10) are together again as they were with expanded nodes, approaching zero runtime. Addressed in analysis.
- Interesting to note that (1,10) has slower runtimes than (3,3), despite its suboptimality upper bound being slightly higher. Addressed in analysis.



- **Description:** Figure 3F above shows the zoomed in runtime of SMHA* with different weights per board size. A* and IMHA* with (1,1) are omitted for greater clarity.

- (3,3) and (10,10) are together again as they were with expanded nodes, approaching zero runtime. Addressed in analysis.
- (1,10) has slower runtimes than (3,3) as well, despite its suboptimality upper bound being slightly higher. Addressed in analysis.

IV. Analysis

The Analysis section is divided into four parts:

1) Solution Cost
2) Expanded Nodes
3) Runtime
4) Full Picture Analysis

Section1: Solution Cost:

Figures 1B and 1C show that for smaller weights, the solution cost for each algorithm is only slightly less optimal than for A*. This is a good result since this means that MHA* algorithms really achieve the goal, a slight decrease in optimality results in a significant increase in speed and decreased usage of memory, which is what node expansion signifies.

From figure 1A it is noteworthy that exchanging w1 and w2 does not have a significant effect on the optimality for either MHA* algorithm, at least for smaller differences between the weights. This is most likely because the suboptimality is bounded by w1*w2 (as proven in the research paper), and so exchanging the weights does not change the upper suboptimality bound (it does change other factors however as will be seen below in the next sections). However at the same time, their cost data is not identical. This can be explained by the fact that w1 weights the heuristic while w2 is the primary factor that binds suboptimality (see the pseudocode in the article, lines 21 for IMHA* and line 29 for SMHA*). Therefore exchanging these weights yields differing results.

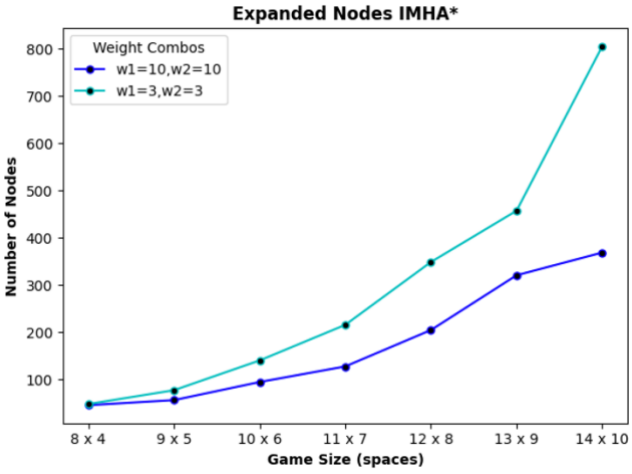
From figure 1B and 1C there are multiple intriguing results, specifically as solution cost relates to runtime and expanded nodes, but this will be addressed further on. For (1,1), it was expected that IMHA* and SMHA* would behave like A*, and indeed with regard to optimality, they return the optimal paths in all the weight combination trials that were run. The green A* plotline cannot be seen because the orange (1,1) plotline completely overlaps it in both diagrams. For (10,10) the paths are significantly less optimal, which is also consistent with what was expected. There is a point of convergence where the suboptimality of a solution cannot get worse, and this was verified with weights of (4000,4000), but the data is not shown here.

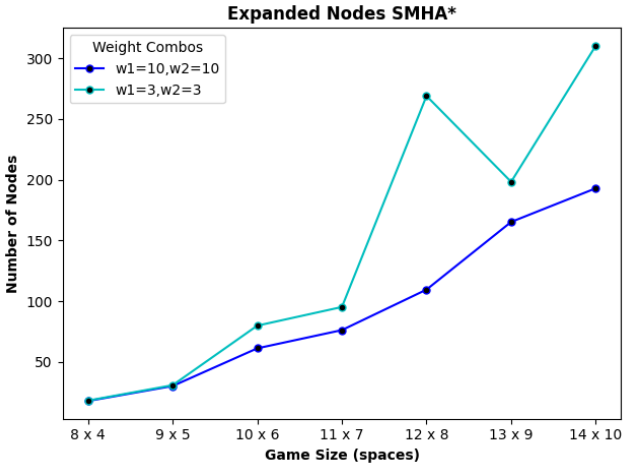
Another interesting observation is that (3,3) returned a much worse solution than (10,10) for SMHA* for board size of 12x8, and a slightly worse solution for 14x10. For IMHA*, (10,10) outperformed (3,3) surprisingly for boards 9x5, 10x6, and 14x10. It is unclear why this is, but a possible explanation could be that the layout of the walls, food, and Pacman's initial position created conditions where the weighted heuristic values happened to align with a more optimal solution for (10,10). However this needs to be tested further.

