Problem 5: Description of our own Heuristic vs Hamming Distance A*

We had several different implementations of our own heuristic functions which varied in how different it was from the original heuristic function (the hamming distance one) we had to implement. The *getOurPossibleActionHeur* function simply broke the ties given by the original heuristics function by adding 1 to the h(n) value if it found the same value already on the list. The *getOurPossibleActionHeur2* used the *getOurHeuristic* function, which got a different h(n) values by dividing the subtraction of the current node from the final node, and adding the absolute value of this to the hamming distance. *getMyHeuristic* was a function that broke ties by creating a greater h(n) value the closer the different digit position is to the ones digit.

As for its optimality, the *getOurPossibleActionHeur* is very similar to the original hamming distance heuristics so the difference is hardly noticeable. *getOurPossibleActionHeur2* also simply breaks the ties and keeps the values in original sorted order by associating different decimal values added to them and as such, the searching path distance is not different than the original hamming distance.

Problem 6: write a report on your assignment

Description of the problem and the algorithm used to solve problems 1-5

For all 5 of the problems we had to solve the puzzle of primes. We are given one prime number and another prime number of the same or fewer digits. The path from the first prime number has to consist of prime numbers as well and with a change of only one digit from the preceding number.

Our *getPossibleActions* with parameters *currentPrime* (a whole number) returns the list of prime numbers reachable from the current prime, not including the prime numbers that have already been used either in the frontier or the closed list. We take *currentPrime* and make it go through a loop that makes all possible mutations of the right most digit and checks to see if any of the mutations are prime numbers, by making it go through the helper functions *isPrime* with the parameter of number – this simply returns true if the number is prime and false otherwise. Any mutation of the right most digit of *currentPrime* that is prime eventually goes into the list

that gets returned (*listOfPrimes*). This is done to the rest of the digits in *currentPrime*; it is added to the *listOfPrimes*. The list is sorted and returned.

The pseudocode for *isPrime* is that it checks to see if all numbers up to the square root of the square that is the closest but smaller than *number* is able to divide the number evenly. If none of the numbers are able to divide *number* then return true else return false.

For problem one, we programmed a breadth first solver for the puzzle. This is an uninformed search in which "we expand the shallowest unexpanded node." (Lecture 3). We used a list of tuples that had each tuple consist of the current prime number and the the list of primes in the path to the current prime (including the current prime number). We have a set that keeps track of the nodes we have already visited. There is a while loop that goes through the list of tuples. In the loop, we pop the first element in the list of tuples and put its values in another tuple. We add the current tuple's first element into the visited set. If the current tuple's first element is equal to the prime we want to reach, then return the current path and end of function. Otherwise for all the possible next prime numbers - found out by the *getPossibleActions* method - if the prime hasn't been visited, add the prime number to the visited set and append the tuple of the prime number and the current path plus the path to the list of tuples. We keep doing all of this until the list of tuples has nothing in it.

For problem two, we programmed the depth-limit search algorithm. In the depth-first search in general we "expand the deepest unexpanded node." (Lecture 3) With the depth-limit, there is a limit to how deep we can go in the path from the starting prime. The depth limit was 5 where the root node has a depth of 0. This is a recursive function that takes the parameters: starting prime number or the current prime number we are on, the final prime number we need, the limit of the depth (which in this case is 5), the set of visited prime numbers, the path so far from the original starting prime to the current prime number, and the current layer. First if the current prime is the same as the final prime then we return the path with the current prime. If we are out of layers then return the current path without the current prime.

