STRINGS-AND-COINS AND NIMSTRING ARE PSPACE-COMPLETE

Erik D. Demaine
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, Massachusetts
edemaine@mit.edu

Yevhenii Diomidov
MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, Massachusetts
diomidov@mit.edu

Received: 1/15/21, Accepted: 7/2/21, Published: 12/20/21

Abstract

We prove that Strings-and-Coins — the combinatorial two-player game generalizing the dual of Dots-and-Boxes — is strongly PSPACE-complete on multigraphs. This result improves the best previous result, NP-hardness, argued in *Winning Ways*. Our result also applies to the Nimstring variant, where the winner is determined by normal play; indeed, one step in our reduction is the standard reduction (also from *Winning Ways*) from Nimstring to Strings-and-Coins.

– In memoriam Elwyn Berlekamp (1940–2019),
John H. Conway (1937–2020),
and Richard K. Guy (1916–2020)

1. Introduction

Elwyn Berlekamp loved Dots and Boxes. He wrote an entire book, *The Dots and Boxes Game: Sophisticated Child's Play* [3] devoted to explaining the mathematical underpinnings of the game, after they were first revealed in Berlekamp, Conway, and Guy's classic book *Winning Ways* exploring many such combinatorial games [2, ch. 16]. At book signings for both books,[1] and after talks he gave about these topics [1], Elwyn routinely played simultaneous exhibitions of Dots and Boxes — him against dozens of players, in the style of Chess masters.

As many children will tell you, Dots-and-Boxes is a simple pencil-and-paper game taking place on an $m \times n$ grid of dots. Two players alternate drawing edges of the

[1]The first author had the honor of playing such a game against Elwyn at a book signing on April 13, 2004, at Quantum Books in Cambridge, Massachusetts. Elwyn won.

grid, with one special rule: when a player completes the fourth edge of one or two 1×1 boxes, that player gains one or two points, respectively, and *must* immediately draw another edge (a "free move" that is often a blessing and a curse). The game ends when all grid edges have been drawn; then the player with the most points wins. (Draws are possible on boards with an even number of squares.)

An equivalent way to think about Dots-and-Boxes is in the dual of the grid graph. Think of each 1×1 square as a dual vertex or *coin* worth one point, "tied down" by four incident *strings* or dual edges. Interior strings connect two coins, while boundary strings connect a coin to the *ground* (not worth any points). (Equivalently, boundary edges have only one endpoint.) Now players alternate cutting (removing) strings, and when a player *frees* one or two coins (removing the last strings attached to them), that player gains the corresponding number of points and must move again. The game ends when all strings have been cut; then the player with the most points wins.

Strings-and-Coins [2, pp. 550–551], [3, ch. 2] is the generalization of this game to arbitrary graphs, where vertices represent coins and edges represent strings which can connect up to two coins (the other endpoints being considered "ground"). *Nimstring* [2, pp. 552–554], [3, ch. 6] is the closely related game where we modify the win condition to *normal play*: the first player unable to move loses. Nimstring is known to be a special case of Strings-and-Coins, a fact we use in our results; see Lemma 1 below.

Related work. Dots-and-Boxes, Strings-and-Coins, and Nimstring are surprisingly intricate games with intricate strategy [2, 3]. On the mathematical side, even $1 \times n$ Dots-and-Boxes is largely unsolved [8, 5].

To formalize this difficulty, *Winning Ways* [2] argued in 1984 that deciding the winner of a Strings-and-Coins position is *NP-hard* by a reduction from vertex-disjoint cycle packing. Around 2000, Eppstein [7] pointed out that this reduction can be adapted to apply to Dots-and-Boxes as well; see [6].

This work left some natural open problems, first explicitly posed in 2001 [6]: are Dots-and-Boxes, Strings-and-Coins, and Nimstring NP-complete or do they extend into a harder complexity class? Being bounded two-player games, all three naturally lie within PSPACE; are they PSPACE-complete?

Results. In this paper, we settle two out of three of these 20-year-old open problems by proving that Strings-and-Coins and Nimstring are PSPACE-complete. This is the first improvement beyond NP-hardness since the original *Winning Ways* result from 1984. Our reductions from Game SAT are relatively simple but subtle. Along the way, we prove PSPACE-completeness of a new Strings-and-Coins variant called *Coins-Are-Lava*, where the first person to free a coin loses.

