6.890: Algorithmic Lower Bounds: Fun With Hardness Proofs Fall 2014

Lecture 5 — September 18, 2014

Prof. Erik Demaine *Scribe: Quanquan Liu, Jonathan Weed*

1 Overview

We'll start this week with a few more examples of classic Nintendo games, most of which are hard via reductions from 3SAT or a variant. Then, we'll look at a few more puzzles, including one whose reduction was thorny enough to exhibit almost all the parity problems you might run into during a reduction of your own.

2 Main Section

2.1 Super Mario Bros.

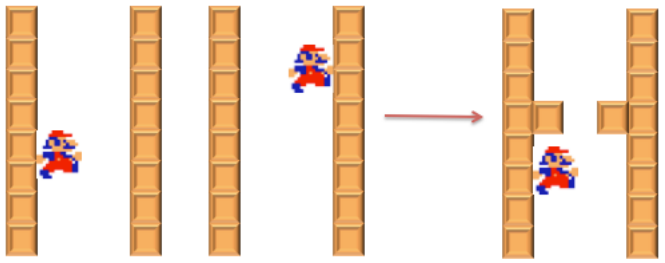
We saw in the first lecture that Super Mario Bros. is NP-Hard [**?**]. The reduction is from 3SAT, and the main lesson there was to construct clause, variable, and crossover gadgets.

(Super Mario World is also hard [**?**]—we just need to develop analogous gadgets.)

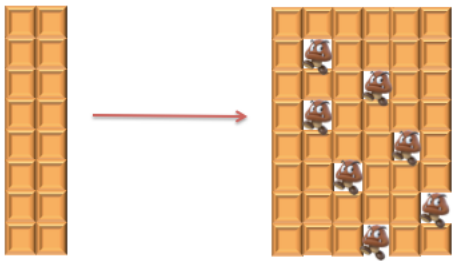
What we didn't discuss the first time was *glitches*. The original proof worked on an ideal Super Mario Bros. without bugs, but the real life version sometimes allows Mario to move in ways that should be impossible.

2.1.1 Glitches

- **Wall Jump:** Sometimes you can jump up walls. This isn't generally possible with human reflexes, but is possible on computer-assisted speed runs. *Solution:* Add bars on each wall so that you're always blocked when you jump upwards.



- **Jump Through Walls:** Sometimes you can jump through walls (even with human reflexes). *Solution:* Replace walls by walls filled with monsters.



2.2 Legend of Zelda

Most Zelda games contain a block-pushing puzzle, so they're hard by standard block pushing results. (In fact, Zelda's block pushing is a little more constrained. It's called *Push-Once*: not only can you only push one block at a time, but you can only push a block once. The lock mechanism from Push-1 can be modified for a Push-Once environment, so Push-Once is also hard.)

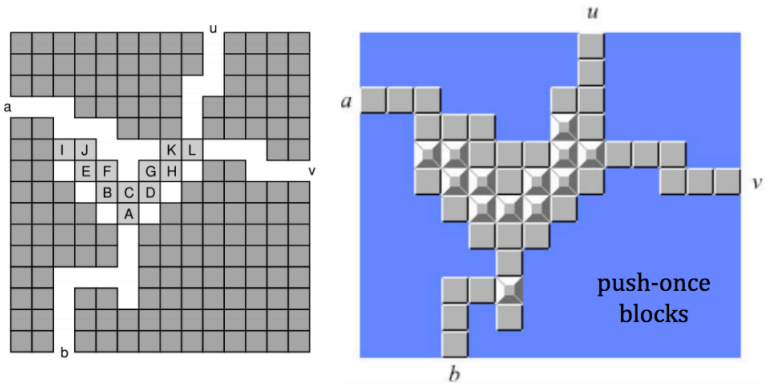


Figure 1: The modification of the Push-1 lock to the Push-Once lock prevents Link from pushing a block more than once (assuming Link does not have a raft) because blocks can't be pushed into the water.

However, it would be nice to give a Legend of Zelda reduction that doesn't follow directly from block-pushing, and in fact such a reduction exists, so Legend of Zelda is NP-Hard [**?**]. The reduction uses a Zelda item called a hookshot, which allows the player to jump over holes as long as there's an object on the other side; the reduction from 3SAT requires creating variable, clause, and crossover gadgets, as haul.

