SEARCH ALGORITHMS: TOWER OF HANOI

# PROBLEM REPRESENTATION

The Tower of Hanoi problem is implemented through state space representation specified by black box description to the search algorithm in the code. Every state in my state space representation is a 6-tuple (S, A, c, T, I, SG ) where:

* **Set of states S**: In the state, I store the information about the discs on each peg. So, every state is a vector of length 4 (P1, P2, P3, P4) where each Pi contains a list of some discs on the ith peg in a descending order i.e, from top to bottom. Namely,

S = {(P1, P2, P3, P4) | Pi = (da, db,…,dz) s.t. 0 ≥ da > db >….>dz > n | = n }

* where n = number of discs
* da > db specifies that da is larger than db
* every di is a unique number from 0 to n-1 for n discs
* Pi can be empty
* number of discs on all pegs is n i.e. = n

Graphical user interface

Description automatically generated

For ex, for n = 6, corresponds to { ( ), (1), (0), (5,4,3,2) }

* **Set of actions A**: Every action is defined as:

{ move(Pi, Pj) | < } meaning move topmost disc from Pi to Pj given topmost disc on Pj is larger than topmost disc on Pi.

* **Cost function c**: Unit 1 for each action i.e., for one disc movement.
* **Transition Relation T**: { {(d3,d2), (d1), (d4), ()}, move(P1, P3), {(d3), (d1), (d4, d2), ()} } where n = 4 and accordingly for every n.
* **Initial State I:** { (dn, dn-1, ….,d2 ,d1), (), (), () }
* **Goal states** SG **:** { (), (), (), (dn, dn-1, ….,d2 ,d1) }

The above problem was *rational* to represent the Tower of Hanoi problem as every state is unique giving full description of position of each disc on 4 pegs. Also, every peg contains an ordered list of discs with smallest disc on top. This was implemented by representing each disc as a number from 0 to n-1 for n discs as mentioned in state representation. So, the disc number can be directly used as an indication of its size namely, disc represented by 1 will be smaller than disc represented by 5. This allows the comparison to satisfy the problem constraint of moving a disc to only a larger disc.

# SEARCH ALGORITHMS

The search algorithms chosen to solve the problem are: A\* search and BFS.

## A\* Search

It is an informed search algorithm having the knowledge of the goodness of expanding a state S which is given in the form of a heuristic function h(s) that estimates the cost of an optimal (cheapest) path from s to the goal. For every expansion, it considers the value of an evaluation function f(n) = g(n) + h(n) where

* g(n) = cost so far to reach n
* h(n) = estimated cost to goal from n
* f(n) = estimated total cost of path through n to goal

and expands the state with the smallest evaluation function value at each expansion. With this approach, the algorithm avoids expanding paths that are already expensive. It achieves completeness meaning if a solution exists to a problem, A\* is guaranteed to find it. But whether this solution is best possible solution to the problem or not depends on the chosen heuristic values. So, A\* is optimal only with consistent heuristic functions.

## Breadth First Search

It is a blind search algorithm where it always expands the shallowest unexpanded node first. It explores all the nodes at the present depth before moving on to the nodes at the next depth level. So, it guarantees completeness given the goal is at finite depth and the branching factor at each level is also finite. But since it follows the expansion order regardless of what the problem is, it is only optimal in case of unit action costs.

# PROBLEM HEURISTICS

## HEURISTIC 1: Count of discs not on rightmost peg

This heuristic value takes the number of discs which are not on the rightmost peg at any state.

* At the initial state, all the discs will be on the leftmost peg ⇒ h(s) = n
* At the goal state, all the discs will be on the rightmost peg ⇒ h(s) = 0.

**Idea**: This heuristic was formulated taking the intuitive idea of obtaining the goal distance within a relaxed version of the problem. For tower of Hanoi, let the relaxed problem be the problem where there is no constraint on size of discs i.e., any disc can be moved on top of any disc regardless of the size. The optimal solution from initial to goal state in this relaxed version will be n (i.e., number of discs) which is what our heuristic will return for initial state of our original Tower of Hanoi Problem.

