Complexity of the Path Multi-Peg Tower of Hanoi

Daniel Berend* Amir Sapir ** ¶

Abstract

The Tower of Hanoi problem with $h \geq 4$ pegs is long known to require a sub-exponential number of moves in order to transfer a pile of n disks from one peg to another. In this paper we discuss the $\texttt{Path}_h$ variant, where the pegs are placed along a line, and disks can be moved from a peg to its nearest neighbor(s) only. Whereas in the simple variant there are $h(h-1)/2$ bi-directional interconnections among pegs, here there are only $h-1$ of them. Despite the significant reduction in the number of interconnections, the task of moving n disks between any two pegs is still shown to grow sub-exponentially as a function of the number of disks.

1 Introduction

In the well-known Tower of Hanoi problem, composed over a hundred years ago by Lucas [8], a player is given 3 pegs and a certain number n of disks of distinct sizes, and is required to transfer them from one peg to another. Initially all disks are stacked (composing a tower) on the first peg (the source) ordered by size, with the smallest at the top and the largest at the bottom. The goal is to transfer them to the third peg (the destination), moving only topmost disks, and never placing a disk on top of a smaller one. The known recursive algorithm, which may be easily shown to be optimal, takes $2^n - 1$ steps to accomplish the task.

Work on this problem still goes on, studying properties of solution instances, as well as variants of the original problem. Connections between Pascal's triangle, the Sierpiński gasket and the Tower of Hanoi are established in [6]. In [1] it is shown that, with a certain way of coding the moves, a string which represents an optimal solution is square-free. Another direction was concerned with various generalizations, such as having any initial and final configurations [4], and assigning colors to disks [9].

A natural extension of the original problem is obtained by adding pegs. One of the earliest versions is "The Reve's Puzzle" [3, pp. 1-2]. There it was presented in a limited form: 4 pegs and specified numbers of disks. The general setup of the problem, with any number $h > 3$ of pegs and any number of disks, was suggested in [12], with solutions in [13] and [5], shown recently to be identical [7]. An analysis of the algorithm reveals, somewhat surprisingly, that the solution grows sub-exponentially, at the rate of $\Theta(\sqrt{n}2^{\sqrt{2n}})$ for $h = 4$ (cf. [14]). The lower bound issue was considered in [15] and [2], where it has been shown to grow roughly at the same rate.

An imposition of movement restrictions among pegs generates many variants, and calls for representing variants by di-graphs, where a vertex designates a peg, and an edge − the permission to move a disk in the appropriate direction. In [11] the "three-in-a-row" arrangement (the $\texttt{Path}_3$) is discussed, as well as the (uni-directional) $\texttt{Cyclic}_4$. In [14], other 4-peg variants are dealt with: the $\texttt{Star}_4$ and the $\texttt{Path}_4$. Whereas for $\texttt{Star}_4$ a sub-exponential algorithm is presented, the complexity issue for $\texttt{Path}_4$ is left open.

In this paper, we prove that the number of moves required to transfer n disks between any two pegs, in $\texttt{Path}_h$, grows sub-exponentially as a function of n, for any $h \geq 4$. We present an algorithm which accomplishes the task.

2 Some notations

A *configuration* is any legal distribution of the disks among the pegs. A *perfect configuration* is one in which all disks reside on the same peg. Such a configuration will be denoted by $R_{i,n}$, where n is the number of disks and i is the peg containing the disks.

A problem instance is given by the number h of vertices, the number n of disks and two specified pegs *src* and *dst*, and we are required to move from the configuration $R_{src,n}$ to the configuration $R_{dst,n}$, in a minimal number of moves. We denote this task by $R_{src,n} \to R_{dst,n}$. Set $t_{h,n} = \max_{1 \leq i,j \leq h} |R_{i,n} \to R_{j,n}|$.