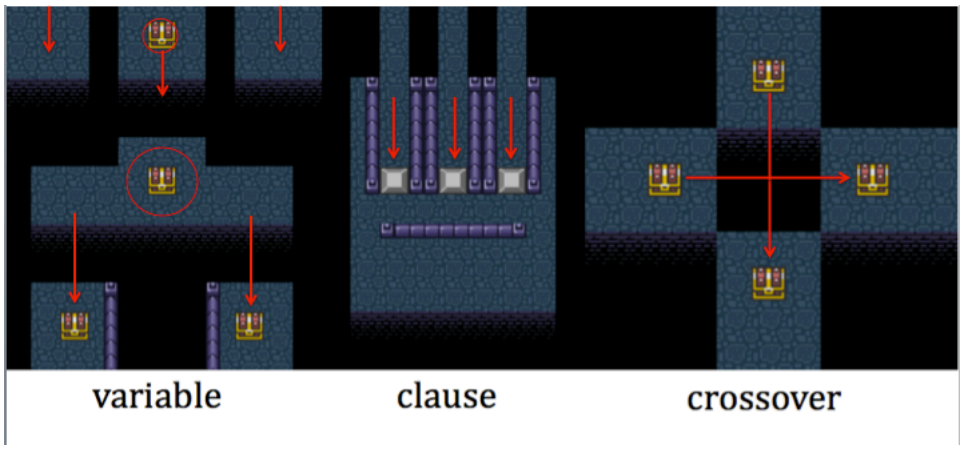


Figure 2: The variable gadget lets Link hookshot from the chests in the center to either of the chests on the bottom. Link can push any of the blocks down so that he can later hookshot from the left. Finally, the crossover gadget only allows Link to hookshot to chests that are directly across.

2.3 Metroid

Metroid is NP-complete [**?**]. The variable gadgets work just as they do in Super Maro Bros.; activating a variable in a clause gadget allows the player to kill all the enemies who would otherwise block her path to finishing the level. The crossover gadget involves a well-timed group of enemies that cycle around the gadget—a player only has time to cross in one direction and would be killed by turning onto a forbidden path. See the paper for more details!

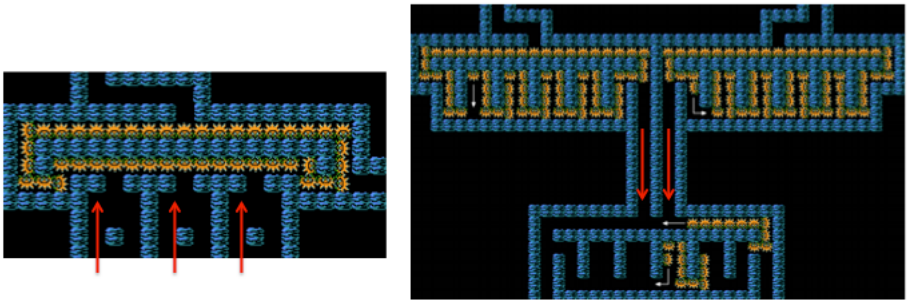


Figure 3: The clause gadget allows Samus to come through one of the tunnels below the monsters and shoot up to kill them. However, if she came from the top path, then there is nothing she could do because she can't shoot down. The cross over gadget allows her to either come from the left or right. Depending on which tunnel she chose she has to either roll left or roll right to avoid the monsters that are moving in opposite directions.

Remark: We actually claim NP-completeness for Metroid because the states are simple enough to be described in polynomial time. Most of the other problems in this lecture may not be in NP because verifying a certificate could take too long.

2.4 Donkey Kong Country

Donkey Kong Country is NP-Hard [**?**]. By now, the reduction is familiar: variable gadgets involve exclusively choosing a true or false paths; clause gadgets allow the player to remove an obstruction (in this case, a bee), which would otherwise block her path to the exit. The crossover gadget in Donkey Kong Country is a cross of four shooter barrels: since the player cannot control Donkey Kong while he's in the air, players can use crossover gadgets only for the intended paths.

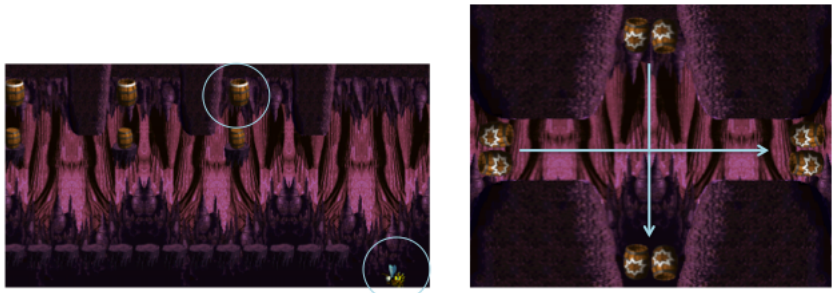


Figure 4: Donkey Kong picks up a barrel to kill the bee to later traverse the path in the clause gadget. Then, for the crossover gadget, the roll forward and backward barrel prevents the vertical path from leaking into the horizontal path and vice versa.

2.5 Pokemon

"Pokemon is a somewhat complicated game." In the reduction considered in [**?**], there are weak trainers and strong trainers. The player always wins against weak trainers and always loses against strong trainers. Trainers you've beaten (all weak trainers, of course) act as obstacles for strong trainers once they're activated. Satisfying a clause corresponds to activating several barrier trainers, which allow the player to exit all clause gadgets if and only if they're all satisfied. (We imagine that trainers have a larger field of vision than they do in the real game, though the reduction still requires only a bounded constant range.) With this setup, Pokemon is NP-Hard.

