Explorations in 4-peg Tower of Hanoi

Ben Houston
exocortex.org

Hassan Masum
hmasum.com

November 29, 2004

Abstract

Finding an optimal solution to the 4-peg version of the classic Tower of Hanoi problem has been an open problem since the 19th century, despite the existence of a "presumed-optimal" solution. We verify that the presumed-optimal Frame-Stewart algorithm for 4-peg Tower of Hanoi is indeed optimal, for up to 20 discs. We also develop a distributed Tower of Hanoi algorithm, and present 2D and 3D representations of the state transition graphs. Finally, two variants (k-out-of-order and k-at-a-time) and an analogy with distributed agent software are suggested.

1 Introduction: History and Overview

The Tower of Hanoi is a well known problem in recreational mathematics; it has also been widely used in computer science as a paradigmatic teaching example for recursive solution methods. In the basic version, a stack of discs of mutually distinct sizes is arranged on one of three pegs, with the size restriction that no larger disc is atop a smaller disc. The problem is then to move the entire stack of discs to another of the three pegs by moving one disc at a time, and always maintaining the size restriction.

As the instruction sheet for the puzzle (first sold in 1883) said: *"According to an old Indian legend, the Brahmins have been following each other for a very long time on the steps of the altar in the Temple of Benares, carrying out the moving of the Sacred Tower of Brahma with sixty-four levels in fine gold, trimmed with diamonds from Golconde. When all is finished, the Tower and the Brahmins will fall, and that will be the end of the world!"*

It is well known that this basic version is solvable in 2^n-1 moves. But in generalizing the problem to four or more pegs, the analysis gets much

harder. (Details and variants of the 4-peg problem can be found in [Stockmeyer 1994].) There is a solution method - the Frame-Stewart algorithm or "presumed-optimal solution" - which is the best known for 4 pegs, but has not been proved to be optimal. A lower bound to the order of magnitude of moves required is given in [Szegedy 1999], and these results are sharpened in [Chen and Shen 2004] to show that the presumed-optimal solution has the same order of magnitude as the actual optimal solution.

Since no optimal solution is known, we have verified that the presumed-optimal solution is indeed optimal for up to 20 discs, via an exhaustive search of the possible moves. Our work builds on that of [Hinz and Bode 1999], who found that the presumed-optimal solution was in fact best for up to 17 discs. We verify and extend their results, and give results for 5 and 6 pegs as well.

Two further approaches toward solving the problem are discussed. First, the solution method can be modified to work on a distributed network of machines, which could allow verification for larger numbers of discs. Second, we construct 2D and 3D representations of the state transition graphs for various Tower of Hanoi puzzles, which may help point out symmetries and suggest potential lines of attack on the general problem.

Finally, we close with suggestions for k-out-of-order and k-at-a-time variants, and an analogy between the Hanoi problem and future distributed agent software.

2 Verifying 4-peg Tower of Hanoi

2.1 The Presumed-Optimal Solution

The presumed-optimal solution for 4-peg Tower of Hanoi (henceforth referred to as 4-peg ToH) has the following general form:

Step 0 Number the pegs from 0 to 3. Our goal is to move all discs from peg 0 to peg 3, respecting the constraints that only one disc moves at a time, and a smaller disc is never under a larger disc.

Step 1 Move the smallest d-k discs from peg 0 to peg 1, using all 4 pegs.

Step 2 Move the largest k discs from peg 0 to peg 3, *without* using peg 1. (In other words, move these discs using the optimal 3-peg method, which is already known.)

Step 3 Move the smallest d-k discs from peg 1 to peg 3, using all 4 pegs.

Denoting by T(d, p) the minimal number of moves for a d-disc problem on p pegs, we see that the above method implies a recursive solution:

$$T(d, 4) = 2 * T(d - k, 4) + T(k, 3)$$

Analysis of this recursion results in a closed-form solution for the minimizing value of k. One can also derive closed-form expressions for the total number of moves required (see [Stockmeyer 1994] and [Klavzar and Milutinovic 2002]). However, this solution is only an answer to the original problem (how many moves does a 4-peg ToH require) under the assumption that the optimal solution has the general structure given above.

