
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 20, no. 2, pp. 461–479 (2016)
DOI: 10.7155/jgaa.00403

Exponential vs. Subexponential Tower of Hanoi Variants

Daniel Berend[1] *Amir Sapir*[2,3]

[1]Departments of Mathematics and Computer Science,
Ben-Gurion University of the Negev, Israel
[2]Department of Computer Science, Sapir Academic College, Yehudah, Israel
[3]The Center for Advanced Studies in Mathematics at Ben-Gurion University,
Beer-Sheva, Israel

Abstract

We deal here with Tower of Hanoi variants played on digraphs. A major source for such variants is achieved by adding pegs and/or restricting direct moves between certain pairs of pegs. It is natural to represent a variant of this kind by a directed graph whose vertices are the pegs, and an arc from one vertex to another indicates that it is allowed to move a disk from the former peg to the latter, provided that the usual rules are not violated. We denote the number of pegs by h. For example, the variant with no restrictions on moves is represented by the Complete graph $\mathtt{K}_h$; the variant in which the pegs constitute a cycle and moves are allowed only in one direction is represented by the uni-directional graph $\mathtt{Cyclic}_h$.

For all 3-peg variants, the number of moves grows exponentially fast with n. However, for $h \geq 4$ pegs, this is not the case. For example, for $\mathtt{Cyclic}_h$ the number of moves is exponential for any h, while for a path on 4 vertices it is $O(\sqrt{n}3^{\sqrt{2n}})$.

This paper characterizes the graphs for which the transfer of a tower of size n of disks from a peg to another requires exponentially many moves as a function of n.

To this end we introduce the notion of a shed, as a graph property. A vertex v in a strongly-connected directed graph $G = (V, E)$ is a *shed* if the subgraph of G induced by $V(G) - \{v\}$ contains a strongly connected subgraph on 3 or more vertices. Graphs with sheds will be shown to be much more efficient than those without sheds, for the particular domain of the Tower of Hanoi puzzle. Specifically, we show how, given a shed, we can indeed move a tower of disks from any peg to any other within $O(\lambda^{n^\alpha})$ moves, where $\lambda > 1$ and $\alpha = \frac{1}{2} + o(1)$. For graphs without a shed, this is impossible.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
March 2014	December 2015	July 2016	August 2016	October 2016

Published:
October 2016

Article type:
Regular paper

Communicated by:
M. Kaufmann

Research supported in part by the Sapir Academic College, Israel

E-mail addresses: berend@cs.bgu.ac.il (Daniel Berend) amirsa@cs.bgu.ac.il (Amir Sapir)

1 Introduction

Given are 3 pegs and a certain number n of disks of distinct sizes. Initially, the disks form a tower: the largest at the bottom of one of the pegs (the source), the second largest on top of it, and so on, until the smallest at the top of that peg. The well-known Tower of Hanoi problem asks: how do we optimally move the tower to another peg (the destination peg), subject to the following rules:

1. At each step only one disk is moved.
2. The moved disk must be a topmost one.
3. At any moment, no disk may reside on a smaller one.

We shall refer to constraints 1-3 as the *Hanoi rules* (HR).

The puzzle was invented over a hundred years ago by Lucas [24]. Ever since then, it was studied from numerous points of view. For example, in [2, 3, 1], algebraic properties of the solution to the Tower of Hanoi are discussed, and it is shown that the string representing the optimal solution – where the i^{th} character denotes the disk moving at the i^{th} step – is square-free. This line of work was extended in [4]. The combinatorial aspect has been considered too (cf. [19, 21]). As computer science education has evolved, the Tower of Hanoi problem has been used as a common example, demonstrating the elegance of recursive programming. The reader is referred to [35, 10] for a review of the history of the problem, to [36] for an extensive bibliography of papers on various lines of research in the field and to the recent (and first!) book [18] on many of the mathematical aspects of the subject.

Many variants of the original puzzle have come up, some of which will be described here, though not chronologically. Without changing the basic peg structure (3 pegs, each pair being connected bi-directionally), one direction is solving the problem for any initial and final configurations – arrangements of disks among the pegs such that HR3 is not violated (cf. [15]). Other challenging versions have been proposed and solved in [26, 27, 28, 29, 25]. In another direction, a disk may reside on top of a smaller one, with various limitations, [22, 12].

Another version of the original problem is where we impose restrictions on the movements between pegs, which was discussed in several papers. In [32, 35, 17], the "three-in-a-row" ($\texttt{Path}_3$) arrangement is studied. The uni-directional cycle ($\texttt{Cyclic}_3$) has been solved in [5, 14]. It is natural to represent a variant by a directed graph. A necessary and sufficient condition for a variant (on 3 or more pegs) to be solvable, for any source and destination pegs and any number of disks, is that the corresponding graph is strongly connected [23]. For 3 pegs there are 5 (up to isomorphism) strongly connected variants. A single optimal algorithm for all these variants was obtained in [31], accompanied with an explicit formula for the minimum number of moves for each variant. (Note that individual algorithms and explicit formulas were known beforehand for the common variant $\texttt{K}_3$, for $\texttt{Path}_3$, and for $\texttt{Cyclic}_3$, as mentioned above.)

Probably the first version with four or more pegs is "The Reve's Puzzle" [13, pp. 1−2], in which there are 4 pegs and various specific numbers of disks. It has been generalized to any number of pegs and any number of disks in [33], with solutions in [34] and [16], which were (among several other solutions) proved to be identical in [20]. Analysis of the algorithms given for these problems reveals, somewhat surprisingly, that the number of moves in the solution grows sub-exponentially as a function of the number of disks n. In the case of 4 pegs, it grows like $\Theta(\sqrt{n}2^{\sqrt{2n}})$ (cf. [35]). The lower bound issue was considered in [37] and [11], where it has been shown to grow at a rate close to that yielded by the algorithm.

