

OR/MS Games: 2. Towers of Hanoi

Moshe Sniedovich
Department of Mathematics and Statistics
The University of Melbourne
Parkville, VIC 3052, Australia

m.sniedovich@ms.unimelb.edu.au

tery close to the rods. The monks' task in life is to transfer all the rings onto another rod. The only operation permitted consists of moving a single ring from one rod to another, in such a way that no ring is ever placed on top of another smaller one. When the monks have finished their task, according to the legend, the world will come to an end.

Brassard and Bratley (1988, p. 64)

Abstract

In this discussion we examine the famous Towers of Hanoi puzzle from an OR/MS perspective, focusing on its educational content. We show that this puzzle provides an excellent environment for illustrating a number of fundamental OR/MS problem-solving concepts in general and dynamic programming concepts in particular. In addition to the popular 'min' version of the problem, we also present an interesting but rather neglected 'max' version. On-line interactive modules are included.

This popular description suggests that the puzzle is of ancient Eastern origin. However, it seems that it was invented only in 1883 by the French mathematician Edouard Lucas.

Some toy shops sell inexpensive (plastic, metal, or wooden) implementations of the original diamond-gold version, typically with a small number of rings (much smaller than the original 64). Here is picture of a size 5 version of the puzzle.



Figure 1

Introduction

The Towers of Hanoi is a classical puzzle. It is regarded as a must in computer science subjects dealing with recursions. It is normally cast as a legend concerning - among other things - the end of the world. The good news is, however, that due to the explosive nature of 2^n we have a long way to go. From an OR/MS point of view the puzzle is a combinatorial optimization problem and it will be therefore treated as such in this discussion.

Here is a typical wordy description of the problem:

It is said that after creating the world God set on Earth three rods made of diamond and 64 rings of gold. At the creation they were threaded on one of the rods in order of size, the largest at the bottom and the smallest at the top. God also created a monas-

In what follow we present an OR/MS anatomy of the puzzle, including a detailed derivation of the functional equation of dynamic programming for the optimal policy. It is then shown that this popular recursive formulation can be simplified into a non-recursive incremental policy with minimal (constant) memory requirements. We also consider the (neglected) case where the objective is to maximize the number of (acyclic) moves.

The main objective is to show that the puzzle is very rich educationally and that it offers excellent material for OR/MS subjects.

Math Formulation

The wordy description of the problem is somewhat ambiguous with regard to a very important aspect of the problem: are the monks trying to prolong their task, or are they trying to complete it as quickly as possible? However, the literature is dominated by the 'min version' of the problem: the monks are trying to complete the task as quickly as possible!

Therefore, we shall first consider the popular 'min version' and then the 'max version' of the problem.

Also, we do not have the resources to play this game with diamond rods and gold rings. We shall replace the rods with wooden platforms and the rings with 3D trapezoidal plastic pieces. For ease of identification, the pieces will be labelled 1,2,...,n. Let P denote the set of all pieces, namely set $P=\{1,2,3,...,n\}$. We shall refer to the three platforms as Left, Right and Center, and assume (with no loss of generality) that initially all the pieces are on Left and they are to be moved to Right. And just for good measure we shall regard the number of pieces as a parameter, call it n. This is the picture then for a game of size n=5:

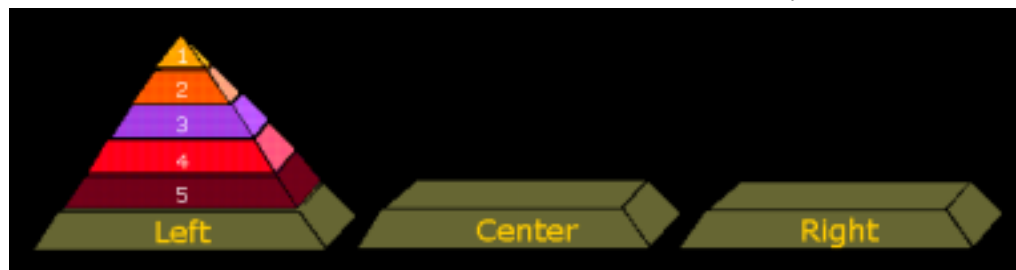


Figure 2

Two things are of interest to us, namely:

- what is the minimum number of moves that will be required to solve a problem of size n?
- what is the optimal sequence of moves for a problem of size n?

It turns out that the answers to these two related questions can be obtained in a straight forward manner using standard dynamic programming techniques (see Sniedovich (1992)). However, before we do this let us consider the following issue.

What is a proper Math translation of the wordy description of the problem given above? As indicated in Sniedovich (2001) , games of this nature can be viewed as sequential decision problems and be stated mathematically as follows:

Problem Q:

$$z^* := \min_{x_1, \dots, x_k} k \quad (1)$$

$$\text{s.t.} \quad s_{k+1} = s \quad (2)$$

$$s_{j+1} = T(s_j, x_j), j=1,2,\dots,k \quad (3)$$

$$x_j \in D(s_j), j=1,2,\dots,k \quad (4)$$

where

x_j = decision made at stage j of the process.

s_j = state of the process at stage j. (s_1 is given)

s = target state of the process (given).

$D(s_j)$ = set of feasible decisions when the process is in state s_j .

T = Transition function. That is, $s_{j+1}=T(s_j, x_j)$ is the next state of the process given that the current state is s_j and the current decision is x_j .

In words, the objective is to minimize the number of transitions required to move from the initial state of the process (s_1) to the specified target state (s) subject to the transition function (T) and the feasibility conditions (imposed by D). Note that in this context the initial state of the process is given. By definition then, z^* is the optimal value of k , namely the minimum number of transitions required to solve the puzzle.