**Consistency**: For any transition s s’, a will be the movement of 1 disc from 1 peg to another. In this case, h(s’) = h(s) if we are not moving this disc to rightmost peg or h(s’) – h(s) = 1 if this disc is moved to rightmost peg. So, we can see that h(s) – h(s’) is never greater than 1 which is our action cost (our tower of Hanoi assumes unit cost of each action). This implies that, h(s) – h(s’) ≤ c(a) and the heuristic value is not decreasing by more than the cost of a. So, we can conclude that our heuristic is *consistent*.

**Admissible**: For any state s, the true cost to goal h\*(s) will be minimum h(s) in the best case where each disc can be moved to rightmost peg without considering the size. So, we can say that our heuristic value acts as a lower bound on goal distance h(s) ≤ h\*(s). This concludes that our heuristic is also admissible. Also, we can consider the fact that consistency ⇒ admissibility to consider this heuristic as admissible since we proved its consistency.

Hence, this heuristic will result in an optimal A\* search algorithm returning the best possible solution.

## HEURISTIC 2: Distance of the largest disc from the goal

This heuristic value will return the distance of the disc of the largest size as the number of pegs it is away from the rightmost peg.

* At the initial state, the largest disc will be on the leftmost peg ⇒ h(s) = 3
* At the goal state, the largest disc will be on the rightmost peg ⇒ h(s) = 0.

**Idea**: This heuristic was formulated taking the intuitive idea of first moving each of the n-1 smaller discs to any peg between leftmost to rightmost peg and then moving the largest disc to rightmost peg. This could be taken as an estimate to the goal distance as the largest disc can never reach rightmost peg without first moving other discs.

**Consistency**:

Let s = {( ),(0),(1), (2)} and s’ = {(2),(0),(1), ())} for n = 3

Here, h(s) = 0 and h(s’) = 2 as per above definition.

For transition s s’, h(s’) – h(s) = 2 ≥ c(a)

So, we can conclude that this heuristic is *not* *consistent*.

**Admissible**:

Let s = { (1), ( ), ( ), (0) } for n = 2

h\*(s) = 1 -> Disc 1 can be moved to rightmost peg in 1 move making it a goal state

h(s) = 2 -> Largest disc i.e., 1 is 2 pegs away from the rightmost peg

Here, h(s) ≥ h\*(s) so this heuristic is not admissible. Also, we can consider the fact that consistency ⇒ admissibility to conclude that this heuristic is not admissible since we proved it to be not consistent.

Hence, this heuristic will result in a non-optimal A\* search algorithm.

We will below evaluate the performance of these heuristics, when used to solve our tower of Hanoi problem, with A\* search.

## HEURISTIC COMPARISON

### Results for Heuristic 1: Count of discs not on rightmost peg

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number of discs (n)** | **Runtime (in second)** | **Plan length** | **States expanded** | **States visited** |
| 2 | 0.008 | 3 | 9 | 15 |
| 3 | 0.012 | 5 | 25 | 54 |
| 4 | 0.009 | 5 | 25 | 54 |
| 5 | 0.410 | 13 | 669 | 897 |
| 6 | 1.516 | 17 | 2299 | 3161 |
| 7 | 9.069 | 25 | 13617 | 15414 |
| 8 | 30.543 | 33 | 59489 | 63493 |
| 9 | 138.558 | 41 | 236346 | 251715 |
| 10 | 223.959 | 49 | 834610 | 926414 |
| 11 | 1297.438 | 65 | 4045066 | 4130145 |

### Results for Heuristic 2: Distance of the largest disc from the goal

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number of discs (n)** | **Runtime (in second)** | **Plan length** | **States expanded** | **States visited** |
| 2 | 0.003 | 3 | 6 | 13 |
| 3 | 0.014 | 5 | 37 | 39 |
| 4 | 0.139 | 9 | 241 | 253 |
| 5 | 0.613 | 13 | 1007 | 1021 |
| 6 | 2.548 | 17 | 4041 | 4093 |
| 7 | 10.821 | 25 | 16373 | 16381 |
| 8 | 47.219 | 33 | 65525 | 65533 |
| 9 | 156.348 | 41 | 262129 | 262141 |
| 10 | 237.692 | 49 | 1048495 | 1048571 |
| 11 | 1298.236 | 65 | 4194293 | 4194301 |

### Observations

For smaller values of n, both the heuristics perform almost the same. For larger values of n i.e., n > 6, heuristic 1 performs better giving slightly lower runtime and lesser number of states expanded.

