The Complexity of Sliding-Block Puzzles and Plank Puzzles

Robert A. Hearn

Sliding-block puzzles have long fascinated aficionados of recreational mathematics. From Sam Loyd's infamous 14-15 puzzle to the latest whimsical variants such as Rush Hour$^{\mathrm{TM}}$, these puzzles seem to offer a maximum of complexity for a minimum of space.

In the usual kind of sliding-block puzzle, one is given a box containing a set of rectangular pieces, and the goal is to slide the blocks around so that a particular piece winds up in a particular place. A popular example is Dad's Puzzle, shown in Figure 1; it takes 59 moves to slide the large square to the bottom left. What can be said about the difficulty of solving this kind of puzzle, in general? Martin Gardner devoted his February, 1964 Mathematical Games column to sliding-block puzzles. This is what he had to say [2]:

> These puzzles are very much in want of a theory. Short of trial and error, no one knows how to determine if a given state is obtainable from another given state, and if it is obtainable, no one knows how to find the minimum chain of moves for achieving the desired state.

Forty years later, we still do not have such a theory. It turns out there is a good reason for this: sliding-block puzzles have recently been shown to belong to a class of problems known as *PSPACE-complete*. These problems are thought to be even harder than their better-known counterparts, the NP-complete problems (such as the traveling salesman problem).

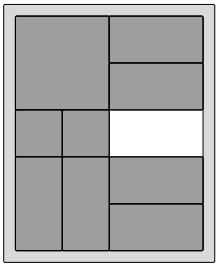


Figure 1: Dad's Puzzle.

In 2002, Gary Flake and Eric Baum showed that Rush Hour is PSPACE-complete. Inspired by their work, Erik Demaine and I were able to show that ordinary sliding-block puzzles (without the lengthwise movement restrictions that Rush Hour imposes) are also PSPACE-complete.

Here I will sketch how to build computers out of sliding-block puzzles, one of the oldest staples of recreational mathematics. I will also introduce *plank puzzles*, one of the newest mathematical recreations, and show that they too possess this computational character.

Complexity Theory

Let's begin with the concept of PSPACE-completeness. Technically, a problem is called PSPACE-complete if it is equal in computational power to a particular mathematical model of computation (called "polynomial-space-bounded Turing machines"). Practically, this means that one can build computers out of elements of the problem, just as one can with wires and transistors.

But how can a sliding-block puzzle have computational power? That is what is so intriguing about these results! Of course, puzzles don't *do* anything on their own (except lure the unwary); they require users, and the users may slide the blocks around any which way they wish, without regard to any supposed computational sequence.

But a puzzle *can* correspond to a computer, in the following sense: if you give me a mathematical problem that such a computer can solve (say, by printing YES or NO), then I can give you back a sliding-block puzzle that has a solution if and only if the computer would print YES.

As a result of this computational property, it is natural to expect that sliding-block puzzles can be made that are very hard indeed: computers can solve difficult mathematical problems, and this difficulty can be translated directly into the difficulty of a puzzle. "Very hard" in this case means that we should not expect to do better than a brute-force search of all the possible move sequences.

Sliding-Block Logic Gates

To make all this a little more concrete, let's see how we might go about building computers out of sliding-block puzzles. Figure 2 shows how to make various logic gates and wiring elements from sliding-block puzzles.