Given a di-graph G and a positive integer n, the corresponding *configuration graph* is the graph whose

*Departments of Mathematics and Computer Science, Ben-Gurion University, Beer-Sheva, 84105, Israel. Email:berend@cs.bgu.ac.il

**Department of Computer Science, Ben-Gurion University, Beer-Sheva, 84105, Israel. Email:amirsa@cs.bgu.ac.il

¶Supported in part by the Lynn and William Frankel Center for Computer Sciences.

vertices are all configurations of n disks over G, where there exists a directed edge from a vertex to another if one can pass from the former to the latter by a single disk move. Thus the problem of passing by a minimal number of moves from one configuration to another may be formulated as the problem of finding the shortest path between the corresponding vertices of the configuration graph.

3 The main result

The main question the paper addresses is: what is the complexity of $R_{1,n} \to R_{h,n}$ in $\texttt{Path}_h$? It is answered by

THEOREM 3.1. *For $h \geq 4$, the minimal number of moves for moving disks between any perfect configurations in $\texttt{Path}_h$ grows subexponentially as a function of n. Moreover, it is bounded above by $C_h n^{\alpha_h} 3^{\theta_h n^{\frac{1}{h-2}}}$, where:*

$$\theta_h = ((h-2)!)^{\frac{1}{h-2}} \ ,$$
$$\alpha_h = \tfrac{h-3}{h-2} \ ,$$
$$C_h = \tfrac{(h-2)\cdot 6^{h-3}}{\theta_h} \ .$$

Note that the bound holds also for $h = 3$ pegs, but since in this case the number of moves can be determined exactly, we will limit the discussion to $h \geq 4$.

Next, we present an algorithm which produces a (sub-exponentially long) list of moves to accomplish the task of moving n disks from any peg to any other peg. Basically, we follow the Frame-Stewart scheme of dividing the disks to two groups: m largest disks, and $n - m$ smallest disks. The general idea is to transfer the smallest disks to the farthest peg using all the pegs, move the largest disks to the destination using one less peg, and then move the smallest disks to the destination using all the pegs. Unlike the original problem, in $\texttt{Path}_h$ we often encounter the situation in which the best available temporary peg is also our destination, which makes it necessary to perform some additional steps.

The size of the group of larger disks $- m$, and hence of the group of smaller disks, is an important issue. It is determined by the number of available pegs and the number of disks, as can be seen in the algorithm.

A set of consecutive disks will be designated by specifying the smallest and largest disks in the set $- disk_{\mathrm{s}}$ and $disk_l$, respectively. The variable $free$ is the set of pegs available for this task (i.e., not occupied by any smaller disks). Note that src and dst are not necessarily endpoints of $free$.

4 Algorithm 1 and proof of Theorem 3.1

We start by proving the correctness of Algorithm 1. The Analysis of the length of the move sequence it produces will serve as a basis for proving Theorem 3.1.

Algorithm 1 $\texttt{move}(disk_{\mathrm{s}}, disk_l, src, dst, free)$

/* Moves a column of consecutive disks from *src* */
/* to *dst*, using the pegs in the set *free*. */
/* It is assumed that $|free| \geq 2$; */
/* moreover, if $|free| = 2$ then $disk_{\mathrm{s}} = disk_l$ */
$h \leftarrow |free|$
$n \leftarrow disk_l + 1 - disk_{\mathrm{s}}$
if $n > 1$ **then**
  $\alpha \leftarrow \frac{h-3}{h-2}$
  $m \leftarrow \left\lceil \frac{((h-2)!)^{\alpha}}{(h-3)!} n^{\alpha} \right\rceil$
  if $|src - dst| = h - 1$ **then**
    /*src and dst are at opposite endpoints of *free*/
    $st \leftarrow \frac{|dst-src|}{dst-src}$
    move$(disk_{\mathrm{s}}, disk_{l-\mathrm{m}}, src, dst, free)$
    move$(disk_{l-\mathrm{m}+1}, disk_l, src, dst-st, free-\{dst\})$
    move$(disk_{\mathrm{s}}, disk_{l-\mathrm{m}}, dst, src, free)$
    move$(disk_{l-\mathrm{m}+1}, disk_l, dst-st, dst, free-\{src\})$
    move$(disk_{\mathrm{s}}, disk_{l-\mathrm{m}}, src, dst, free)$
  else
    /*either *src* or *dst* (or both) is not an endpoint*/
    if $(src = free[1])$ or $(dst = free[1])$ **then**
      $tempT \leftarrow free[|free|]$
    else
      $tempT \leftarrow free[1]$
    end if
    move$(disk_{\mathrm{s}}, disk_{l-\mathrm{m}}, src, tempT, free)$
    move$(disk_{l-\mathrm{m}+1}, disk_l, src, dst, free-\{tempT\})$
    move$(disk_{\mathrm{s}}, disk_{l-\mathrm{m}}, tempT, dst, free)$
  end if
else
  slide(src, dst)
end if