Despite a century of exploration into the problem, the optimality of the above solution has not been proven. It is therefore worthwhile to actually try all possible solutions to 4-peg ToH, to verify by demonstration that no shorter method exists.

2.2 Algorithm Details and Implementation

This section is devoted to explaining our verification technique, which has demonstrated that no better solution than the above exists for 20 or fewer discs.

The basic verification technique uses a breadth-first search through the space of all states reachable from the starting state. In other words, starting from an initial state, we exhaustively check the set of all states reachable by 1 move, then by 2 moves, and so on.

If a final state has not been found before the critical number of states implied by the presumed-optimal solution, then the presumed-optimal solution has been verified to be optimal for the current number of pegs and discs. (And since the presumed-optimal solution is a constructive method, we will not only know of the existence of an optimal solution with the given number of states, but also of its specific move sequence.)

Clearly ToH is a memoryless process: regardless of how a state was reached, the minimal number of moves to go from that state to a final state remains the same. Hence we can maintain a cache of previously-visited states, which starts out empty. Then at each level of the BFS search, we add each state not already in the cache. This allows a substantial saving of computational time - by the nature of the BFS procedure, any state which is in the cache must already have its successor states included in the set of states being searched. Hence any new state which is found to already be in the cache can safely have the corresponding BFS branch terminated.

It's important to have an efficient implementation of cache lookup and storage. We used the optimizations in (Hinz and Bode 1999), along with a few of our own[1]. One can save a substantial amount of work by just checking when the bottom (largest) disc moves - we assumed that, by symmetry, once the bottom disc moves then the remaining moves should be the inverse of the previous move sequence.

2.3 Results

For ToH with 4 to 6 discs, we give results in Tables 1 to 3, at the end of this document. The key figure is in the second column, showing the minimum # of moves that we have proved through exhaustive search.

The computing environment was a single-processor Windows 2000 machine with 1 GB RAM, at 600 MHz. Time and memory used are also included for illustrative purposes; the nontrivial values for 1 disc largely represent fixed computational overhead for startup, data structure creation, etc. (Of course, time and memory are machine-specific, but note that the exhaustive search method is exponential in both quantities.)

3 Distributed ToH

3.1 Distributing memory and processing across machines

It is difficult to reduce the memory requirements without significantly increasing the running time of a ToH solver on a single machine. However, we have come up with a distributed solution for ToH. The idea is to arrange n computers into a ring, where:

- x computers store the previous move data,
- y computers store the current move data, and
- z computers store the next move data

with x + y + z = n.

This distributed solution is theoretically scalable to any number of computers - allowing one to postpone the point at which memory problems overwhelm the solver.

We estimate that, given a modest number of computers with plenty of RAM each, one could figure out up to ToH(4,23) in a week or so of processing

[1] *See the code for details: www.exocortex.org/toh/*

time. As a proof of concept, we implemented a prototype distributed solver, which can run on 3 machines at once. (For decent speed it requires at least 100Mbit Ethernet, and preferably gigabit Ethernet).

3.2 Algorithm for three machines

Given three machines, we assign each one a distinct role: an S[n-1] client, an S[n] client and an S[n+1] client.

The operation of the cluster is as follows:

Startup The S[n] client will be initially seeded with the starting state. Both the S[n-1] and the S[n+1] clients will have their stores empty.

Step 1 The S[n] client generates all the adjacent states to the states in its store and sends those new states to the S[n-1] client. The S[n-1] client compares all the states it receives from the S[n] client against its store and only sends the ones that are not present in its store to the S[n+1] client. The S[n+1] client adds the states it receives from the S[n-1] client to its store if the states are not already present.