Our constructed game positions rely on *multigraphs* with multiple copies of some

edges/strings, a feature not present in instances corresponding to Dots-and-Boxes. Thus our results do not apply to Dots-and-Boxes. A generalization of Dots-and-Boxes that we might be able to target is *weighted* Dots-and-Boxes, where each grid edge has a specified number of times it must be drawn before it is "complete" and thus can form the boundary of a 1×1 box. This game corresponds to Strings-and-Coins on planar multigraphs whose vertices can be embedded at grid vertices such that edges have unit length. However, our multigraphs are neither planar nor maximum-degree-4, so they cannot be drawn on a square grid, so our approach does not resolve the complexity of weighted Dots-and-Boxes.

In independent work, Buchin, Hagedoorn, Kostitsyna, and van Mulken [4] proved that (unweighted) Dots-and-Boxes is PSPACE-complete by a reduction from G_{pos}(POS CNF) [9] (roughly the same problem that we reduce from, G_{pos}(POS DNF) [9]). They construct an instance where, after variable setting, one player's winning strategy is to select a maximum set of disjoint cycles. This approach works well for Dots-and-Boxes (and thus Strings-and-Coins) where the goal is to maximize score, but not for Nimstring like our approach does. Thus the two approaches are incomparable.

2. Nimstring

We begin with more formal definitions of the games of interest, and some known lemmas about them:

Definition 1 (Coin–String Multigraph)**.** A *multigraph* G consists of vertices, also called *coins*, and edges, also called *strings*, where each edge $e \in E$ is a set of at most two vertices in V. Notably, we allow edges incident to zero or one vertices in V; we view the missing endpoints of such an edge as being connected to the *ground*.

Definition 2 (Strings-And-Coins(G) and Nimstring(G))**.** Games Strings-And-Coins(G) and Nimstring(G) are played on a multigraph G by two players who alternate removing edges and, if a player *frees* one or two coins by removing their last incident edges, then that player gains the corresponding number of points (one or two) and must move again. The games end when there are no more strings; in Strings-And-Coins(G), the player with the most points wins, while in Nimstring(G), the first player unable to move loses.

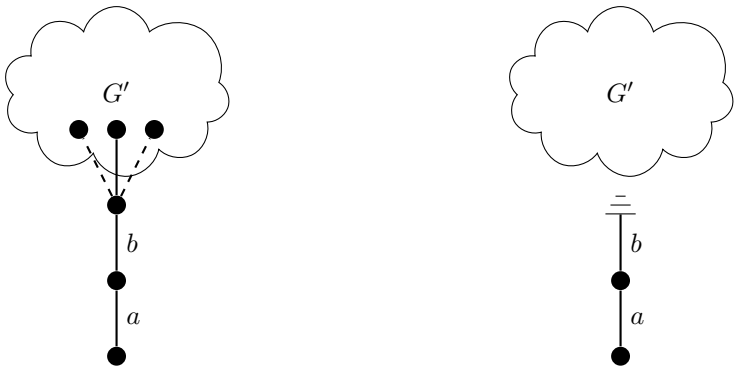
Next we prove the standard result that Nimstring is equivalent to a special case of Strings-and-Coins, and thus hardness of the former implies hardness of the latter:

Lemma 1 ([2, p. 552])**.** *For every graph G, there exists an efficiently computable graph H such that the winner of* Nimstring(G) *is the same as the winner of* Strings-And-Coins(H)*.*

Proof. Let $H = G \cup C_n$ where C_n is a cycle on $n > |V(G)|$ vertices. If a player cuts any string in this cycle, then the opponent can claim $n > \frac{|V(H)|}{2}$ coins in a single turn, winning the game. Therefore the players will try to only cut edges in G, and the player who cannot do so loses. This goal is equivalent to just playing NIMSTRING(G). □

A final known result we will need is about "loony" positions in Nimstring:

Lemma 2 ([2, p. 557])**.** *If G has a degree-2 vertex adjacent to exactly one degree-1 vertex, then the first player can always win in* NIMSTRING(G)*. Such positions are known as loony positions.*



(a) String b connects to a vertex of degree at least 2.

(b) String b connects to the ground.

Figure 1: Two loony positions

Proof. Let a be the string between the two coins, b be the other string connected to a degree-2 coin, and G' be the rest of the graph (Figure 1). One of the players has a winning strategy in NIMSTRING(G').