At each invocation, the algorithm moves a set of disks from a specified peg to another specified peg, according to the values of the following variables. Assuming, for example, we have to move the disks from peg 1 to peg h, the variables are initialized as follows:

$disk_{\mathrm{s}} \leftarrow 1$; $disk_l \leftarrow n$;
$src \leftarrow 1$; $dst \leftarrow h$; $free \leftarrow \{1..h\}$;

The algorithm first determines whether it needs to move only one disk. If so, it calls the `slide` procedure – the only procedure that actually transfers disks – to move that single disk from *src* to *dst*, as detailed in Procedure 2.

Procedure 2 `slide`(src, dst)

```
/* Moves the top disk from src to dst. The */
/* procedure is invoked only when there is no */
/* obstruction along the way. */
st ← |dst−src| / (dst−src)
while src ≠ dst do
   take the topmost disk from src to src + st
   src ← src + st
end while
```

The main part of the algorithm handles the situation of $n > 1$ disks. The (near optimal) number of large disks, m, is determined. Then the relationship between *src*, *dst* and $free$ is examined. If the source and destination are the endpoints of the group of free pegs, certain additional steps are required.

Sketch of correctness proof. In order to perform correctly, the algorithm requires that:

a1. The group of pegs in $free$ will be consecutive; both *src* and *dst* must be members of $free$.

a2. Any disk that may reside on any peg in the group of free pegs, other than those in the set to be moved, will be larger than the largest disk in the set, thus not interfering with any move the algorithm may do.

a3. All the disks belonging to the range $disk_{\mathrm{s}}..disk_l$ are present and are on *src* at the beginning.

b. Either $|free| \geq 3$, or $|free| = 2$ and $disk_{\mathrm{s}} = disk_l$.

It is quite straightforward to see that, assuming requirements a1, a2, a3, are initially fulfilled, they are kept also in the recursive calls to `move`.

Recall that $h \leftarrow |free|$ and $n \leftarrow disk_l + 1 - disk_{\mathrm{s}}$. As to the initial conditions $h \geq 3$ or $h = 2 \Rightarrow n = 1$ (requirement b), we notice that, when $h = 3$, the value of m is 1. This ensures that those subsequent calls to `move` such that $h = 2$ will be given a single disk to move, thus performed by `slide` without further recursion. Moreover, if the procedure is invoked with some $n \geq 2$, each recursive call is actually with n' satisfying either $n' < n$, or $n' = n$ and a decrease of h by 1. Hence the depth of the recursion is bounded above by the initial value of n.

Denote the number of moves, needed by the algorithm, to transfer n disks using h pegs, by $f(h, n)$. This number depends on the choice of *src* and *dst*, so we will refer to $f(h, n)$ as the maximum over all possible (src, dst) pairs.

Proof. of Theorem 3.1. The recursive calls in Algorithm 1 have the property that the parameters h and n are not larger than in the original procedure. Moreover, at least one of them becomes strictly smaller. Hence it is natural to employ double induction on h and n. If invoked with $|free| = h$, the number of moves the algorithm requires to transfer n disks is $|dst - src|$ if $n = 1$, and is less than $C_h n^{\alpha_h} 3^{\theta_h n^{\frac{1}{h-2}}}$ for $n \geq 2$.

