**Project Two Summary**

**Breakdown**

The heuristic can be viewed as being built one column at a time, going from the bottom to top of each. Three nested four loops are used to compare the *current state* to the *goal state*.

The first calculation done is a difference estimation between the length of a column of the *goal state* and length of the same indexed column in the *current state*. This prevents an out of bounds error from occurring, and in the case that the *current state* has more nodes in a column than the *goal state* then that difference is multiplied by two and added to the weight. This is because all of these blocks will need to be moved out of that column to eventually reach the goal (the two times multiplier was experimentally determined).

The blocks are then compared one by one within each column. If a match is found, meaning that a block is in the exact same position in both the *goal state* and *current state*, then it is evaluated if there are any values below that block that are out of place (this evaluation is done through the use of a boolean flag that will be set to true before each column iteration, but will change to false if two blocks are encountered that do not match). If there are no blocks below it out of place, then no weight will be added. If there are blocks below it out of place, then a weight of 4 will be added. This is because that block will have to be moved out of its correct position, so that the blocks below can be moved, before then being moved back (the weight of 4 was experimentally determined).

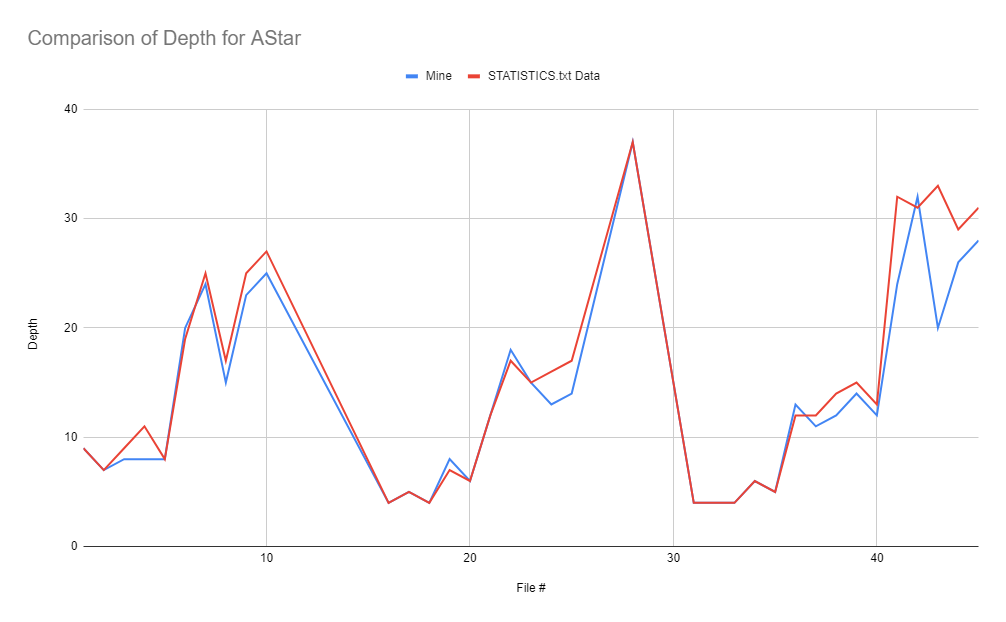
In the event that the blocks being compared between the *goal state* and *current state* do not match, then a weight of 2 is added, and the boolean flag for keeping track of whether two blocks do not match will be turned to false.