Step 2 After the S[n] client has send all adjacent states to the S[n-1] client, the S[n-1] client has sent its last state to the S[n+1] client and the S[n+1] client has processed the final state from S[n-1], the clients will *rename themselves* using the following mapping:

- S[n+1] $\mapsto$ S[n]
- S[n] $\mapsto$ S[n-1]
- S[n-1] $\mapsto$ S[n+1]

Now go to step 1 if a halting state has not yet been found.

This algorithm doesn't send too much information around, since it exploits the "client renaming" trick.

Although we have not implemented it, two natural strategies suggest themselves for extension to more than three machines.

First, one could have each machine store and be responsible for just a certain equivalence class of states, in order to partition the BFS search. New states could be stored up and then collectively transmitted to the appropriate machine for cache lookup. As one adds machines, they would automatically configure to partition the states across equivalence classes in a balanced way

(which should be feasible since we are searching the whole Hanoi graph and know some of its properties).

Second, one could also partition the machines by BFS iteration (carrying the idea in the previous section further, by having separate machines for more BFS iteration steps). Analyzing the schemes to find the best partitioning of work between machines, to maximize overall effective computation and minimize inter-machine communication, is an interesting distributed programming challenge - one which we leave to the interested reader as an exercise.

4 Visualizing Tower of Hanoi

4.1 State Transition Graphs

The state transition graph for ToH is a graphical representation of all possible states, for a given number of discs and pegs.

For n discs and 4 pegs, we represent each possible state as an n-dimensional vector. Each coordinate has a value from 1 to 4, where that coordinate value represents the peg which the associated disc is on. For example, (2, 4) would be one state in the 2-disc, 4-peg problem, with disc 1 on peg 2 and disc 2 on peg 4.

Discs can be numbered from 1 being the smallest up to n being the largest. Without loss of generality, call peg 1 the starting peg and peg 4 the goal peg. Of course, many times we will have multiple discs on the same peg - for example, the starting state is (1, 1, 1, ..., 1). However, there is no ambiguity, since by the conditions of the problem multiple discs on the same peg can only be arranged in one valid way: from largest on the bottom to smallest on the top.

To use this vector representation of states to see all the states graphically, a layout is first chosen. In the examples shown here, a 2D or 3D position for each state is chosen according to properties of the state (e.g. minimal distance of the state from the starting state). So, each possible state (i.e. legal arrangement of the discs) is identified with a distinct node of the state transition graph.

Next, edges are drawn between each pair of states which are mutually accessible - i.e. for which one can go from either state to the other via a legal move. (Since all moves in the standard ToH are reversible, there is no need to consider directed edges.) In our simulations, these edges were drawn during the breadth-first search of all possible states for a given number of pegs and discs.

4.2 2D Representations

For 3-peg ToH, it is well known that the above procedure using a triangular layout generates a state transition graph. For larger and larger numbers of discs, these graphs are better and better approximations to the Sierpinksi Gasket triangular fractal; see [Bogomolny] for an explanation of why this occurs. Analysis of these approximation graphs and some inferences for the properties of the Sierpinski Gasket are in [Hinz and Schief 1990].

However, the corresponding graph for 4 pegs is not obvious, and so we experimented with several different layouts. It is of interest to try to find a layout for 4-peg ToH which shows some of the patterns of the state transition graph in an understandable way. These graphs have interesting regularities which may aid in analysis.

In Figure 1, we have a layout of the state space of the T(4,4) problem - Tower of Hanoi with 4 discs on 4 pegs. One can see the many symmetries, and repeated subproblem structure at various scales. (Note that a projection onto 2D or 3D requires a choice of layout, which makes some symmetries easier to see than others; there may well be another layout which more clearly projects the symmetries.)

When generating the graph, one can shade the edges according to the first time that each possible state transition is visited in a breadth-first search from the initial state. This gives an intuitive visual depiction of the "distance" of each edge from the starting state[2].

These visualizations may help infer properties of generalized ToH problems, and are interesting mathematical objects to study in their own right. Note that solving Hanoi is equivalent to finding particular shortest paths in this graph.

