The Panex Puzzle

Mark Manasse, Danny Sleator, Victor K. Wei, Nick Baxter

Introduction

The Panex puzzle is a one-person board game created by Toshio Akanuma and manufactured by TRICKS Co., Ltd. of Tokyo, Japan. On first sight, the puzzle reminds one of the Tower of Hanoi. A little thought reveals that they are intriguingly different.

The puzzle consists of a flat board with three vertical tracks laid in the board with a horizontal track at the top connecting the three. The board is made of two flat pieces fastened together by screws, with intricate shapes hidden between them. Rectangular tile pieces can be moved along and inside the tracks, but cannot be lifted out of the board, nor rotated. Thus, for example, one tile cannot fly over another tile. For the Silver version (see Figure 1), there are ten tiles with blue markings, and ten with orange markings. The markings of the same color are such that they form a narrow, ten-story high triangle-shaped tower, with each tile taking up one floor (or layer). For the Gold version (see Figure 3), each tower is made up of tiles that appear identical, but otherwise is constructed identically to the Silver version.

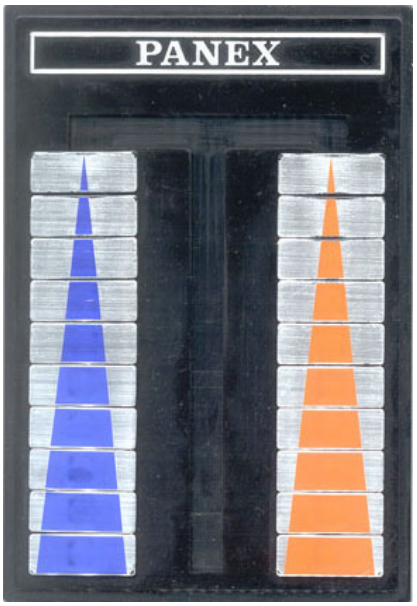


Fig 1. Panex Silver

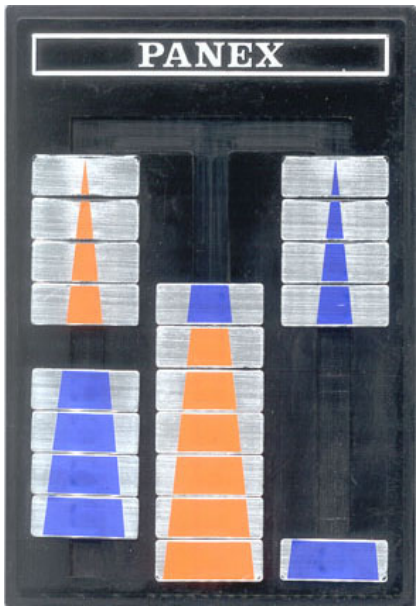


Fig 2. Sample position

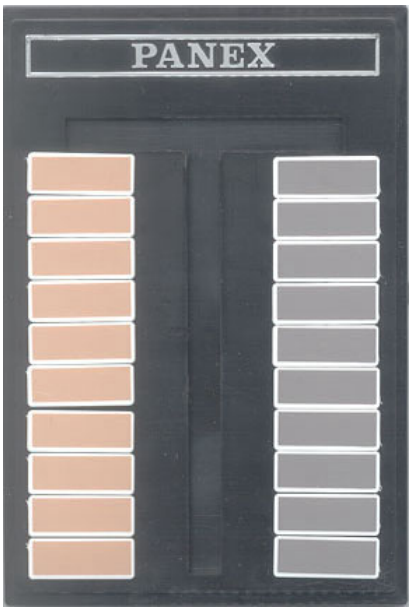


Fig. 3. Panex Gold

The rule of the game is such that the tile with the i^{th} highest floor can only stay in layers 0 through i in any track. (The top layer of a track, where it is connected to another track, is numbered 0. The *cul-de-sac* of a track is numbered 10.) This rule is enforced by the mechanics of the Panex board (see figure 2). There is a tongue (or tang) underneath each tile which decreases in size from first to the tenth tiles. The throat of the track narrows from layer 1 to layer 10. In summary, the rules of the game are

---

Mark Manasse is a senior researcher at Microsoft. **Danny Sleator** is professor of computer science at Carnegie Mellon University. **Victor K. Wei** is professor of information engineering at the Chinese University of Hong Kong. **Nick Baxter** is a mathematician and puzzle expert with a specialty in sliding block puzzles.

This chapter is almost exclusively based on a paper originally written in 1983 by M. Manasse, D. Sleator, and V. K. Wei [7], but never formally published. The original paper, a description of the authors' search program, and other related information is at N. Baxter's Panex Resources [2].

1. The tiles can only move along the tracks. They cannot fly over one another.
2. The tile with the i^{th} highest floor of the triangular tower can only stay in track layers 0 through i.

Initially, the blue tiles are stacked on the left track, and the orange tiles are on the right track. There are two goals to be played on the Panex puzzle. The first is to transfer one tower from a side track to the center track. The second is to exchange two side towers.