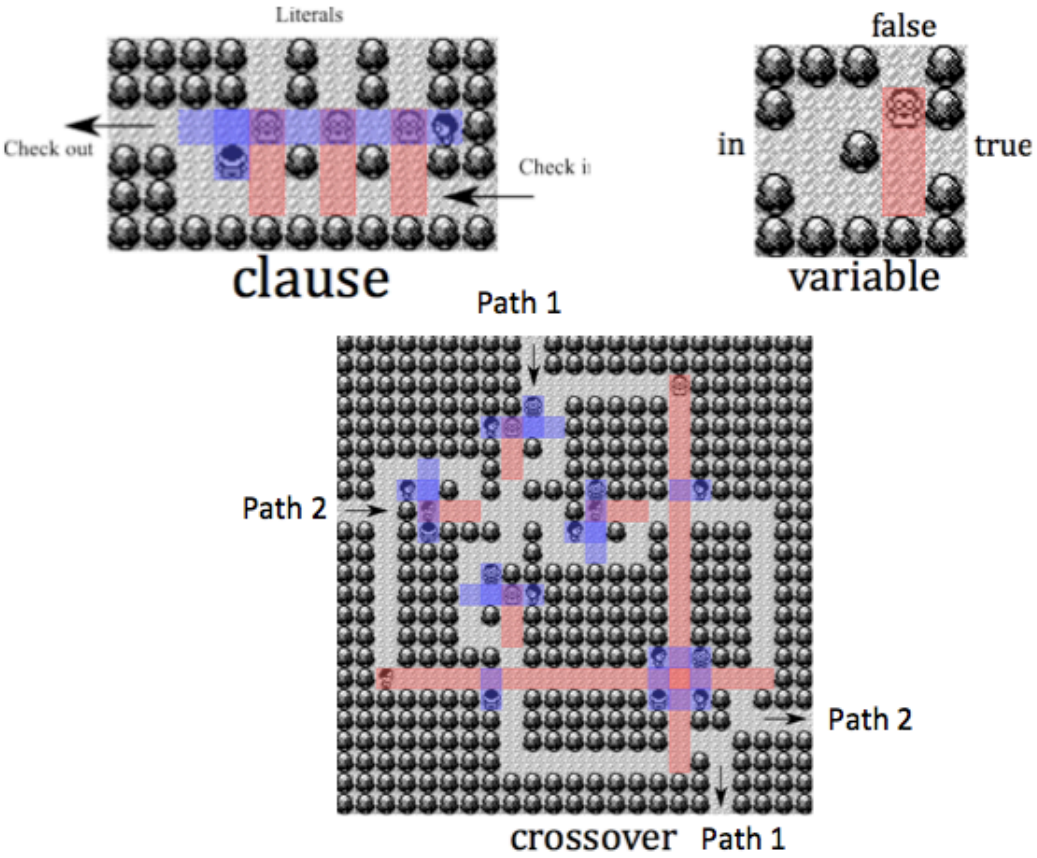


Figure 5: Variable, clause, and crossover gadget. The purpose of the clause gadget is to come down from the top and eliminate one of the red trainers in advance.

Erik would like to establish his Pokemon cred, so he read us this slide: "Weak Trainers each hold a Level 100 Electrode with maximum Speed and equipped with only the Self Destruct move. Strong Trainers each hold two Snorlaxes, with Speed of 30. The player has no items, and only one Pokémon in his team. For Generation I and II games (Red/Blue/ Yellow and Gold/Silver/Crystal versions respectively), the player holds a Gastly which has learned Self Destruct using TM36, and its PP for its other moves have all been expended, so it can only use Self Destruct in battle. When the player encounters a weak Trainer, the enemy Electrode will move first and use Self Destruct, which deals no damage to Gastly since Self Destruct is a Normal type attack and Gastly is Ghost type, so the weak Trainer immediately loses. When the player encounters a strong Trainer, Gastly moves first and uses Self Destruct, causing the player to lose (even if it defeats the enemy Snorlax, the opponent holds another one). This implementation only works in Generations I and II since TM36 exists only in Generation I and the Time Capsule feature in Generation II allows a Gastly with Self Destruct to be traded from Generation I to Generation II. In Generations III, IV, and V, Gastly can be replaced by Duskull, which is allowed to learn the move Memento, which serves the same purpose as Self Destruct, via breeding."

2.6 Conway's Phutball (Philosopher's Football)

Phutball is a 2-player game played with black and white stones on a square lattice (or Go board). Players move by either adding another black stone to the board or moving the white stone by jumping over contiguous groups of black stones and removing them. The goal is to move white stone to one of the goals at the end of the board.

This game is not solved, and a large class of problems are still unsolved. The game is PSPACE-hard [**?**] and even a simple problem such as deciding whether a current position is a mate-in-1 is NP-complete [**?**]. Whether the game is EXPTIME-complete is still open.