In the case of the Towers of Hanoi problem the state variable can be defined as a triplet $s=(L,C,R)$ where: L = set of pieces on Left, C = set of pieces on Center and R = set of pieces on Right. If we let E denote the empty set then the initial state is $s=(P,E,E)$ and the target state is $s=(E,E,P)$, recalling that P represents the set of all the pieces, namely $P=\{1,2,\dots,n\}$.

Next, let us define the decision variable. Since the state variable provides full information on how the pieces are distributed among the platforms, and since the pieces are arranged in decreasing size on each platform, a move (decision) can be described as a pair $x=(a,b)$ where a represents the platform from which the piece is taken and b represents the platform to where the piece is moved. Thus, $x=(a,b)$ represents the decision "move a piece from platform a to platform b ". For example, $x=(\text{Right},\text{Left})$ represents the decision "move a piece from Right to Left".

In view of the restrictions on the manner in which pieces are arranged on the platforms, it follows that $x=(a,b)$ is feasible if and only if either the smallest piece on platform a is smaller than the smallest piece on platform b , or there are no pieces on platform b . Thus, $D(s)$ is the set of all feasible pairs $x=(a,b)$ associated with state $s=(L,C,R)$. For example, consider the case where $P=\{1,2,3,4,5\}$, $L=\{5\}$, $C=\{1,2,3,4\}$, $R=E$. Then for this case $D(s)=\{(L,R),(C,R),(C,L)\}$. The picture is then this:

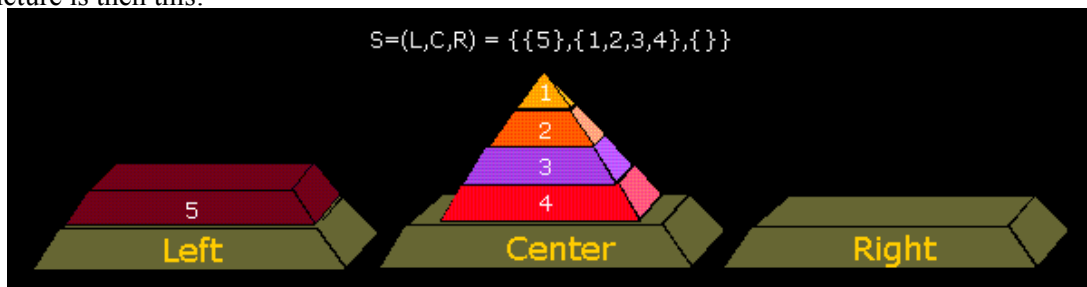


Figure 3

In view of these definitions, the transition function T operates as follows: $T(s,x)$ is the new arrangement of pieces on the platforms given that the current arrangement is $s=(L,C,R)$ and the decision $x=(a,b)$ has been implemented (a piece was moved from platform a to platform b). For example, for $P=\{1,2,3,4,5\}$, $L=\{5\}$, $C=\{1,2,3,4\}$, $R=E$ we have

$$T(s,(L,R)) = (E,C,\{5\})$$

$$T(s,(C,R)) = (L,\{2,3,4\},\{1\})$$

$$T(s,(C,L)) = (\{1,5\},\{2,3,4\},E).$$

We are interested in two things:

- the optimal sequence of decisions associated with this problem
- the length of this sequence

How then do we determine these two related things? We shall use dynamic programming (DP) for this purpose.

Decomposition

In the usual DP manner we decompose the original problem into related subproblems so that the solutions to the subproblems could be used to construct the solution to the original problem. In the context of the sequential decision problem (1)-(4), the subproblems are represented by the states of the process. The question is then: in the context of the Towers of Hanoi problem, what states should be used in the construction of an optimal solution to the original problem?

The answer is this: In order to move the largest piece from Left to Right it is necessary to move the other n-1 pieces from Left to Center. Given this, we can

then move the largest piece to Right and to complete the job we have to move the n-1 pieces from Center to Right. The following pictures illustrate this observation in the case where n=5.