Another important note is that in 1B and 1C, for (1,10), while technically having a higher suboptimality bound of w1*w2=1*10 > 9 (which is the suboptimality bound of (3,3)), this weight combination consistently found as good or better of a path. This is also unclear as to why this is, however it points to the fact that inflating the heuristic cost (w1) has a more potent effect on the suboptimality than w2. It could be that the times when the inflated heuristic values passes the suboptimality check of the algorithm causes the solution to be more suboptimal than an uninflated heuristic value (when w1=1) with a more lenient suboptimality check (w2). However further testing is required to verify this postulate.

Section 2: Expanded Nodes:

From figures 2C and 2D, there are multiple interesting results that were received. For both IMHA* and SMHA* the (3,3) and (10,10) weight combinations opened extremely few nodes compared to the rest of the algorithms. Their lines in these graphs are overlapping, but when zoomed both are visible as shown in the graph below:





Sometimes they expanded below 100 nodes, and only on board 14x10 did they expand much at all, for IMHA* around 800 nodes on average and SMHA* around 300 nodes on average. This ratio between them makes sense since SMHA* expands each node at most twice, while IMHA* expands each node at most n+1 times such that n is the number of heuristics used (in this case we used 4 heuristics so n=5). Therefore, it is very possible that about 150 nodes were expanded by SMHA* twice (for a total of 300) and for IMHA* each was expanded 5 times (for a total of around 750 which is close to 800). When zooming in on the data of just (10,10) and (3,3) for both IMHA* and SMHA*, weight combination (3,3) consistently expands more nodes than (10,10). This makes sense since the more weighted the heuristic, usually the less nodes are expanded since nodes closer to the goal are preferred, similar to WA*.

When comparing weight combinations (1,10) and (3,3), (3,3) has much less expanded nodes in both MHA* algorithms. This makes sense since when w1 is inflated, the entire algorithm can skip faster towards finding a solution, despite the fact that the suboptimality bound (w2) is much higher. It is likely that none of the heuristics being used ever return values near to 10 times that of the admissible heuristic, while if inflated by 3 it does seem more likely to arrive at 3 times the admissible heuristic value. (See the pseudocode of the algorithm for how the suboptimality check works in lines 2 and 21 for IMHA* and lines 2 and 29 for SMHA*).

With regard to weight combination (1,1) for SMHA*, the amount of nodes expanded was about the same as for A*, which is a very surprising result since SMHA* is known for expanding less nodes overall, but on the other hand makes sense since SMHA* technically becomes A* with weights (1,1) since there is no inflation and the suboptimality bound is purely from the admissible heuristic. Also note that SMHA* returned the optimal solution every time for combination (1,1). For IMHA* however, (1,1) expanded astronomically more nodes than astar or in fact any other algorithm anywhere in this project. This is true particularly for 14x10 boards. Yet somehow the runtime still easily surpassed that of astar and SMHA* for this weight combination. The reason it expanded so many nodes is definitely because each node is expanded at most n+1 times as opposed to the others . However the runtime is surprising and will be addressed in the runtime section below. This is the most jarring example of the fact that the amount of expanded nodes is not necessarily correlated with runtime. IMHA* also returned the optimal solution for each board when using this weight combination as mentioned previously.