The reduction in [**?**] lays out variables and clauses on a large 2-dimensional board, with variable choices along the left and right edge and clauses along the top and bottom. The player makes a series of long horizontal jumps to choose which variables to set to true, and is able to traverse top to bottom if and only if none of the crucial squares have been cleared.

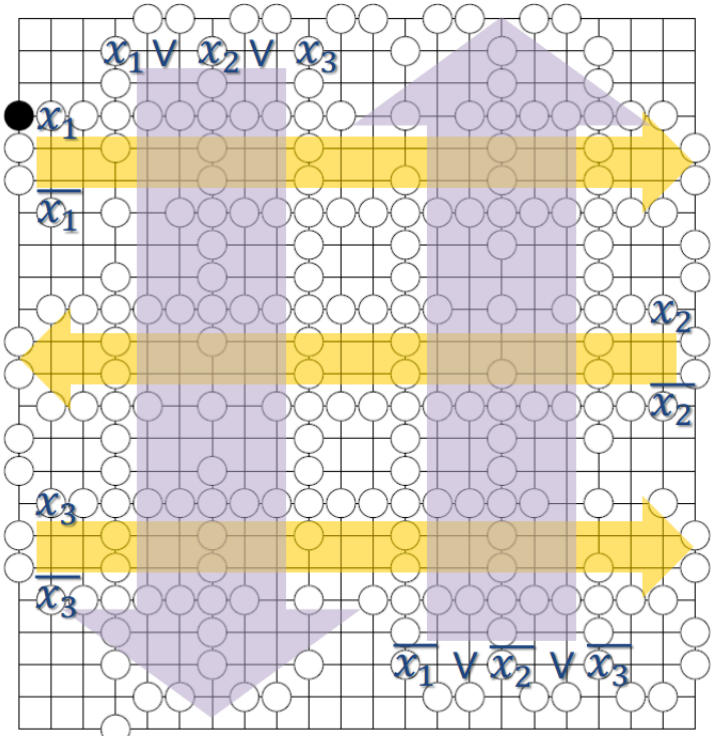


Figure 6: Here clauses are traversed by going top to bottom or bottom to top. Variable values are set by going left to right and vice versa.

One heuristic for understanding why even this simple question is NP-complete is that one move in phutball can comprise a large number of individual jumps. However...

2.7 Checkers

Checkers mate-in-1 also seems to involve a large number of jumps, but deciding whether a checkers position is a mate-in-1 is in P: the problem reduces to the question of finding an Eulerian path on

a graph, which is very easy to decide [**?**]

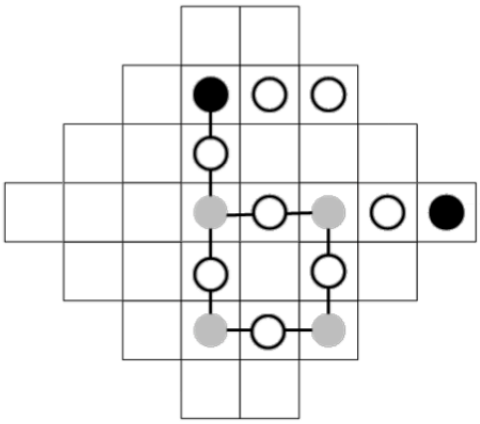


Figure 7: We can rotate the board by 90 degrees and now jumps are vertical and horizontal. All horizontal and vertical jumps preserve x and y parity.

2.8 Cryptarithms/Alphametrics

Cryptarithms are classic puzzles involving arithmetic on words.

	S	E	N	D
+	M	O	R	E
M	O	N	E	Y

Table 1: The SEND MORE MONEY Cryptarithm.

The goal is to find a bijection between letters and numbers making the arithmetic operations correct. [1]

Cryptarithms are NP-complete [**?**], which Epstein showed by reducing from 3SAT. (We have to allow numbers with a large base, but with that proviso the problem is strongly NP-complete.) Variables and clauses correspond to contiguous sets of columns, with the rightmost three columns reading:

$$\begin{array}{r} 0\,p\,0 \\ 0\,p\,0 \\ \hline 1\,q\,0 \end{array}$$

These three columns force the letter "0" to have the value 0 and the letter "1" to have the value 1. A variable v_a is considered true if $v_a \equiv 1 \pmod 4$ and false if $v_a \equiv 0 \pmod 4$—the variable gadget

[1] For the puzzle in Table **??**, $S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2$.

sets both v_a and its opposite $\overline{v_a}$. This is accomplished through a string of intermediate sums:

$$\begin{aligned}
b_i &= 2a_i \\
v_i &= 2b_i + C \qquad C = \text{carry}(y_i + y_i) \in \{0, 1\} \\
&= 4a_i + C \equiv C \pmod 4 \\
d_i &= 2c_i + C \\
e_i &= d_i + 1 + C \\
&= 2c_i + 1 + 2C
\end{aligned}$$

$$\begin{aligned}
\overline{v_i} &= d_i + e_i \\
&= 4c_i + 1 + 3C \\
&\equiv 3C + 1 \equiv 1 - C \pmod 4
\end{aligned}$$

d_i	0	1	y_i	0	c_i	y_i	0	b_i	y_i	0	a_i	0
e_i	0	d_i	y_i	0	c_i	y_i	0	b_i	y_i	0	a_i	0
$\overline{v_i}$	0	e_i	z_i	0	d_i	z_i	0	v_i	z_i	0	b_i	0

Table 2: The variable gadget.