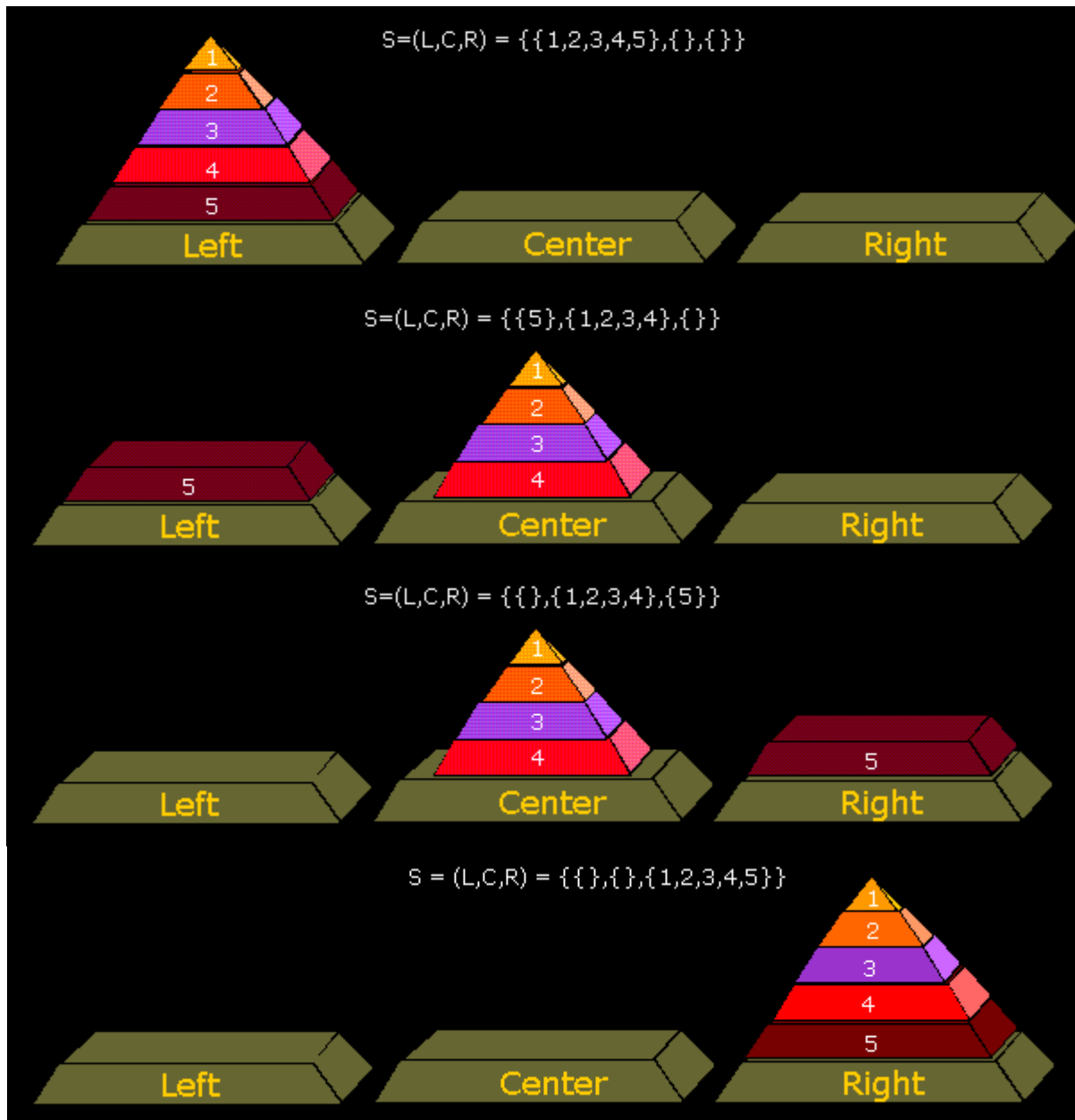


Figure 4

Formally, to define the subproblems we simply let the initial and target states in (1)-(4) be parameters. The result is this:

Problem Q(s, σ):

$$f(s, \sigma) := \min_{x_1, \dots, x_k} k \quad (5)$$

$$\text{s.t.} \quad s_1 = s \quad (6)$$

$$s_k + 1 = \sigma \quad (7)$$

$$s_j + 1 = T(s_j, x_j), j=1, 2, \dots, k \quad (8)$$

$$x_j \text{ in } D(s_j), j=1, 2, \dots, k \quad (9)$$

where s and σ are given states.

Note that by construction, Problem Q is equivalent to Problem Q(s^*, σ^*) where s^* is the initial state of the original problem, namely $\sigma^* = (P, E, E)$ and s^* is the target state of the original problem, namely $s^* = (E, E, P)$. So in view of the discussion above it follows that the optimal solution to Problem P will pass through state $s' = \{\{n\}, \{1, 2, \dots, n-1\}, E\}$ and then through state $s'' = (E, \{1, 2, \dots, n-1\}, \{n\})$ on the way to the target state $\sigma^* = (E, E, P)$.

This conclusion can be stated formally as follows:

$$f(s^*, \sigma^*) = M(s^*, s') + 1 + M(s'', \sigma^*) \quad (10)$$

where $M(a, b)$ is the number of moves required by the optimal solution to the original problem to move from state a to state b .

The following question then arises: how do we determine the values of $M(s^*, s')$ and $M(s'', \sigma^*)$?

Principle of Optimality

Since our objective is to minimize the total number of moves, and since the tasks represented by $M(s^*, s')$ and $M(s'', \sigma^*)$ are independent of each other (there are no loops), it follows that $M(s^*, s')$ is equal to the minimum number of transitions required to move from state s^* to state s' . Similarly $M(s'', \sigma^*)$ is equal to the minimum number of transitions required to move from s'' to σ^* . So in view of the definition of Problem P(s, σ) it follows that

Theorem 1

$$M(s^*, s') = f(s^*, s'), \quad M(s'', \sigma^*) = f(s'', \sigma^*). \quad (11)$$

This property of the optimal solution of the original problem is a common manifestation of Bellman's Principle of Optimality. In this context it implies that any subpath of the optimal solution is optimal with respect to its end-points, that is its own initial and final states (Sniedovich [1992]).

Functional Equation

The following is an immediate consequence of (10)-(11)

Theorem 2

$$f(s^*, \sigma^*) = f(s^*, s') + 1 + f(s'', \sigma^*) \quad (12)$$

or more specifically

$$\begin{aligned} f(\{1, \dots, n\}, \{\}, \{\}) &= f(\{1, 2, \dots, n\}, \{\}, \{\}) + 1 + f(\{n\}, \{1, \dots, n-1\}, \{\}) \\ &= f(\{1, 2, \dots, n\}, \{\}, \{\}) + 1 + f(\{n\}, \{1, \dots, n-1\}, \{\}) \end{aligned} \quad (13)$$

This is the DP functional equation for the minimum number of transitions required to complete the puzzle. We can write down a similar functional equation for the optimal decisions but it is more convenient to do this in the context of the much more attractive formulation that we consider in the next section.

Similarities

A close examination of the DP model used in the preceding sections reveals that many of the subproblems are equivalent in the sense that their $f(s, \sigma)$ values are identical and their optimal decisions are similar. In particular, the minimum number of transitions required to move m pieces from platform a to platform b , assuming that all the pieces on the other platform are greater than all those placed on these two platforms, depends only on m .

Let then $F(m)$ denote the minimum number of transitions required to move m pieces from one platform to another given that all the remaining pieces are larger than those being transferred. Then it would follow from (12)-(13) that

Theorem 3

$$F(m) = F(m) + 1 + F(m) = 2F(m) + 1 \quad (14)$$

observing that by definition $F(1) = 1$.

