# Using the Hill Climbing Algorithm to Solve Towers of Hanoi Problem

By Matthew Shore, Jacob Hunter and Alexander Hoerr

**Abstract: State-space problems such as the "Farmer, Fox, Goose and Grain", "Cannibals and Missionaries", "Towers of Hanoi", or "Eight-Puzzle" are useful problems for artificial intelligence search algorithms to solve. Searching using A* function allows artificial intelligence to search for solutions to these problems. In the paper, we will demonstrate how using the hill-climbing algorithm can be used for solving the "Towers of Hanoi" problem. We will apply this algorithm to find an optimal solution. We will be using LISP as our primary programming language to create the problem in a graph and test solutions.**

## 1. INTRODUCTION

The project was to take a problem that is not polynomially solvable and convert the problem to a state space problem. That requires changing the problem into a graph that can be iterated on by underestimating heuristic algorithms. We then used the hill climbing underestimate heuristic algorithm to find a path that would then solve the problem. The problem chosen was the towers of Hanoi problem which involves moving a set number of rings that are grouped together to another pole in the same order. To solve the project research was done on converting the towers of hanoi problem to a graph based implementation. Then a graph based implementation was then converted into an adjacency list in the LISP language.

### *1.1 LISP*

Programming languages to create artificial intelligence have become common. LISP is particularly useful due to its specialization in artificial intelligence, and remains in use due to its simplicity and flexibility.[1]
LISP was one of the first languages to use tree data structures. It is particularly suited to using the hill-climbing algorithm used to solve the Tower of Hanoi problem.[2]

### *1.2 Hill Climbing Algorithm*

The hill-climbing algorithm uses a greedy approach to solve the problem. It continuously moves to minimize the number of moves to get to the solution. This algorithm was useful for optimizing the Tower of Hanoi problem since the problem itself lends to having a good heuristic. Additionally, it doesn't require too much processing power since it only maintains a single state at a time rather than rewriting the graph each step.

## 2. TOWERS OF HANOI PROBLEM

The Towers of Hanoi problem involves shifting a stack of rings from one pole position to another. In towers of Hanoi there are two main objects: a set of three rods and a number of rings with varying colors or widths to distinguish them. Our solution used the three ring version of the problem. The user or solver is allowed to move one ring at a time off of the top of the ring stack to another pole which can either contain rings or be empty. The goal of the problem is to shift all of the rings to another pole from the starting pole and have the same pole arrangement as the original pole. So to move

from the first pole to the third pole the moves aaa -> caa -> cba -> bba -> bbc -> abc -> acc -> ccc would need to be made. Each letter represents one ring on the tower of Hanoi and its pole position. (See figure 1)

## 3. EXECUTION

The execution of the graph is completely successful. Though the algorithm takes time the graph prints A -> C -> F -> E -> W -> V -> Y -> AB (See figure 3).

Which in terms of the original graph equals aaa -> caa -> cba -> bba -> bbc -> abc -> acc -> ccc. By looking at the graph that is a correct solution and it only took 8 iterations. By working out the computations by hand the best possible result for a Hanoi problem solution using a graph is 8 moves either to position bbb or ccc like our algorithm did.

This is actually the optimal solution in this case, changing the weights it would be possible to path find to any state desired. Overall our algorithm was not only successful but also produced an optimal result.