A clause gadget is a string of columns that checks whether the sum of three variables is congruent to 1, 2, or 3 modulo 4; if it is, then one of the variables was true.

u_{ab}	0	v_a	0	1	r_i	0	g_i	w_i	0	f_i	0
v_c	0	v_b	0	h_i	r_i	0	g_i	w_i	0	f_i	0
t_i	0	u_{ab}	0	t_i	s_i	0	h_i	x_i	0	g_i	0

Table 3: The clause gadget.

$$\begin{aligned}
g_i &= 2f_i \\
h_i &= 2g_i + \{0, 1\} \\
&= 4f_i + \{0, 1\} \\
t_i &= h_i + 1 + \{0, 1\} \\
&= 4f_i + 1 + \{0, 1, 2\} \\
&= 4f_i + \{1, 2, 3\}
\end{aligned}$$

$$v_a + v_b + v_c = t_i \equiv \{1, 2, 3\} \pmod 4$$

This construction can be simplified considerably by instead reducing from 1-in-3SAT, which is still NP-complete. We don't need negations anymore, and we can then check a clause by seeing whether its three variables sum to 1 (mod 4):

$$
\begin{aligned}
g_i &= 2f_i \\
h_i &= 2g_i \\
&= 4f_i \\
t_i &= h_i + 1 \\
&= 4f_i + 1
\end{aligned}
$$

$$v_a + v_b + v_c = t_i \equiv 1 \pmod 4$$

It's easy to check that a solution to the cryptarithms gives a solution to 3SAT, but the reverse is less clear: we still need a way to assign the letters in a distinct fashion. In particular, we would like to set v_i and $\overline{v_i}$ such that the sums $v_i + v_j + v_k$ are all distinct. We can show inductively that a polynomial number of values suffice; we assign the v_i in order of the index i, selecting v_i to avoid the situation where $v_i = v_j + v_k + v_l - v_m - v_p$, for $j, k, l, m, p < i$. Since there are only n^5 such quintuples, as long as we have more than n^5 numbers it will always be possible to select a v_i satisfying this condition.

Since the parameters remain polynomial, this construction proves that cryptarithms are strongly NP-hard.

2.9 Star Tessellation

Deciding whether a crease pattern has a flat folded state is NP-hard [**?**]. The easy part is deciding where the polygons in a folding pattern end up in the plane; the difficult part is deciding which pieces of paper lie atop others.

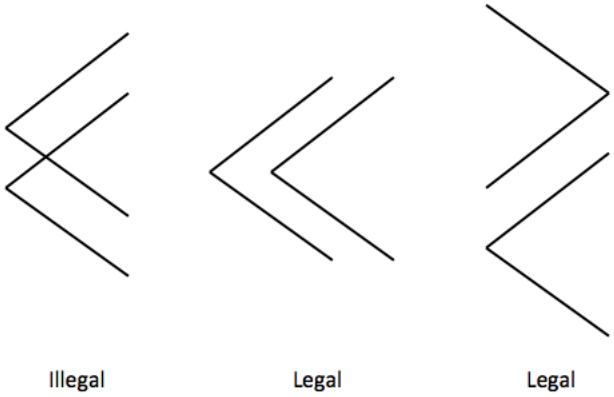


Figure 8: Some crease patterns are legal while others are not. If the creases locally intersect, then the pattern is illegal because a piece of paper can't both be on top and under itself.

We build our reduction out of pleats (two parallel segments). Two parallel creases can't both be "mountains" or both be "valleys," because then the paper would locally self-intersect, so there has to be one fold of each type. Variables will correspond to orientations of pleats.

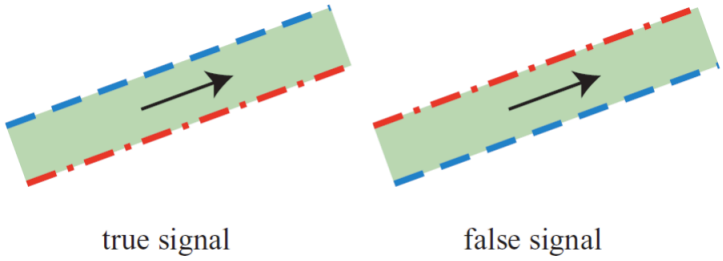


Figure 9: Pleats represent different variables. Depending on which crease is a mountain and which is a mountain fold and which is a valley fold, we can represent True or False.