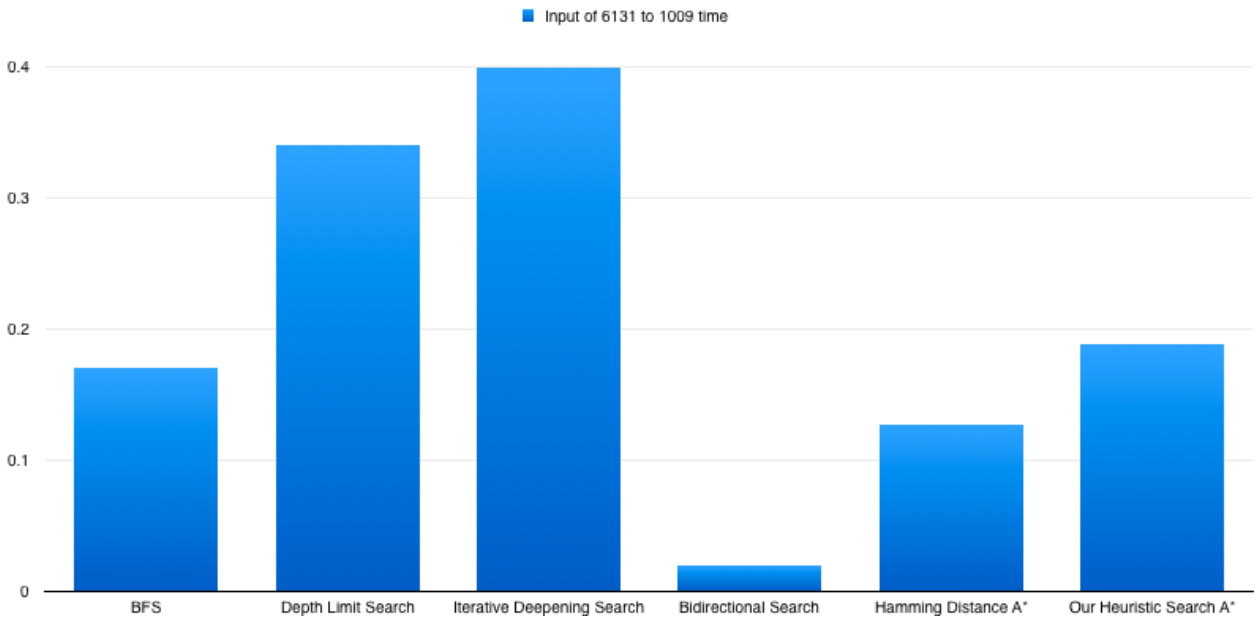
The third problem utilizes an iterative depth first search. It essentially calls problem two's algorithm in a for loop that keeps on increasing the depth from 1 to 8. This function returns a path (an int list) with the lowest depth (that is less than 8) or else it returns an empty list

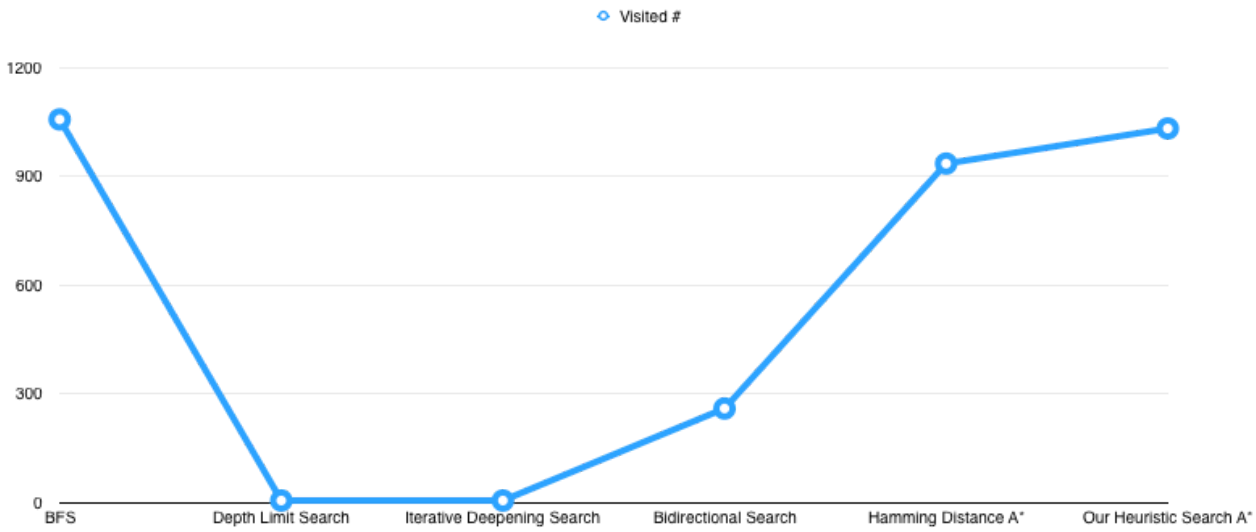
The fourth problem was extra credit. We implemented the bidirectional search algorithm. It runs two searches - one that starts from the beginning and searches forward and the other starts from the end and searches backward. We create two tuple lists with each of the tuple having the first element being the first prime number and the second element being a list of prime numbers which is the path. The first tuple starts from the first given prime number and the second tuple starts from the second given prime number. We also have respective visited primes list. While the two tuple list have tuples, we pop the first element from each tuple lists and put them into a tuple. And the first tuples in the list into the corresponding visited sets. If we find that the two searches have found a common prime, then return the two paths (from the backward search and the forward search) of the prime number they both have in common; otherwise, continue searching from both sides. For both searches *getPossibeActions* from the current prime number from the respective searches. If the two prime numbers are not visited in the respective search, then add the prime number to the respective visited set. Append the respective tuple list with the new prime number from the *getPossibeActions* list and append that new prime number to the recently appended tuple to the current path.