- If the first player has a winning strategy in NIMSTRING(G'), then we cut strings a and b in this order. We get exactly G' and it is still our turn. By assumption we can win.
- If the second player has a winning strategy in NIMSTRING(G'), then we just cut string b. We get graph G' (plus an extra edge that does not affect the game), and it is our opponent's turn. By assumption opponent cannot win. □

3. Coins-are-Lava

We introduce a variant game played on strings and coins that we find easier to analyze, called *Coins-are-Lava*:[2]

Definition 3 (Coins-Are-Lava(G))**.** Game Coins-Are-Lava(G) is played on a multigraph G by two players who alternate removing edges and, if a player frees a coin, that player loses. Equivalently, players are forbidden from removing an edge that would free a coin, and the winner is determined according to normal play.

Now we show that Coins-are-Lava is a special case of Nimstring. Thus, its hardness will imply the hardness of both Nimstring and (by Lemma 1) Strings-and-Coins.

Lemma 3. *For every graph G, there exists an efficiently computable graph H such that the winner of* Coins-Are-Lava(G) *is the same as the winner of* Nimstring(H)*.*

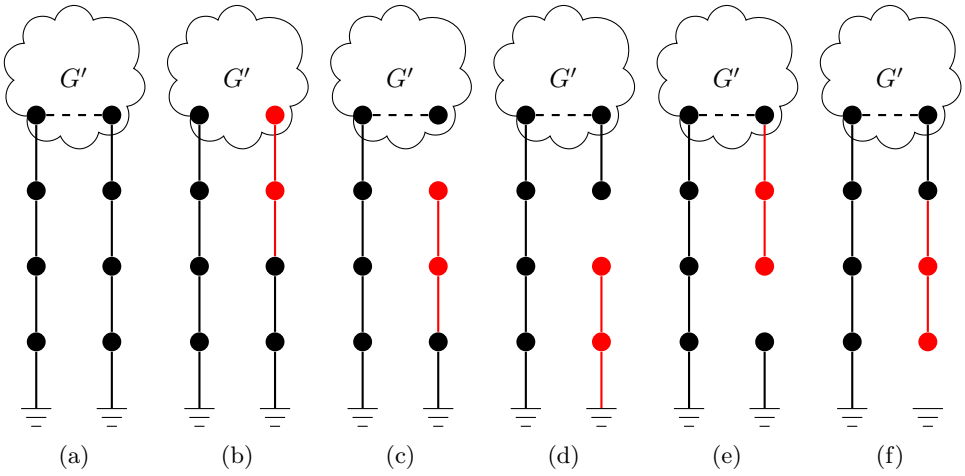


Figure 2: (a) The graph H; (b) freeing a coin in G results in a loony position; (c–f) cutting a string outside G results in a loony position.

Proof. Let H be a graph obtained from G by connecting every coin to the ground with a long chain (length ≥ 5); see Figure 2a.

If a player cuts a string in one of these chains, or cuts all strings in G attached to the same coin, this creates a loony position and ends their turn; see Figure 2. By Lemma 2, their opponent can then win.

[2]For a "practical" motivation for this game, consider the 1933 Double Eagle U.S. coin: until 2002, possession of this coin could result in imprisonment [10].

Therefore the players will try to avoid cutting strings outside G or freeing a coin in G. The first player to fail to do so loses. This goal is equivalent to Coins-Are-Lava(G). □

4. PSPACE-Hardness

It remains to prove that Coins-Are-Lava(G) is PSPACE-complete. Our reduction is from the following known PSPACE-complete problem.

Definition 4 (Game-SAT($\mathcal{F}$)). Given a positive DNF formula $\mathcal{F}$ (an OR of ANDs of variables without negation), Game-SAT($\mathcal{F}$) is the following game played by two players, Trudy and Fallon. Initially each variable is *unset*. In each turn, the player may *set* a variable to TRUE or FALSE, or the player may *skip* their turn (do nothing). The game ends when all variables are set; then Trudy wins if formula $\mathcal{F}$ is true, while Fallon wins if formula $\mathcal{F}$ is false.

We allow players to skip turns and to set variables to the "wrong" value (Trudy to false or Fallon to true). The player with a winning strategy can always avoid such moves, however, replacing them with dominating "good" moves that do not skip and play the "right" value (Trudy to true or Fallon to false), as such moves never hurt the winning player's final goal.

Schaefer [9] proved that this game is PSPACE-complete, under the name G_{pos}(POS DNF).