Allowing four or more pegs, and imposing restrictions on whether a direct move of a disk from peg i to peg j is allowed, for each pair (i, j) of pegs, we obtain a huge number of graphs (83 non-isomorphic strongly connected digraphs on 4 vertices already), and no algorithm seems a natural candidate to be optimal. The question whether a variant is sub-exponential had been resolved only for particular ones: Star [35] (which was proved to be sub-exponential), Cyclic [7] (exponential) and Path [9] (sub-exponential; specifically, for $h = 4$ pegs, the number of moves grows slower than $1.6\sqrt{n} \cdot 3^{\sqrt{2n}}$.)

The fact that, even for the original multi-peg variants on complete graphs, it is not known whether the proposed algorithms are optimal, indicates that the complexity issue for this directed-graph generalization is non-trivial. Facing the wealth of variants, we would like an easy way to determine, given a variant, whether the number of moves required grows exponentially or sub-exponentially fast. This paper presents a simple necessary and sufficient condition for a variant to be sub-exponential. In addition, it shows that

- almost all graphs are sub-exponential;
- the exponential graphs form a family with a concise description; and
- all the sub-exponential graphs are at most "slightly worse" than $\mathtt{K}_4$ and $\mathtt{Path}_4$.

In Section 2 we describe the problem domain. The main results are introduced in Section 3. The proofs of the theorems are presented in Section 4.

2 Problem domain and notations

Any arrangement of pegs and their immediate connections, such that each peg is reachable from each other, constitutes a variant. As mentioned above, it is natural to represent a variant by a digraph G, whose vertices are the pegs, and an arc from one vertex to another designates the ability of moving a disk from the former peg to the latter, provided that the HR are obeyed. In the rest of the paper, when we mention a *variant graph*, we mean a strongly connected

simple directed graph (which amounts to requiring that each peg is reachable from each other) on $h \geq 3$ vertices.

A *configuration* is a distribution of the disks among the pegs, in accordance with HR.3. A configuration is *perfect* if all disks reside on the same peg. Such a configuration will be denoted by $R_{i,n}$, where n is the number of disks (disk 1 being the smallest and disk n the largest) and i the peg containing the disks.

Given a variant graph G and a positive integer n, the corresponding *configuration graph* $G^{(n)}$ is the graph whose vertices are all configurations of n disks over G, where there is an arc from a vertex to another if one can pass from the former to the latter by a single disk move. More generally, for any pair of configurations there is a corresponding *task*, of passing from the first to the second. Thus, an optimal solution of a task corresponds to a shortest path between the vertices of $G^{(n)}$ representing the task's initial and final configurations. The diameter of $G^{(n)}$ is denoted by $D_n(G)$.

A task is *perfect* if both its initial and final configurations are perfect. We use the notation $R_{i,n} \to R_{j,n}$ both for the task and for a minimal length solution of it. The length of such a minimal solution is expressed by $|R_{i,n} \to R_{j,n}|$. We set $d_{i,j,n}(G) = |R_{i,n} \to R_{j,n}|$. An interesting quantity is $d_n(G) = \max_{i,j} d_{i,j,n}(G)$, which we call the *little diameter* of $G^{(n)}$. When the identity of the graph to which we refer is clear, we may omit it from this notation. For example, we may write D_n instead of $D_n(G)$.

Formally, a *move* is composed of the disk being moved, the peg on which it resides prior to the move, and the peg to which it is transferred. A *solution* to a task is a sequence of moves accomplishing it. The algorithms constructed in the proofs of our results produce solutions to all perfect tasks.

A variant graph G is *H-exp* if $D_n(G)$ grows exponentially fast as a function of the number of disks, namely there exist $C > 0$ and $\lambda > 1$ such that $D_n(G) \geq C\lambda^n$ for all n. G is *H-subexp* if for every $\varepsilon > 0$ there exists a constant $C = C(\varepsilon)$ such that $D_n(G) \leq C(1+\varepsilon)^n$. In principle, it could have been the case that a graph is neither H-exp nor H-subexp. However, it will follow, in particular, from Theorem 1, below, that the rate of growth of $D_n(G)$ is sufficiently regular to ensure this dichotomy. In addition, H-subexp graphs admit an explicit subexponential bound. Namely, we prove that, if G is H-subexp, then $D_n(G) \in O(\lambda^{n^\alpha})$ moves, where $\lambda > 1$ and $\alpha = \frac{1}{2} + o(1)$.

3 Main results

The main problem we study is how to identify, given a variant graph, whether it is H-subexp or H-exp. We start by proving that the number of moves behaves regularly as a function of the number of disks.

For a variant graph G, denote $\lambda_G = \inf_{n \geq 1} \sqrt[n]{d_{n+1}(G)}$.

Theorem 1 *For any variant graph G*

$$\lim_{n \to \infty} \sqrt[n]{d_n(G)} = \lambda_G \,.$$

Let us recall Corollary 1 of [8]:

Proposition 1 *For any variant graph and any number of disks,*

$$D_n \leq (2n-1)d_n\,.$$

Combining Theorem 1 and Proposition 1 one can infer

Corollary 1 *For every graph G and $\varepsilon > 0$ there exists an n_0 such that*

$$\lambda_G^{n-1} \leq d_n(G) \leq D_n(G) \leq (\lambda_G + \varepsilon)^{n-1}, \qquad n \geq n_0.$$

Remark 1 Corollary 1 states that d_n and D_n are not too far apart, justifying that it suffices to focus on d_n.

The main question this paper answers is: what property must G have so that $\lambda_G = 1$? To answer this, we need the following notion.

Definition 1 A *shed* in a strongly connected digraph G (see Fig. 1) is a vertex w with the property that the graph induced by $V(G) - \{w\}$ contains a strongly connected subgraph of size at least 3.