### Conclusion

The first heuristic is admissible and as expected it was more efficient in solving the problem. This difference of performance between two heuristics will become even more significant for greater values of number of discs, n.

# OPTIMAL VS SUB-OPTIMAL SOLUTION RUNTIME

Here, we will indicate the maximum number of discs n that can be handled optimally and sub-optimally with a runtime limit of 10 minutes. To understand the same, we will consider the results of:

* *A\* search algorithm* with first consistent heuristic which is expected to give optimal solution
* *Breadth First Search* which will give us sub-optimal solution

|  |  |  |
| --- | --- | --- |
| **Number of discs, n** | **Runtime (in seconds)** | |
| **Optimal Solution** | **Sub-Optimal Solution** |
| 2 | 0.003 | 0.000 |
| 3 | 0.014 | 0.015 |
| 4 | 0.139 | 0.110 |
| 5 | 0.613 | 0.470 |
| 6 | 2.548 | 2.006 |
| 7 | 10.821 | 8.610 |
| 8 | 30.543 | 73.284 |
| 9 | 138.558 | 158.107 |
| 10 | 223.959 | 841.181 > 10 min |
| 11 | 1297.438 > 10 min |  |

The maximum number of discs that can be handled optimally within runtime of 10 minute (as highlighted) is 10 and that can be handled sub-optimally is 9.

The maximum number of discs handled optimally is k = 10. Let us compare the optimal and sub-optimal length of the solutions for number of discs, n = k =10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Runtime (in second)** | **Plan length** | **States expanded** | **States visited** |
| Optimal Solution | 223.959 | 49 | 834610 | 926414 |
| Sub-optimal Solution | 841.181 | 49 | 1048565 | 1048575 |

We can see that for the same plan length; the states visited, and states expanded along with the runtime is way higher for sub-optimal solution than the optimal solution which is as expected.

# ALGORITHM RUNTIME COMPARISON

Here, we will show the tables with the runtime of each algorithm vs. number of discs n.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Discs** | **Runtime (in seconds)** | | |
| **A\* with first optimal heuristic** | **A\* with second non- optimal heuristic** | **Breadth First Search** |
| 2 | 0.008 | 0.003 | 0.000 |
| 3 | 0.012 | 0.014 | 0.015 |
| 4 | 0.009 | 0.139 | 0.110 |
| 5 | 0.410 | 0.613 | 0.470 |
| 6 | 1.516 | 2.548 | 2.006 |
| 7 | 9.069 | 10.821 | 8.610 |
| 8 | 30.543 | 47.219 | 73.284 |
| 9 | 138.558 | 156.348 | 158.107 |
| 10 | 223.959 | 237.692 | 841.181 |

Hence, we can say that A\* search algorithm with optimal solution performs the best, A\* search with sub-optimal solution works equally well and BFS gives the least performance where the runtime increases exponentially for larger values of n.

# PATTERN DATABASE (PDB) HEURISTIC

Pattern database is a heuristic stored as a lookup table that stores the lengths of optimal solutions for subproblem instances. These subproblem instances are abstracted by choosing a subset of variables to be solved and ignoring the values of all other variables.

In my implementation, a typical entry of key-value pair in Pattern database is:

Key: Abstraction of state taking positions of only largest m discs.

Value: Length of optimal solution for abstract state calculated by doing BFS from the goal.

#### PATTERN DATABASE KEY

Graphical user interface

Description automatically generatedFor instance, the below state for n = 6 will be abstracted to a pattern as shown for m = 3. This pattern will be hashed and stored as an index in the pattern database.

A picture containing chart

Description automatically generatedGraphical user interface

Description automatically generated

Lookup

{ (), (1), (0), (5,4,3,2) } { ( ), ( ), ( ), (5,4,3) }

#### PATTERN DATABASE VALUE

The value stored for the above state will be minimum of cost to goal of all the states that will match the patter. For instance,

A picture containing text

Description automatically generated

The heuristic for all the above states will be same as they all match the pattern as defined above. So there will be a single entry in database for all these states. Hence, the heuristic returned will be minimum of cost to goal of each of these states.