The case $n = 1$ is clear: exactly $|dst - src|$ moves are needed for a single disk to be transferred from *src* to *dst*, as suggested by the theorem. The bound is

$$C_h 1^{\alpha_h} 3^{\theta_h \cdot 1^{\frac{1}{h-2}}} = C_h 3^{\theta_h} = \frac{(h-2)\cdot 6^{h-3}}{((h-2)!)^{1/(h-2)}} 3^{((h-2)!)^{\frac{1}{h-2}}}$$
$$> 6^{h-3}\cdot 3 > h-1 \geq |dst-src| = f(h,1).$$

For $h = 3$, the algorithm works exactly as does Algorithm1 in [10] for the `3-in-a-row` – the $\mathtt{Path}_3$ graph. The number of moves required (assuming *src* and *dst* are at opposite sides of *free*) is $3^n - 1$. The substitution $h = 3$ in the upper bound suggested by the theorem yields

$$C_h n^{\alpha_h} 3^{\theta_h n^{\frac{1}{h-2}}} = 1 \cdot n^0 \cdot 3^{1\cdot n} = 3^n > f(3, n) \ .$$

Assume, for arbitrary fixed $h \geq 3$ and $n > 1$, that the bound in the theorem holds for all (n', h') with either $h' < h$ or both $h' = h$ and $n' < n$. The algorithm clearly gives:

$$f(h, n) \leq 3f(h, n - m) + 2f(h - 1, m) \ .$$

By the induction hypothesis:

$$f(h,n) < 3C_h(n-m)^{\alpha_h} \cdot 3^{\theta_h (n-m)^{\frac{1}{h-2}}} + 2C_{h-1} m^{\alpha_{h-1}} \cdot 3^{\theta_{h-1} m^{\frac{1}{h-3}}} \tag{4.1}$$

Now:

$$(n-m)^{\alpha_h} = n^{\alpha_h}\left(1 - \tfrac{m}{n}\right)^{\alpha_h} < n^{\alpha_h}\left(1 - \tfrac{\alpha_h m}{n}\right) \ . \tag{4.2}$$

Similarly

$$\begin{aligned}\theta_h(n-m)^{\frac{1}{h-2}} &= \theta_h n^{\frac{1}{h-2}}(1-\tfrac{m}{n})^{\frac{1}{h-2}} \\ &< \theta_h n^{\frac{1}{h-2}}\left(1-\frac{\theta_h^{h-3}n^{\frac{h-3}{h-2}}}{(h-2)n\theta_{h-1}^{h-3}}\right) \\ &= \theta_h n^{\frac{1}{h-2}} - 1\end{aligned} \tag{4.3}$$

and

$$\begin{aligned}\theta_{h-1}m^{\frac{1}{h-3}} &\le \theta_{h-1}\left(\frac{\theta_h^{h-3}}{\theta_{h-1}^{h-3}}n^{\frac{h-3}{h-2}}+1\right)^{\frac{1}{h-3}} \\ &= \theta_h n^{\frac{1}{h-2}}\left(1+\frac{\theta_{h-1}^{h-3}}{\theta_h^{h-3}}n^{-\frac{h-3}{h-2}}\right)^{\frac{1}{h-3}} \\ &< \theta_h n^{\frac{1}{h-2}} + \frac{(h-3)!\theta_h^2}{(h-3)(h-2)!}n^{-\frac{h-4}{h-2}} \\ &= \theta_h n^{\frac{1}{h-2}} + \frac{\theta_h^2}{(h-3)(h-2)}n^{-\frac{h-4}{h-2}} \\ &< \theta_h n^{\frac{1}{h-2}} + 1\,.\end{aligned} \tag{4.4}$$

Substituting (4.2), (4.3) and (4.4) in (4.5), we obtain:

$$\begin{aligned}f(h,n) &< 3C_h n^{\alpha_h}(1-\tfrac{\alpha_h m}{n})3^{\theta_h n^{\frac{1}{h-2}}-1} \\ &\quad +2C_{h-1}m^{\alpha_{h-1}}3^{\theta_h n^{\frac{1}{h-2}}+1} \\ &= C_h n^{\alpha_h}(1-\tfrac{\alpha_h m}{n})3^{\theta_h n^{\frac{1}{h-2}}} \\ &\quad +6C_{h-1}m^{\alpha_{h-1}}3^{\theta_h n^{\frac{1}{h-2}}} \\ &= (C_h n^{\alpha_h} - \tfrac{C_h n^{\alpha_h}\alpha_h m}{n} \\ &\quad +6C_{h-1}m^{\alpha_{h-1}})3^{\theta_h n^{\frac{1}{h-2}}}\,.\end{aligned} \tag{4.5}$$