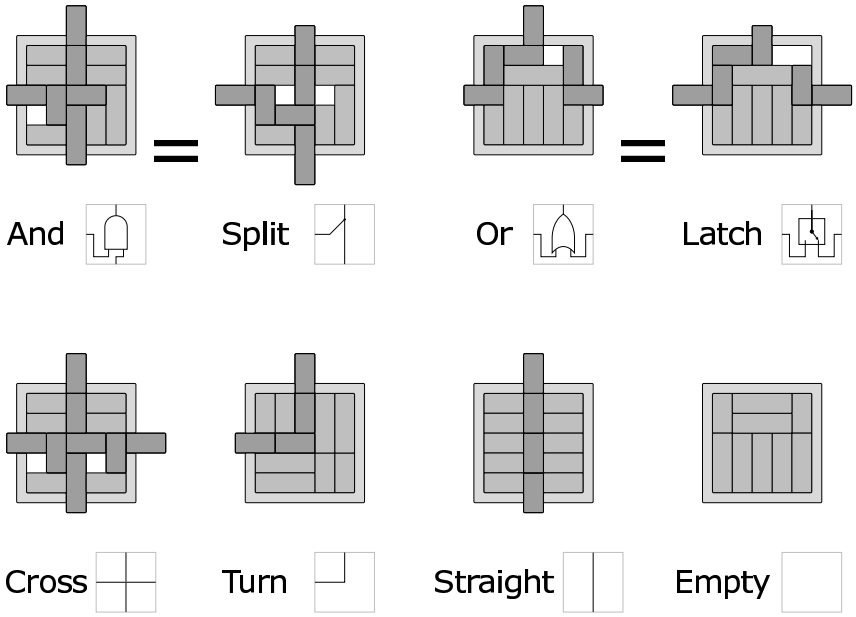


Figure 2: Sliding-block logic gates.

Consider the AND mini-puzzle. Suppose the goal is to slide the upper protruding block into the box – how can we do this? The answer is that first the left block and the bottom block must both be slid out one unit. This will let the internal blocks slide to free up space for the upper block to slide in. (The light gray blocks are spacers, which don't ever need to move.) So, this puzzle does indeed have an AND-like property: both the left *and* the bottom blocks must slide out before the top one may slide in. Likewise, the OR mini-puzzle has an OR-like property: if either the left *or* the right block slides out, the internal blocks can be manipulated to allow the upper block to slide in.

Just as real computers are made by wiring together logic gates, so we can make a sliding-block computer by connecting a lot of these mini-puzzles together, in a very large box. Figure 3 shows a 4×6 grid of these gates assembled into one large puzzle. (Can the reader see how to move block X?) The gates are arranged so that the "input" and "output" blocks are shared between adjoining gates. This is how signals flow throughout the puzzle.

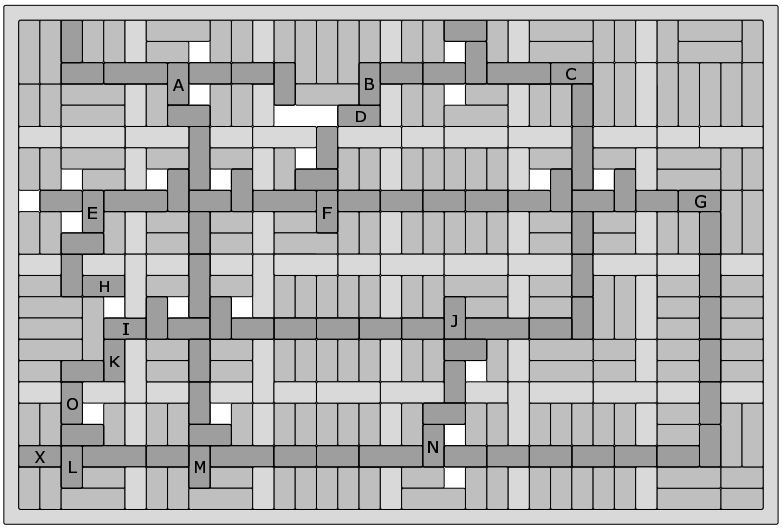


Figure 3: A puzzle made from logic gates. Try to move block X.

Sometimes, wires have to cross over each other; the CROSS mini-puzzle accomplishes this: the left-right motion is independent from the up-down motion. Also, sometimes signals need to be split, and sent off in multiple directions. The SPLIT mini-puzzle does this; if the upper block slides out, then either or both of the bottom and left blocks can be made to slide in. But notice that this is really just an AND gate, used backwards!