For weight combinations ($\sqrt{2},\sqrt{2}$) , ($\sqrt{1.5},\sqrt{2}$), ($\sqrt{2},\sqrt{1.5}$), and ($\sqrt{1.5},\sqrt{1.5}$) Figures 2A and 2B are the most informative graphs with regard to their behavior. From 2A it is clear that for both IMHA* and SMHA* that ($\sqrt{2},\sqrt{1.5}$) expands more nodes than ($\sqrt{1.5},\sqrt{2}$), while from 2B it is clear that for both algorithms ($\sqrt{1.5},\sqrt{1.5}$) expands more than ($\sqrt{2},\sqrt{2}$). This result in 2B is consistent with the expectation that higher suboptimality bounds cause the heuristic to become slightly less admissible and therefore faster. The result in 2A is highly informative, since at these smaller weight scales, it seems that higher suboptimality upper bounds allow for the algorithm to progress further and more easily through the search space. The inadmissible heuristics themselves are not exaggeratedly inadmissible, therefore a slight nudge of inflation from w1 and enough leeway from w2 allow for better performance than when w2 < w1, which has more inflation and a lower chance of passing the given suboptimality bound.

Overall the general trend from all these graphs on expanded nodes (particularly 1D and 1E) seems to be that the more suboptimal the weight combinations cause the algorithm to be, the less nodes are expanded, and the differences are in the tens of thousands of nodes, which is significant. Regarding WA*, occasionally it expands more nodes than a comparable IMHA* algorithm and sometimes less, more comprehensive data needs to be generated for conclusions to be made. WA* also seems to expand less nodes than comparable SMHA* runs, but this is not surprising since SMHA* is a faster algorithm overall and prioritizes expanding less nodes. Another overall trend is that SMHA* opens many less nodes than IMHA* does, yet as seen in the next section, the runtime relation between them is surprisingly reversed. This is discussed below.

Section 3: Runtime

Regarding runtime, A* and W* are far outperformed by IMHA* and SMHA*. These results are particularly stark for board sizes larger than 11x7 in which WA* and A* took hours to complete if at all, hence parts of the incomplete data for these board sizes.

Even while both SMHA* and IMHA* expanded as many or more nodes for weight combination (1,1) as shown previously in figures 2C and 2D, their runtime was still vastly superior to A*'s as seen in 3C and 3D (and note that A* technically has weight combination of (1,1)). The reason for this must be the combination of heuristics being used in the MHA* algorithms. While IMHA* can expand nodes n+1 times, if one priority queue being used is moving closer to the goal at a faster rate than the others, then when it finds the solution faster, the algorithm ends faster. A* has only one heuristic to work with in comparison, which can be hindered by depression regions in the search space with no alternative direction. For SMHA*, the use of multiple heuristics actually makes a shared path (as described above in the introductions) which allows the complementary heuristics to avoid depression regions all together. Seemingly the heuristics chosen have complementary depression regions. An exact example of this using the heuristics in this project needs to be found by finding these depression region, a topic that can be pursued in a further research project.

For considering the runtimes differences of all weight combinations other than (1,1), more zoomed in graphs 3A,3B,3E, and 3F need to be used.

For weight combinations (10,10) and (3,3), their runtimes were consistently below 1 second for all board sizes. They also expanded far less nodes than all others and returned some of the most suboptimal paths for these searches, as seen from the solution cost section graphs, Figures 1D and 1E.

For weight combination (1,10), SMHA* takes about double the time that IMHA* takes but both finish all boards in under 20 seconds. As mentioned before, (1,10) has a lot more expanded nodes than (3,3) or (10,10), therefore this runtime is appropriate.

To compare weight combinations, (√1.5,√2), (√2,√1.5), and (√1.5,√1.5), (√2,√2) , figures 3A and 3B will be considered. In figure 3A, (√1.5,√2) and (√2,√1.5) are compared, and for each board combination that solutions were generated for in this graph, SMHA* had a larger runtime than IMHA* for both combos. It is interesting to note that in this graph, SMHA* with (√2,√1.5) was slower (as more clearly seen in Figure 3F) while with (√1.5,√2) was faster, and same with IMHA*. The lone exception is for board size 12x8 where the runtimes are switched for these weight combos, while for IMHA* no change occurs. This outlier in 12x8 can be explained away as a fluke based on the randomness of the boards generated, especially since these values are averages anyway. The placement of Pacman, the food, and the inner walls can easily have this kind of impact. As noted previously, the amount of nodes expanded by (√2,√1.5) was more than expanded by (√1.5,√2), which affords these findings with consistency, and a potential reason has already been discussed in the Expanded Nodes section.