Theorem 1. *Coins-are-Lava is PSPACE-complete.*

Proof. Let $\mathcal{F}$ be a positive DNF formula with n variables, m clauses, and k_i occurrences of each variable x_i. Without loss of generality, every clause contains at least 2 variables and every variable appears in at least 1 clause. Fix a sufficiently large number $N \gg m^2n^2$.

First we define several useful gadgets, which will be connected together via shared coins (merging the output coin of one gadget with the input coin of another gadget). Many of these gadgets are parametrized by an integer *level*. Intuitively, doing anything to a level-$(\ell+1)$ gadget requires an order of magnitude more time than doing anything to a level-ℓ gadget. This way we can make sure that players interact with gadgets in the right order. However since each level-ℓ gadget uses $N^{\theta(\ell)}$ strings, we can only use a constant number of levels.

A *rope* (Figure 3) is a collection of strings that share both endpoints. The number of strings in a rope is called its *width*. We say that a rope has been *cut* when all of its strings have been cut. When the game ends, every rope has either been cut completely, or it has only 1 string remaining. (Otherwise, a string in the rope can always be safely cut without freeing any coin.)

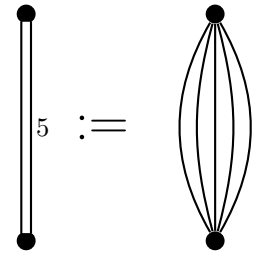
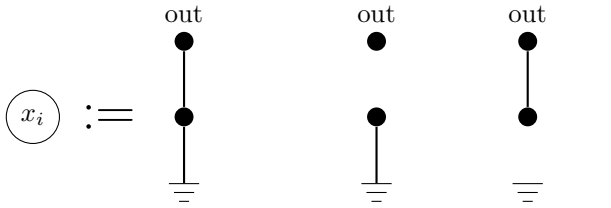


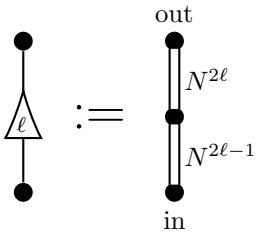
Figure 3: A width-5 rope



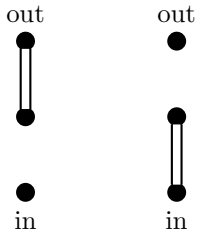
(a) Initial state (unset)

(b) Variable set to true and false respectively

Figure 4: Variable gadget



(a) Initial state



(b) Disabled and activated wires

Figure 5: Wire gadget

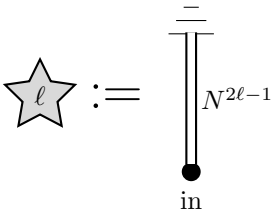


Figure 6: Clause gadget

A *variable* gadget (Figure 4) consists of a chain of two strings, where the bottom string is connected to the ground and the top string is connected to an output coin. We say that is *set to false* if the bottom string has been cut, *set to true* if the top string has been cut, and *unset* if neither string has been cut. A variable implicitly has level 0.

A level-ℓ *wire* gadget (Figure 5) consists of a chain of two ropes, a width-$N^{2\ell-1}$ bottom rope connected to an input coin and a width-$N^{2\ell}$ top rope connected to an output coin. We say that it is *disabled* if the input rope has been cut, *activated* if the top rope has been cut. The *HP* (Hit Points) of the wire is the number of strings remaining in the bottom rope. Note that activating a wire takes a factor of N more moves than disabling it. This means that, if one player is racing to activate a wire and the other is racing to disable it, then the disabler will win the race. Intuitively, the only case where a wire will get activated is if disabling the wire would free a coin.

A level-ℓ *clause* gadget (Figure 6) consists of a single width-$N^{2\ell-1}$ rope connected to an input coin and the ground. We say that it is *disabled* if the rope has been cut. The *HP* of the clause is the number of strings remaining in the rope.

The winner is determined solely by the parity of the number of removed strings.

We can easily flip this parity, for example by adding an extra ground-to-ground string. So without loss of generality, Fallon wins if (but not only if) every variable and wire has one string remaining and all m clauses have no strings. Then Trudy wins if every variable and wire has one string remaining, $m-1$ clauses have no strings, and the final clause has one string. In fact, we will show that the game has to end in one of these two specific ways.

Let $\mathcal{F}'$ be a new formula with the following clauses:

- all clauses from $\mathcal{F}$, which we call *real* clauses;
- for every variable, a *singleton* clause containing just that variable; and
- one additional *empty* clause that contains no variables and is always satisfied.