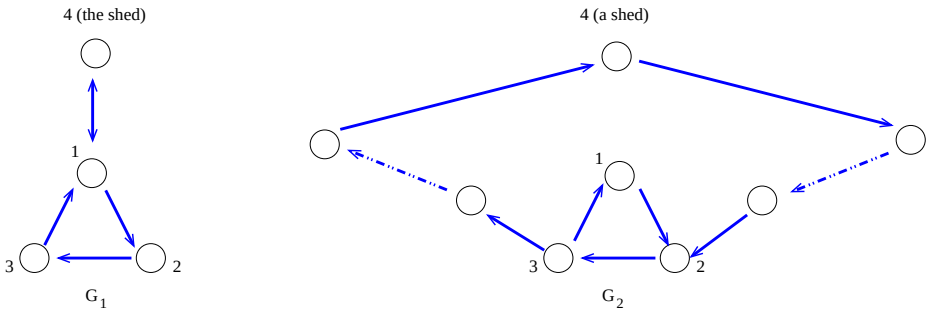


Figure 1: Two graphs with sheds

Our main result is

Theorem 2 *A variant graph G with $h \geq 3$ vertices is H-subexp if and only if it contains a shed.*

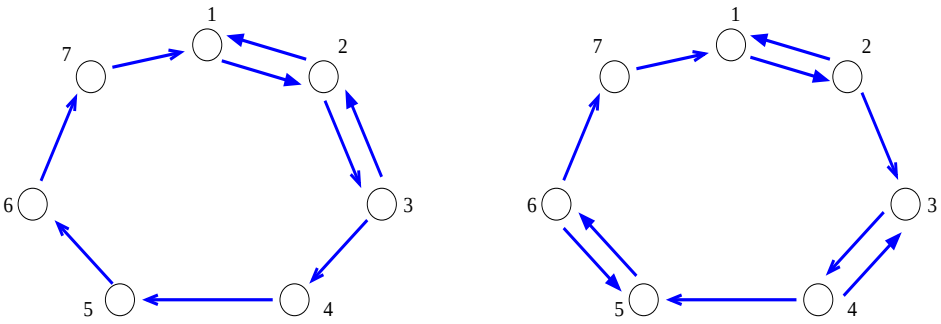
Of course, for $h = 3$ the graph cannot contain a shed, so such a graph is always H-exp (which is trivial anyway).

Unlike the proof of sub-exponentiality for $\mathtt{K}_4$ (cf. [35]) or for the path $\mathtt{P}_h$ [9] for $h \geq 4$, the proof for general graphs containing a shed is quite cumbersome. The reason is, intuitively, that the former graphs have (at least) two sheds. This means that there are a lot more options of keeping a block of small disks on some peg for a while, taking care in the meantime of other (large) disks. On

the other hand, if the graph has a single shed, then no obvious sub-exponential algorithm comes to mind, and in fact it may seem surprising at first glance that such an algorithm exists at all.

It is possible to give a very explicit description of those strongly connected graphs not containing a shed. To this end, we need

Definition 2 A *moderately enhanced cyclic* graph (see Fig. 2) is a graph composed of a single uni-directional cycle containing all the vertices, augmented with some arcs in the reverse direction, provided that no two such arcs are adjacent.



(a) A non moderately enhanced cyclic graph (since it has two consecutive back arcs)

(b) A moderately enhanced cyclic graph

Figure 2: A moderately enhanced cyclic graph and a graph with a shed

Proposition 2 gives a simple characterization of shedless graphs.

Proposition 2 *A strongly connected graph on $h \geq 4$ vertices has no shed if and only if it is a moderately enhanced cyclic graph.*

4 Proofs

To prove Theorem 1, we first need

Lemma 1 *For every strongly connected graph G,*

$$d_{m+n-1} \leq d_m d_n, \qquad m, n \geq 1.$$

Proof: We need to show that, for every $1 \leq i, j \leq h$ with $i \neq j$, the task $R_{i,m+n-1} \to R_{j,m+n-1}$ can be performed within at most $d_m d_n$ moves. Let T_m be an optimal solution for the task $R_{i,m} \to R_{j,m}$, and $T_{i',j',n}$ with $i' \neq j'$ be optimal solutions for the tasks $R_{i',n} \to R_{j',n}$. Clearly, $|T_{i',j',n}| \leq d_n$ for every (i', j').

Denote by $T_{m,(l)}$ the l-th move within T_m. We are going to combine the above-mentioned solutions into a (not necessarily optimal) solution for $R_{i,m+n-1} \to R_{j,m+n-1}$, as follows. Consider the smallest n disks $1, 2, \ldots, n$ as a single disk − number 1 − and the largest $m-1$ disks $n+1, n+2, \ldots, n+m-1$ as disks $2, 3, \ldots, m$, respectively. Write the solution for $T_{i,j,m}$, then replace each move of disk 1 from peg i' to peg j' by $T_{i',j',n}$ and each move of a disk $k \geq 2$ by the same move, but of disk $k+n-1$. The procedure is described in Algorithm 1.

Algorithm 1 uses several pre-defined functions:
`DiskNumber`$(T_{m,(l)})$ retrieves the disk being moved in $T_{m,(l)}$.
`UpdateDiskNumber`$(T_{m,(l)}, s)$ replaces the disk performing the move $T_{m,(l)}$ by disk s.
`SourcePeg`$(T_{m,(l)})$ and `DestinationPeg`$(T_{m,(l)})$ retrieve the peg on which the disk resides prior to move l and after that move, respectively.