4.3 3D Representations

Is there a natural 3-dimensional representation generated by the state transition graph for 4-peg ToH? The representations are not just finite approximations to the "Sierpinski Tetrahedron" in 3-space as one might first guess, as the graph contains many more edges. Not much is known yet about this graph, aside from basic information about the number of edges [Klavzar 1998], and the fact that it is non-planar for 3 or more discs [Hinz and Parisse 2002].

One way to generate such a 3D representation is to lay out the state transition graph in 3D space, and use a force layout method: nodes repel

[2] *Shaded versions of some graphs can be found at www.exocortex.org/toh/.*

each other and change position until a stable state is reached where forces are in balance. This stable state is the graphical depiction.

We have made some videos using this process[3]. However, this is just a beginning - the reader is encouraged to try to find a natural 3D representation and explore its analytic properties. It may be that such representations of state space graphs can be useful for other variants of Hanoi as well, e.g. if shortest paths through such graphs can be approximated.

5 Conclusions, Variants, and an Analogy

Our empirical work may interest both those seeking to get more data on this classic problem, and those looking for interesting computational challenges. Of course, an analytical proof of an optimal solution method remains the ultimate goal, but meanwhile each small step helps.

We hope that the reader who may not have been familiar with the 4-peg version has learned a little about a long-standing puzzle. And if the reader is a mathematician or programmer, we encourage trying out solution methods for the puzzle or its variants.

In fact, the basic Hanoi setup has been generalized in a number of ways, as seen in the literature - we suggest two that may be new below. There may also be an application (or at least a metaphor) of Hanoi to large-scale software systems of the future.

5.1 Hanoi with Movement Constraints

In traditional Hanoi, discs can move to any other peg as long as the resulting configuration is legal. Variants have been suggested where there are various constraints on which kinds of inter-peg moves are permissible, in addition to the universal constraint that no disc is above a smaller disc - see [Stockmeyer 1994]. An example is the cyclic puzzle, where the pegs are arranged in a cycle.

One could generalize and consider Hanoi where movement is restricted to travel along an arbitrary (possibly directed) graph connecting the pegs. Developing a theory which covers this more general case is one route to answering questions about particular subcases (besides being an interesting question in its own right).

[3] *Results can be seen at www.exocortex.org/toh/videos/.*

5.2 k-at-a-time

Suppose that, instead of moving only one disc at a time, one is allowed to move k discs at a time. It is easy to see that an upper bound for the number of moves required can be found through relating k-at-a-time to the traditional Hanoi version. Denote by T(d, p, k) a d-disc problem with p pegs and k at a time; then:

$$T(d, p, k) \leq T(\lceil d/k \rceil, p, 1)$$

The idea is that, given a starting configuration of all discs on the same peg, we can consider the move-k-at-a-time ability as redefining groups of discs to be "meta-discs". So for moving k at a time, one would start from the top of the discs and label each group of k as a meta-disc. Now the traditional Hanoi problem on these meta-discs is also a valid solution to the k-at-a-time variant, and hence the upper bound is valid.

5.3 k-out-of-order

A variant is to move just one disc at a time, but relax the smaller-above-larger condition, so that a larger disc can be above a smaller one as long as it is not too much larger. We call this "k-out-of-order", to mean that a larger disc can be above a smaller one if they differ in size by at most k (where the size of a disc is its distance from the top in the traditional starting configuration, ranging from 1 smallest to d largest).

An upper bound is again possible using the "meta-disc" idea, though it is weaker than in the k-at-a-time case:

$$T(d, p, k) \leq T(\lceil d/\lfloor (k+1)/2 \rfloor \rceil, p, 1) * \lfloor (k+1)/2 \rfloor$$

One can see this by noting that it corresponds to using meta-discs as above, but that the component discs of each meta-disc may be inverted if that meta-disc has been moved an odd number of times. Hence one could in the worst case have two adjacent groups of discs, both of which are inverted. The top group's bottom disc and bottom group's top disc will differ by 2M-1 in size, where M is the maximum size of a meta-disc, and so M = $\lfloor \frac{k+1}{2} \rfloor$ for k-out-of-order. Combine this with the fact that each movement of a disc in a meta-disc counts as a separate move, and we have the above bound.