That is one of the intriguing properties of this strange, nondeterministic kind of logic: signals can flow both forwards and backwards, and in fact they don't even need to respect normal notions of input and output! For example, block O in Figure 3 appears to be joining an AND input to an OR input. They key property, as it turns out, is that the AND and OR *constraints* are satisfied by any configuration of the puzzle, without regard to what would normally be considered an input or an output.

Perhaps the reader has noticed one omission from the menagerie of logic gadgets: there is no inverter, or NOT gate. Inverters are essential in ordinary digital logic, but they are not possible in this style of logic. To build an inverter, we would need a gate that enabled an output block to slide based on the presence, rather than the absence, of an input block. But a block may only enable another block to move by opening up a hole as it

moves out of the way. And yet, it turns out that with this nondeterministic, constraint-based logic, all you need to build computers are AND and OR, and the ability to wire them together; you can get by without inverters.

The gates in Figure 2 are all made from 1×2 and 1×3 blocks (dominoes and trominoes). Can they be built using only dominoes? The answer is yes, but then they are much more complicated [3].

Sliding-Block Computers

The formal details of the PSPACE-completeness proof are beyond the scope of this essay, but the intuition that one can assemble a sliding-block computer from a collection of logic gates and wiring is the essential idea, and turns out to be valid. With a sufficiently large puzzle, we can arrange things so that the only way to solve the puzzle is to effectively perform a computation, by sliding the blocks following the sequence of logic gate activations a real computer would perform.

Flake and Baum showed explicitly how to build a kind of reversible computer using logic gates made from Rush Hour configurations, similar to the sliding-block gates presented here [1]. Their construction is quite ingenious and elegant.

Demaine and I took a slightly different approach. We started with a mathematical formulation of the underlying logic in terms of "constraint graphs", which turn out to be equivalent to the kinds of circuits shown here. Rather than explicitly build a computer, we showed how a problem (called Quantified Boolean Formulas) which is known to be PSPACE-complete can be translated into a constraint graph problem, and from there into a sliding-block puzzle. This is a common approach in computational complexity theory: by reducing problem A to problem B, one shows that B is at least as hard as A.

One of the subcircuits used in our proof is called a *universal quantifier* (Figure 4); this circuit is the basis for the puzzle in Figure 3. A long string of these circuits, connected together, effectively forces a particular computation to occur in order to solve the puzzle. Each extra universal quantifier also doubles the number of moves required – a string of n quantifiers takes on the order of 2^n moves to solve. This means that not only are there puzzles for which it is difficult to determine whether there is a solution; there are also puzzles that require a vast number of moves to actually solve.

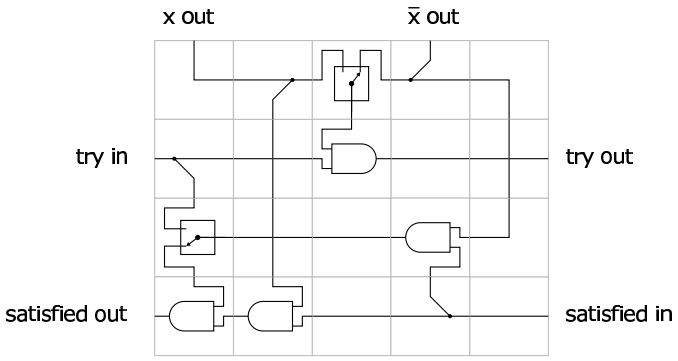


Figure 4: A “universal quantifier” circuit.

Plank Puzzles

Sliding-block puzzles date from the 19th century. Now let’s jump to the 21st century, for the latest in combinatorial challenges: *plank puzzles*, invented by UK maze enthusiast Andrea Gilbert. Like sliding-block puzzles, plank puzzles can hide enormous complexity behind a tame appearance.