Algorithm 1 `Combine()`

```
/* Combining existing solutions with m and n disks to obtain a solution */
/* with m + n − 1 disks. The symbol '*' stands for concatenation. */
T_{m+n−1} ← [ ]                                  /* The empty sequence */
/* Producing the required solutions with m and n disks. */
T_m ← (R_{i,m} → R_{j,m})
for i' ← 1 to h do
  for j' ← 1 to h, j' ≠ i' do
    T_{i',j',n} ← (R_{i',n} → R_{j',n})
  end for
end for
for l ← 1 to |T_m| do
  r ← DiskNumber(T_{m,(l)})
  if r ≥ 2 then
    T_{m+n−1} ← T_{m+n−1} * UpdateDiskNumber(T_{m,(l)}, r + n − 1)
  else
    i' ← SourcePeg(T_{m,(l)})
    j' ← DestinationPeg(T_{m,(l)})
    T_{m+n−1} ← T_{m+n−1} * T_{i',j',n}
  end if
end for
return T_{m+n−1}
```

Since each move in T_m is replaced either by $T_{i',j',n}$ for some (i', j') (if it is a move of disk 1) or by a single move of some other disk (otherwise), the total number of moves increases at most by a factor of d_n. In particular:

$$|R_{i,m+n-1} \to R_{j,m+n-1}| \leq d_m d_n\,, \qquad m, n \geq 1\,.$$

Consequently:

$$d_{m+n-1} = \max_{i,j} d_{i,j,m+n-1} \leq d_m d_n.$$

Proof of Theorem 1: Putting $b_n = \ln d_{n+1}$, we obtain by Lemma 1:

$$b_{m+n} \leq b_n + b_m\,, \qquad m, n \geq 0\,.$$

The sequence $(b_n)_{n=0}^{\infty}$ is thus sub-additive, which implies that the sequence $\frac{b_n}{n}$ converges to its greatest lower bound (cf. [30, p. 198]), and hence so does the sequence

$$e^{b_n/n} = \sqrt[n]{d_{n+1}}\,,$$

thus proving the theorem.

Proof of Corollary 1: Since $\sqrt[n]{d_{n+1}(G)}$ converges to λ_G from above, for any $\varepsilon > 0$ and sufficiently large n

$$\lambda_G \leq \sqrt[n]{d_{n+1}(G)} \leq \lambda_G + \varepsilon\,,$$

or, equivalently,

$$\lambda_G^{n-1} \leq d_n(G) \leq (\lambda_G + \varepsilon)^{n-1}\,.$$

In combination with Proposition 1, this proves the corollary.

To establish the proof of the 'if' part (which is the main part) of Theorem 2, we will need several definitions. Unless explicitly stated otherwise, G will denote a strongly connected graph with $h \geq 4$ vertices containing a shed w, and G' – a strongly connected subgraph of size at least 3 of the graph left after w is removed. (Note that G may contain more than one shed, and for each choice of a shed there may be several appropriate strongly connected subgraphs of G. For our purposes, though, any choice of a shed vertex and a strongly connected component in accordance with that shed is suitable.) Taking a shortest path from $w \in V(G) - V(G')$ to $V(G')$, we denote the *entrance vertex* to G' – the last vertex on this path and the only one that belongs to G' – by e. Similarly, an *exit vertex* $x \in V(G')$ is the first vertex on a shortest path leading from $V(G')$ to w. These paths will be denoted by $p_{w,e}$ and $p_{x,w}$, respectively. The sequence of moves of disk r along, say, the path $p_{w,e}$ is expressed by $t_{w,e,r}$. $S_{(l)}$ is the l-th move in a sequence of moves S.

Example 1 In each of the graphs G_1 and G_2 in Figure 1, vertex $w = 4$ is a shed. (In fact, in G_2 all vertices but 2 and 3 are sheds.) For G_1 we have $e = x = 1$, whereas for G_2 we have $e = 2, x = 3$.

As hinted above, most of the work is required for the 'if' part of Theorem 2. The following lemma shows that some of the tasks are of sub-exponential complexity. Let `Inner` $= V(G') - \{e, x\}$. We first show that tasks in which the source is the shed and the destination lies in `Inner` (or vice versa) are sub-exponential.

Lemma 2 *Let G, G', w be as above and $v \in$ `Inner`. Then $|R_{w,n} \to R_{v,n}|$ and $|R_{v,n} \to R_{w,n}|$ grow sub-exponentially fast as functions of n.*

Proof: Fix a number $\lambda \geq \lambda_{G'}$. It suffices to show that for some $\alpha < 1$ there exists a constant C such that

$$|R_{w,n} \to R_{v,n}| \leq C\lambda^{n^\alpha}, \qquad n = 1, 2, \ldots ,$$

and

$$|R_{v,n} \to R_{w,n}| \leq C\lambda^{n^\alpha}, \qquad n = 1, 2, \ldots .$$

In fact, we will show that this is the case for every $\alpha > \frac{1}{2}$. The proof is based on the procedures `ShedToInner` (Algorithm 2) and `InnerToShed` (Algorithm 3), which perform, respectively, the tasks $R_{w,n} \to R_{v,n}$ and $R_{v,n} \to R_{w,n}$ in

$$f_{w,v}(n) \leq C\lambda^{n^\alpha} \tag{1}$$

and

$$f_{v,w}(n) \leq C\lambda^{n^\alpha} \tag{2}$$

moves. We discuss `ShedToInner` in detail.

Algorithm 2 `ShedToInner`(w, v, n)

```
/* The algorithm moves a tower of n disks from the shed w to a vertex
v ∈ V(G') − {e, x}, where G', e and x are defined following the proof of
Corollary 1. */
if n ≥ 1 then
  m ← ⌈n − n^α⌉
  T ← ShedToInner(w, v, m)
  for r ← m + 1 to n do
    T ← T * t_{w,e,r}
    T ← T* InnerToShed(w, v, m)
    T ← T* Accumulate(e, v, r, m + 1 ... r − 1)
    T ← T* ShedToInner(w, v, m)
  end for
end if
return T
```

For each $i, j \in V(G')$ we choose a simple path from i to j. The sequence of moves of a single disk r (henceforth *transfer*) along this path is denoted by $t_{i,j,r}$. A sequence of moves joining a single disk r, currently on peg i, to the set of disks $\{m+1, \ldots, r-1\}$, currently on peg j, utilizing only the subgraph G', is produced by `Accumulate`$(i, j, r, m+1 \ldots r-1)$ (detailed in Algorithm 4), provided that there are no other disks in G'.