We construct a multigraph G by connecting the gadgets as follows; refer to Figure 7:

- a variable gadget for each variable;
- a clause gadget for each clause;
- a level-1 wire from each variable x_i to each of k_i real clauses that contain that variable;
- k_i-1 level-1 wires from each variable to the corresponding singleton clause;
- a single vertex called the *root coin*;
- a level-2 wire from the root coin to every real clause and every singleton clause; and
- $n+m-1$ level-2 wires from the root coin to the empty clause.

First we describe how typical gameplay in G should look (without proofs) to give some intuition for why this construction makes sense, and then we prove that it works more formally. Typical gameplay divides into four sequential phases:

1. First Trudy and Fallon set variable gadgets to true and false respectively.
2. Then the players disable all wires from false variables, and disable all but one wire from each true variable (disabling all wires from a true variable would free a coin). Then they activate the level-1 wires that have not been disabled (one from each true variable). Note that almost half the wires from each variable go to singleton clauses. If all real clauses are false, then those wires form the majority, and Fallon can ensure that one of them gets activated. But if even one real clause is true, the wires to true clauses (real or singleton) now form a majority, and Trudy can ensure that one of them gets activated.

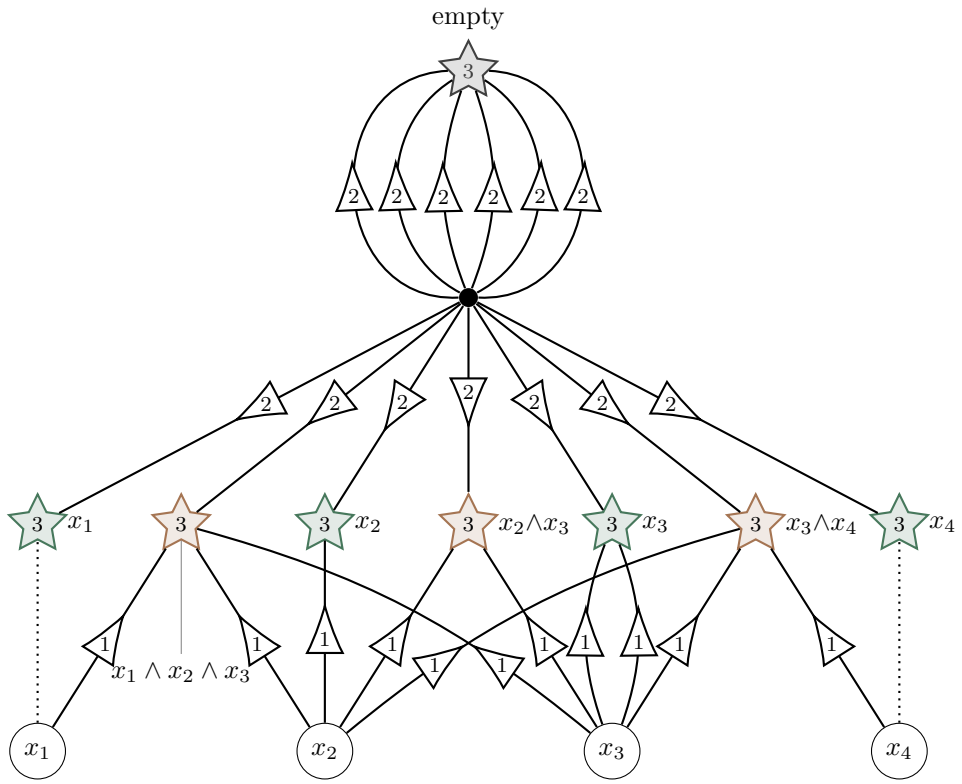


Figure 7: Graph G for formula $(x_1 \wedge x_2 \wedge x_3) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4)$. Clauses are labeled and colored according to whether they are empty ("empty" and gray, at the top), singleton ("x_i" and green), or real ("$x_i \wedge x_j \cdots$" and orange). Dotted lines indicate that there are supposed to be $k_i - 1$ wires there, but $k_i - 1 = 0$.

3. Then the players disable all but one level-2 wire and activate the remaining level-2 wire (disabling all of them would free a coin). Almost half of these wires go to the empty clause. If all real clauses are false, then they form a majority, and Fallon can ensure that one of them gets activated. But if even one real clause is true, then it together with the empty clause forms a majority, and Trudy can ensure that one of them gets activated.