We implemented problem five with the A* algorithm with hamming distance as the heuristic. We used a tuple list again with the first element being the current prime and the second element being a list of primes which is the path to the current path. If the tuple list is not empty, we take the first tuple and add the first element of the tuple to the visited set. If the current prime is the final prime then we return the path. Or else we look through the possible heuristics actions. In order, to get the heuristics, we convert the current prime and the final prime into a string and then we look through both to see if there are any differences character by character; if there are any differences, we increment the difference and at the end of iterating through the two numbers, we return the difference. We made our own version of *getPossibleActions,* called *getPossActionHeuri*. The only difference is that we added the heuristics factor to it. This function returns all next possible actions in the order of best next prime number based off of heuristics to worst. We also have a similar set up of our heuristics. We have its description In that list, if the prime number has been visited, we add it to the visited set and append the prime number to the

tuple list and the prime number to the current path. We keep doing this until we have found the final prime number, then we return the path; otherwise we return an empty list.

time in seconds	BFS	Depth limit search	Iterative Deepening Search	Bidirectional search	Hamming distance A*	Our heuristic Search A*
Input of 6131 to 1009 time	0.17	0.34	0.40	0.02	0.127	0.189
Nodes visited	1057	5	5	259	935	1032





In terms of optimality with regards to time, the input with a sufficient amount of digits in the test inputs gave a clearer result to compare each algorithm with. Since the depth limited search with a limited depth couldn't find the path after a certain point, the simpler input of 6131 going to 1009 gave a good enough result. The simple BFS gave a time (in seconds) of 0.17, the depth limited search 0.34, the iterative deepening search a 0.40, the bidirectional search a 0.02, the hamming distance A* a 0.127, and our heuristic search a 0.189. The best search seemed to be the bidirectional search most of the time for our inputs. But in terms of number of nodes visited, best case (least amount visited) was tied with 5 nodes for the depth limited search and the iterative deepening search. Which makes sense since They're only going to a certain amount of depth to search for the correct path, where they won't be able to visit more than the maximum amount of nodes for the limit given by the depth. This compares very similar to the Big-O's of each algorithm as all of the similar timed ones have Big-O of $O(b^d)$ except for bidirectional search which is $2*O(b^{d/2})$ with the shortest recorded time and is known to be much shorter than the single searches' $O(b^d)$ times.

Alvin Paragraph:

Implemented the original BFS, depth limited search, iterative deepening search, and A*. Came up with the idea for bidirectional search as well. Set the standard for using strings to represent the prime numbers in our programs except for the functions checking whether the

numbers were prime. This simplified the generation of the list of neighbors for each prime number. Hard carried. GG

Woo Paragraph:

Helped with different problems that came up with the coding process and came up with the hamming distance heuristic function and some of the custom heuristic function. Implemented analysis output into the code.

Sarah Paragraph:

I worked on looking through all of the code, analyzing how we solved each given problem and how that relates to the lectures and other materials. In the process, I also kept an eye out for bugs in the code. I helped with the final analysis of code performance by creating a table and graph for time performance and the number of nodes visited.

Inputs used to test our algorithm:

1009 4969
4969 1009
103 199
199 103
103 269
269 103
1009 6131

6131 1009
1009 1009
007 107
107 007
107 7
7 107
107 1009
100003 199999

Output Problem 1:
1009 1019 4019 4919 4969
visited number: 761
time it took in seconds: 0.0697238445282
max queue length it took: 426

4969 4909 4409 1409 1009
visited number: 532
time it took in seconds: 0.0432090759277
max queue length it took: 358

103 193 199
visited number: 45
time it took in seconds: 0.000975847244263
max queue length it took: 35

199 193 103
visited number: 90
time it took in seconds: 0.00286793708801
max queue length it took: 61

103 163 263 269
visited number: 143
time it took in seconds: 0.0114390850067
max queue length it took: 73

