Applying A* and Beam Search Based On Admissible and Non-admissible Heuristic Function To The Puzzle Towers of Corvallis

Zhiyuan He[1] and Zhengxian Lin[2]

***Abstract*— This paper mainly focus on applying A* and beam search based on two heuristic function, one admissible and one non-admissible heuristic, to solve a puzzle named Towers of Corvallis.**

I. INTRODUCTION

This paper mainly focus on applying A* and beam search based on two heuristic function, one admissible and one non-admissible heuristic, to solve a puzzle named Towers of Corvallis. The goal of this puzzle is to get from an initial position to a goal position. In this paper, one admissible heuristic function and one non-admissible heuristic function is designed and implemented. Based on this two heuristic function, A* and beam search algorithms was also designed and implemented to solve this problem. This paper will introduce two heuristic functions as well as A* and beam search algorithms for this problem. This paper will also describe the experiment of testing the performance of A* and beam search algorithms by using each heuristic function. For the beam search algorithms, different beam widths will also be applied to get the performance. The performance is measured by the number of searched nodes, the problem size (number of disks), CPU time, the wall-clock time and the solution length.

II. DESCRIPTION

A. Admissible Heuristic Function

An admissible heuristic is one that never overestimates the cost to reach the goal. Because g(n) is the actual cost to reach n along the current path, and $f(n) = g(n) + h(n)$, we have as an immediate consequence that $f(n)$ never overestimates the true cost of a solution along the current path through n. And admissible heuristics can be generated from relaxed problem.

For the Towers of Corvallis, we have mainly 2 conditions. One is only the top disk can be removed. The second is we can only add the disk on the top of stack. An admissible heuristic function is designed by removing the first condition.

The admissible heuristic function in this paper only consider one condition which is the disk will only be added on the top of stack. A solution for this relaxed Towers of Corvallis is remove disks in Reg A that start from the first wrong place (bottom to top) to top. To which disk is no matter, as we don't have the limitation that only the top disk can be removed. For the disks on Reg B and C, each node at least need be moved once to Reg A. And if they are not in the right order, they will need one more step to change the order. For example, we have a initial state (70213,64,5), and the goal state is (76543210,-,-). (Leftmost is the bottom, rightmost is the top), the first wrong disk is 0, start from this place, 5 disks are needed to remove, and there are 3 disks on Reg B and C, and the 4 in the Peg B is not at the right position, it should be moved. So the h(n) of this heuristic function is $4 * 2 + 3 + 1 = 12$. The equation is $h(n) =$ the number of disks from the first wrong place to the top on Reg A$*2$ +the number of disks on Reg B and C + the number of disks from the first wrong order to the top on Reg B and C.

```
For each disk in Peg A
    Check the bottom disk is right position
        If yes, check the upper disk
        If no, h += the number of upper disks (include this disk)*2
For each disk in Peg B and C
    h ++
    Check the bottom disk is right order
        If yes, check the upper disk
        If no, h += the number of upper disks (include this disk), and break
```

Fig. 1. Pseudocode of admissible heuristic function

B. Non-admissible Heuristic Function

A non-admissible heuristic function is one that could overestimates or underestimates the cost to reach the goal which means the the cost could be bigger or smaller than the actual cost.

And in the Towers of Corvallis, a non-admissible heuristic function also can be generated from relaxed problem. The non-admissible heuristic function in this paper removed one condition like the admissible heuristic which means the disk can be added no matter it is on which position. A solution for this relaxed Towers of Corvallis is close to the admissible heuristic but non-admissible removes all disks from the wrong position to top not only for the first wrong disk but also follow disks as well. For example, we have a initial state (7021,643,5), and the goal state is (76543210,-,-). (Leftmost is the bottom, rightmost is the top), the first wrong disk is 0, start from this place, 3 disks are needed to remove, and there

[1]Zhiyuan He, Computer Science, Oregon State University. Email: hezhi@oregonstate.edu