Although the TRICKS Company implemented the Panex puzzle with towers of height ten only, we can treat the general puzzle with towers of height n. Let $T(n)$ denote the minimum number of moves to transport a tower of height n from a side track to the center track, and let $X(n)$ denote the minimum number of moves to exchange two towers on the side tracks. In this paper, we will give a formula for $T(n)$ and give upper and lower bounds on $X(n)$. Note that the displacement of a tile is counted as one move. It does not matter how far along the tracks the tile travels, or how many turns it makes.

For $n \geq 3$, the minimum number of moves to transport a tower is

$$T(n) = c_1(1+\sqrt{2})^n + c_2(1-\sqrt{2})^n + \frac{(-1)^n}{2} - 1 ,$$

where

$$c_1 = \frac{7}{4}(-1+\sqrt{2}), \text{ and } c_2 = \frac{7}{4}(-1-\sqrt{2}) .$$

As for the minimum number of moves to exchange towers on the side tracks, $X(n)$, we have obtained the exact value of $X(n)$ for $1 \leq n \leq 8$ by exhaustive computer search. For $n \geq 5$, we have the following upper and lower bounds, $L(n) \leq X(n) \leq U(n)$. The upper bound is given by

$$U(n) = c_3(1+\sqrt{2})^n - 8n - 2 + c_4(1-\sqrt{2})^n ,$$

where

$$c_3 = \frac{7}{2}(7-4\sqrt{2}), \text{ and } c_4 = \frac{7}{2}(7+4\sqrt{2})$$

The lower bound is given by

$$L(n) = 4(T(n) + T(n-1) - 2) .$$

Below is a tabulation of the results for $n <= 10$.

n	$T(n)$	$X(n)$ [1]	$U(n)$	$L(n)$
1	1	3		
2	3	13		
3	9	42		
4	24	128		
5	58	343	343	320
6	143	881	881	796
7	345	2,189[2]	2,189	1,944
8	836	5,359[2]	5,359	4,716
9	2,018	?	13,023	11,408
10	4,875	?	31,537	27,564

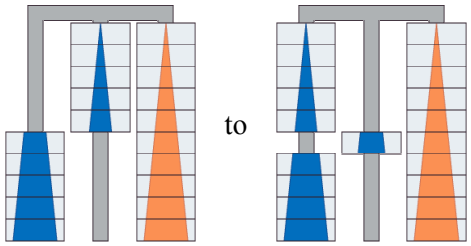
We first present the optimum algorithm for transferring a single tower, then an algorithm for exchanging two towers in $U(n)$ moves. Lastly, we describe our modification of Dijkstra's Algorithm for finding optimum solutions.

[1] The $X(n)$ values are results verified by the authors' computer search program.
[2] Verified first by David Bagley in 2002

The Optimum Sequence of Moves for Transferring Towers

In this section, we will give an algorithm for transferring a tower from a side track to the center track and prove that the moves used are the fewest possible. Specifically, we shall focus on the task of transferring the blue tower from left to center. We will accomplish the transfer problem by a series of *atomic operations*. Each atomic operation is a fixed sequence of moves that achieves a sub-goal toward the eventual goal of tower transfer. Initially, we will prohibit the movement of orange tiles. Later, we will show that the removal of this restriction does not help.

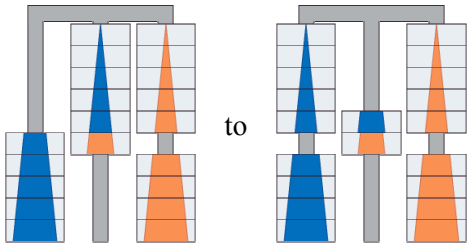
Let T_n denote the operation of transferring a tower of height n from a side track to the center track. Let S_n denote the operation which takes the position where an $(n-1)$-tower is in the center and the n^{th} tile is alone on the side, to the position where the $(n-1)$-tower is on the side and the n^{th} tile is alone in the center. For example, the operation S_6 is illustrated below.



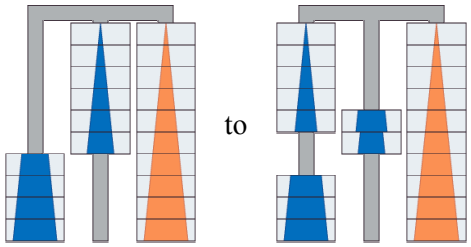
The task T_n can be accomplished in $T(n)$ but not fewer moves. The task S_n can be accomplished in $S(n)$ but not fewer moves. We use the letter "S" to name the second operation because it *sinks* the #n tile to the bottom of the center track.

There are two operations similar to S_n that we will assume take the same minimum number of moves. This assumption will be justified later. In the meantime, these two operations will also be referred to as S_n. These situations are caused by particular combinations on the lower portion of the board. Ambiguity should not arise from reading the context.

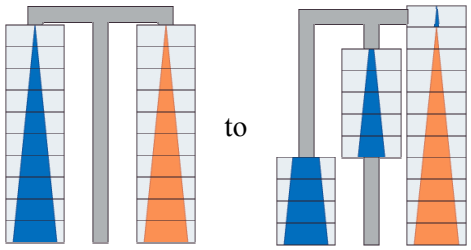
For the first variation of S_n, there is an single extra tile at height n in the center track. (Illustrated for S_6.)