269 263 163 103
visited number: 99
time it took in seconds: 0.00418496131897

max queue length it took: 58

1009 1709 6709 6701 6101 6131
visited number: 1059
time it took in seconds: 0.181212186813
max queue length it took: 442

6131 6101 6701 6709 1709 1009
visited number: 1057
time it took in seconds: 0.168427944183
max queue length it took: 460

1009
visited number: 1
time it took in seconds: 5.96046447754e-06
max queue length it took: 1

UNSOLVABLE
time it took in seconds: 9.89437103271e-05

UNSOLVABLE
time it took in seconds: 0.014230966568

UNSOLVABLE
time it took in seconds: 0.0149569511414

UNSOLVABLE
time it took in seconds: 9.79900360107e-05

UNSOLVABLE
time it took in seconds: 0.0139300823212

100003 100103 109103 199103 199109 199909 199999
visited number: 43068
time it took in seconds: 25.2798120975
max queue length it took: 24866

Output Problem 2:
1009 1609 8609 8669 8969 4969
visited number: 5
time it took in seconds: 0.0119318962097

4969 4909 4409 1409 1609 1009
visited number: 5
time it took in seconds: 0.000969886779785

103 193 593 599 199
visited number: 4
time it took in seconds: 0.000519037246704

199 599 569 563 503 103
visited number: 5
time it took in seconds: 0.00232791900635

103 193 593 599 569 269
visited number: 5
time it took in seconds: 0.000509023666382

269 263 863 563 503 103
visited number: 5
time it took in seconds: 0.00447487831116

1009 1709 6709 6701 6101 6131
visited number: 5
time it took in seconds: 0.349662065506

6131 6101 6701 6709 1709 1009
visited number: 5
time it took in seconds: 0.338929176331

UNSOLVABLE
time it took in seconds: 1.90734863281e-06

UNSOLVABLE

time it took in seconds: 0.000370025634766

UNSOLVABLE
time it took in seconds: 0.575058937073

UNSOLVABLE
time it took in seconds: 0.58703994751

UNSOLVABLE
time it took in seconds: 0.000458002090454

UNSOLVABLE
time it took in seconds: 0.603121995926

UNSOLVABLE
time it took in seconds: 5.72976303101

Output Problem 3:
1009 1019 4019 4919 4969
visited number: 4
time it took in seconds: 0.0505950450897

4969 4909 4409 1409 1009
visited number: 4
time it took in seconds: 0.0142431259155

103 193 199
visited number: 2
time it took in seconds: 0.000296831130981

199 193 103
visited number: 2
time it took in seconds: 0.000643014907837

103 163 263 269
visited number: 3
time it took in seconds: 0.00806593894958

269 263 163 103
visited number: 3
time it took in seconds: 0.00150108337402

1009 1709 6709 6701 6101 6131
visited number: 5
time it took in seconds: 0.551283836365

6131 6101 6701 6709 1709 1009
visited number: 5
time it took in seconds: 0.403335809708

1009
visited number: 1
time it took in seconds: 6.19888305664e-06

UNSOLVABLE
time it took in seconds: 0.00183701515198

UNSOLVABLE
time it took in seconds: 25.5437531471

UNSOLVABLE
time it took in seconds: 24.8496828079

UNSOLVABLE
time it took in seconds: 0.0018298625946

UNSOLVABLE
time it took in seconds: 24.9030380249

100003 100103 109103 199103 199109 199909 199999
visited number: 6
time it took in seconds: 24.4709148407

Output Problem 4:

1009 1019 4019 left ring visited nodes: 23
4969 4919 4019 right ring visited nodes: 15
max queue for left ring: 20 max queue for right ring: 12
time it took in seconds: 0.00123310089111

4969 4919 4019 left ring visited nodes: 15
1009 1019 4019 right ring visited nodes: 23
max queue for left ring: 12 max queue for right ring: 20
time it took in seconds: 0.00117802619934

103 109 left ring visited nodes: 9
199 109 right ring visited nodes: 10
max queue for left ring: 8 max queue for right ring: 9
time it took in seconds: 0.000211954116821

199 109 left ring visited nodes: 10
103 109 right ring visited nodes: 9
max queue for left ring: 9 max queue for right ring: 8
time it took in seconds: 0.000211954116821

103 163 left ring visited nodes: 14
269 263 163 right ring visited nodes: 14
max queue for left ring: 12 max queue for right ring: 12
time it took in seconds: 0.00049614906311