[2]Zhengxian Lin,Computer Science, Oregon State University. Email: linzhe@oregonstate.edu

are 4 disks on Reg B and C, and the 4 in the Peg B is not at the right position, it should be moved. And the 3 in the Peg B is not at the right position, it should also be moved. So the h(n) of this heuristic function is $3 * 2 + 4 + 2 + 1 = 13$. By doing this, all the situation can be solved. The equation is $h(n) =$ the number of disks from the first wrong place to the top on Reg A$*2$ +the number of disks on Reg B and C +the number of disks from the first wrong order to the top on Reg B and C+the number of disks from the second wrong order to the top on Reg B and C + so on.

```
For all disk in Peg A
    h += number of all disks on Reg A * 2
For each disk in Peg B and C
    h ++
    Check the bottom disk is right order
            If yes, check the upper disk
            If no, h += the number of upper disks (include
            this disk), continue check the upper disks.
```

Fig. 2. Pseudocode of non-admissible heuristic function

C. A* search algorithm

The most widely known form of best-first search is called A* search. It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal: $f(n) = g(n) + h(n)$. The A* search algorithm was implemented for finding the paths between initial states to a given goal state in the problem of Tower of Corvallis. It is a variant of the uniform cost search by using a heuristic function for each node in addition to path costs. And the pseudocode of A* search algorithm is as following:

```
A-Star-Search(problem, NMax, isUsingAdmissibleHeristic)
loop do
    if NMax $<$ length of showedNodes then break
    if length of notVisitedState $<= 0$ then break
    currentState$< -$notVisitedState [i] which have
    minimum f(n) = g(n) + h(n)
    if problem.Goal-Test(currentState) then break
      for each child in currentState.expand() do
          if child not at showedNodes
            add child into notVisitedState
            add child into showedNodes
      notVisitedState.remove(currentState)
    if goalState exist
      return each parent of goalState
```

Fig. 3. Pseudocode of A* search algorithm

D. Beam search algorithm

Beam search is a heuristic search algorithm that explores a graph by expanding the most promising node in a limited set. Beam search algorithm is an excellent way to use memory. Many kinds of algorithm, like A*, run out of memory when the data is very big. Beam search algorithm solve this problem through using beam widths, which is the number of successors that have smallest value: f(n) = g(n) + h(n). Because limiting the number of successors, some successors will be pruned. Thus, the solution of Beam Search is not always optimal solution. However, when beam width is infinity, it is equivalent to A* algorithm. The solution is optimal.

```
Beam-Search(problem, NMax, isUsingAdmissibleHeristic,
BeamWidth)
loop do
    if NMax $<$ length of showedNodes then break
    if length of notVisitedState $<= 0$ then break
    currentState$< -$notVisitedState [i] which have
    minimum f(n) = g(n) + h(n)
    if problem.Goal-Test(currentState) then break
    for each child in currentState.expand() do
        if child not at showedNodes
          if the length of notVisitedState ¡ BeamWidth
            then add child into notVisitedState
          else
            add child instead of the node of
            notVisitedState which has max f(n)
          add child into showedNodes
        notVisitedState.remove(currentState)
    if goalState exist
        return each parent of goalState
```

Fig. 4. Pseudocode of beam search algorithm

III. EXPERIMENTAL SETUP

As mentioned previously, this paper experiment the performance of admissible and non-admissible heuristic A* search, as well as admissible and non-admissible heuristic beam search with different beam width to solve the problem named Towers of Corvallis.

Towers of Corvallis is close to the famous puzzle, Towers of Hanoi which consists of three pegs and a number of disks of different sizes in ascending order. Unlike in Towers of Hanoi, in Towers of Corvallis, any disk can go on any other. The goal of this puzzle is to get from an initial position to a goal position.