The rules are simple. You have to cross a crocodile-infested swamp, using only wooden planks supported by tree stumps. You can pick up planks and put them down between other stumps, as long as they are exactly the right distance apart. You are not allowed to cross planks over each other, or over intervening stumps, and you can carry only *one* plank at a time.

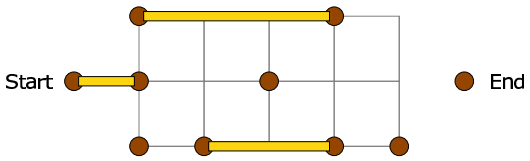


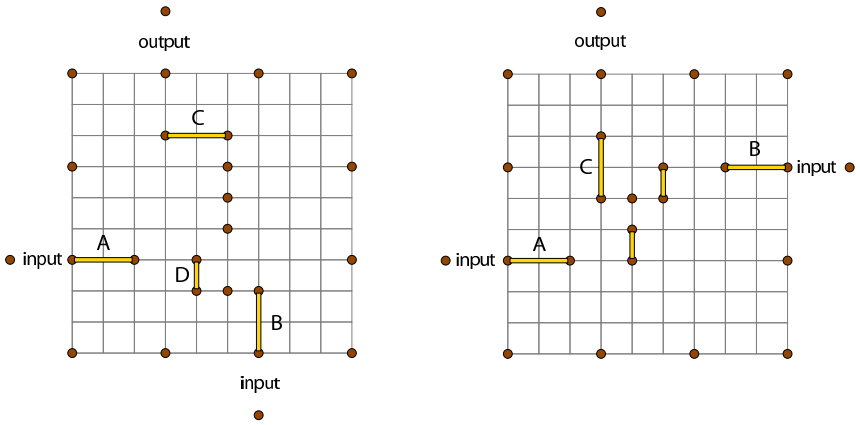
Figure 5: A plank puzzle.

A sample plank puzzle is shown in Figure 5. The first few moves of the solution are as follows: walk across the length-1 plank; pick it up; lay it down to the south; walk across it; pick it up again; lay it down to the east; walk across it again; pick it up again; walk across the length-2 plank; lay the length-1 plank down to the east... Can the reader see how to finish the sequence, and safely cross the swamp?

Many challenging plank puzzles are available on Ms. Gilbert's website, www.clickmazes.com, as playable Java applets. Plank puzzles are also available in physical form, from Binary Arts (called River Crossing^{TM}).

Plank-Puzzle Logic Gates

Just as with sliding-block puzzles, we can build nondeterministic, constraint-based computers out of plank puzzles. Thus, plank puzzles are also PSPACE-complete, and we should not be surprised that they can be very difficult.



(a) AND (b) OR

Figure 6: Plank-puzzle AND and OR gates.

The plank-puzzle AND and OR gates are shown in Figure 6. The length-2 planks serve as the input and output ports. To see how these gates work, consider the AND gate. Both of its input port planks (A and B) are present, and thus activated; therefore, you should be able to move its output port plank (C) outside the gate. Suppose you are standing at the left end of plank A. First walk across this plank, pick it up, and lay it down in front of you, to reach plank D. With D you can reach plank B. With B and D, you can reach C, and escape the gate. (Note that in sliding-block gates, signals propagate by blocks sliding *backwards* to fill holes, but in plank-puzzle gates, signals propagate by planks moving *forwards*.)

To complete the construction, we must have a way to wire these gates together into large puzzle circuits. It's all well and good to activate a single AND gate, but when you've done that, you're stuck standing on its output plank - now what?

Figure 7 shows a puzzle made from six gates. For reference, the equivalent circuit is shown in Figure 8. The gates are arranged on a staggered grid, in order to make matching inputs and outputs line up. The port planks are shared between adjoining gates. Notice that two length-3 planks have been added to the puzzle. These are the key to moving around between the gates. If you are standing on one of these planks, you can walk along the edges of the gates, by repeatedly laying the plank in front of you, walking across it, then picking it up. This will let you get to any port of any of the gates. However, you can't get inside any of the gates using a length-3 plank, because there are no interior stumps exactly three grid units from a border stump.