The second and third terms on the right-hand side may be omitted since:

$$\begin{aligned}&6C_{h-1}m^{\alpha_{h-1}} - \tfrac{C_h n^{\alpha_h}\alpha_h m}{n} \\ &= m^{\alpha_{h-1}}\left(6\tfrac{(h-3)6^{h-4}}{\theta_{h-1}} - \tfrac{(h-2)6^{h-3}}{n\cdot\theta_h}n^{\frac{h-3}{h-2}}\cdot\tfrac{h-3}{h-2}\cdot m^{\frac{1}{h-3}}\right) \\ &= m^{\alpha_{h-1}}(h-3)6^{h-3}\left(\frac{1}{\theta_{h-1}} - \frac{n^{\frac{-1}{h-2}}m^{\frac{1}{h-3}}}{\theta_h}\right) \\ &\le m^{\alpha_{h-1}}(h-3)6^{h-3}\left(\frac{1}{\theta_{h-1}} - \frac{n^{\frac{-1}{h-2}}}{\theta_h}\left(\frac{\theta_h^{h-3}}{\theta_{h-1}^{h-3}}n^{\frac{h-3}{h-2}}\right)^{\frac{1}{h-3}}\right) \\ &= 0\,.\end{aligned}$$

Thus we find that:

$$f(h,n) < C_h n^{\alpha_h}3^{\theta_h n^{\frac{1}{h-2}}}\,.$$

Hence (4.5) implies the required inequality.

References

[1] J. P. Allouche, D. Astoorian, J. Randall, and J Shallit, *Morphisms, squarefree strings, and the Tower of Hanoi puzzle*, Amer. Math. Monthly, 101 (1994), pp. 651–658.

[2] X. Chen and J. Shen, *On the Frame-Stewart conjecture about the Towers of Hanoi*, SIAM J. on Computing, 33(3) (2004), pp. 584–589.

[3] H. E. Dudeney, *The Canterbury Puzzles (and Other Curious Problems)*, E. P. Dutton, New York, 1908.

[4] M. C. Er, *The complexity of the generalised cyclic Towers of Hanoi*, J. Algorithms, 6 (1985), pp. 351–358.

[5] J. S. Frame, *Solution to advanced problem 3918*, Amer. Math. Monthly, 48 (1941), pp. 216–217.

[6] A. M. Hinz, *Pascal's triangle and the Tower of Hanoi*, Amer. Math. Monthly, 99 (1992), pp. 538–544.

[7] S. Klavzar, U. Milutinovic, and C. Petr, *On the Frame-Stewart algorithm for the multi-peg Tower of Hanoi problem*, Discrete Applied Math., 120 (2002), pp. 141–157.

[8] E. Lucas, *Récréations Mathématiques*, volume III, Gauthier-Villars, Paris, 1893.

[9] S. Minsker, *The Towers of Hanoi rainbow problem: coloring the rings*, J. Algorithms, 10 (1989), pp. 1–19.

[10] A. Sapir, *The Tower of Hanoi with forbidden moves*, The Computer Journal, 47(1) (2004), pp. 20–24.

[11] R. S. Scorer, P. M. Grundy, and C. A. B. Smith, *Some binary games*, Math. Gazette, 280 (1944), pp. 96–103.

[12] B. M. Stewart, *Advanced problem 3918*, Amer. Math. Monthly, 46 (1939), p. 363.

[13] B. M. Stewart, *Solution to advanced problem 3918*, Amer. Math. Monthly, 48 (1941), pp. 217–219.

[14] P. K. Stockmeyer, *Variations on the four-post Tower of Hanoi puzzle*, Congr. Numer., 102 (1994), pp. 3–12.

[15] M. Szegedy, *In how many steps the k peg version of the Towers of Hanoi game can be solved?* Lecture Notes in Computer Science, 1563 (1999), pp. 356–361.