4. Finally, the players disable the clause gadgets. A clause can be disabled unless all wires pointing at it got activated (in that case, disabling it would free a coin). If the formula is not satisfied, then all clauses get disabled and Fallon wins. If the formula is satisfied, then exactly one clause remains and Trudy wins.

We want to show that the winner of Coins-Are-Lava(G) is the same as the winner of Game-SAT($\mathcal{F}$). We do a case split on the winner of Game-SAT($\mathcal{F}$), and in each case provide a winning Coins-Are-Lava(G) strategy for that player.

If Fallon can win Game-SAT($\mathcal{F}$), then they can win Coins-Are-Lava(G) using the following strategy (where numbers match the phases of intended gameplay above):

1. There is a natural mapping f from states of Coins-Are-Lava(G) to states of Game-SAT($\mathcal{F}$): a variable x_i in Game-SAT($\mathcal{F}$) is set to true if the corresponding variable gadget is set to true, set to false if the gadget is set to false, and unset if the gadget is unset. Every move in Coins-Are-Lava(G) maps to a valid move in Game-SAT($\mathcal{F}$), where moves outside of variable gadgets map to skip moves. Also, if we played Coins-Are-Lava(G) for less than $2n$ moves, then we can perform any move that is valid in the corresponding Game-SAT($\mathcal{F}$) state. This does not free a coin, because the relevant coin has degree $\Omega(N) \gg 2n$. So we can transfer the strategy from Game-SAT($\mathcal{F}$) to Coins-Are-Lava(G): for every opponent's move in Coins-Are-Lava(G), map it to Game-SAT($\mathcal{F}$), find the best response, and map it back to Coins-Are-Lava(G). We remain in this phase until we have set all variable gadgets to some assignment that does not satisfy $\mathcal{F}$, as guaranteed by the winning strategy in Game-SAT($\mathcal{F}$).

2. Call a level-1 wire from a true variable x_i *good* if it points at a real clause and *bad* if it points at a singleton clause. Wires from false variables are *neutral*. Each true variable x_i has k_i good wires and $k_i - 1$ bad ones. For each true variable x_i, the total HP of bad wires is still at most $(k_i - 1)N^1$ and total HP of all good wires is at least $k_iN^1 - O(n) > (k_i - 1)N^1$ (the opponent could cut up to $O(n)$ strings here while we were setting variables).

   (a) Disable all bad wires, Specifically, if the opponent reduced HP of a good wire connected to some true variable x_i, we respond by reducing HP of a bad wire connected to the same x_i; if the opponent did something else or x_i has no bad wires left, we reduce HP of a bad wire connected an arbitrary variable x_j. This maintains the invariant that for each true variable x_i, HP of x_i's good wires is higher than HP of x_i's bad wires. The opponent cannot activate any bad wires because that would take $\Theta(N^2) \gg \sum_i k_iN^1$ moves.

   (b) Disable good and neutral level-1 wires until there is only one good wire remaining per true variable. Once again, the opponent cannot activate these wires because that would take too many moves.

   (c) Activate the remaining good wires. Opponent cannot disable these wires, because that would free a coin.

(d) We have activated exactly one good wire per true variable. There are no activated level-1 wires pointing at satisfied clauses, because real clauses are unsatisfied and singleton clauses are bad.

3. Call a level-2 wire *good* if it points to a real or singleton clause and *bad* if it points to the empty clause. The total HP of the $n + m - 1$ bad wires is at most $(n + m - 1)N^3$ and the total HP of the $n + m$ good wires is still (after $O(nmN^2)$ moves spent in the first two stages) at least $(n + m)N^3 - O(nmN^2) > (n + m - 1)N^3$.

   (a) Disable all bad wires. The opponent cannot disable all good wires before we disable the bad ones because good wires have more HP.

   (b) Disable all but one good wire. Because all disabling and activating steps done so far are for wires of HP $\Theta(N^3)$, and activating a level-2 wire requires $\Theta(N^4)$ moves, the opponent cannot afford to activate any of these good wires before we disable them.

   (c) Activate the last good wire. The opponent cannot disable it because that would free the root coin.

4. Disable all clause gadgets. This will not free a coin, because every clause has at least one disabled wire: real clauses are unsatisfied so there is a false variable whose adjacent wire we disabled in Step 2(b); singleton clauses have bad level-1 wires that we disabled in Step 2(a); and the empty clause has a bad level-2 wire that we disabled in Step 3(a). We win because there are no clause gadgets remaining.