The algorithm `ShedToInner`(w, v, n) splits the disks into two sets. One is that of the smaller disks $- \{1, 2, \ldots, m\}$, where $m = \lceil n - n^\alpha \rceil$, which will be repeatedly moved between the shed and vertex v. The other set $- \{m+1, m+2, \ldots, n\} -$ is that of the larger ones, that will be accumulated one by one on vertex v. At the beginning, the smallest m disks are moved (recursively) to

Algorithm 3 `InnerToShed`(w, v, n)

```
/* The algorithm moves a tower of n disks from a vertex v ∈ V(G') − {e, x}
to the shed w. */
if n ≥ 1 then
  m ← ⌈n − n^α⌉
  T ← InnerToShed(w, v, m)
  for r ← n downto m + 1 do
    T ← T∗ Split(v, x, r, m + 1 . . . r − 1)
    T ← T∗ ShedToInner(w, v, m)
    T ← T ∗ t_{x,w,r}
    T ← T∗ InnerToShed(w, v, m)
  end for
end if
return T
```

vertex v, revealing the larger ones. Then the algorithm performs $n-m$ iterations, in each of which the next disk from the set $\{m+1, m+2, \ldots, n\}$ is handled. We enumerate the first iteration by $m+1$, the second by $m+2$, and so on. Immediately prior to iteration r, $m < r \leq n$, disks $1, 2, \ldots, m, m+1, \ldots, r-1$ are at vertex v, and disks $r, r+1, \ldots, n$ are at the shed vertex, w. In iteration r:

- Disk r (whose transfer is the goal of the iteration) is temporarily moved to vertex e.
- The smallest m disks are moved from vertex v to the shed.
- Disk r is joined to disks $m+1, \ldots, r-1$, using G' only.
- The smallest m disks are returned from the shed to vertex v.

By the end of the iteration, disks $1, 2, \ldots, m, m+1, \ldots, r$ are at v, while the larger disks, $r+1, \ldots, n$, are still at the shed.

Principally, Algorithm `InnerToShed` works in a similar way. It splits the disks into the set of the smaller disks $-$ $\{1, 2, \ldots, m\}$, where $m = \lceil n - n^{\alpha} \rceil$, which will be repeatedly moved between vertex v and the shed. The other set $-$ $\{m+1, m+2, \ldots, n\}$ $-$ is that of the larger disks, that will be accumulated one by one on the shed. The procedure `Split` serves as the opposite to `Accumulate`.

The procedure `Accumulate` is detailed in Algorithm 4.

The procedure `MoveIn`$G'(i, j, D)$ moves a set of disks D from vertex i to j, using G' only. The call `MoveIn`$G'(v, e, m+1 \ldots r-1)$ moves the tower consisting of disks $m+1, \ldots, r-1$ from v to e. The call `MoveIn`$G'(e, v, m+1 \ldots r)$ performs an analogous task (from e to v, with the additional disk). Thus, by Corollary 1, the number of moves produced by the $(r-m)$-th invocation of `Accumulate` is bounded above by $C'\lambda^{r-m}$, for some constant $C' = C'(G')$. (In fact, Algorithm 4 is not necessarily the most efficient way to perform the accumulation, but it suffices for our needs.)

Algorithm 4 `Accumulate`$(e, v, r, m+1 \ldots r-1)$

/* Starting with disks $m+1 \ldots r-1$ on peg v and disk r on peg e, the algorithm */
/* unites them so that all disks will reside on peg v, using G' only. */
$T \leftarrow$ `MoveIn`G'`(`$v, e, m+1 \ldots r-1$`)`
$T \leftarrow T *$ `MoveIn`G'`(`$e, v, m+1 \ldots r$`)`
return T

Since each step carried out by `ShedToInner`, `Accumulate` and `MoveIn`G' obeys the HR, and at the end all disks reside on vertex v, the correctness of the algorithms is straightforward.

Inequalities (1) and (2) are proved simultaneously by induction on n, where the constant C will be determined later. Since $f_{w,v}(1) \leq h-1$ and $f_{v,w}(1) \leq h-1$, both inequalities are valid for $n=1$ provided $C \geq h-1$. Assume that (1) and (2) hold with n replaced by every smaller number. For simplicity, we consider $f_{w,v}$ (consideration of $f_{v,w}$ is analogous) for the case of n disks. Algorithm 2 makes:

- $n-m+1$ calls to `ShedToInner`(w, v, m),
- $n-m$ calls to `InnerToShed`(w, v, m),
- $n-m$ transfers of a single disk along a simple path in G, and
- $n-m$ calls to `Accumulate`$(e, v, r, m+1 \ldots r-1)$, for $r = m+1, m+2, \ldots, n$.

Since $m = \lceil n - n^\alpha \rceil$, we have

$$\begin{aligned} f_{w,v}(n) &\leq (n-m+1)f_{w,v}(m) + (n-m)f_{v,w}(m) \\ &\quad + (n-m)h + \sum_{i=1}^{n-m} C'\lambda^i \\ &\leq (2n^\alpha + 1)\max(f_{w,v}(m), f_{v,w}(m)) + nh + \tfrac{C'\lambda}{\lambda-1}\lambda^{n^\alpha} \qquad (3) \\ &\leq (2n^\alpha+1)C\lambda^{m^\alpha} + C''\lambda^{n^\alpha} \\ &\leq (2n^\alpha+1)C\lambda^{(n-n^\alpha+1)^\alpha} + C''\lambda^{n^\alpha} \\ &< 3n^\alpha C\lambda^{(n-\frac{1}{2}n^\alpha)^\alpha} + C''\lambda^{n^\alpha}, \end{aligned}$$

where $C'' = \frac{C'\lambda}{\lambda-1} + 1$, for sufficiently large n.
Now $(n - \frac{1}{2}n^\alpha)^\alpha = n^\alpha(1-\frac{1}{2}n^{\alpha-1})^\alpha = n^\alpha(1 - \frac{\alpha}{2}n^{\alpha-1} + O(n^{2\alpha-2}))$, and hence

$$f_{w,v}(n) \leq C\lambda^{n^\alpha}\tfrac{3n^\alpha}{\lambda^{\frac{\alpha}{2}n^{2\alpha-1}}} + C''\lambda^{n^\alpha} \leq C\lambda^{n^\alpha},$$

where we assume that $C > C''$.