For combination (√1.5,√1.5), (√2,√2), from Figure 3B it follows clearly that IMHA* is faster than SMHA* for comparable weights, no exceptions from the data averages here. Combination (√1.5,√1.5) clearly runs slower than (√2,√2) based on the data for both MHA* algorithms. This also fits with the previous conclusions, since (√2,√2) expanded fewer nodes than (√1.5,√1.5) did for both algorithms. This was also discussed previously in the Expanded Nodes section.

Overall, IMHA* is found to have better runtimes than SMHA* in almost all cases when compared using the same weight combinations. The largest contributor to this result is the node expansion step of SMHA*, which is significantly more expensive than in IMHA*. The reason for this is because SMHA* updates all its priority queues at the same time upon each favorable expansion of a node. IMHA* on the other hand, only updates the single priority queue belonging to the heuristic search that expanded that node. Therefore in the Pacman search problem, the advantages of having a shared path are outweighed (but only slightly) by the expensiveness of the expansion stage. The outlier is for weight combination (1,1) where from figures 3C and 3D the difference is a whopping ~1500 second difference, where SMHA* is by far slower. But using (1,1) on these searches almost defeats the purpose of the MHA* search algorithms, because the greatest advantages of these algorithms is in the weight values w1 and w2.

Section 4: Full Picture Analysis:

From the data, IMHA* and SMHA* have seemed to accomplish the purpose of their creation. Multiple arbitrarily inadmissible heuristics were used along with a simple and not so special admissible heuristic (Manhattan Distance), and the results were rather incredible. IMHA*'s speed far surpassed its predecessors of A* and WA* and even in cases where no suboptimality can occur to speed up the search (like for weight combination (1,1)). While IMHA*'s amount of nodes expanded was far greater than all other algorithms, it was even faster than SMHA* which also has the advantage of multiple heuristics. SMHA* was special in that it almost always expanded less nodes than WA* and A*, in addition to the improved runtimes and relative optimality. The reasons for these massive improvements in runtime is most likely because of a combination of the inadmissible heuristics themselves along with the avoidance of depression regions. As explained previously in the Runtime section, A* and WA* have only one heuristic to turn to to guide them to a solution, while IMHA* and SMHA* have multiple that can work together to avoid these depression regions. While this area was not researched, it probably can be assumed that since the heuristics used were mostly very different in their assessment of the current Pacman board, that their depression regions can be avoided or chosen between. For example, the Four Corners heuristic may have a depression region in boards where there is food in each corner of the board, which means it can rule out any of them, so its choice for a next step may not be good until that point. Therefore, another heuristic such as Max Manhattan Distance can help it make the choice of where to go and guide it. Then, once more of the food is cleared, Four Corners may hypothetically be more powerful. This is conjecture however and needs to be researched. Despite this need for further research, the results of this project can be considered a success.

Three issues will be addressed in the following paragraphs. The first issue that has come up a lot in the analysis is whether the amount of expanded nodes and runtime are correlated. Intuitively it makes sense that they would be, since the bulk of the work in A*-like searches is the relatively expensive node expansion to find a solution path. This indeed is what was witnessed from the data for each algorithm as compared to itself; the more nodes expanded using the specified weight combination and board size, the longer the runtime, and vice versa. For example, for IMHA* and SMHA*, weights (3,3) and (10,10) both expanded extremely few nodes and had below 1 second of average runtime. However when different algorithms were compared, different results occurred. IMHA* expanded tens of thousands of more nodes than A* or WA*, yet it finished many hours faster on some of the same boards, sometimes finding the optimal path, as shown for weight combination (1,1) where no suboptimality is allowed. This can only be attributed to the multi-heuristic property of IMHA*. The ability to choose the best path from several searches is a very powerful tool that indeed improves results. Therefore, the amount of nodes expanded is not indicative of IMHA*'s runtime as compared to A*. Another example where the correlation is debunked when look across algorithms is when comparing SMHA* and IMHA*. SMHA* expanded much less nodes than even A* or WA* and of therefore of course less than IMHA*, yet IMHA* was still so much faster. The way to explain this is as explained in the Runtime section about how much more expensive expanding nodes is in SMHA*. If this process could be made more efficient with some tweaks, then it is very likely that SMHA* can be faster than IMHA*.