It is not difficult to show (by induction on m) that this implies that

Corollary 1

$$F(m) = 2^m - 1, \quad m=1,2,\dots \quad (15)$$

This result is consistent with the output of Module 1 provided below: for $n=5$ we have $F(5) = 2^5 - 1 = 31$.

Similarly, as far as the optimal decisions are concerned, what really matters is how many pieces are

being moved from where to where. Thus, let $X(m,a,b)$ be the optimal sequence of decisions for moving the top m pieces from platform a to platform b , assuming that these pieces are all smaller than all the other pieces placed on the three platforms. Then clearly,

Corollary 2

$$X(m,a,b) = X(m-1,a,c) ; X(1,a,b) ; X(m-1,c,b) \quad (16)$$

where c denote the platform which is neither a nor b and x, y denotes the sequence obtained by concatenating sequence x and sequence y , eg for $x=(1,2,4)$ and $y=(6,5,3)$ we have $x, y = (1,2,4,6,5,3)$.

In words, the optimal sequence of decisions for moving m pieces from platform a to platform b is the concatenation of three sequences:

- optimal sequence of decisions for moving the top $n-1$ pieces from platform a to platform c
- moving the piece that was left on platform a to platform b
- moving the $m-1$ pieces that were placed temporarily on platform c to platform b .

This fact can be deduced directly from the discussion in Decomposition.

Since $X(1,a,b)$ is trivial, namely it is simply "move one piece from platform a to platform b ", Corollary 2 can be easily implemented (recursively) for small values of m . For example, for $m=3$ we have

$$X(3,a,b) = X(2,a,c) ; X(1,a,b) ; X(2,c,b) \quad (17)$$

Applying Corollary 2 to the terms involving $m=2$ we obtain

$$\begin{aligned} X(2,a,c) &= X(1,a,b) ; X(1,a,c) ; X(1,b,c) ; X(2,c,b) \\ &= X(1,c,a) ; X(1,c,b) ; X(1,a,b) \end{aligned} \quad (18)$$

Hence

$$\begin{aligned} X(3,a,b) = & X(1,a,b) ; X(1,a,c) ; X(1,b,c) \\ & ; X(1,a,b) ; X(1,c,a) ; X(1,c,b) ; \\ & X(1,a,b) ; \end{aligned} \quad (19)$$

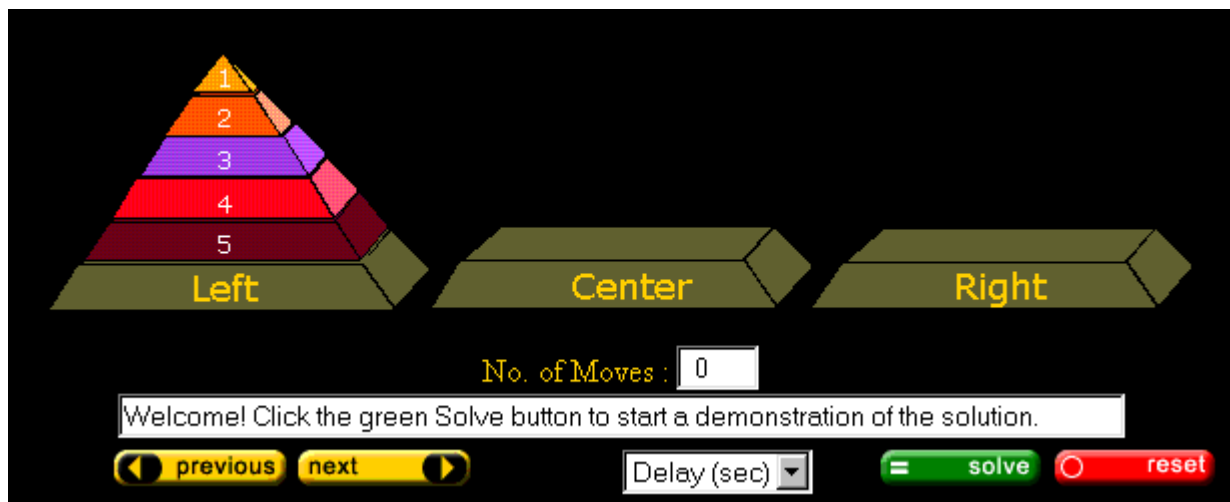
The English translation is as follows:

Table 1

Move a piece		
Transition No.	To	From
1	Left	Right
2	Left	Center
3	Right	Center
4	Left	Right
5	Center	Left
6	Center	Right
7	Left	Right

For the benefit of readers who are not familiar with this puzzle we provide below an eToy version of the game based on (19) so that they can observe the behaviour of the optimal solution.

Since $X(m,a,c)$ is easily obtained from $X(m,e,d)$ by a simple substitution, it is only necessary to compute $X(n,?,?)$ for one pair of platforms. The optimal solutions for all other combinations of platforms for the given value of m can be easily derived from this pair by simple substitution. For example, observe that $X(2,c,b)$ can be obtained from $X(2,a,c)$ by the following substitution: $a \rightarrow c$; $b \rightarrow a$; $c \rightarrow b$. This means that the functional equation can be solved iteratively (bottom-up) rather than recursively (top-down). That is, such a procedure will start by computing say $X(1,L,C)$ and then $X(2,L,C)$ and then $X(3,L,C)$ and so on up to $X(n-1,L,C)$. Whenever $X(m,a,b)$ is needed for other combinations of platforms, the respective $X(m,L,R)$ sequences will be transformed accordingly to yield the desired sequence.