## 4. WORKS CITED

[1] https://apro-software.com/top-programming-languages-in-ai/

[2] http://www.paulgraham.com/icad.html

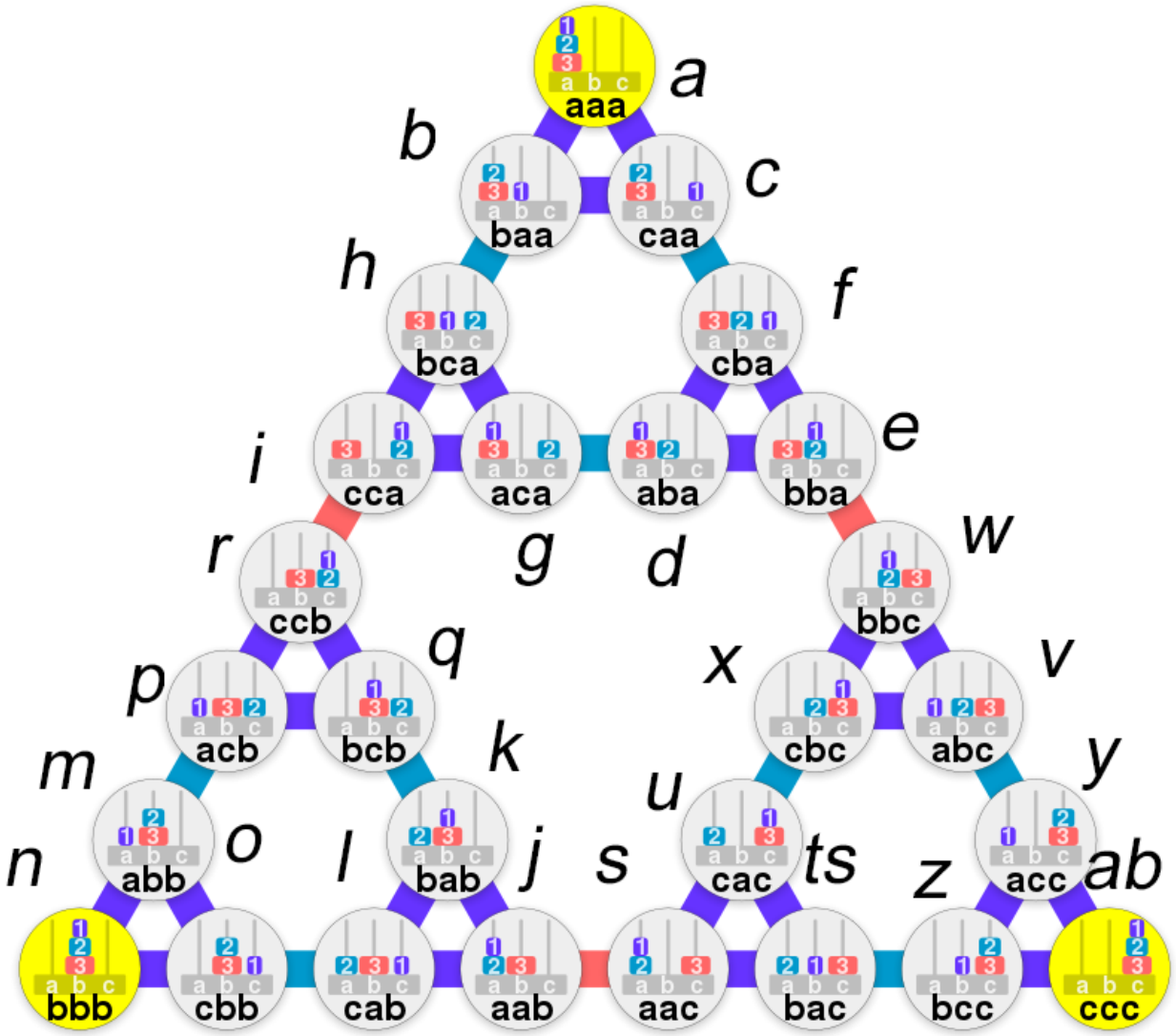


Figure 1

Used this graph from wikipedia to create a graph that is understandable in clisp

```
(setq hanoi '(
        (a (b 7) (c 6))
        (b (a 7) (c 6) (h 8))
        (c (a 7) (b 7) (f 5))
        (d (e 4) (f 5) (g 6))
        (e (d 5) (f 5) (w 3))
        (f (c 6) (d 5) (e 4))
        (g (d 5) (h 8) (i 7))
        (h (b 8) (g 6) (i 7))
        (i (g 6) (h 8) (r 7))
```

```
        (j (k 5) (l 6) (s 3))
        (k (j 4) (l 6) (q 6))
        (l (j 5) (k 6) (o 6))
        (m (n 7) (o 6) (p 7))
        (n (m 7) (o 6))
        (o (l 6) (m 7) (n 7))
        (p (m 7) (q 6) (r 7))
        (q (k 6) (p 7) (r 7))
        (r (i 7) (p 7) (q 6))
        (s (j 4) (ts 2) (u 3))
        (ts (s 3) (u 3) (z 1))
        (u (s 3) (ts 2) (x 3))
        (v (w 3) (x 3) (y 1))
        (w (e 4) (v 2) (x 3))
        (x (u 3) (v 2) (w 3))
        (y (v 2) (z 1) (ab 0))
        (z (ts 2) (y 2) (ab 0))
        (ab (y 1) (z 1)))
)
```

Figure 2: Graph with Heuristics for Towers of Hanoi

```
;; Loading file hillclimb.lsp ...
WARNING: DEFUN/DEFMACRO: redefining MEMBER; it was traced!
WARNING: DEFUN/DEFMACRO: redefining LENGTH; it was traced!
WARNING: DEFUN/DEFMACRO: redefining CDRLIST; it was traced!
WARNING: DEFUN/DEFMACRO: redefining HILLCLIMB; it was traced!
WARNING: DEFUN/DEFMACRO: redefining EXPANDNODE; it was traced!
(A C F E W V Y AB)
;; Loaded file hillclimb.lsp
T
Break 1 [2]>
```

Figure 3

```
 (setq temp hanoi)

(defun member (A L)
   (cond
      ((NULL L) '())
      ((EQ (car L) A) T)
      (T (member A (cdr L)))))

(defun length (L)
   (do ((M L)(sum 0))
      ((NULL M) sum)
```

```
      (setq M (cdr M))
      (setq sum(+ sum 1))))

(defun cdrList (start graph)
   (cond
      ((> (length graph) 2)
      (cond
         ((eq (car(car graph)) start) graph)
         ((cdrList start (cdr graph))))
      (cdr graph))))

(defun hillClimb (start target graph &optional path)
   (cond
      ((eq start target)
      (push start path)
      (write (nreverse path)))
      ((eq (car(car graph)) start)
      (push start path)
      (setq start (expandNode start target (cdr(car graph)) path))
      (hillClimb start target (cdrList start temp) path))
      ((hillClimb start target (cdrList start graph) path))))

(defun expandNode (node target connections path)
   (setq shortest 9999)
   (do ((M connections))
      ((NULL M) nextNode)
      (cond
         ((member target (car M))
         (setq nextNode target)
         (setq M (cdr M)))
         ((member (car(car M)) path) (setq M (cdr M)))
         ((cond
            ((< (car(cdr(car M))) shortest)
            (setq shortest (car(cdr(car M))))
            (setq nextNode (car(car M)))
            (setq M (cdr M)))
            ((setq M (cdr M))))))))
```

Figure 4: Rest of Code Used