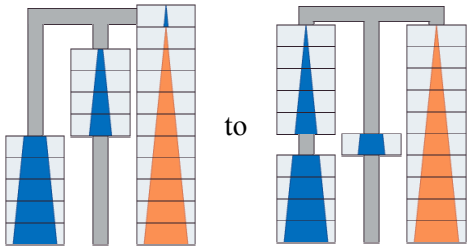
For the second variation of S_n, tile $\#(n+i)$, where $i \geq 1$, instead of tile #n, is sunk to the bottom of a tower of height $n-1$ in the center track. (Illustrated for S_6 and $i=1$)



Let $\overline{T}_n$ denote the operation of transporting a tower of height n from the side track to the center, but with the 1st tile ending up in the opposite corner position. (Illustrated for $\overline{T}_6$).



Assume $\overline{T}_n$ can be accomplished in $\overline{T}(n)$ but not fewer moves. Let $\overline{S}_n$ denote the operation of S_n, but with the 1st tile starting in the opposite corner. (Illustrated for $\overline{S}_6$).



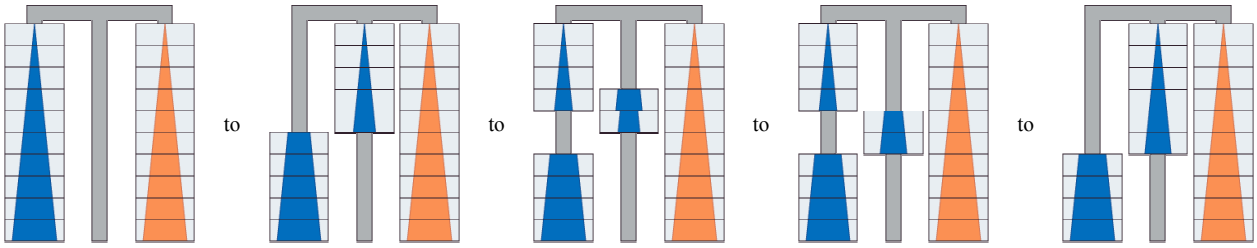
Also assume $\overline{S}_n$ can be accomplished in $\overline{S}(n)$ but not fewer moves. Similar assumptions are made for $\overline{S}_n$ as for S_n.

Let T_n^{-1}, S_n^{-1}, $\overline{T}_n^{-1}$, and $\overline{S}_n^{-1}$ denote the inverse operations of T_n, S_n, $\overline{T}_n$, and $\overline{S}_n$, respectively. Clearly, T_n^{-1} (and S_n^{-1}, $\overline{T}_n^{-1}$, and $\overline{S}_n^{-1}$ respectively) uses the same number of moves as T_n (and S_n, $\overline{T}_n$, and $\overline{S}_n$ respectively).

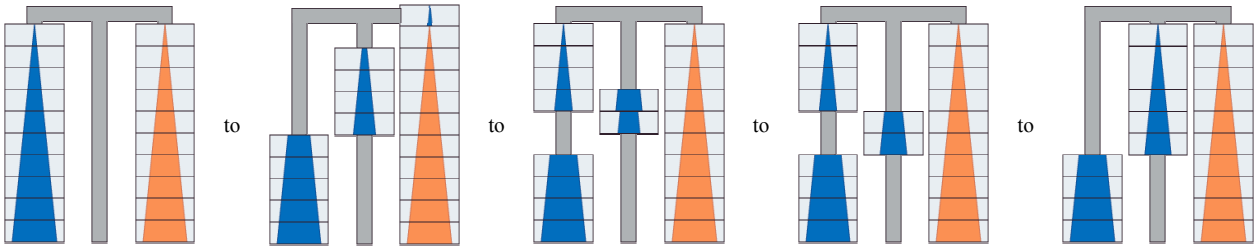
Without much effort, the readers can verify that

$$T(1)=\overline{T}(1)=1, T(2)=S(2)=3, \overline{T}(2)=\overline{S}(2)=2, T(3)=\overline{T}(3)=9$$
$$\text{and } S(3)=\overline{S}(3)=8.$$

For $n\geq 4$, assume we know the shortest sequence of moves to accomplish $T(i), \overline{T}(i), S(i), \overline{S}(i)$ for all $i\leq n$, then the task T_n can be accomplished by applying T_{n-1}, S_{n-1}, a special sequence of four moves, and T_{n-2}, in that order. The positions after each stage are shown below, for n=6.



Another way to accomplish T_n is by applying $\overline{T}_{n-1}$, $\overline{S}_{n-1}$, a special sequence of four moves, and T_{n-2}, in that order. The intermediate positions reached at each stage is shown below, for n=6.