Module 1

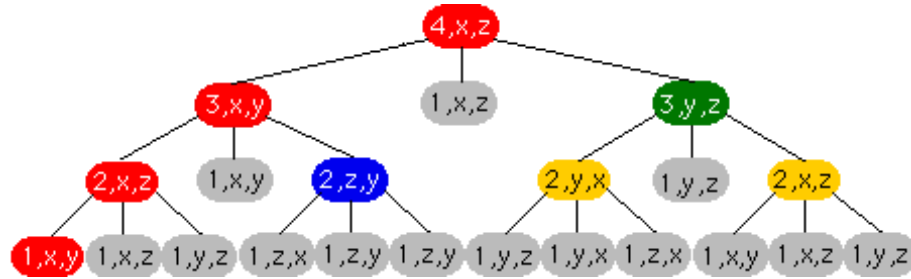


Figure 5

It should be pointed, however, that while this could be an improvement on the strictly recursive procedure, the main difficulty still remains: to solve a problem of size $m+1$ it is necessary to store an optimal sequence of decisions for a problem of size m , and this sequence is of length 2^m . Even for moderate values of m this will be prohibitively demanding as far as memory is concerned. In particular, in relation to the original version of the problem,

$$2^{63} = 9223372036854776000$$

is a very large number. Indeed, your current computer does not have enough memory to cope with this requirement and it is doubtful that you will ever own a computer capable of storing the sequence of optimal solutions for a problem of size $n=64$.

Pattern Recognition

It would be nice if we could develop an iterative **incremental** solution procedure for this problem such that to determine the next optimal decision it is not necessary to know the entire sequence of previous decisions. Rather, it will be sufficient to know, say just the content of each platform and which piece was moved last.

The following observations regarding the structure of the optimal policy are derived directly from the fact that according to the rules of the game the smallest piece is always the top piece on one of the platforms and that there are only three platforms:

- **Lemma 1:** The next piece to be moved is the smallest eligible piece, where "an eligible piece" means a piece satisfying the following conditions: (i) it is not the piece that was moved in the last transition (ii) it is the smallest piece on its platform. Hence, there are at most two eligible pieces.
- **Lemma 2:** The rule implied by Lemma 1 means that if the smallest eligible piece is not the smallest piece (the piece labelled 1) then there is no choice in the destination of the piece: the piece must be moved to the third platform (the one that does not contain the smallest piece). For example, if the top pieces on the three platforms are 1, 5 and 4 and piece 1 was moved in the last transition, then we now have to move piece 4 on top of piece 5.

Lemma 2 implies that there is a simple iterative rule for moving the next piece if this piece is not the smallest piece. Thus, all that needs to be determined is how to move the smallest piece.

It turns out that this is not difficult to do. In fact, the rule for this can be observed, by inspection, from the decision tree of the recursive solution, or even from the recursive formula of the optimal rule, namely from (16), recalling that $X(m,a,b)$ denotes the optimal solution to a problem involving m pieces which are to be moved from platform a to platform b . This rule can be summarized as follows:

• **Lemma 3:**

The first move generated by $X(m,a,b)$ is "move a piece from platform a to platform b" if m is odd, and it is "move a piece from platform a to platform c" if m is even. For example, the first move of $X(3001, \text{Left}, \text{Right})$ is "move a piece from Left to Right", and the first move of $X(60456, \text{Left}, \text{Right})$ is "move a piece from Left to Center".

• **Lemma 4:**

The moves generated by $X(m,a,b)$ have the following properties:

- The odd pieces move in accordance with the pattern (a,b,c,a,b,c,a,...) if m is odd; if m is even they follow the pattern (a,c,b,a,c,b,a,...).
- The even pieces follow the opposite direction: they move in accordance with the pattern (a,c,b,a,c,b,a,...) when m is odd and the pattern (a,b,c,a,b,c,a,...) when m is even. For example, $X(3097, \text{L}, \text{R})$ generates moves such that the odd pieces follow the pattern (L,R,C,L,R,C,L,...) and the even pieces follow the pattern (L,C,R,L,C,R,L,...).

Note that there are two basic patterns: one for cases where m and n have the same parity and one for cases where m and n have different parity. Thus it is useful to introduce the following:

• **Definition:** Patterns.

The Match Pattern for moving pieces from platform a to platform b is (a,b,c,a,b,c,...) and the MisMatch pattern is (a,c,b,a,c,b,...).

So here is Lemma 4 expressed in terms of the two basic patterns.

• **Lemma 5:**

The moves generated by $X(m,a,b)$ have the following property: Each piece moves (throughout the game) either in accordance with the Match Pattern or in accordance with the MisMatch Pattern, depending on whether the piece has the same parity as m.

The optimal policy has many other interesting properties. For example:

- **Lemma 6:** In a game of size n the optimal policy moves the m-th smallest piece for the first time in transition number 2^{m-1} and altogether this piece is moved exactly $2^{(n-m)}$ times. In particular, the smallest piece is moved exactly $2^{(n-1)}$ times, that is every second transition moves the smallest piece.

Thus, the total number of moves generated by the optimal policy for a game of size n is equal to

$$2^{n-1} + 2^{n-2} + \dots + 2^0 = 2^n - 1 \quad (20)$$

which is in line with Corollary 1.

Recall that it is not necessary to have an explicit rule to determine how pieces other than the smallest piece should be moved. If you have to move a piece other than the smallest one you basically do not have a choice. That is, immediately after you move the smallest piece, you must move the second smallest top piece and you have no choice as to where it should be moved: it must be moved to the platform whose top piece is the largest.