5.4 An Analogy with Distributed Agents

It is intriguing to consider potential situations where a Tower of Hanoi problem might arise. One might consider a system with discrete queues, or (in

an engineering context) physical piles each of which has some load-bearing capacity.

Another system could be some kind of security problem, with a larger disc corresponding to a more secure protocol or firewall. This actually turns out to admit an analogy with a number of similarities to Hanoi:

Disc Corresponds to a software agent, prioritized task, or OS layer.

Smaller above Larger If a disk is a prioritized task, then we're not allowed to put a lower-priority task on a machine where a higher-priority task is already running. And if a disc is an OS layer or software agent with some set of weaknesses, then this restriction might correspond to the idea that a less secure protocol or code can only be used on top of a more secure one, and not vice versa (i.e. weaknesses are cumulative and each protocol is "installed on top of" the previous one).

Pegs and Adjacency A peg is a machine or environment (of which a limited number might exist, perhaps for reasons of security or trust). Adjacency (and hence position) of discs would be important if there were discrete layers that could not be bypassed - as would happen for security reasons.

Move 1 at a time Logistics or asynchrony, or failsafe protocols (i.e. only when the success of each move is reported is the next move taken). A more secure (or otherwise better) agent or task refuses to run or be installed on top of a less secure one.

Just 1 of each disc Why not just copy tasks or programs? One can imagine future difficult-to-copy software - a personalized agent or avatar, or an application from a security-conscious agency. Difficult copyability could be implemented with a cryptographically-strong OS combined with communication with some trusted external environment. (In fact, some variant might well be the case for future personal software agents that travel through potentially hostile computing environments, containing personal information and directives.)

Move all discs between pegs Replicate the full security stack or environment in another computational environment.

This speculative analogy suggests that problems like Hanoi could be used to demonstrate how difficult computational tasks might get in an environment where code blocks or layers do not trust each other, and cannot simply

be copied. These constraints imply that the logistics of a simple move of code or software agents between machines would get much harder.

If agents have more complex interactions (regarding which other agents and environments are required or prohibited to be present before a move) than just the one-dimensional relation implied by the analogy to size of disc, then deadlocks would easily occur. And even without this, having too many layers or security levels would lead to a computationally intractable situation.

5.5 Acknowledgments

We thank Paul Stockmeyer and Andreas Hinz for suggestions and feedback.

5.6 References

Alexander Bogomolny. Sierpinski Gasket and Tower of Hanoi. Online at www.cut-the-knot.com/triangle/Hanoi.shtml.

A. Brousseau. Tower of Hanoi with more pegs. In J. Recreational Math., Volume 8 (1975-76), pp 169-178.

Xiao Chen and Jian Shen. On the Frame-Stewart Conjecture about the Towers of Hanoi. SIAM Journal on Computing. Volume 33, Number 3 (2004), pp 584-589.

Andreas M Hinz and JP Bode. Results and open problems on the Tower of Hanoi. In Congressus Numerantium, Volume 139 (1999), pp 113-122.

Andreas M Hinz and Daniele Parisse. On the Planarity of Hanoi Graphs. Exposition. Math. 20(2002), 263-268.

Andreas M Hinz and Andreas Schief. The average distance on the Sierpinski gasket. Probability Theory Related Fields Volume 87 (1990), pp 129-138.

S. Klavzar and U. Milutinovic. Simple explicit formulas for the Frame-Stewart numbers. Annals of Combinatorics, Volume 6 (2002), pp 157-167. Online at www.birkhauser.ch/journals/2600/papers/2006002/20060157.pdf.

S. Klavzar. Square-edge graphs, partial cubes and their subclasses (1998). Online at citeseer.nj.nec.com/klavzar00squareedge.html.

Paul Stockmeyer. Variations on the Four-Post Tower of Hanoi Puzzle. In Congressus Numerantium, Volume 102 (1994), pp 3 - 12. Online at www.cs.wm.edu/~pkstoc/h_papers.html.