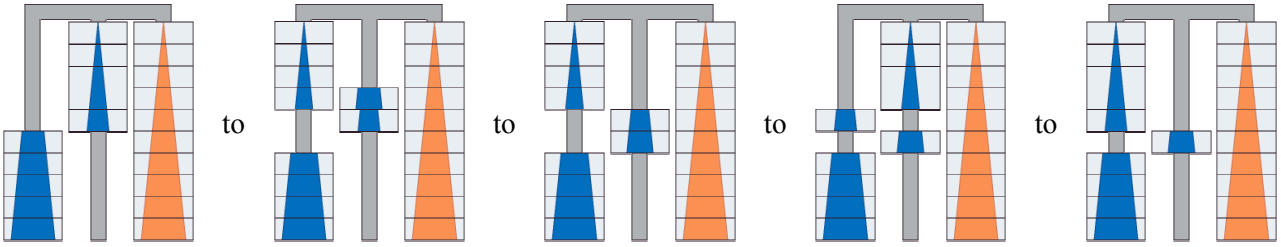
Therefore, the minimum number of moves to achieve T_n is bounded by

$$T(n)\leq \min\{T(n-1)+S(n-1); \overline{T}(n-1)+\overline{S}(n-1)\}+4+T(n-2)\ .$$

Similarly, the task $\overline{T}_n$ can be accomplished by combining smaller tasks, and we have

$$\overline{T}(n) \le \min\left\{T(n-1)+S(n-1);\overline{T}(n-1)+\overline{S}(n-1)\right\}+4+\overline{T}(n-2)\ .$$

The task S_n can be accomplished by applying S_{n-1}, a special sequence of four moves, S_{n-1}^{-1}, and T_{n-2}^{-1}, in that order. The positions reached after each stage are shown below, for n=6.



Another way to accomplish S_n is to combine S_{n-1}, a special sequence of four moves, $\overline{S}_{n-1}^{-1}$, and $\overline{T}_{n-2}^{-1}$, in that order. Therefore the minimum number of moves to accomplish S_n is bounded by

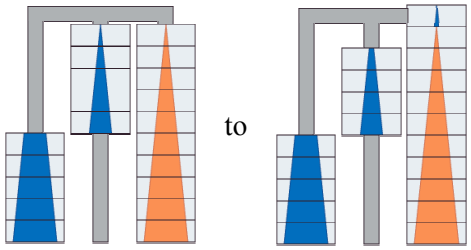
$$S(n) \le S(n-1)+4+\min\left\{S(n-1)+T(n-2);\overline{S}(n-1)+\overline{T}(n-2)\right\}.$$

Similarly, the task $\overline{S}_n$ can be accomplished by combining smaller operations, and we have

$$\overline{S}(n) \le \overline{S}(n-1)+4+\min\left\{S(n-1)+T(n-2);\overline{S}(n-1)+\overline{T}(n-2)\right\}.$$

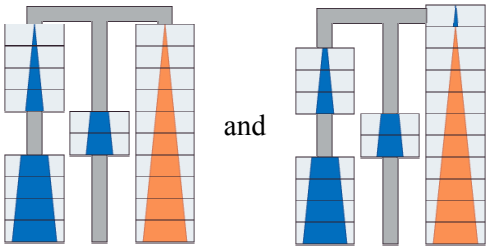
Next, we will show that the equalities hold in all four cases.

In any shortest sequence of moves accomplishing T_n, $n\ge4$, consider the position just before the #n tile is moved for the first time. In order for the #n tile to make a meaningful move, we must have either of the following two positions, illustrated here for n=6:



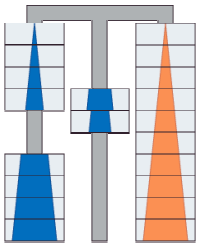
This is because the #n tile must move all the way to the top and around the corner to be meaningful, and these are the only two possible placements of the other tiles to make the move feasible. These two positions are called an *unavoidable* set of positions. The operation from the initial position to the unavoidable position on the left is exactly T_{n-1}, and the operation from the initial position to the unavoidable position on the right is exactly $\overline{T}_{n-1}$. Therefore, the first part of any shortest sequence of moves to accomplish T_n must be a shortest sequence for either T_{n-1} or $\overline{T}_{n-1}$.

Next, consider the position just after the #*(n-1)* tile is moved for the last time in a shortest sequence. There are two possible positions, as shown below for n=6.



But there is no meaningful sequence of moves that can lead into the position on the right, considering the *#(n-1)* file has just been moved. Therefore, the position on the left is an unavoidable position for any shortest sequence. The task from this position to the end is exactly T_{n-2}. Therefore, the last part of any shortest sequence of moves for accomplishing T_n must be a shortest sequence for accomplishing T_{n-2}.

Tracing backwards from this unavoidable position, we see that four moves earlier, we must have been in the position



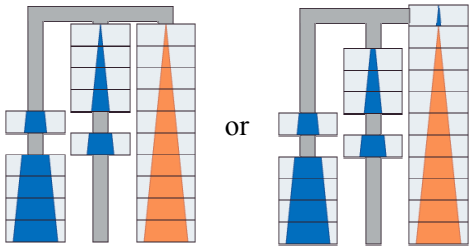
in any shortest sequence. This position is also unavoidable. The sequence of moves before reaching this position must be a shortest sequence for S_{n-1} or $\overline{S}_{n-1}$. Summarizing the arguments, we have

$$T(n) = \min\{T(n-1) + S(n-1); \overline{T}(n-1) + \overline{S}(n-1)\} + 4 + T(n-2)$$

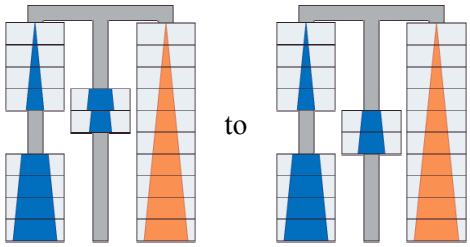
for $n \geq 4$. Similarly, we can argue on unavoidable positions and prove that, for $n \geq 4$,

$$\overline{T}(n) = \min\{T(n-1) + S(n-1); \overline{T}(n-1) + \overline{S}(n-1)\} + 4 + \overline{T}(n-2)$$

In any shortest sequence of moves that accomplishes S_n, $n \geq 4$ consider the last time the *#(n-1)* tile is moved. We must be in either of the following unavoidable set of positions, illustrated for $n=6$.