For example, suppose that you are in the middle of the game, trying to move 5 pieces from Left to Right and you observe the following arrangement:

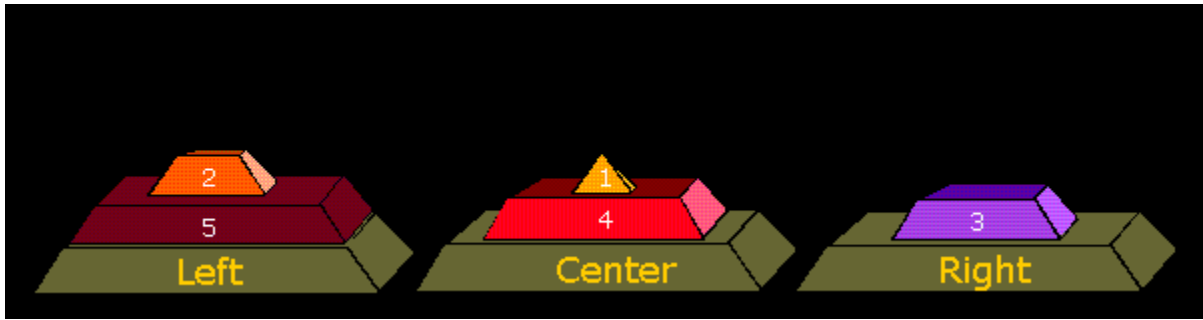


Figure 6

What should be your next move?

There are two possibilities, depending on which piece was moved last: if the smallest piece was moved last, then you have no choice: the only thing you can do is move the 2nd smallest top piece (piece number 2) to Right. If the smallest piece is eligible, then (according to the Match Pattern) you move it to Left.

Thus, as far as on-line games are concerned, this recipe can deal with puzzles of any size. The limiting factor is the player's time!

In view of the role that the smallest piece plays in this analysis, the reader may wish to observe on-line the path of this piece using Module 1.

So here is a simple recipe for the Towers of Hanoi puzzle of size n where the pieces are to be moved from platform a to platform b and c denotes the third platform.

The memory requirements of this recipe are minimal: only four very small pieces of information are needed:

- Three static details:

The pattern to be used (Match or MisMatch), the 'origin' platform and the 'target' platform.

- Dynamic details:

Either the iteration number or the content of the three platforms (this is uniquely determined by the content of any two platform).

The dynamic details, in particular the information about the content of the three platforms, is consistent with the structure of the state variable of the dynamic programming model (5)-(9).

Towers of Hanoi Recipe
How to move n pieces from platform a to platform b
(using the spare platform c)

If n is odd set Pattern = $(a, b, c, a, b, c, \dots)$, otherwise set Pattern = $(a, c, b, a, c, b, \dots)$.
For $k=1, 2, \dots, 2^n-1$ Do:
 If k is odd, move the smallest piece according to Pattern.
 If k is even, move the second smallest top piece.
End Do

Towers of Hanoi Module

Problem Size <input type="text" value="5"/>	Origin <input type="text" value="Left"/>	Destination <input type="text" value="Right"/>	n = <input type="text" value="5"/> 2 ⁿ = <input type="text" value="32"/>	<input type="button" value="Reset"/>
--	---	---	--	--------------------------------------

☒ Terse ☐ Full







Range

Status:

Idle

G'day : Click the 'report', 'iterate',
'next' and 'previous' buttons to experiment with the
module.
Specific instructions are provided below .

Instructions

Problem Size	Use a small (<10) size if you want to use the 'report' button to generate a report of the entire sequence of optimal decisions (there are 2^n-1 such decisions). You can use larger size values to generate individual decisions via the 'iterate', 'next' and 'previous' buttons.
>9	Use this option in the Problem Size menu if you want to use the 'report' button for problems of size greater than 9. Once you use the 'report' button with this option you will be allowed to try to generate reports for problems of any size listed in the Problem Size menu (and in all likelihood run out of memory!).
	Use this button to terminate the operation triggered by the 'report' button if it takes too long.
 <input checked="" type="radio"/> Terse <input type="radio"/> Full	Generates the optimal sequence of moves for the problem. Use only for small problems (<9), otherwise your browser will run out of memory, especially if you select the 'Full' option.
	Generates the optimal decision for the iteration specified by the # field. Can handle large problems (say of size 1-40) depending on your browser's ability to deal with very large integers.
#	Use this field to specify the iteration number you are interested in. You can use the 'cut and paste' tool of your browser to fill in this field.
	Displays the record of all the iterations generated by the 'next' and 'previous' buttons since the last click on the 'iterate' button.
Range	The range of feasible values of the entry specified in the # field, namely $\{1, 2, \dots, 2^n-1\}$, where n is the problem's size.
	Generates the next optimal decision relative to the iteration specified by the # field.
	Generates the previous optimal decision relative to the iteration specified by the # field.

Module2

More details on this game and its history can be found in Roth (1974) and Hayes (1977). Needless to say, the game is irresistible for animators and this is reflected in the many interactive modules for this game available on the web (see Yahoo's directory of Math Problems, Puzzles and Games). Such modules also feature in the OR/MS oriented tutOR and tutORial web sites and IFORS On-line Encyclopedia (IOE).

Multipeg Problems

One natural generalization of the conventional Towers of Hanoi game is the Reve's Puzzle (see Rohl (1986), Lu (1989)) introduced by England's greatest puzzlist H.E. Dudeney in 1907. In this puzzle there are four stools rather than three pegs and the pieces are not golden rings but rather blocks of cheese!

Needless to say, there are also so called "multipeg" Towers of Hanoi games, namely games where the number of pegs can be larger than 4 (see Boardman et al(1986) and Lu (1990)).

OR/MS Perspective

Students are often puzzled that the DP functional equations (14) and (16) do not include the beloved 'min' operator. After all, aren't we supposed to minimize the number of moves? If so, how come that this is not reflected explicitly in the DP functional equation?

For this reason it is pedagogically useful to consider the following very natural (but very neglected!) version of the Tower of Hanoi problem: What happens if we try to maximize rather than minimize the number of moves? After all, to delay the end of the world, the monks might have decided to adopt the 'max' rather than the 'min' policy!