This completes the proof of (1). The proof of (2) is analogous.

The following lemma is required for the proof of the 'only if' part of Theorem 2.

Lemma 3 *There exists a constant C with the following property. For every graph G, not containing a strongly connected component of size 3 or more, and for any initial configuration, the number of different disks which can participate in any legal sequence of moves is bounded above by $Ch^{\frac{1}{2}\lg h+2}$, where $h = |V(G)|$. Here $\lg h \equiv \log_2 h$.*

Proof: Let T be any legal move sequence and i any peg. Set $l = \lfloor (h-2)Ch^{\frac{\lg h}{2}} \rfloor + 2$ (where the constant C is sufficiently large; see below). Consider the first move of the disk residing at the l-th place (counting from the top) of peg i before T is started. Right before this move, all smaller disks (residing on peg i prior to T) have to be spread on $h-2$ of the other pegs, making it necessary for at least one peg to accept more than $Ch^{\frac{1}{2}\lg h}$ disks from peg i, contradicting [6, Theorem 1.3] if C is sufficiently large. It follows that the overall number of disks that can participate in any task is bounded above by $h(h-2)Ch^{\frac{1}{2}\lg h} + 2h$, proving the lemma.

Proof of Theorem 2: (a) Here we prove the 'if' part of the theorem. Let G', w, e, x be as before. Take $\lambda > \lambda_{G'}$. We will show that, if $\alpha > \frac{1}{2}$ is arbitrarily fixed, then for every pair of vertices $i, j \in V(G)$, there exists a constant K such that

$$|R_{i,n} \to R_{j,n}| \leq K\lambda^{n^\alpha}, \qquad n = 1, 2, \ldots . \tag{4}$$

Take $v \in V(G') - \{e, x\}$. Due to Lemma 2, it remains to consider the following cases:

- Case 1: $i = w$, $j \in V(G') - \{v\}$,
- Case 2: $i \in V(G') - \{v\}$, $j = w$,
- Case 3: $i, j \in V(G')$,
- Case 4: $i, j \in V(G)$, where at least one of i, j does not belong to $V(G') \cup \{w\}$.

Remark 2 *In fact, there is some overlap between the cases. For example, in Case 1 we need only to take care of the case where $j = e$, and in Case 2 only of $i = x$, but this is of no consequence.*

Case 1: We employ `ShedTo`$G'(w, j, n)$ (Algorithm 5), which moves a tower of n disks from the shed to any vertex $j \in V(G') - \{v\}$. In order to place the next largest disk at the destination, it performs the following:

- Moves all the disks from the shed to vertex v.
- Moves all disks but the largest in the set from v back to the shed.
- Moves the largest disk from vertex v to the destination.

Remark 3 *Setting the set of largest disks to contain more than one disk at a time, we could get a more efficient algorithm; however, for our purpose, this algorithm suffices.*

The procedures `ShedToInner` and `InnerToShed` are as in the proof of Lemma 2. Algorithm 5 provides the details.

Algorithm 5 `ShedTo`$G^{'}(w, j, n)$

```
/* The algorithm moves a tower of n disks from the shed w to any j ∈
V(G') − {v}. */
if j ∈ V(G') − {e, x} then
  T ← ShedToInner(w, j, n)
else
  v ← a vertex in V(G') − {e, x}
  T ← [ ]                    /* The empty sequence */
  for r ← n downto 2 do
    T ← T* ShedToInner(w, v, r)
    T ← T* InnerToShed(w, v, r − 1)
    T ← T * t_{v,j,r}
  end for
  T ← T * t_{w,j,1}
end if
return T
```

If a single disk can be moved uninterruptedly from a vertex to another, the number of moves is less than h. This is the situation whenever the $t_{v,j,r}$ is performed (disk r is moved along a simple path from v to j, where these two vertices are in $G^{'}$ and all other disks currently in $G^{'}$ are larger than r). `ShedToInner`(w, v, n) and `InnerToShed`(w, v, n) require at most $C\lambda^{n^{\alpha}}$ moves each, for an appropriate C. We now bound the number of moves $f_{w,j}(n)$ performed by `ShedTo`$G^{'}(w, j, n)$:

$$f_{w,j}(n) \quad \leq nC\lambda^{n^{\alpha}} + nC\lambda^{n^{\alpha}} + nh \leq 3Cn\lambda^{n^{\alpha}} .$$

Case 2: We employ $G^{'}$`ToShed`, which moves a tower of n disks from any vertex $i \in V(G^{'}) - \{v\}$ to the shed. Clearly:

$$f_{i,w}(n) \leq nC\lambda^{n^{\alpha}} + nC\lambda^{n^{\alpha}} + nh \leq 3Cn\lambda^{n^{\alpha}} ,$$

as for $f_{w,j}$.

Algorithm 6 $G^{'}$`ToShed`(w, i, n)

```
/* The algorithm moves a tower of n disks from any vertex i ∈ V(G') − {v}
to the shed. */
if i ∈ V(G') − {e, x} then
  T ← InnerToShed(w, i, n)
else
  v ← a vertex in V(G') − {e, x}
  T ← t_{i,w,1}
  for r ← 2 to n do
    T ← T ∗ t_{i,v,r}
    T ← T∗ ShedToInner(w, v, r − 1)
    T ← T∗ InnerToShed(w, v, r)
  end for
end if
return T
```

Case 3 is a consequence of the first two cases, by first moving all disks from i to w, and then moving them from w to j.