* This will ensure that the size of pattern database is *controlled* as there will be only one entry for several states matching the pattern. This will, in turn, limit the size of abstract search space leaving enough memory to do actual search.

Graphical user interface

Description automatically generated with medium confidenceIn this example, the heuristic value returned will be 0 as they match the pattern of a goal state.

**Idea**: This heuristic was formulated taking the intuitive idea of directly using the optimal length to goal as heuristic by first traversing the state space from goal to initial state storing all the optimal lengths. But storing each state for a big state space will require a lot of a memory and hence a pattern-based database is created where the optimal solutions are stored for patterns which is some abstraction of the original states. For every state, this heuristic will return the length of the optimal solution of its pattern. This heuristic is equivalent to the optimum solution in a relaxed problem where only largest m out of n discs should be placed correctly and hence looks like a good estimate of the goal distance.

**Consistency**: For any transition s s’, a will be the movement of 1 disc from 1 peg to another. In this case, h(s’) = h(s) if we are not moving one of the m larger discs or h(s’) – h(s) ≤ c(a) if one of the m larger discs is moved as heuristic is directly taking the cost of moving the disc. So, we can see that h(s) – h(s’) is never greater than action cost. This implies that the heuristic value is not decreasing by more than the cost of a. So, we can conclude that our heuristic is *consistent*.

**Admissible**: For any state s, the cost of its abstract state i.e., h(s) will always be less than the actual cost i.e., h\*(s)as it will only consider the cost of m out of n discs. So, we can say that our heuristic value acts as a lower bound on goal distance h(s) ≤ h\*(s). This concludes that our heuristic is also admissible. Also, we can consider the fact that consistency ⇒ admissibility to consider this heuristic as admissible since we proved its consistency.

This will result in an optimal A\* search algorithm returning the best possible solution.

# PDB RESULTS

Let us find the biggest m such that constructing the PDB (m largest out of n) takes less than 5 minutes. We will capture these results by fixing the value for n = 7 (not too large, not too small in order to capture results in genuine runtime)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Number of discs** | **Chosen m** | **Time to construct PDB (in seconds)** | **Search time**  **(in seconds)** | **Plan Length** | **States Expanded** | **States Visited** |
| 7 | 1 | 1.483 | 0.378 | 25 | 5207 | 6624 |
| 2 | 1.518 | 0.172 | 25 | 2436 | 3593 |
| 3 | 1.458 | 0.081 | 25 | 1249 | 2069 |
| 4 | 1.672 | 0.0062 | 25 | 695 | 1193 |
| 5 | 1.655 | 0.036 | 25 | 471 | 857 |
| 6 | 1.610 | 0.020 | 25 | 314 | 665 |
| 7 | 2.140 | 0.092 | 25 | 226 | 561 |

The biggest m is 7 for n = 7, where constructing the PDB took less than 5 minutes.

Now taking this value of m, we will increment the value of n (currently n = 7) and capture the results.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Chosen m** | **Number of discs** | **Time to construct PDB (in seconds)** | **Search time**  **(in seconds)** | **Plan Length** | **States Expanded** | **States Visited** |
| 7 | 8 | 10.412 | 0.133 | 33 | 714 | 1447 |
| 9 | 65.616 | 0.279 | 41 | 1305 | 2516 |
| 10 | 444.055 | 0.239 | 49 | 1171 | 2141 |
| 11 | 1124.673 > 10 min | 0.821 | 65 | 9300 | 14799 |

From the results, we can see that no solution can be found in less than 10 minutes after n = 10.

## OBSERVATIONS

From the obtained results, we can observe that:

* Runtime worsens exponentially as the value of n increases. This is because as the value of n increases, the time to construct the pattern database also increases, thereby significantly increasing the total runtime.
* The observed search time is very low even with greater values of n so PDB significantly reduces the search time.

## CONCLUSION

With an increasing number of discs, the PDB heuristic provides very low values of runtime, but the total time taken to construct the pattern database is very large. One must be careful while choosing the value of m so that the creation of PDB as well as search algorithm runs in a considerable amount of time. Hence, a careful trade-off between the time to construct the PDB as well as search runtime is required.

***End of report***