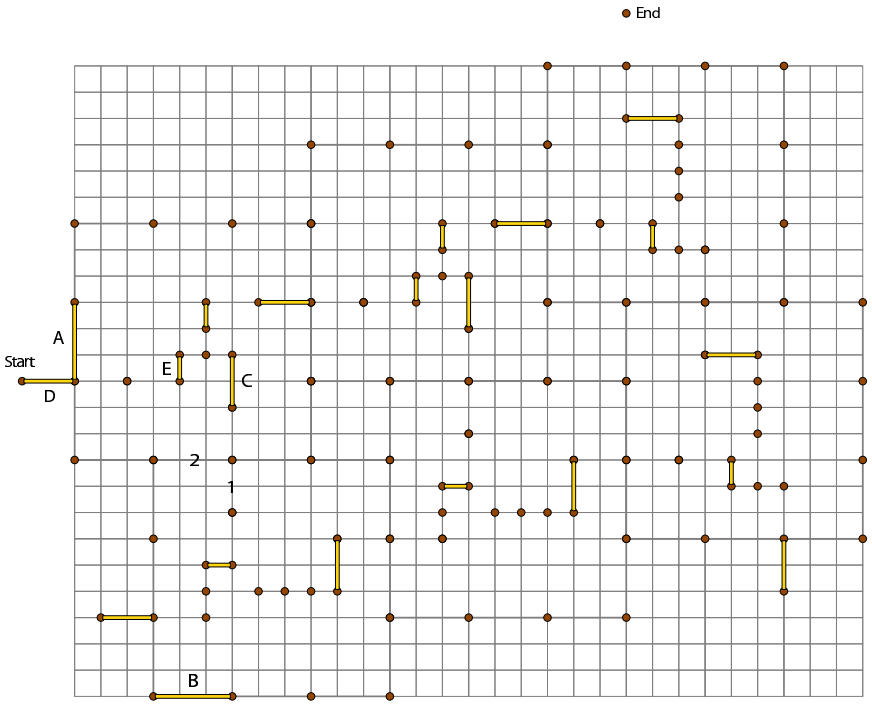


Figure 7: A plank puzzle made from logic gates.

Suppose you want to move plank C to position 1 (thus activating an OR output). This is what you do: first, walk plank A over to plank B. You can walk both of these planks together until B is in position 2, by alternately laying each plank in front of the other. Then walk A back to its starting position. Now, when you activate the OR, by using planks D and E, plank B is sitting there waiting for you at the gate exit. You can then use it and plank A to position yourself for the next gate activation. Can the reader see how to finish solving the puzzle?

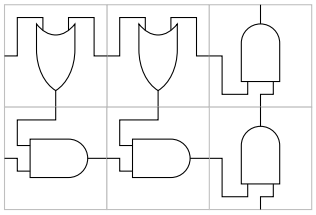


Figure 8: The equivalent circuit for Figure 7.

As a result of this construction, everything we have said about sliding-block puzzles also applies to plank puzzles: it can be very difficult to determine whether there is a solution, and there are puzzles that take exponentially many moves to escape.

Conclusion

Puzzle PSPACE-completeness is not reserved for sliding-block puzzles and plank puzzles alone. The methods described here have been applied to several other kinds of puzzles and problems as well. The list so far includes Rush Hour, ordinary sliding-block puzzles, plank puzzles, Sokoban, hinged polygon dissections, and many related block-pushing problems. One might speculate that any sufficiently interesting motion-planning puzzle is PSPACE-complete, but there seems no hope of proving this in the abstract.

The lack of a successful theory of sliding-block puzzles, after all these years, could be seen as a mathematical failure. But to me, the reasons for the failure are inextricably linked to the very reason the puzzles are interesting: one can build all sorts of weird and wonderful gadgets with them. Furthermore, there is *still* a theoretical possibility for a successful theory!