In any case, it turns out that this version of the Tower of Hanoi problem is more challenging than the conventional one. More importantly, this unorthodox version of the problem is instrumental in explaining the DP formulation of the conventional problem and provides a distinct OR/MS flavour to the problem as a whole.

Of course, this version of the problem is of no interest if cycles are allowed, for in this case the number of moves is unbounded. The challenge is then to find an acyclic optimal policy. In the sequel we shall refer to this version of the problem as the 'max version' and assume that no cycles are allowed.

We now show that the optimal solution to the 'max version' consists of $3^n - 1$ moves and present a $O(3^n)$ time and constant memory algorithm for generating this solution. But before we do this, let us investigate where the missing 'min' is hiding in the DP functional equations for the conventional ('min') version of the problem, namely in (14) and (16).

Unlike most DP functional equations that appear in the OR/MS literature for optimization problems, here there are no explicit decision variables, hence nothing to minimize explicitly. The minimization is nevertheless very much behind the scene. More specifically, it plays a major role in two places:

- Definitions of $F(1)$ and $X(1, \dots)$.

The reason why we assume in (14) that $F(1)=1$ is that the optimal ('min') solution for a problem

of size $m=1$ is to move the piece from the origin directly to the destination. Hence, it is clear that $X(1, a, b) = \text{"move a piece from a to b"}$.

- Elimination of obviously non-optimal decisions.

We implicitly ignore - and rightly so - the possibility of using policies according to which the first time the largest piece is moved, it is moved to the spare platform.

In short, the 'min' is hiding in the logic used in Figure 4 to justify the structure of $F(m)$ and $X(m, a, b)$ given in (14) and (16), respectively: obviously-non-optimal policies were eliminated on the fly implicitly. As we shall see shortly, in the context of the 'max version' of the problem, things are not so obvious and therefore this issue must be discussed explicitly.

With this in mind let $Y(m, a, b)$ denote the policy defined as follows:

$$Y(m, a, b) := Y(m-1, a, b) ; y(1, a, c) ; Y(m-1, b, a) ; y(1, c, b) ; Y(m-1, a, b) , m=2, 3, \dots \quad (21)$$

where

$$Y(1, a, b) := y(1, a, c) ; y(1, c, b) \quad (22)$$

$$y(1, a, b) := \text{"move the top piece from platform a to platform b"} \quad (23)$$

In words, $Y(m, a, b)$ operates as follows: it

- moves the top $m-1$ pieces from a to b ; then
- moves the remaining piece from a to c ; then
- moves the $m-1$ pieces from b to a ; then
- moves the piece from c to b ; then
- moves the $m-1$ pieces from a to b .

Pictorially it looks like this:



Figure 7

Let $G(m)$ denote the total number of (single piece) moves generated by $Y(m, a, b)$. Then by construction $G(1)=2$ and

$$G(m) = G(m-1) + 1 + G(m-1) + 1 + G(m-1) = 3G(m-1) + 2, m > 1 \quad (25)$$

Hence,

Theorem 4

$$G(m) = 3^n - 1, \quad n=1,2,3,\dots \quad (26)$$

Here is a sample of $G(m)$ values and the respective $F(m)$ values:

m	$F(m) = 2^m - 1$	$G(m) = 3^m - 1$
1	1	2
2	3	8
3	7	26
4	15	80
5	31	242
6	63	728
7	127	2,186
8	255	6,560
9	511	19,683
10	1,023	59,048
20	1,048,575	3,486,784,400
<input type="text" value="30"/>	<input type="text" value="1 073741824"/>	<input type="text" value="205891132094649"/>
Fill in the value of m and then click the $F(m)$ or $G(m)$ field.		

Table 2

To show that $Y(m,a,b)$ is indeed an optimal policy for the 'max version' consider the following question: when the largest piece (m) is moved for the first time, where should it moved to? Should it be moved to platform c or to platform b? Under Y it is moved to c, so to show that Y is optimal it is necessary to show that Y is at least as good as any policy Z according to which the first time the largest piece (m) is moved in line with $Z(m,a,b)$ it is moved to platform b.

Observe then that because there are three platforms, it is impossible to move the largest piece more than twice without creating a cycle. This means that the optimal policy should move the largest piece first to the spare platform (c).

The following table depicts the states generate by an attempt to move the largest piece to the destination (b) in its first move. It shows that a deadlock occurs after 7 transitions.

Iteration No.	Origin (a)	Spare (c)	Destination (b)	Explanation
0	{1,...,m}	{}	{}	Initial state
1	{m}	{1,...,m-1}	{}	Intermediate state required to reach the assumed state.
2	{}	{1,...,m-1}	{m}	Assumed state.
3	{1,...,m-1}	{}	{m}	We have to go this way to avoid a clash with the state observed at Iteration 1.
4	{1,...,m-1}	{m}	{}	We have to go this way to avoid a clash with the previous state.
5	{}	{m}	{1,...,m-1}	We have to go this way to avoid a clash with the previous state.
6	{m}	{}	{1,...,m-1}	We have to go this way to avoid a clash with the previous state.
7	{m}	{1,...,m-1}	{}	We have to go this way to avoid a clash with the previous state.

Table 3

Hence

Theorem 5

Policy Y defined by (21) generates the maximum number of feasible moves (without cycles) for the 'max version' of the problem. Therefore, $G(m) := 3^m - 1$ is the maximum number of feasible moves (without cycles) for the 'max version' of a problem of size m.

Proof. In view of the above, any optimal policy must first move the largest piece to the spare platform and then to the destination. In preparation for each such move the remaining m-1 pieces are moved to the appropriate platform. Each such move is independent of the other moves, hence such moves are uniquely defined by the (relative) origin and destination platforms and the number of pieces moved (m-1). This is precisely how Y is defined.