269 263 163 left ring visited nodes: 14
103 163 right ring visited nodes: 14
max queue for left ring: 12 max queue for right ring: 12
time it took in seconds: 0.000411987304688

1009 1709 6709 6703 left ring visited nodes: 136
6131 6133 6733 6703 right ring visited nodes: 125
max queue for left ring: 110 max queue for right ring: 99
time it took in seconds: 0.0273718833923

6131 6133 6733 6703 left ring visited nodes: 125
1009 1709 6709 6703 right ring visited nodes: 136
max queue for left ring: 99 max queue for right ring: 110
time it took in seconds: 0.0229279994965

1009 left ring visited nodes: 1
1009 right ring visited nodes: 1
max queue for left ring: 1 max queue for right ring: 1
time it took in seconds: 8.10623168945e-06

UNSOLVABLE
time it took in seconds: 0.000519037246704

UNSOLVABLE
time it took in seconds: 0.000518083572388

UNSOLVABLE
time it took in seconds: 0.000514030456543

UNSOLVABLE
time it took in seconds: 0.000509023666382

UNSOLVABLE
time it took in seconds: 0.162350893021

100003 100103 109103 199103 left ring visited nodes: 211
199999 199909 199109 199103 right ring visited nodes: 221
max queue for left ring: 178 max queue for right ring: 188
time it took in seconds: 0.128858089447

Output Problem 5a:
1009 1069 8069 8969 4969
visited number: 624
maximum queue: 1
time it took in seconds: 0.0588450431824

4969 4909 4409 1409 1009
visited number: 524
maximum queue: 1

time it took in seconds: 0.0447371006012

103 193 199
visited number: 44
maximum queue: 1
time it took in seconds: 0.00127005577087

199 193 103
visited number: 45
maximum queue: 1
time it took in seconds: 0.00126695632935

103 163 263 269
visited number: 124
maximum queue: 1
time it took in seconds: 0.00727677345276

269 263 163 103
visited number: 71
maximum queue: 1
time it took in seconds: 0.00315690040588

1009 1709 6709 6701 6101 6131
visited number: 1060
maximum queue: 1
time it took in seconds: 0.22318816185

6131 6101 6701 6709 1709 1009
visited number: 935
maximum queue: 1
time it took in seconds: 0.127193927765

1009
visited number: 1
maximum queue: 1
time it took in seconds: 4.05311584473e-06

UNSOLVABLE
time it took in seconds: 0.000138998031616

UNSOLVABLE
time it took in seconds: 0.018651008606

UNSOLVABLE
time it took in seconds: 0.0187578201294

UNSOLVABLE
time it took in seconds: 0.000150918960571

UNSOLVABLE
time it took in seconds: 0.0195949077606

100003 100103 109103 199103 199109 199909 199999
visited number: 35913
maximum queue: 1
time it took in seconds: 19.2809011936

Output Problem 5b:
1009 1069 2069 2969 4969
visited number: 620
maximum queue: 1
time it took in seconds: 0.0700759887695

4969 2969 2069 1069 1009
visited number: 549
maximum queue: 1
time it took in seconds: 0.0493180751801

103 193 199
visited number: 44
maximum queue: 1
time it took in seconds: 0.00126504898071

199 193 103
visited number: 45
maximum queue: 1
time it took in seconds: 0.00144982337952

103 163 263 269
visited number: 124
maximum queue: 1
time it took in seconds: 0.00822401046753

269 263 163 103
visited number: 71
maximum queue: 1
time it took in seconds: 0.00355100631714

1009 1709 6709 6701 6101 6131
visited number: 1060
maximum queue: 1
time it took in seconds: 0.245523929596

6131 6101 6701 6709 1709 1009
visited number: 1032
maximum queue: 1
time it took in seconds: 0.189197063446

1009
visited number: 1
maximum queue: 1
time it took in seconds: 5.00679016113e-06

UNSOLVABLE
time it took in seconds: 0.000166177749634

UNSOLVABLE
time it took in seconds: 0.0207831859589

UNSOLVABLE
time it took in seconds: 0.0203731060028

UNSOLVABLE
time it took in seconds: 0.000266075134277

UNSOLVABLE
time it took in seconds: 0.0202729701996

100003 100103 109103 199103 199109 199909 199999
visited number: 35913
maximum queue: 1
time it took in seconds: 19.7666349411