PSPACE-complete problems are indeed believed to be very hard, but nobody has yet proven that there is not some efficient algorithm that solves them all. An efficient method for solving sliding-block puzzles would yield the most important result in computer science of all time.

Answers

"Universal quantifier" puzzle: Only key moves are given. Slide blocks as far as possible. E left, D left, F up, G left, N right, J down, C left, I right, H left, K up, J left, B up, N up, F right, D right, A right, F up, G left, M up, L up, X right.

Small plank puzzle conclusion: pick up length-2, lay it north, pick up length-1, walk to far end of length-3, lay length-1 south, go back and pick up length-2, go back to far end of length-1, lay length-2 east, go back and pick up length-3, go back to end of length-2, lay length-3 east, escape.

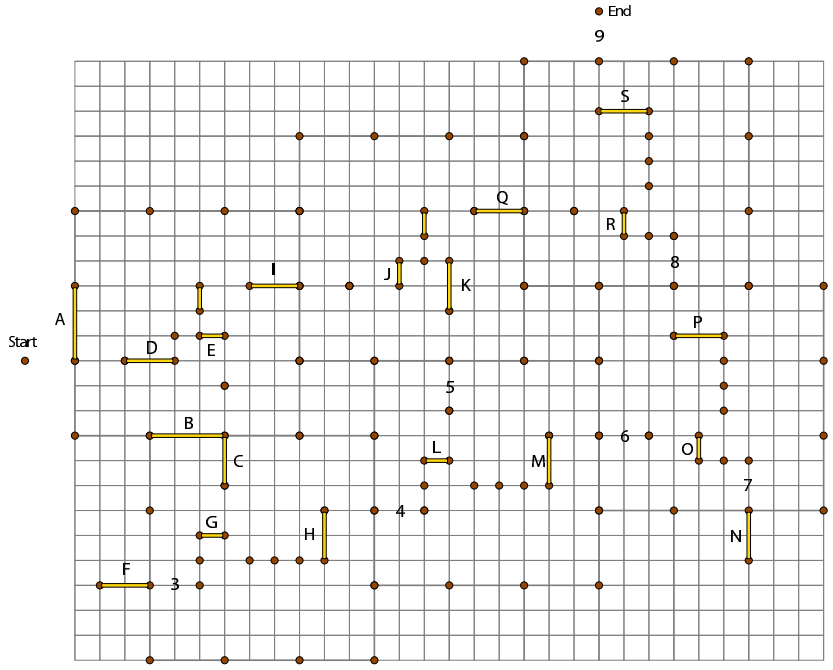


Figure 9: Figure 7 after activating an OR.

Large plank puzzle: Figure 9 shows the puzzle after moving plank C as described. Before each step below, position A and B as needed. Move F to 3. Using C, G, and F, move H to 4. Using I and J, move K to 5. Using K, L, and H, move M to 6. Move N to 7. Using M, O, and N, move P to 8. Using Q, R, and P, move S to 9 and escape.

Acknowledgments

Thanks to Erik Demaine of MIT for pointing me towards the complexity problem for sliding-block puzzles, and for all the help in deriving the results. Thanks to Michael Albert of the University of Otago for introducing me to plank puzzles, and for help in designing the logic gates. Thanks to Andrea Gilbert for permission to reproduce one of her plank puzzles. Thanks to Martin Gardner for 30 years of Mathematical Games.

References

[1] Gary William Flake and Eric B. Baum. *Rush Hour* is PSPACE-complete, or "Why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1–2):895–911, January 2002.

[2] Martin Gardner. The hypnotic fascination of sliding-block puzzles. *Scientific American*, 210:122–130, 1964.

[3] Robert A. Hearn and Erik D. Demaine. The nondeterministic constraint logic model of computation: Reductions and applications. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 401–413, 2002.

[4] Edward Hordern. *Sliding Piece Puzzles*. Oxford University Press, 1986.