If Trudy can win Game-SAT($\mathcal{F}$), then they can win Coins-Are-Lava(G) using the following strategy (where numbers match the phases of intended gameplay above):

1. Set the variable gadgets to some assignment that satisfies $\mathcal{F}$. Let $C \in \mathcal{F}$ be a satisfied clause.

2. Call a level-1 wire from a true variable $x_i \in C$ *good* if it points at a singleton clause or C and *bad* otherwise. Wires from variables not in C are *neutral*. Each variable $x_i \in C$ has $k_i - 1$ bad wires (k_i to real clauses, but one of them is C) and k_i good ones ($k_i - 1$ to the singleton clause and one to C). Disable all bad wires, then disable all neutral wires and all-but-one good wire per variable in C, and then activate the remaining good wires. Each activated wire points to a singleton clause or to C. Then either all of them point to C, or at least one of them points to a singleton clause. Either way, we have some satisfied real or singleton clause C' with only activated level-1 wires.

3. Call a level-2 wire *good* if it points to C' or to the empty clause. There are $n+m-1$ bad wires ($n+m$ to real clauses, but one of them is C') and $n+m$ bad ones ($n+m-1$ to the empty clause plus one to C'). Disable all bad wires and activate exactly one good wire. Let C'' be the clause pointed by the activated wire (either C' or the empty clause).

4. Disable all clause gadgets other than C''. This will not free a coin, because every clause other than C'' has a disabled level-2 wire. But C'' cannot be disabled, because all of the wires pointing at it have been activated. We win because there is exactly one clause gadget remaining. □

Corollary 1. *Nimstring is PSPACE-complete.*

Proof. This follows immediately from Theorem 1 and Lemma 3. □

Corollary 2. *Strings-and-Coins is PSPACE-complete.*

Proof. This follows immediately from Corollary 1 and Lemma 1. □

5. Open Problems

We have proved PSPACE-completeness of Strings-and-Coins and Nimstring on multigraphs, while Buchin et al. [4] proved PSPACE-completeness of Dots-and-Boxes, and thus Strings-and-Coins on grid graphs. The main open problem is whether Dots-and-Boxes with normal play instead of scoring, i.e., Nimstring on grid graphs, is also PSPACE-complete. Toward this goal, we could also aim to prove PSPACE-completeness of Nimstring on simple graphs (with only one copy of each edge/string) or planar graphs.

Acknowledgements. This work was initiated during an MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.892, Spring 2019). We thank the other participants of the class for providing an inspiring research environment.

References

[1] American Mathematical Society, Elwyn Berlekamp Gives Arnold Ross Lecture, http://www.ams.org/programs/students/arl2004, 2003.

[2] E. Berlekamp, J. Conway, and R. Guy, *Winning Ways for Your Mathematical Plays, Volume 3*, A K Peters, Wellesley MA, 2003.

[3] E. Berlekamp, *The Dots and Boxes Game: Sophisticated Child's Play*, A K Peters, Massachusetts, 2000.

[4] K. Buchin, M. Hagedoorn, I. Kostitsyna, and M. van Mulken, Dots & boxes is PSPACE-complete, https://arXiv.org/abs/2105.02837, 2021.

[5] S. Collette, E. Demaine, M. Demaine, and S. Langerman, Narrow misère dots-and-boxes, *Games of No Chance 4*, Cambridge University Press, 2015.

[6] E. Demaine and R. Hearn, Playing games with algorithms: algorithmic combinatorial game theory, *Games of No Chance 3*, Cambridge University Press, 2009.

[7] D. Eppstein, Computational Complexity of Games and Puzzles, http://www.ics.uci.edu/~eppstein/cgt/hard.html.

[8] R. Guy and R. Nowakowski, Unsolved problems in combinatorial games, in *More Games of No Chance*, Cambridge University Press, 2002.

[9] T. Schaefer, On the complexity of some two-person perfect-information games, *J. Comput. System Sci.* **16**(1978), 185–225.

[10] United States Mint, The United States government to sell the famed 1933 double eagle, the most valuable gold coin in the world, https://www.usmint.gov/news/press-releases/20020207-the-united-states-government-to-sell-the-famed-1933-double-eagle-the-most-valuable-gold-coin-in-the-world, 2002.