In the experiment, the admissible and non-admissible A* search and beam search will be tested to solve 4 different sizes (number of disks) of problems. And for each size of problem, 20 distinct permutations in Peg A are used as the initial state. NMAX is the maximum number of nodes the program have the time to search and it is set 1000 in this experiment to avoid spend too much time on searching. The result of the the experiment is the solution length if successful, the number of nodes expanded, and the total CPU time spent on evaluating the heuristic and on solving the whole problem. The whole process of the experiment can be described as following:

```
For each heuristic function h,
    For beam widths 5, 10, 15, 20, 25, 50, 100, infinity
        For at least 4 different sizes (number of disks)
            For each of the 20 problems p,
                1) Solve p using h or until NMAX
                nodes are expanded.
                2) Record needed data.
```

Fig. 5. The process of the experiment (beam search is A* search when beam width is infinity)

IV. RESULTS

After the experiment, the result of the number of nodes searched against the problem size (number of disks) for the two algorithms and the two heuristics is gained. The result is showed in Fig.6 and Fig.7. the result of the CPU time against the number of disks for the two algorithms and the two heuristics. The result is showed in Fig.8 and Fig.9.

The average solution lengths found by the different heuristics for each problem size (number of disks), heuristic, and search algorithm of this experiment is showed in TABLE I:

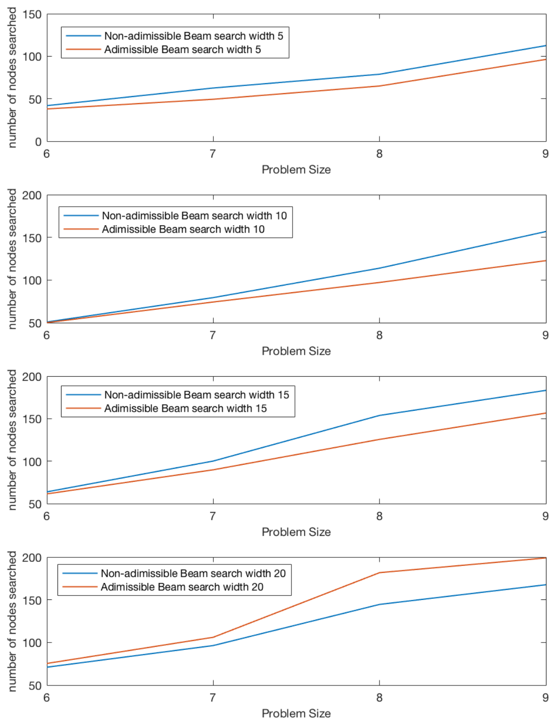

Fig. 6. the number of nodes searched against the problem size (number of disks) Part1

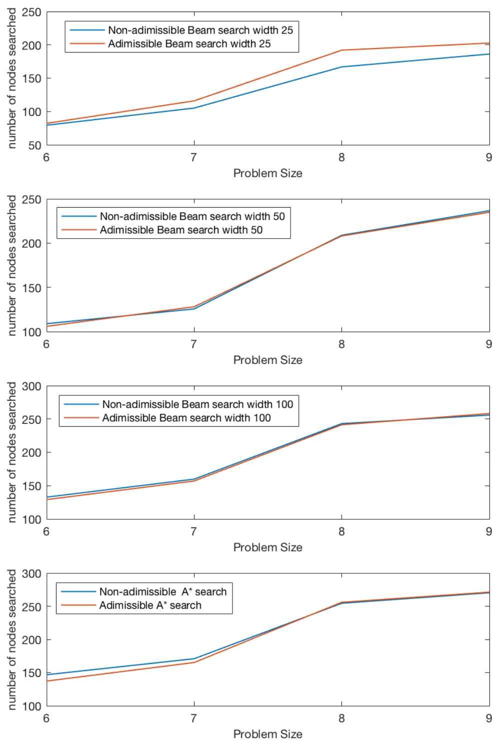

Fig. 7. the number of nodes searched against the problem size (number of disks) Part2