Mario Szegedy. In How Many Steps the k Peg Version of the Towers of Hanoi Game Can Be Solved? In STACS '99, LNCS 1563, pp 356-361, 1999.

# Discs	# Moves	Time (secs)	Mem (Mb)
1	1	0.02	0.02
2	3	0.02	0.02
3	5	0.02	0.02
4	9	0.02	0.02
5	13	0.03	0.02
6	17	0.03	0.02
7	25	0.03	0.02
8	33	0.05	0.02
9	41	0.07	0.03
10	49	0.10	0.05
11	65	0.32	0.13
12	81	1.09	0.37
13	97	3.10	0.76
14	113	9.28	1.68
15	129	24.91	5.54
16	161	135.28	19.19
17	193	563.89	66.17
18	225	1888.41	201.21
19	257	-	-
20	289	-	-

Table 1: 4-peg Tower of Hanoi Results

# Discs	# Moves	Time (secs)	Mem (Mb)
1	1	0.64	0.17
2	3	0.65	0.17
3	5	0.71	0.17
4	7	0.71	0.17
5	11	0.73	0.17
6	15	0.75	0.17
7	19	0.76	0.17
8	23	0.81	0.19
9	27	0.89	0.22
10	31	1.03	0.34
11	39	1.97	1.10
12	43	3.51	2.26
13	47	7.00	4.88
14	53	19.88	16.06

Table 2: 5-peg Tower of Hanoi Results

# Discs	# Moves	Time (secs)	Mem (Mb)
1	1	0.62	0.17
2	3	0.67	0.17
3	5	0.72	0.17
4	7	0.74	0.17
5	9	0.77	0.17
6	13	0.78	0.17
7	17	0.83	0.17
8	21	0.88	0.20
9	25	1.01	0.34
10	29	1.44	0.74
11	33	2.69	2.09
12	37	6.99	6.64

Table 3: 6-peg Tower of Hanoi Results

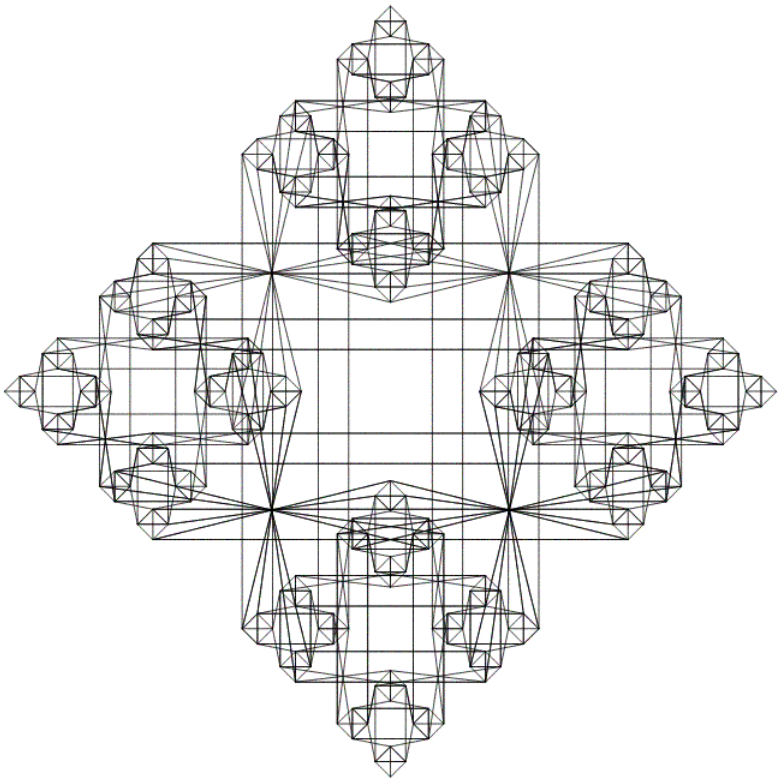

Figure 1: State Transition graph - 4 pegs, 4 discs