The second issue was only realized too late in the research process. All the data compares data on paths found on different board sizes. Board sizes that are the same were averaged together, independent of the randomly generated properties of the board itself. While the data came out

intelligible and therefore seems to indicate a correlation between increased runtime, and expansion of nodes as board sizes increase, it is possible that a smaller board size could have a longer optimal path to find the food, and therefore could take IMHA* or SMHA* longer to solve it than on a larger board. As a suggestion for further research, an effort should be made to find boards with similar or different sizes but with the same optimal path costs and data should be collected from these kinds of controlled runs.

The third issue that needs to be addressed is the discrepancies between the results of this paper and of the research papers the MHA* algorithms were adapted from. Their results showed most significantly that SMHA* performed faster than IMHA* across multiple search problems, as complex as a robot finding an item in a building and as simple as the eight-puzzle. This is not what the data of this paper shows. There are several possible explanations for this discrepancy, the first of which is that Pacman may be a purely different problem for some unknown fundamental reason. This really could be since IMHA* and SMHA* have not been tested on every search problem in existence (and never will be) and there are so many variables in each search space that it is likely that there are an abundance of cases where IMHA* performs faster and better than SMHA* in addition to Pacman. Another suggestion could be that the implementation of these algorithms in the code used for this paper could be tweaked to make SMHA* more efficient as alluded to earlier, specifically regarding node expansion. A possible implementation would be to instead of have n+1 priority queues that are updated on each expansion, instead implement one master priority queue that is actually shared between all the heuristic search problems going on within SMHA*, as opposed to each priority queue having the same nodes inside it. The pseudocode would a large overhaul for this to happen, but the runtime improvements could be significant.

Another discrepancy between this paper and the other is that they make the claim that SMHA* is more memory intensive than IMHA* (page 25). This assertion was given with no explanation within the paper and seems to contradict the basic intuition that IMHA* is built to be able to expand each node at most n+1 if need be. In the results in this paper IMHA* expanded the most nodes generally and often by tens of thousands. However it is possible that the overhead created by update n+1 priority queues in node in expansion in SMHA* accounts for this claimed memory intensiveness, and therefore would be solved by carrying out the suggested implementation above.

V. Conclusions and Further Study

- IMHA* and SMHA* seem to be better algorithms and help achieved the objectives of this paper by finding solutions faster without compromising completeness (as shown in the article from Carnegie Mellon) and by minimally sacrificing suboptimality. Heuristics were used despite not being proven consistent or admissible, as hoped for, and depression regions were presumably (moreso) avoided than in A* and WA*.
- Further testing is necessary for IMHA* and SMHA* with bigger layouts, more and less heuristics, and with better and worse heuristics of different kinds. More testing needs to be done on boards of different sizes with similar optimal path costs, and with boards of similar sizes and optimal path costs. Boards with different shapes, like circles and triangles or a randomized blob shaped board could also yield interesting results.
- Heuristics used should be analyzed for their exact depression regions so that way they can be paired up with complementary heuristics that fill in those disadvantages and then IMHA* and SMHA* should be run.
- Another interesting area that can be researched is the following: There are multiple points of convergence to the suboptimality or optimality of the path costs that IMHA* and SMHA* find. These points of convergence can be found on specific boards with specific internal variables (food location, Pacman location, etc) and by changing the weight combinations and determining at what weight combinations they converge to the optimal or suboptimal solutions.
- An improvement than can be made is running and attaining all the necessary data on WA* which was lacking because of time constraints, some of these runs were taking hours and were not finishing and A* was prioritized. The data would be more complete if WA* was more involved, although most of the conclusion would remain the same most likely, except for the occasional outlier in the data.
-

VI. How to run the algorithms?

See the README attached with the code.

VII. References

[1] Hebrew University - 67842 – Introduction to Artificial Intelligence Course Slides Weeks 1-3

[2] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA. (2015). *Multi-Heuristic A**. Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang and Maxim Likhachev. https://www.cs.cmu.edu/~maxim/files/mha_ijrr15.pdf

[3] *A* and Weighted A* Search*. (2015). Carnegie Mellon University CS. https://www.cs.cmu.edu/~motionplanning/lecture/Asearch_v8.pdf

[4] Univeristy of Berkeley AI Course Search http://ai.berkeley.edu/project_overview.html

[5] *When does Weighted A* Fail? University of New Hampshire CS.* https://www.cs.unh.edu/~ruml/papers/wted-astar-socs-12.pdf