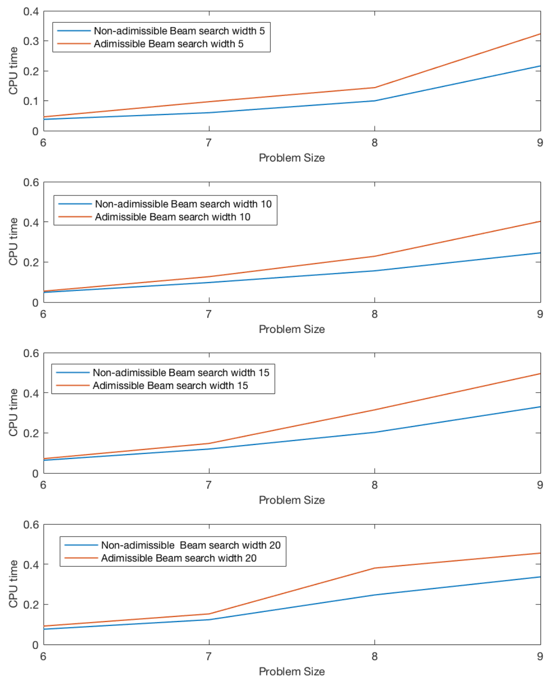

Fig. 8. The CPU time against the problem size (number of disks) Part1

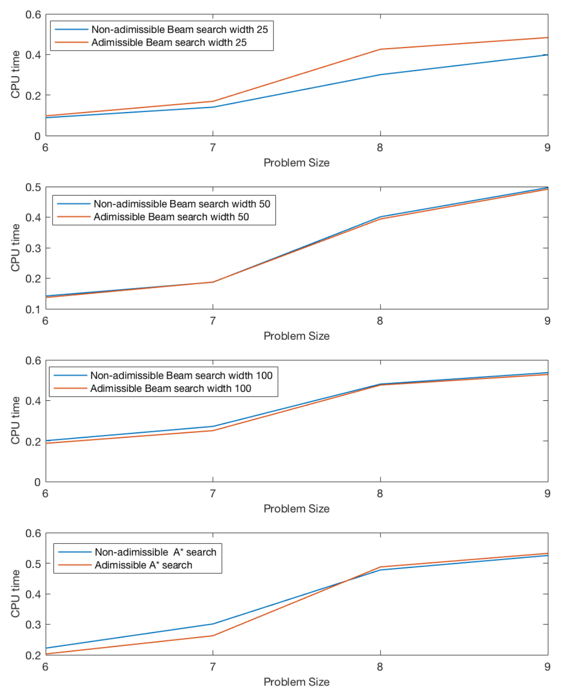

Fig. 9. The CPU time against the problem size (number of disks) Part1

V. DISCUSSION

1) Show an example solution sequence for each algorithm for the largest size you tested. Answer:
Adimissible A* search algorithm:
1 : [[7, 6, 3, 0, 5, 8, 2, 1, 4], [], []]
2 : [[7, 6, 3, 0, 5, 8, 2, 1], [], [4]]
3 : [[7, 6, 3, 0, 5, 8, 2], [1], [4]]
4 : [[7, 6, 3, 0, 5, 8], [1, 2], [4]]
5 : [[7, 6, 3, 0, 5, 8], [1, 2, 4], []]
6 : [[7, 6, 3, 0, 5], [1, 2, 4, 8], []]
7 : [[7, 6, 3, 0], [1, 2, 4, 8, 5], []]
8 : [[7, 6, 3], [1, 2, 4, 8, 5], [0]]
9 : [[7, 6], [1, 2, 4, 8, 5], [0, 3]]
10: [[7, 6], [1, 2, 4, 8], [0, 3, 5]]
11: [[7], [1, 2, 4, 8], [0, 3, 5, 6]]
12: [[], [1, 2, 4, 8], [0, 3, 5, 6, 7]]
13: [[8], [1, 2, 4], [0, 3, 5, 6, 7]]
14: [[8, 7], [1, 2, 4], [0, 3, 5, 6]]
15: [[8, 7, 6], [1, 2, 4], [0, 3, 5]]
16: [[8, 7, 6, 5], [1, 2, 4], [0, 3]]
17: [[8, 7, 6, 5, 4], [1, 2], [0, 3]]
18: [[8, 7, 6, 5, 4, 3], [1, 2], [0]]
19: [[8, 7, 6, 5, 4, 3, 2], [1], [0]]
20: [[8, 7, 6, 5, 4, 3, 2, 1], [], [0]]
21: [[8, 7, 6, 5, 4, 3, 2, 1, 0], [], []]
Non-Adimissible A* search algorithm:
1 : [[7, 6, 3, 0, 5, 8, 2, 1, 4], [], []]
2 : [[7, 6, 3, 0, 5, 8, 2, 1], [], [4]]