A triangular twist is the intersection of three pleats in such a way that the twist can only be folded if the pleats are not all oriented the same direction. This is a Not-All-Equal gadget.

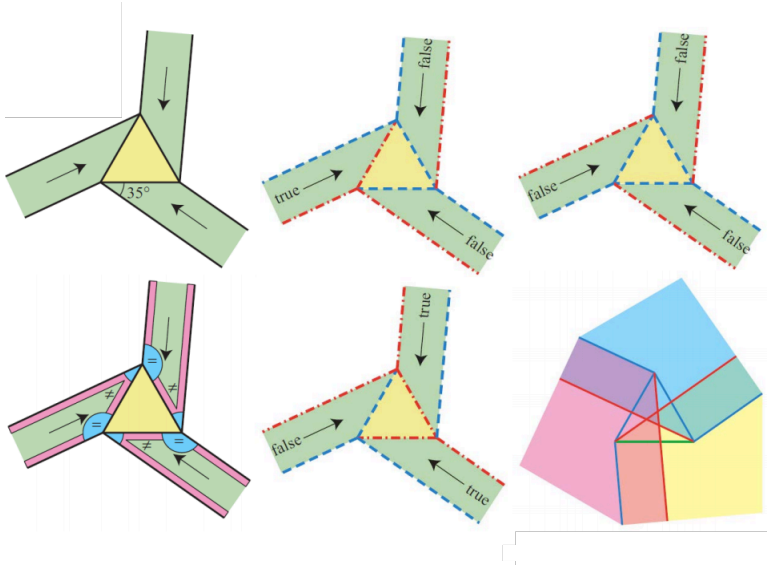


Figure 10: Arrows within the triangular gadget all point into the gadget. Because not all of the folds pointing into the center of the gadget are the same, we can produce a flat fold. However, if they are all the same, then we can't produce a flat fold because each fold of the paper is on top of another.

Remark: A nice reason to hope for a reduction from Not-All-Equal-3SAT is that "true" and "false" are totally symmetric in origami, and Not-All-Equal-3SAT is a totally symmetric formulation of satisfiability.

Checking that these folds can be represented polynomially is a little tricky, but serves to establish that the problem is in NP.

The splitter/negation gadget forces a fold to be false if another fold next to it is true.

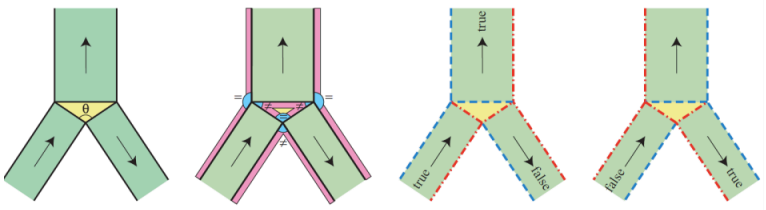


Figure 11: Splitter/negation gadget.

The crossover gadget makes sure that a fold can fold one way or another without being able to fold both ways.

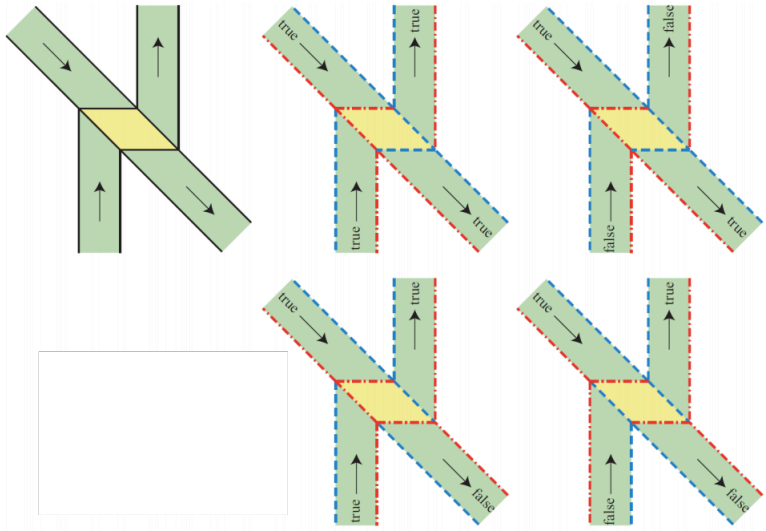


Figure 12: Crossover gadget.

The reduction is from Not-All-Equal-SAT: a triangle of folds represents a clause, and it is foldable if and only if the three incoming folds are not all of the same type. We can represent negations using the splitter gadget and the crossover gadget prevents folds from crossing themselves. For more, see [**?**].

2.10 Vertex-Disjoint Paths

The Vertex-Disjoint Paths problem asks, given a graph G and a set of starting and ending points, are there paths between each pair not sharing any vertices?