Focusing on the time when tile *#n* and tile *#(n-1)* change relative positions, we find the following two unavoidable positions, illustrated for $n=6$. We must reach the unavoidable position on the left, and four moves later, reach the unavoidable position on the right.



Therefore, we have

$$S(n) = S(n-1) + 4 + \min\{S(n-1) + T(n-2); \overline{S}(n-1) + \overline{T}(n-2)\}$$

for $n \geq 4$. Similarly, we have

$$\overline{S}(n) = \overline{S}(n-1) + 4 + \min\{S(n-1) + T(n-2); \overline{S}(n-1) + \overline{T}(n-2)\}$$

for $n \geq 4$.

Based on the values for small n, and the four equations, we can derive that, for $n \geq 4$,

$$\overline{T}(n) = \begin{cases} T(n), \, n \text{ odd} \\ T(n) - 1, \, n \text{ even} \end{cases}$$

$$\overline{S}(n) = S(n) = \overline{T}(n) - 1 .$$

Hence,

$$\begin{aligned} \overline{T}(n) &= \overline{T}(n-1) + \overline{S}(n-1) + 4 + \overline{T}(n-2) \\ &= 2\overline{T}(n-1) + \overline{T}(n-2) + 3 \end{aligned}$$

Solving by the techniques of difference equations, we obtain

$$\overline{T}(n) = c_1(1+\sqrt{2})^n + c_2(1-\sqrt{2})^n - \frac{3}{2}$$

for $n \geq 4$, where

$$c_1 = \frac{7}{4}(-1+\sqrt{2}), \text{ and } c_2 = \frac{7}{4}(-1-\sqrt{2})$$

The equation for *T(n)* can be obtained easily, and has been given in the Introduction. The values of $T(n)$, $\overline{T}(n)$, $S(n)$ and $\overline{S}(n)$ are shown here.

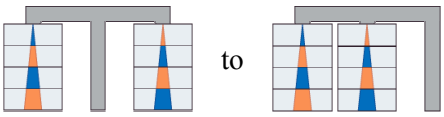
n	$T(n)$	$\overline{T}(n)$	$S(n)$	$\overline{S}(n)$
1	1	1		
2	3	2	3	2
3	9	9	8	8
4	24	23	22	22
5	58	58	57	57
6	143	142	141	141
7	345	345	344	344
8	836	835	834	834
9	2,018	2,018	2,017	2,017
10	4,875	4,874	4,873	4,873

Earlier, we have assumed that two variations of the operation S_n takes the same number of moves as S_n. Judging from the algorithm we presented for accomplishing S_n, and the arguments based on unavoidable positions, we see that these assumptions are indeed true. Earlier, we have also prohibited the movement of the orange tiles. Again, judging from the algorithm and the unavoidable positions, we see that the removal of this restriction does not help shorten the sequence of moves needed to accomplish T_n.

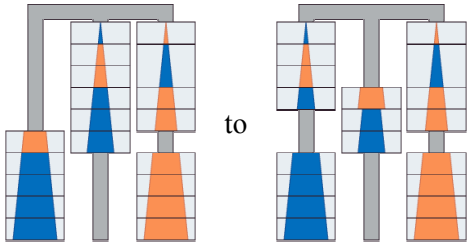
A Good Algorithm for Exchanging Two Towers

In this section we present an algorithm for exchanging two towers on the side tracks.

We need some more detailed notations to present our algorithm. For $Y = T, \overline{T}, S$ or $\overline{S}$, let ${}_LY_n$ denote the operation of performing the task on the left and center tracks, and let ${}_RY_n$ denote the operation of performing the task on the right and center tracks. The combination of colors of the tiles on the tracks is irrelevant. For example, the following operation is denoted ${}_RT_4$:



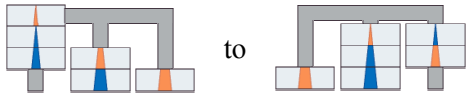
As before, Y^{-1} denote the inverse operation, and the sink task S_n requires the same number of moves even if the sinking tile is $\#(n+i)$, $i>0$, and/or the i^{th} layer of the center track is already occupied. For example, the following operation is also denoted ${}_LS_5$:



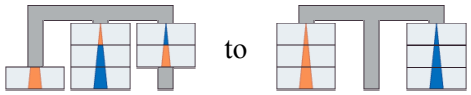
The readers can easily convince themselves that $X(1)$=3 and $X(2)$=13. The following sequence of operations accomplishes X_3 in 42 moves:

$$ {}_L\overline{T}_2 + {}_L\overline{S}_3 + {}_R\overline{T}_2 + Y + Z $$

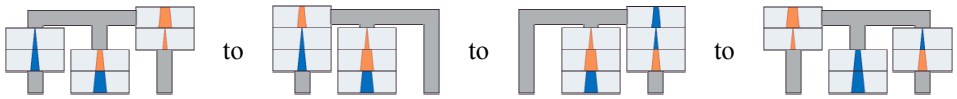
where Y is the following operation which requires 15 moves:



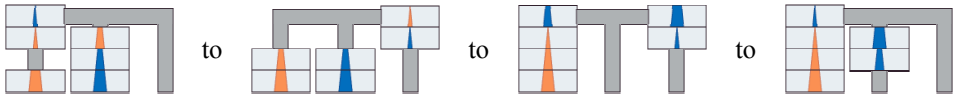
and Z is the following operation which also requires 15 moves:



In Y, intermediate positions after 3, 6, 9, 12 moves are:



In Z, intermediate positions after 3, 6, 9, 12 moves are

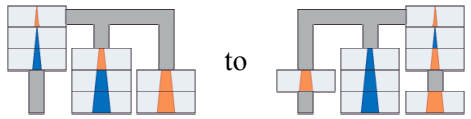


The readers can easily figure out the complete details of Y and Z.

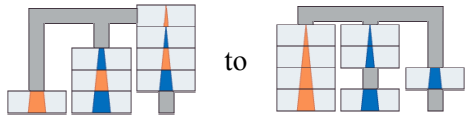
The following sequence of operations accomplishes X_4 in 128 moves:

$$ {}_L\overline{T}_3 + {}_L\overline{S}_3 + 4 + {}_R\overline{T}_2 + \overline{Y} + {}_L\overline{T}_2^{-1} + {}_R\overline{T}_2 + {}_R\overline{S}_4 $$
$$ + {}_L\overline{S}_4^{-1} + W + {}_RS_3 + 4 + {}_R\overline{S}_3^{-1} + {}_R\overline{T}_3^{-1} $$

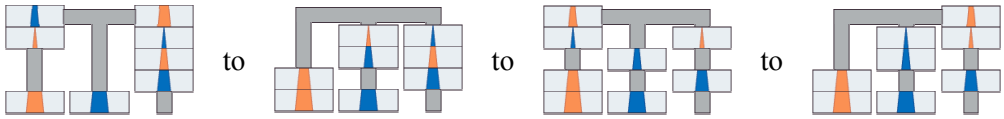
where $\overline{Y}$ is the following operation which requires 15 moves:



and W is the following operation which requires 13 moves:



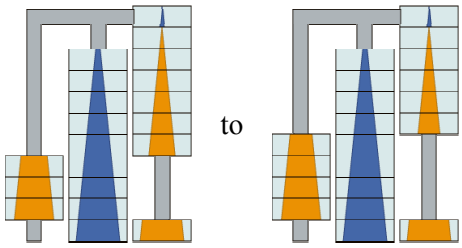
The operation "4" swaps two pieces in the center track in four moves. The first 12 moves of $\overline{Y}$ and Y are identical. The reader can easily figure out the last three moves of $\overline{Y}$. In W, the positions reached after 3, 6, 9, 11 moves are



For convenience, define C_n as the following combination move moves:

$$ {}_L\overline{T}_{n-1}^{-1} + {}_R\overline{T}_{n-1} + {}_R\overline{S}_n + {}_L\overline{S}_n^{-1} $$

which moves the #*i* tile from the right track to the left (shown for C_6).



For $n \geq 5$, the task X_n is accomplished by the following four series of operations:

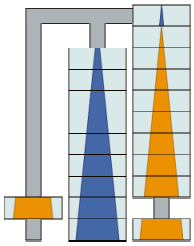
(1) $$ {}_L\overline{T}_{n-1} + {}_L\overline{S}_{n-1} + 4 + {}_R\overline{T}_{n-2} + {}_R\overline{S}_{n-1} + {}_L\overline{S}_{n-1}^{-1} $$

(2) $$ C_{n-2} + C_{n-3} + \ldots + C_5 + C_4 $$

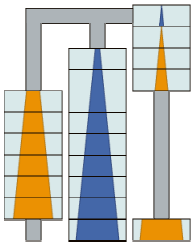
(3) $$ {}_L\overline{T}_2^{-1} + {}_R\overline{T}_2 + \overline{Y} + {}_L\overline{T}_2^{-1} + {}_R\overline{T}_2 + {}_R\overline{S}_n $$

(4) $$ {}_L\overline{S}_n^{-1} + V + {}_R\overline{T}_{n-1}^{-1} $$

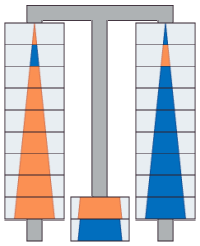
The first sequence takes the initial position into the following position, illustrated for n=10,



The second sequence moves the orange *#(n-2)* through *#4* tiles from the right track to the left, resulting in the following position:

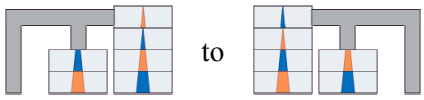


The third sequence continues the second sequence with a set of moves roughly equivalent to $C_3 + C_2 + C_1$, except that the #2 tiles are swapped. In doing so, this sequence saves four moves, at the expense of only one additional move later on to correct it. Then ${}_R\overline{S}_n$ reaches the (roughly) *half-way position*,

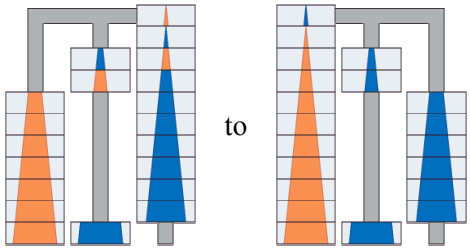


in which all tile pairs except the #*2* tiles have been exchanged and the #10 tiles sit in the center track.