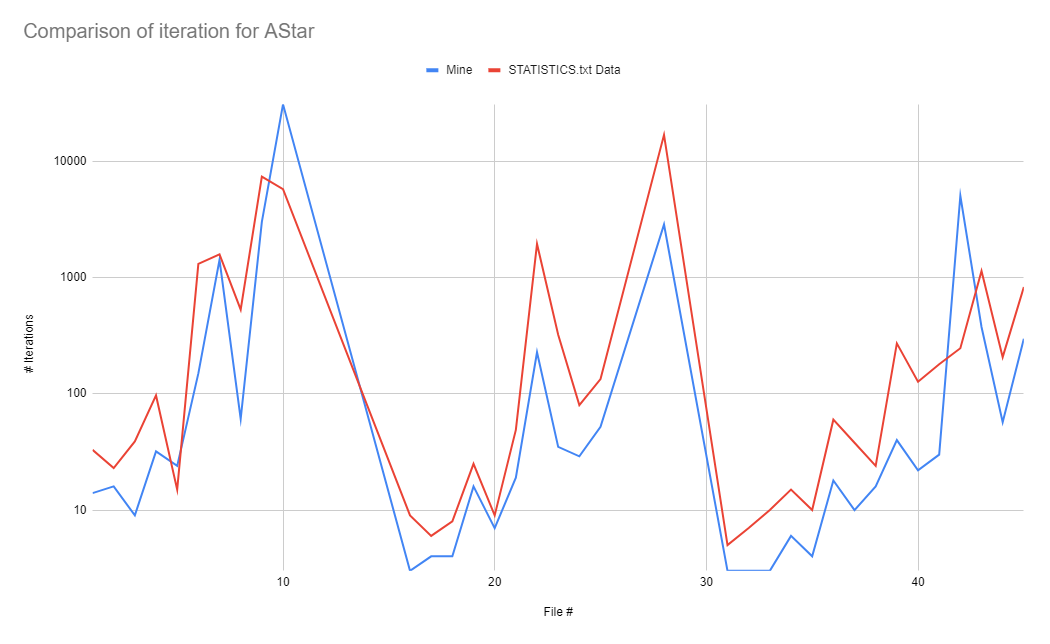
Another for loop is encountered at this point. This for loop compares the currently indexed value of *current state* to all the values in that column of *goal state*, with the exception of course of the value that was already compared. If it is found that a block is in a column that it belongs to, but it is not in the exact correct place, it will then incur an added weight of 2. This is because that block will have to be removed from the column and then returned to it.

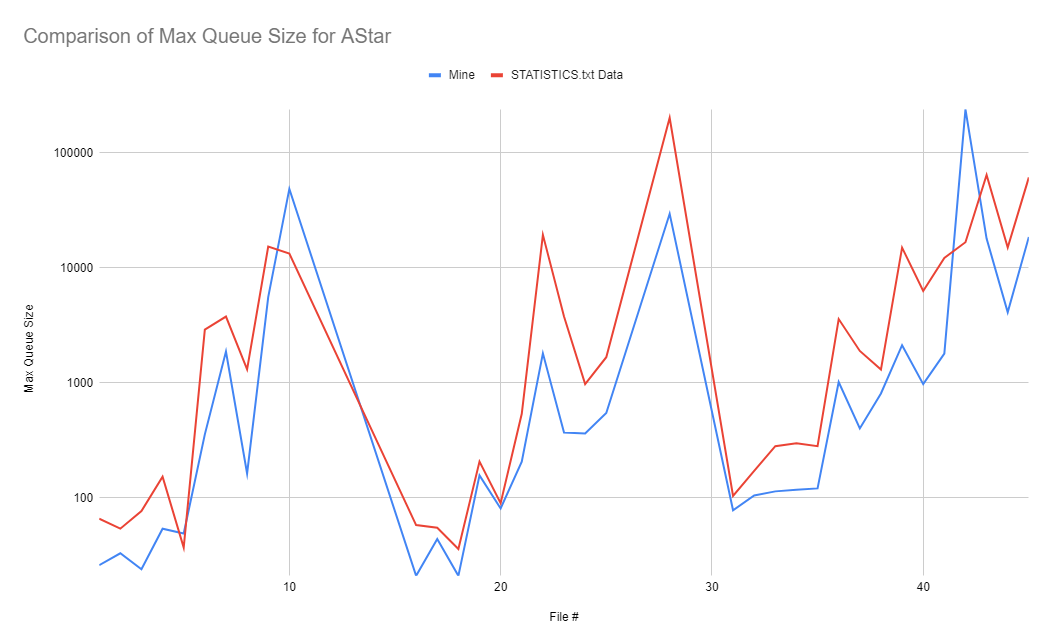
*\*A drawing can be see on the last page of this document to illustrate the h(n) calculation*

**Results**

Overall performance achieved was on par or better when compared to the STATISTICS.txt document **provided**. One noticeable difference in performance between the heuristics was that file *bwchp 27, 29,* & *30* failed according to STATISTICS.txt, but a solution was found with my heuristic. In total I was able to solve all but **six** of the challenge problems. The following are charts comparing my heuristics performance to that of the provided document;







(These graphs are not including the data points for the files that were unable to be computed in STATISTICS.txt)

It is apparent when looking at the data side by side, that my heuristic generated the answer in the same, if not slightly lower, depth. When comparing the number of iterations and max queue size it is difficult to assess as to which heuristic performed better. There is an obvious trend however between the three graphs when a more complex problem is encountered, as they all have sharp peaks in regards to the same file numbers. The formats that neither heuristic was able to solve included those with 3 Columns 20 Blocks, and 5 Columns 20 Blocks. It appears that the problem is considerable easier to solve when there is a low level of blocks and high number of columns, or just when those two values are relatively close to one another.