Case 4: Observe that, once G' has been set, each vertex in $V(G) - V(G')$ may serve as a shed. Thus, for any $i \in V(G')$ and $j \in V(G) - V(G')$, both inequalities $|R_{i,n} \to R_{j,n}| \leq K\lambda^{n^\alpha}$ and $|R_{j,n} \to R_{i,n}| \leq K\lambda^{n^\alpha}$ hold. Now let $i, j \in V(G) - V(G')$. Take a vertex $k \in V(G')$. We move a tower of disks from i to j by first moving it from i to k, and then from k to j. This proves the correctness of (4) in this case.

(b) We turn to the 'only if' part of the theorem. Take any strongly connected graph G without a shed. Start with any configuration C. Denote by v_0 the vertex on which disk 1 resides in C. What is the maximum number of moves which may be done without moving disk 1, without reaching any configuration more than once?

Clearly, as long as we do not move disk 1, all other disks may use only the graph induced by $V(G) - \{v_0\}$, which has no strongly connected component of size at least 3. By Lemma 3, since in this graph there are $h - 1$ vertices, $m = C(h-1)^{\frac{1}{2}\lg(h-1)+2}$ is an upper bound on the number of disks that may participate in any sequence of moves. This implies that the length of any such sequence of moves cannot exceed the length of the longest possible simple path in the configuration graph for m disks, which is at most $b = (h-1)^m - 1$. (The other disks do not participate, and thus we ignore them.) Note that any sequence longer than that must reach at least one configuration more than once, and thus contains a loop, which can never appear in optimal solutions. We emphasize that h, m, and therefore b, do not depend on the number of disks n.

We assert that $d_n(G)$ (and, in fact, even $\min_{i,j} d_{i,j,n}(G)$) grows exponentially fast as a function of n. Take an arbitrary pair of vertices $i, j \in V(G)$. Denote

$a_n = d_{i,j,n}$ for $n \geq 1$. We shall show that

$$a_n \geq A \cdot \left(\frac{b+1}{b}\right)^n$$

for an appropriate $A > 0$.

Let S_{n+1} be an optimal solution for the task $R_{i,n+1} \to R_{j,n+1}$, and $l(S_{n+1})$ its length. Divide S_{n+1} into consecutive subsequences of length $b+1$ (and a remainder of length at most b at the end). Define two operations on such sequences:

(1) `Drop`: Remove all moves of disk 1.

(2) `Decr`: Decrement the disk number in each move by 1.

Starting with a sequence involving disks $1, 2, \ldots, n+1$, and applying `Drop` and then `Decr` to it, we arrive at a sequence involving disks $1, 2, \ldots, n$. Moreover, applying `Drop` and then `Decr` to S_{n+1}, we obtain a (not necessarily optimal) solution S_n for $R_{i,n} \to R_{j,n}$, where each previous subsequence of length $b+1$ shrank to length at most b. Since $l(S_{n+1}) = a_{n+1}$ and $l(S_n) \geq a_n$, this yields

$$a_n \leq l(S_n) \leq \left\lfloor \frac{l(S_{n+1})}{b+1} \right\rfloor \cdot b + b \leq \frac{a_{n+1}}{b+1} b + b\,.$$

Hence:

$$a_{n+1} \geq \frac{b+1}{b} a_n - (b+1)\,.$$

Since $a_n \underset{n\to\infty}{\longrightarrow} \infty$, this implies that

$$a_n \geq A \cdot \left(\frac{b+1}{b}\right)^n\,,$$

thus completing the proof of this part.

Proof of Proposition 2: Denote by $\mathcal{G}_{\text{ns}}$ the family of graphs on $h \geq 4$ vertices which do not have a shed, and by $\mathcal{G}_{\text{ec}}$ the family of moderately enhanced cyclic graphs (see Fig. 2.(b)). Recall that a moderately enhanced cyclic graph is composed of a uni-directional cycle (henceforth the *main cycle*) of all vertices, augmented perhaps by up to $\lfloor \frac{h}{2} \rfloor$ non-adjacent arcs in the reverse direction (henceforth *back arcs*).

It is straightforward to show that $\mathcal{G}_{\text{ec}} \subseteq \mathcal{G}_{\text{ns}}$. For the direction $\mathcal{G}_{\text{ns}} \subseteq \mathcal{G}_{\text{ec}}$, let $G \in \mathcal{G}_{\text{ns}}$. Distinguish between two cases.

- Case 1: G is (strictly) directed: There are two vertices i, j in G such that $(i,j) \in E(G)$ but $(j,i) \notin E(G)$.

  Since G is strongly connected, there is a path from j to i. The path has to pass through at least one vertex in $V(G) - \{i,j\}$, say k. Clearly, the graph

induced by $\{i, j, k\}$ and the other vertices along the path from j to i is strongly connected with at least 3 vertices. The graph is actually G itself. Indeed, had there been a vertex not included in it, that vertex would be a shed – contradicting the 'shedlessness' of G. This path, together with the arc from i to j, constitutes the (only) main cycle in G. Obviously, there are no two consecutive back arcs along this cycle; if there were such arcs between, say, $\{i', j', k'\}$, all other vertices would have been sheds.

- Case 2: The graph is undirected.

  Take any two adjacent vertices i, j. Since G is strongly connected, there is at least one vertex k, adjacent, say, to j. The subgraph induced by $\{i, j, k\}$ is strongly connected. All other vertices of G are sheds, contradicting the assumption that $G \in \mathcal{G}_{\text{ns}}$.

This completes the proof of the proposition.

Acknowledgements

The authors would like to thank the referee for his important suggestion, which made the statement of the paper sharper, and for the rest of his remarks.

References

[1] J.-P. Allouche, D. Astoorian, J. Randall, and J. Shallit. Morphisms, square-free strings, and the Tower of Hanoi puzzle. *Amer. Math. Monthly*, 101:651–658, 1994.

[2] J.-P. Allouche, V. Berthe, and J. Shallit. Sur des points fixes de morphismes du monoïde libre. *RAIRO Inform. Théor. Appl.*, 23:235–249, 1989.

[3] J.-P. Allouche and F. Dress. Tours de Hanoi et automates. *RAIRO Inform. Théor. Appl.*, 24:1–15, 1990.