TABLE I
THE AVERAGE SOLUTION LENGTHS FOR EACH PROBLEM SIZE, HEURISTIC, AND SEARCH ALGORITHM

Problem Size	6	7	8	9
Admissible A*	12.8	15.2	10.9	9.65
Non Admissible A*	13.5	15.5	11.65	10.6
Admissible beam 5	15.3	18.6	22.95	30.2
Non Admissible beam 5	15.95	21.25	25.45	33.35
Admissible beam 10	14.4	17.95	21.35	25.25
Non Admissible beam 10	14.6	18.4	23.45	29
Admissible beam 15	14.25	17.5	20.95	24.6
Non Admissible beam 15	14.55	18.2	22.9	24.95
Admissible beam 20	14.15	16.95	20.45	22.45
Non Admissible beam 20	14.45	17.45	22.15	21.9
Admissible beam 25	14.25	16.85	20.25	21.65
Non Admissible beam 25	14.35	17.3	19.4	19.45
Admissible beam 50	14.15	16.45	17.45	17.3
Non Admissible beam 50	14.2	16.3	17.0	16.05
Admissible beam 100	13.8	16	12.95	11.25
Non Admissible beam 100	13.85	16	13.7	11.3

3 : [[7, 6, 3, 0, 5, 8, 2], [1], [4]]
4 : [[7, 6, 3, 0, 5, 8], [1, 2], [4]]
5 : [[7, 6, 3, 0, 5, 8], [1, 2, 4], []]
6 : [[7, 6, 3, 0, 5], [1, 2, 4, 8], []]
7 : [[7, 6, 3, 0], [1, 2, 4, 8, 5], []]
8 : [[7, 6, 3], [1, 2, 4, 8, 5], [0]]
9 : [[7, 6], [1, 2, 4, 8, 5], [0, 3]]
10: [[7, 6], [1, 2, 4, 8], [0, 3, 5]]
11: [[7], [1, 2, 4, 8], [0, 3, 5, 6]]
12: [[], [1, 2, 4, 8], [0, 3, 5, 6, 7]]
13: [[8], [1, 2, 4], [0, 3, 5, 6, 7]]
14: [[8, 7], [1, 2, 4], [0, 3, 5, 6]]
15: [[8, 7, 6], [1, 2, 4], [0, 3, 5]]
16: [[8, 7, 6, 5], [1, 2, 4], [0, 3]]
17: [[8, 7, 6, 5, 4], [1, 2], [0, 3]]
18: [[8, 7, 6, 5, 4, 3], [1, 2], [0]]
19: [[8, 7, 6, 5, 4, 3, 2], [1], [0]]
20: [[8, 7, 6, 5, 4, 3, 2, 1], [], [0]]
21: [[8, 7, 6, 5, 4, 3, 2, 1, 0], [], []]
Adimissible Beam search with beam width 5:
1: [[7, 6, 3, 0, 5, 8, 2, 1, 4], [], []]
2: [[7, 6, 3, 0, 5, 8, 2, 1], [], [4]]
3: [[7, 6, 3, 0, 5, 8, 2], [1], [4]]
4: [[7, 6, 3, 0, 5, 8], [1, 2], [4]]
5: [[7, 6, 3, 0, 5], [1, 2, 8], [4]]
6: [[7, 6, 3, 0], [1, 2, 8], [4, 5]]