To confirm this result observe that there are 3^n ways in which n distinct pieces can be placed on 3 platforms (ignoring the non-cycling constraint). Hence, $3^n - 1$ is an obvious upper bound on the number of feasible moves in the 'max version' of the problem. Since the policy Y defined above achieves this number, it must be an optimal policy. Observe that the implication of this is that the non-cycling constraint

is not an obstacle for generating **all** the 3^n possible configurations.

It can be shown that policy Y has the following characteristics:

Theorem 6

- In accordance with Y(m,a,b), the smallest piece follows the pattern (a,c,b,c,a,c,b,c,...).
- In accordance with Y(m,a,b), the k-th smallest piece ($k \leq m$) moves exactly $2 \times 3^{m-k}$ times. In particular, the smallest piece moves exactly $2 \times 3^{m-1}$ times and the largest piece (m) moves exactly twice.

Note that according to this, $G(m)$ should be equal to

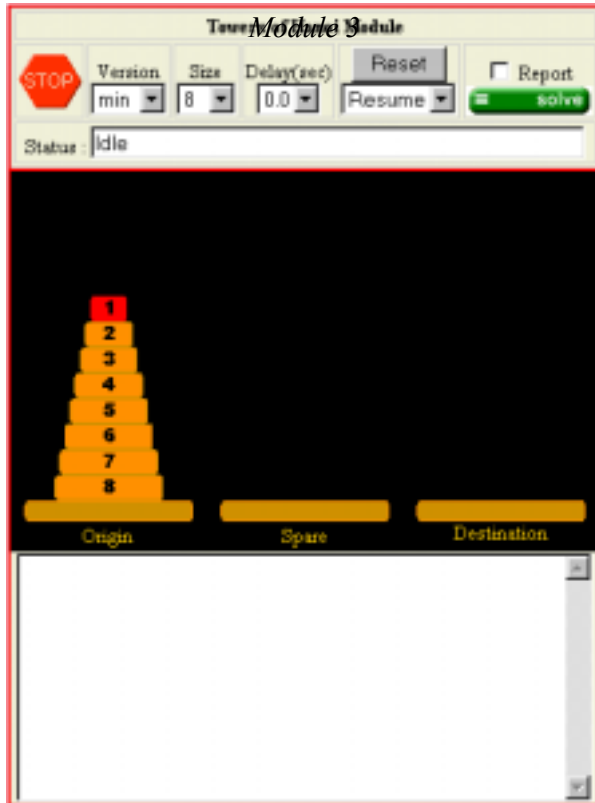
$$2 \times 3^{m-1} + 2 \times 3^{m-2} + \dots + 2 \times 3^{m-m} = 3^n - 1$$

which is consistent with Theorem 4.

Because the pattern used by the smallest piece under Y is so simple, it is straight forward to generate the sequence of decisions induced by Y in an iterative non-recursive manner with a constant (very small) memory requirement. The time requirement is $3^m - 1$ simply because there are that many moves. This means that - with plenty of patience on your

part - it is possible to solve large 'max version' instances of the problem.

You are encouraged to use the following interactive module to experiment with the two versions of the problem. For obvious reasons, reports are not provided for large problems. Adjust the "Delay" parameter to your computer's speed so that you can clearly see the patterns discussed above.



In short, the optimal solution to the 'max version' of the problem is capable of generating all the $3n$ possible configurations of pieces on the platforms and this is achieved by a simple pattern of moves. Although here - as in the case of the 'min version' of the problem - the DP functional equation does not have an explicit 'opt' operator, there is an obvious need to address the question of optimality explicitly.

10. Epilogue

The Towers of Hanoi game contains educationally rich OR/MS material. It has been extensively used for many years in introductory computer science textbooks. OR/MS lecturers engaged in teaching subjects dealing with problems modelling/solving tools should consider the inclusion of this game in their courseware.

This discussion is part of the author's effort Sniedovich (2002) to encourage OR/MS lecturers to incorporate OR/MS games in their lectures.

References

Boardman, J.T., Garrett, C., and Robson, G.C.A. (1986), "A Recursive Algorithm for the Optimal Solution of a Complex Allocation Problem using a Dynamic Programming Formulation," *The Computer Journal*, Vol. 29, No. 2, 182-186.

Brassard, G. and Bratley, P. (1988), *Algorithmics: Theory and Practice*, Prentice-Hall, Englewood-Cliff, NJ, USA.

Hayes, P.J. (1977), "A note on the Towers of Hanoi Problem," *The Computer Journal*, Vol 20, No. 3, 282-285.

IFORS On-line Encyclopedia: www.ifors.org/ioe/

Lu, X.M. (1989), "An Iterative Solution for the 4-peg Towers of Hanoi," *The Computer Journal*, Vol. 32, No. 2, 187-189.

Lu, X.M. (1990), "A loopless approach to the multipeg Towers of Hanoi," *International Journal of Computer Mathematics*, Vol. 33, No. 1/2, 13-29.

Rohl, J.S. and Gedeon, T.D. (1986), "The Reve's Puzzle," *The Computer Journal*, Vol. 29, No. 2, 187-188.

Roth, T. (1974), "The Tower of Brahma Revisited," *Journal of Recreational Mathematics*, Vol. 7, No. 2, 116-119.

Sniedovich, M. (1992), *Dynamic Programming*, Marcel Dekker, NY.

Sniedovich, M. (2001), "The Role of Games in OR/MS Education," submitted to *INFORMS Transactions on Education*.

tutOR Project: www.tutor.ms.unimelb.edu.au

tutORial Project: www.ifors.org/tutorial/

Yahoo's directory of Math Problems, Puzzles and Games: dir.yahoo.com/Science/Mathematics/Problems__Puzzles__and__Games/

eg.

www.mazeworks.com/hanoi/

www.lhs.berkeley.edu/java/tower/tower.html

hanoitower.mkolar.org/

javaboutique.internet.com/Tower/