The final sequence takes the half-way position to the end position. The operation *V* uses $\overline{T}(n-1)+\overline{S}(n)+1$ moves to achieve the tasks ${}_L\overline{T}_{n-1}^{-1}$, ${}_R\overline{S}_n^{-1}$, and the exchange of the #2 tiles; it is ${}_L\overline{T}_{n-1}^{-1}$ followed by ${}_R\overline{S}_n^{-1}$, with slight modifications to the last part of ${}_L\overline{T}_{n-1}^{-1}$ and to the first part of ${}_R\overline{S}_n^{-1}$. According to the algorithm presented in the previous section, the last nine moves of ${}_L\overline{T}_{n-1}^{-1}$ constitute ${}_L\overline{T}_3^{-1}$, and the first nine moves of ${}_R\overline{S}_n^{-1}$ constitute ${}_R\overline{T}_3$ (for $n\geq 5$). Together, these 18 moves accomplish the following task:



In *V*, these 18 moves are replaced by 19 moves which take



The first 13 moves of this sequence are identical to the operation *W* described earlier, the reader can easily figure out the remaining 6 moves.

Since $\overline{S}(n)=\overline{T}(n)-1$, we can accomplish X_n, $n\geq 5$, in *U(n)* moves, where

$$U(n)=3\overline{T}(n)+6\overline{T}(n-1)+3\overline{T}(n-2)+4\sum_{i=2}^{n-3}\overline{T}(i)-2n+6.$$

Algebraic manipulations produce the formula for *U(n)* presented in the Introduction. Exhaustive search shows that this upper bound is in fact required for n≤8. Additionally, we suspect that *X(n) = U(n)* for all $n\geq 9$.

Arguing on unavoidable positions, we can show that the following sequence is one of the shortest for exchanging only the #*n* tiles, for $n\geq 5$,

$${}_L\overline{T}_{n-1}+{}_L\overline{S}_n+{}_R\overline{T}_{n-1}+{}_R\overline{S}_n+{}_L\overline{S}_n^{-1}+{}_L\overline{T}_{n-1}^{-1}+{}_R\overline{S}_n^{-1}+{}_R\overline{T}_{n-1}$$

we obtain the lower bound

$$\begin{aligned}L(n)&=4(\overline{T}(n)+\overline{T}(n-1)-1)\\&=4(T(n)+T(n-1)-2)\end{aligned}$$

If desired, the lower bound can be tightened by considering the fewest moves needed to exchange more than one bottom piece.

The Program

We have written a program that finds the shortest sequence of moves that takes the puzzle from any position to any other position. Obviously the amount of time and space required by the program depends on the initial and final positions, as well as the available hardware[1].

Imagine a graph in which each node corresponds to a position (or configuration, state) of the puzzle, and there is an edge connecting two nodes if there is a move that takes one position into the other. If only the top k tiles of each color are to be exchanged, then the graph needs only to contain a node for each configuration of $2k$ tiles. (Moving pieces of size larger than k cannot help us solve the puzzle, therefore the graph need not contain nodes involving different positions of them.) This graph is undirected since the moves are reversible.

A straightforward method of finding a way to get from an initial position to a final position is to generate the corresponding graph, and find the shortest path between the initial and final positions in this graph using Dijkstra's algorithm [3]. The technique we used is just a refinement of this method.

The first refinement is based on the observation that all edges have unit length. Dijkstra's algorithm then works as follows. At any given time we have reached and marked all nodes that are at a distance d or less from the initial position. We also maintain a list (called the *c-list*) of all those nodes that are at distance exactly d from the starting position. One phase of the algorithm has the effect of extending this distance to $d+1$. It works by probing all the nodes adjacent to nodes in the c-list. Those that are already marked are ignored, and those that are not marked are now marked, and placed in the new version of the c-list (called the *n-list*). These newly marked nodes are exactly those that are not at distance d or less from the start and for which there is a path of length $d+1$ from the start. Therefore, the shortest path to these nodes is of length exactly $d+1$. This reaches all the nodes at distance $d+1$ from the start, because a path to a node at distance $d+1$ must go through a node at distance d. The process starts with only the initial node marked and in the c-list. The process terminates when the final node is marked.

In order to reconstruct the shortest path, one additional data structure must be maintained. When we mark a node at distance $d+1$, we update a pointer in that node to the node at distance d that caused it to be marked. At the end we can reconstruct the shortest path by following these pointers from the final node to the initial node.