Vertex-Disjoint Paths is NP-complete [**?**], and it's very easy to prove: there are horizontal pairs—variable gadgets—and there are exactly two paths between each pair, one true and one false. Clause pairs are vertical, and there are three paths between clause pairs, each one corresponding to a "satisfying" path.

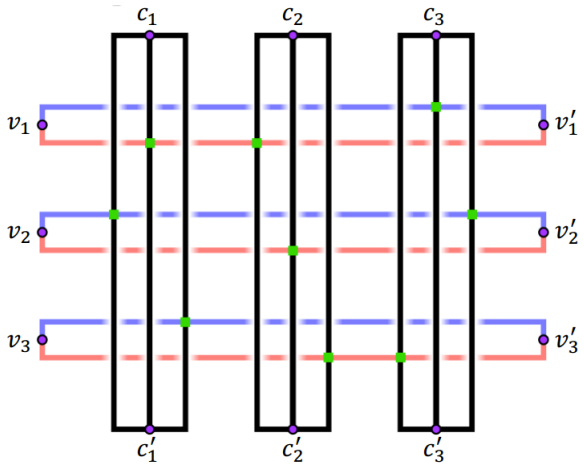


Figure 13: Clauses are vertical and variables horizontal.

This construction can also be carried through even if we require that G be planar by designing crossover gadgets for edges. Crossover gadgets force you to either take one path or the other.

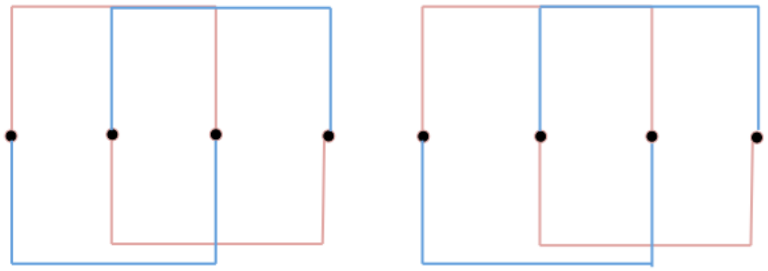


Figure 14: Crossover gadget forces you to take either the red or blue path. Switching paths results in a cross and visiting a vertex more than once.

There is another planar problem, Vertex-Disjoint Paths in a Rectangle. In this problem, we require that every square of our grid be occupied by a path (and that the graph be planar). This problem is NP-complete [**?**].

Erik just completed this proof recently, so he's going to show us some of what can go wrong in a proof like this. The problem was designing the crossover gadget, which is necessary for ensuring the graph's planarity.

2.10.1 What Can Go Wrong

- In order for a region to be fillable, it needs to have an even number of terminals. But Erik's original construction left an odd, unpaired terminal.

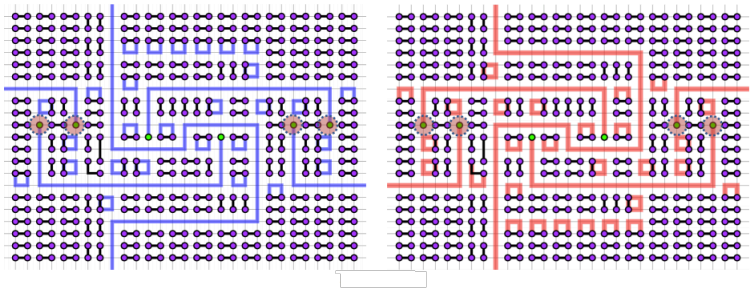


Figure 15: Empty space parity.

- The original gadget was designed by considering the case where a path needed to pass from left to right or right to left. But sometimes a path won't go through a gadget at all, and it still needs to be fillable in that case. This caused another parity problem, which was solved by adding another line of dots.

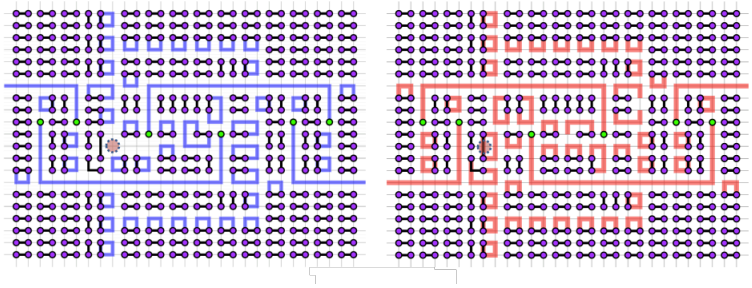


Figure 16: Clause path may be absent.

- Once the gadgets were constructed properly, there was the issue of fitting them together. But the sizes of the variable gadgets and crossover gadgets were inconsistent—another parity problem!