[4] J.-P. Allouche and A. Sapir. Restricted Towers of Hanoi and morphisms. *Lect. Notes in Comput. Sci.*, 3572:1–10, 2005. doi:10.1007/11505877_1.

[5] M. D. Atkinson. The cyclic Towers of Hanoi. *Inf. Process. Lett.*, 13:118–119, 1981. doi:10.1016/0020-0190(81)90123-X.

[6] D. Azriel and D. Berend. On a question of Leiss regarding the Hanoi Tower problem. *Theoretical Comput. Sci.*, 369:377–383, 2006. doi:10.1016/j.tcs.2006.09.019.

[7] D. Berend and A. Sapir. The Cyclic multi-peg Tower of Hanoi. *Trans. on Algorithms*, 2(3):297–317, 2006. doi:10.1145/1159892.1159893.

[8] D. Berend and A. Sapir. The diameter of Hanoi graphs. *Inf. Process. Lett.*, 98:79–85, 2006. doi:10.1016/j.ipl.2005.12.004.

[9] D. Berend, A. Sapir, and S. Solomon. Subexponential upper bound for the Path multi-peg Tower of Hanoi. *Disc. Appl. Math.*, 160(10-11):1465–1483, 2012. doi:10.1016/j.dam.2012.02.007.

[10] J.-P. Bode and A. M. Hinz. Results and open problems on the Tower of Hanoi. *Congr. Numer.*, 139:113–122, 1999.

[11] X. Chen and J. Shen. On the Frame-Stewart conjecture about the Towers of Hanoi. *SIAM J. on Computing*, 33(3):584–589, 2004. doi:10.1137/S0097539703431019.

[12] Y. Dinitz and S. Solomon. Optimality of an algorithm solving the Bottleneck Tower of Hanoi problem. *Trans. on Algorithms*, 4(3):1–9, 2008. doi:10.1145/1367064.1367065.

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

[14] M. C. Er. The Cyclic Towers of Hanoi: a representation approach. *Comput. J.*, 27(2):171–175, 1984. doi:10.1093/comjnl/27.2.171.

[15] M. C. Er. A general algorithm for finding a shortest path between two n-configurations. *Inform. Sci.*, 42:137–141, 1987. doi:10.1016/0020-0255(87)90020-X.

[16] J. S. Frame. Solution to advanced problem 3918. *Amer. Math. Monthly*, 48:216–217, 1941.

[17] D.-J. Guan. Generalized Gray codes with applications. *Proc. Natl. Sci. Counc. ROC(A)*, 22(6):841–848, 1998.

[18] A. M. Hinz, S. Klavžar, U. Milutinović, and C. Petr. *The Tower of Hanoi – Myths and Maths*. Birkhäuser, 2012. doi:10.1007/978-3-0348-0237-6.

[19] S. Klavžar, U. Milutinović, and C. Petr. Combinatorics of topmost discs of multi-peg Tower of Hanoi problem. *Ars Combinatoria*, 59:55–64, 2001.

[20] S. Klavžar, U. Milutinović, and C. Petr. On the Frame-Stewart algorithm for the multi-peg Tower of Hanoi problem. *Disc. Appl. Math.*, 120(1-3):141–157, 2002. doi:10.1016/S0166-218X(01)00287-6.

[21] S. Klavžar, U. Milutinović, and C. Petr. Hanoi graphs and some classical numbers. *Expo. Math.*, 23:371–378, 2005. doi:10.1016/j.exmath.2005.05.003.

[22] C. S. Klein and S. Minsker. The super Towers of Hanoi problem: large rings on small rings. *Disc. Math.*, 114:283–295, 1993. doi:10.1016/0012-365X(93)90373-2.

[23] E. L. Leiss. Solving the "Towers of Hanoi" on graphs. *J. Combin. Inform. System Sci.*, 8(1):81–89, 1983.

[24] É. Lucas. *Récréations Mathématiques*, volume III. Gauthier-Villars, Paris, 1893.

[25] W. F. Lunnon and P. K. Stockmeyer. New Variations on the Tower of Hanoi. *13th Intern. Conf. on Fibonacci Numbers and Their Applications*, 2008.

[26] S. Minsker. The Towers of Antwerpen problem. *Inf. Process. Lett.*, 38(2):107–111, 1991. doi:10.1016/0020-0190(91)90230-F.

[27] S. Minsker. The Linear Twin Towers of Hanoi problem. *ACM SIGCSE Bull.*, 39(4):37–40, 2007. doi:10.1145/1345375.1345410.

[28] S. Minsker. Another brief recursion excursion to Hanoi. *ACM SIGCSE Bull.*, 40(4):35–37, 2008. doi:10.1145/1473195.1473215.

[29] S. Minsker. The classical/linear Hanoi hybrid problem: regular configurations. *ACM SIGCSE Bull.*, 41(4):57–61, 2009. doi:10.1145/1709424.1709446.

[30] G. Pólya and G. Szegő. *Problems and Theorems in Analysis*, volume I. Springer Verlag, 1972.

[31] A. Sapir. The Tower of Hanoi with forbidden moves. *Comput. J.*, 47(1):20–24, 2004. doi:10.1093/comjnl/47.1.20.

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

[33] B. M. Stewart. Advanced problem 3918. *Amer. Math. Monthly*, 46:363, 1939.

[34] B. M. Stewart. Solution to advanced problem 3918. *Amer. Math. Monthly*, 48:217–219, 1941.

[35] P. K. Stockmeyer. Variations on the Four-Post Tower of Hanoi puzzle. *Congr. Numer.*, 102:3–12, 1994.

[36] P. K. Stockmeyer. Tower of Hanoi bibliography, 2005. URL: http://www.cs.wm.edu/~pkstoc/biblio2.pdf.

[37] M. Szegedy. In how many steps the k peg version of the Towers of Hanoi game can be solved? *Lect. Notes in Comput. Sci.*, 1563:356–361, 1999. doi:10.1007/3-540-49116-3_33.