7: [[7, 6, 3], [1, 2, 8, 0], [4, 5]]
8: [[7, 6], [1, 2, 8, 0, 3], [4, 5]]
9: [[7], [1, 2, 8, 0, 3], [4, 5, 6]]
10: [[], [1, 2, 8, 0, 3], [4, 5, 6, 7]]
11: [[3], [1, 2, 8, 0], [4, 5, 6, 7]]
12: [[3, 0], [1, 2, 8], [4, 5, 6, 7]]
13: [[3, 0], [1, 2], [4, 5, 6, 7, 8]]
14: [[3], [1, 2, 0], [4, 5, 6, 7, 8]]
15: [[], [1, 2, 0, 3], [4, 5, 6, 7, 8]]
16: [[8], [1, 2, 0, 3], [4, 5, 6, 7]]
17: [[8, 7], [1, 2, 0, 3], [4, 5, 6]]

```
18: [[8, 7, 6], [1, 2, 0, 3], [4, 5]]
19: [[8, 7, 6, 5], [1, 2, 0, 3], [4]]
20: [[8, 7, 6, 5, 4], [1, 2, 0, 3], []]
21: [[8, 7, 6, 5, 4, 3], [1, 2, 0], []]
22: [[8, 7, 6, 5, 4, 3], [1, 2], [0]]
23: [[8, 7, 6, 5, 4, 3, 2], [1], [0]]
24: [[8, 7, 6, 5, 4, 3, 2, 1], [], [0]]
25: [[8, 7, 6, 5, 4, 3, 2, 1, 0], [], []]
```

Non-adimissible Beam search with beam width 5:

```
1: [[7, 6, 3, 0, 5, 8, 2, 1, 4], [], []]
2: [[7, 6, 3, 0, 5, 8, 2, 1], [], [4]]
3: [[7, 6, 3, 0, 5, 8, 2], [1], [4]]
4: [[7, 6, 3, 0, 5, 8], [1, 2], [4]]
5: [[7, 6, 3, 0, 5], [1, 2, 8], [4]]
6: [[7, 6, 3, 0], [1, 2, 8], [4, 5]]
7: [[7, 6, 3], [1, 2, 8, 0], [4, 5]]
8: [[7, 6], [1, 2, 8, 0, 3], [4, 5]]
9: [[7], [1, 2, 8, 0, 3], [4, 5, 6]]
10: [[], [1, 2, 8, 0, 3], [4, 5, 6, 7]]
11: [[3], [1, 2, 8, 0], [4, 5, 6, 7]]
12: [[3, 0], [1, 2, 8], [4, 5, 6, 7]]
13: [[3, 0], [1, 2], [4, 5, 6, 7, 8]]
14: [[3], [1, 2, 0], [4, 5, 6, 7, 8]]
15: [[], [1, 2, 0, 3], [4, 5, 6, 7, 8]]
16: [[8], [1, 2, 0, 3], [4, 5, 6, 7]]
17: [[8, 7], [1, 2, 0, 3], [4, 5, 6]]
18: [[8, 7, 6], [1, 2, 0, 3], [4, 5]]
19: [[8, 7, 6, 5], [1, 2, 0, 3], [4]]
20: [[8, 7, 6, 5, 4], [1, 2, 0, 3], []]
21: [[8, 7, 6, 5, 4, 3], [1, 2, 0], []]
22: [[8, 7, 6, 5, 4, 3], [1, 2], [0]]
23: [[8, 7, 6, 5, 4, 3, 2], [1], [0]]
24: [[8, 7, 6, 5, 4, 3, 2, 1], [], [0]]
25: [[8, 7, 6, 5, 4, 3, 2, 1, 0], [], []]
```