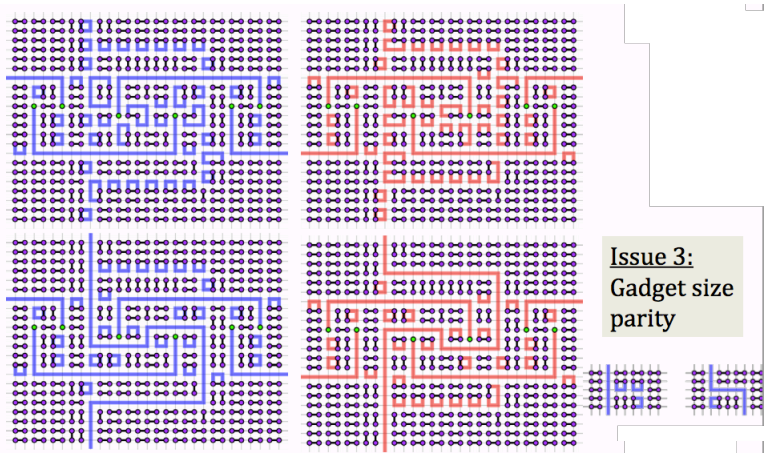


Figure 17: Gadget size parity.

- Once the full crossover versions were completed, it was necessary to create versions which managed to bock edges of a specified type. These were accomplished by adding small terminal pairs to the crossover gadget.

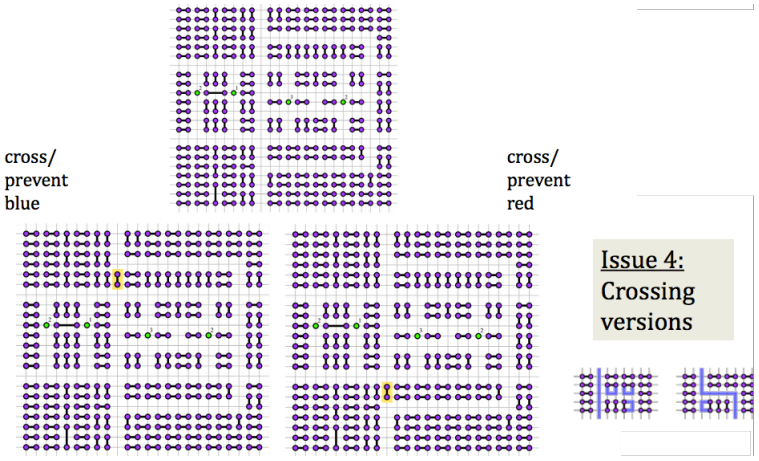


Figure 18: Crossing versions.

2.10.2 Origins

A version of this problem appeared as "The Puzzled Neighbors," created by Sam Loyd for the Brooklyn Daily Eagle in 1897. However, Erik's motivation was the Japanese puzzle Numberlink, by Nikoli, which can be found under the name Flow Free on the Android store. There is some disagreement about the precise rules of the game. The paper [**?**] considers a variant called Zig-Zag Numberlink; another version, "Classic," was considered in 2010 and is also hard [**?**].

References

[ADGV12] Greg Aloupis, Erik D Demaine, Alan Guo, and G Viglietta. Classic Nintendo games are (NP-) hard. *arXiv preprint arXiv:1203.1895*, 2012.

[ADGV14] Greg Aloupis, Erik D Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (computationally) hard. In *Fun with Algorithms*, pages 40–51. Springer, 2014.

[ADO+14] A Adcock, E D Demaine, M P O'Brien, F Reidl, F Sánchez Villaamil, and B D Sullivan. Zig-Zag Numberlink is NP-complete. Under review, 2014.

[BH96] Marshall Bern and Barry Hayes. The complexity of flat origami. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 175–183. Society for Industrial and Applied Mathematics, 1996.

[DDE02] Erik D Demaine, Martin L Demaine, and David Eppstein. Phutball endgames are hard. *More Games of No Chance*, 42:351, 2002.

[Der10] Dariusz Dereniowski. Phutball is PSPACE-hard. *Theor. Comput. Sci.*, 411(44-46):3971–3978, 2010.

[DO07] Erik D Demaine and Joseph O,ÄôRourke. *Geometric folding algorithms*. Cambridge university press Cambridge, 2007.

[Eps87] D Epstein. On the NP-completeness of cryptarithms. *ACM SIGACT News*, 18(3):38–40, 1987.

[FGJ+78] Aviezri S Fraenkel, MR Garey, David S Johnson, T Schaefer, and Yaacov Yesha. The complexity of checkers on an n× n board. In *Foundations of Computer Science, 1978., 19th Annual Symposium on*, pages 55–64. IEEE, 1978.

[KT10] 古妻浩一 and 武永康彦. ナンバーリンクの NP 完全性と問題の列挙. 電子情報通信学会技術研究報告. *COMP,* コンピュテーション, 109(465):1–7, 2010.

[Lyn75] James F Lynch. The equivalence of theorem proving and the interconnection problem. *ACM SIGDA Newsletter*, 5(3):31–36, 1975.