One way to reduce the storage requirement is to do the computation in such a way that the full graph is never actually generated. The observation that makes this possible is the following: since the graph is undirected, it is the case that when we reach a node that has been reached before (a marked node in the above algorithm), that node is either on the c-list, or it was on the c-list at the previous phase. (We cannot otherwise reach a previously marked node, since such a path would have been traversed immediately after that node was first detected.) Thus, if we keep the old c-list around (we call it the *o-list*), then we can determine if the newly reached node was "marked" simply by determining if it is in the c-list or o-list. The o-list is effectively a boundary that forces the search to go in the right direction.

It is necessary to make the test to determine if a node is in the c-list or o-list more efficient than simply scanning these lists. Hashing is a natural way to do this. We keep a hash table that contains all the nodes of the o-list, c-list, and n-list. We also keep these nodes linked together to form the three lists. At the end of each phase the o-list is deleted from the hash table, the c-list becomes the o-list, and the n-list becomes the c-list. (We use separate chaining so that deletion from the table is efficient.) After hashing a node, we need to be able to tell which list that node is in. We do this by keeping a generation found in each node. If the nodes in the c-list are generation g, then those in the o-list are $g-1$ and those in the n-list are $g+1$. (We actually only need to keep this modulo 3.)

A problem with the method of not keeping the whole graph is that it becomes difficult to reconstruct the solution. If time were not a consideration then we could run the algorithm until we found a path from start to finish and remember the node from which we reached the final one. This node is sure to be on a shortest path

[1] In 1983, the original program running on a VAX 11/750 with two megabytes (with up to six megabytes of virtual memory) was able to solve X_5 in about an hour, and X_6 in under a day; but it could not solve X_7 before running out of memory.

from start to finish. We then run the whole program again using the second to last node as our finishing node. The whole solution can be found by repeating this process *n* times, where there are *n* nodes on the shortest path.

A much more efficient way to deal with this difficulty is to search from start and finish simultaneously. The node where the two searches meet is sure to be on a shortest path. This procedure is then recursively applied to find the shortest path from the starting position to the middle position, and then again to find the shortest path from the middle to the final position. With this trick, the time to construct a complete solution is at most *log(n)* times the time required to find the middle position, where *n* is the number of moves on the shortest path.

Making the program search in both directions involves very small changes. We initialize the c-list to contain both the starting and finishing nodes, and we maintain a bit in each node indicating whether that node was found as a result of the search from the start or from the finish. The first common node is the midpoint (or one of a pair of midpoints).

One final trick to reduce the storage requirement takes advantage of the unique color and mirror symmetries of the puzzle. If x is a position, let x' be the position obtained when the left and right columns of x are swapped, and x^* is that obtained if the colors of all the pieces are swapped. Note that $x^{*\prime}$ and x'^* are the same. Consider the set of nodes of the c-list that have been reached from the starting position (at a particular moment during the running of the program). This set of positions is closed under the $*'$ operator, that is, if x is in this set, then so is $x^{*\prime}$. This is true for the following reason. Consider a sequence of moves that takes us form the starting position to a position x. If we take every node in this sequence and apply the $*'$ operator, then we get a legal sequence of positions from the starting position to position $x^{*\prime}$. (Note that the starting position is invariant under the $*'$ operator.) Since the set in question contains all positions reachable in exactly d steps from the starting position it must contain $x^{*\prime}$. By only storing one of the two symmetrical positions we can reduce the memory requirement by a factor of two.

A similar redundancy occurs because the final position is the initial position with ' applied. This means that the set of nodes accessible from the starting position in d moves is the reflection of the set reachable in d moves from the final position. This means that we do not have to store the list of positions reachable from the final position in d moves, which reduces the storage requirement by another factor of two.

Here is how we incorporate these ideas into the program to save space. Initially the c-list contains just the starting position. When we generate a position (say x) adjacent to one in the c-list, we check if x, x^*, x', or $x^{*\prime}$ are in the hash table. If none of them are, then we just insert x into the hash table (x and $x^{*\prime}$ are accessible from the starting position in $d+1$ moves), and continue. If x' or x^* is in the hash table then x is the middle position of a solution; the program computes the number of moves (based on the generation numbers of x and the x' or x^* found, either $2d+1$ or $2d+2$), and terminates.

Acknowledgement

The authors wish to thank Ron Graham for bringing this puzzle to their attention, and F. Chung, Joe Buhler, and Dick Hess for fruitful interactions.

References

[1] D. Bagley, *Panex Applet*, http://gwyn.tux.org/~bagleyd/java/PanexApp.html.

[2] N. Baxter, *Panex Resources*, http://www.baxterweb.com/puzzles/panex.

[3] E. W. Dijkstra, *A note on two problems in connection with graphs*, Numerische Mathematik **1**, pp. 269-271, 1959.

[4] V. Dubrovsky, Nesting Puzzle, Part 1: *Moving Oriental Towers*, Quantum, Jan/Feb 1996, pp 53-57, 49-51.

[5] E. Henderson, *Panex Puzzle* (level 4 only), http://www.cheesygames.com/panex/ (and uncredited at numerous other sites).

[6] L.E. Hordern, *Sliding Piece Puzzles*, Oxford University Press, 1986, pp 144-145, 220.

[7] M. Manasse, D. Sleator, and V. K. Wei, *Some Results on the Panex Puzzle*. Unpublished, 1983.

[8] J. Slocum and J. Botermans, *Puzzles Old & New*, Plenary Publications International, p 135, 1986.