2) How did the search time and the solution quality vary with the beam width? Is there a beam width that gives the best tradeoff for the two heuristics?
   Answer:
   With the beam width goes up, the speed of finding solution is slower, but the radio of finding optimal solution is higher. We think it definitely give the best tradeoff for the two heuristics, because when the beam width are not too big, the algorithm always find a solution, although it is not optimal.
3) Is there a clear preference ordering among the heuristics you tested considering the number of nodes searched and the total CPU time taken to solve the problems for the two algorithms?
   Answer:
   Our result is quite interesting. When the beam width is from 5 to 25, the admissible heuristics is worse than non-admissible heuristics. The admissible heuristics search more nodes and take more CPU time. However, when the When the beam width is 50 to infinity, they are close, even when the size of disk is small, like 6 and 7, the admissible heuristics is better than non-admissible, which means the algorithms with admissible heuristics search less nodes and take less CPU time.
4) Can a small sacrifice in optimality give a large reduction in the number of nodes expanded? What about CPU time?
   Answer:
   Yes, it surely reduces a large number of nodes expanded through take a mall sacrifice in optimality. When the beam width is from 5 to 25, it is not always get an optimal solution, but the amount of CPU time is reduced. For the 6 and 7 sizes, the different of CPU time is quite small. However, it become bigger and bigger for 8 and 8 sizes.
5) How did you come up with your heuristic evaluation functions?
   Answer:
   First of all, we define a relaxed problem that remove the constraint condition which is only can get the top of pegs. This h will definitely be admissible, but the it is slow. So, we realize that the bottommost disks always cost more action if there is smaller than disk above it. So, we add another constraint that is if there is smaller disk, we record the number of from this smaller disk to the top. Thus, it become quite fast, and it is our admissible heuristic Then, we realize that if continue to record the disk which is smaller than the bottommost one of B or C peg, the heuristic will be non-admissible at some condition. Thus, we come up our non-admissible heuristic this way.
6) How do the two algorithms compare in the amount of search involved and the cpu time?
   Answer:
   The A* algorithm search more nodes and spend more cpu-time, but it always find the optimal solution if the number of search nodes is smaller NMax. The beam search with infinity beam width is as good as A*. The number of search node is smaller while the beam width is smaller, and the cpu-time is also smaller too, but it is not always find the optimal path.
7) Do you think that either of these algorithms scale to even larger problems? What is the largest problem you could solve with the best algorithm+heuristic combination? Report the wall-clock time, CPU-time, and the number of nodes searched.
   Answer:
   No, our algorithm can not solve to even larger problems. We try to solve the problem of 10 disks. It takes very long time, and the seared nodes always lager than NMax.
8) Is there any tradeoff between how good a heuristic is in cutting down the number of nodes and how long it took to compute? Can you quantify it?
   Answer:
   Yes, there is for sure. The time complexity of our admissible and inadmissible heuristic are O(n).
9) Is there anything else you found that is of interest?
   Answer:

We found our results are quite interesting. Because we want to achieve a good heuristic function. Our admissible heuristic function is pretty close the actual cost value, and we developed a non-admissible heuristic function based on our admissible heuristic function, our non-admissible heuristic function which could be bigger than actual cost has a close performance as our admissible heuristic function. To figure our why, we tested different admissible heuristic functions and we found that the performance is largely depend on the heuristic functions. And the results of our experiment looks similar is because both of our two heuristic functions is close to the actual cost. And this the non-admissible one could be bigger than the actual cost but only in some cases which is not that often happened in our test. These interesting founds explained the performance of our experiment.

References

[1] S.J.Russell & P.Norvig, Artificial Intelligence: a modern approach, 3rd edition.