function A-Star-Search(problem, NMax, isUsingAdmissibleHeristic) return a solution, or uncomplete solution

loop do

if NMax < length of showedNodes Or 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

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.

function Beam-Search(problem, NMax, isUsingAdmissibleHeristic, BeamWidth) return a solution, or uncomplete solution

loop do

if NMax < length of showedNodes Or 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

1. Show an example solution sequence for each algorithm for the largest size you tested in the following format:

[0123,\_,\_], [123,0,\_], [23,0,1], [23,\_,01] ...

1. 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?

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.

1. 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?

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.

1. Can a small sacrifice in optimality give a large reduction in the number of nodes expanded? What about CPU time?

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.

1. How did you come up with your heuristic evaluation functions?

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.

1. How do the two algorithms compare in the amount of search involved and the cpu-time?

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.

1. 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.

No, our algorithm can not cale 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

1. 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?

Yes, there is for sure. The time complexity of our admissible and inadmissible heristic are O(n)

1. Is there anything else you found that is of interest?