This Game Is Not Going To Analyze Itself

Aviv Adler, Hayashi Layers, Lily Chung, Michael Coulombe, Erik D. Demaine, Jenny Diomidova, Della Hendrickson, and Jayson Lynch

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
{adlera,hayastl,lkdc,mcoulomb,edemaine,diomidova,della,jaysonl}@mit.edu

Abstract. We analyze the puzzle video game *This Game Is Not Going To Load Itself*, where the player routes data packets of three different colors from given sources to given sinks of the correct color. Given the sources, sinks, and some previously placed arrow tiles, we prove that the game is in Σ_2^{P}; in NP for sources of equal period; and NP-complete for three colors and six equal-period sources with player input. Without player input, we prove that just simulating the game is in Δ_2^{P}, and both NP- and coNP-hard for two colors and many sources with different periods. On the other hand, we characterize which locations for three data sinks admit a *perfect* placement of arrow tiles that guarantee correct routing no matter the placement of the data sources, effectively solving most instances of the game as it is normally played.

1 Introduction

This Game Is Not Going To Load Itself (TGINGTLI) [6] is a free game created in 2015 by Roger "atiaxi" Ostrander for the Loading Screen Jam, a game jam hosted on itch.io, where it finished 7th overall out of 46 entries. This game jam was a celebration of the expiration of US Patent 5,718,632 [2], which covered the act of including mini-games during video game loading screens. In this spirit, TGINGTLI is a real-time puzzle game themed around the player helping a game load three different resources of itself — save data, gameplay, and music, colored red, green, and blue — by placing arrows on the grid cells to route data entering the grid to a corresponding sink cell. Figure 1 shows an example play-through.

We formalize TGINGTLI as follows. You are given an $m \times n$ grid where each unit-square cell is either empty, contains a data sink, or contains an arrow pointing in one of the four cardinal directions. (In the implemented game, $m = n = 12$ and no arrows are placed initially.) Each data sink and arrow has a color (resource) of red, green, or blue; and there is exactly one data sink of each color in the grid. In the online version (as implemented), sources appear throughout the game; in the offline version considered here, all sources are known a priori. Note that an outer edge of the grid may have multiple sources of different colors. Finally, there is a loading bar that starts at an integer k_0 and has a goal integer k^*.

During the game, each source periodically produces data packets of its color, which travel at a constant speed into the grid. If a packet enters the cell of an

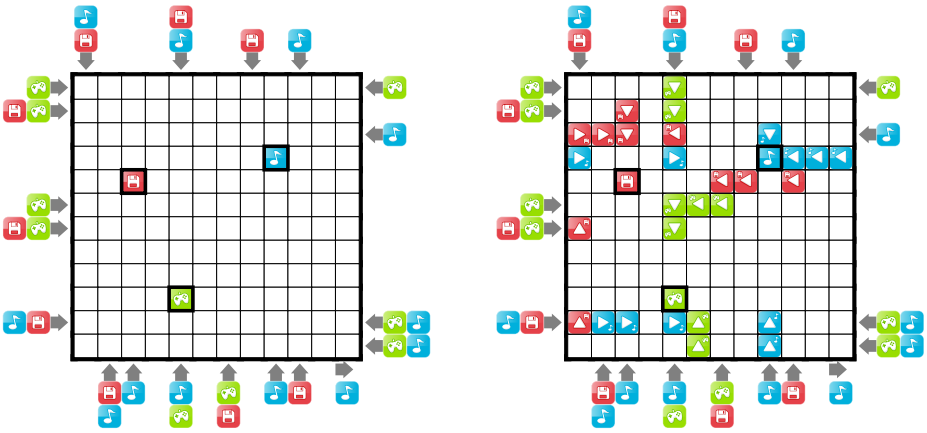


Fig. 1: Left: The (eventual) input for a real-world Level 16 in TGINGTLI. Right: A successful human play-through that routes every packet to its corresponding sink.

arrow of the same color, then the packet will turn in that direction. (Arrows of other colors are ignored, as are other packets.) If a packet reaches the sink of its color, then the packet disappears and the loading bar increases by one unit of data. If a packet reaches a sink of the wrong color, or exits the grid entirely, then the packet disappears and the loading bar decreases by one unit of data, referred to as taking damage. Packets may also remain in the grid indefinitely by going around a cycle of arrows; this does not increase or decrease the loading bar. The player may at any time permanently fill an empty cell with an arrow, which may be of any color and pointing in any of the four directions. If the loading bar hits the target amount k^*, then the player wins; but if the loading bar goes below zero, then the player loses.

In Section 2, we prove NP-hardness of the TGINGTLI decision problem: given a description of the grid (including sources, sinks, and preplaced arrows), is there a placement of arrows that causes the player to win? This reduction works even for just six equal-period sources and three colors; it introduces a new problem, ***3DSAT***, where variables have three different colors and each clause mixes variables of all three colors. In Section 3, we introduce more detailed models for the periodic behavior of sources, and show that many sources of differing periods enable both NP- and coNP-hardness of winning the game, *even without player input* (just simulating the game). On the positive side, we prove that the game is in Σ_2^{P} with player input, and in Δ_2^{P} without player input. We also prove membership in NP when the source periods are all equal, as in our first NP-hardness proof, so this case is in fact NP-complete.

In Section 4, we consider how levels start in the implemented game: a grid with placed sinks but no preplaced arrows. We give a full characterization of when there is a ***perfect layout*** of arrows, where all packets are guaranteed to route to the correct sink, *no matter where sources get placed*. In particular, this result provides a winning strategy for most sink arrangements in the imple-

mented game. Notably, because this solution works independent of the sources, it works in the online setting.

The theme of TGINGTLI — routing from source(s) to the correct sinks — is conceptually related to other games and problems in theoretical computer science. The road coloring theorem [7] characterizes which directed graphs have a coloring of edges that the resulting deterministic finite automaton has a "synchronizing" word which brings any start state to a common state. The Nikoli pencil-and-paper puzzle *Roma* asks the player to complete a square grid with arrows such that every square routes to a given sink square, similar to a one-color version of TGINGTLI, but with additional constraints on arrow placement (given regions that cannot contain arrows in the same direction); this problem is NP-complete [1].

2 NP-Hardness for Three Colors and Six Sources

We first prove that TGINGTLI is NP-hard in the case where there are just six sources of equal period T. That is, every T seconds all six sources simultaneously emit a single packet.

We prove NP-hardness via reduction from a new problem called ***3-Dimensional SAT (3DSAT)***, defined by analogy to 3-Dimensional Matching (3DM). 3DSAT is a variation of 3SAT where, in addition to a 3CNF formula, the input specifies one of three colors (red, green, or blue) to each variable of the CNF formula, and the CNF formula is constrained to have trichromatic clauses, i.e., to have exactly one variable (possibly negated) of each color.

Lemma 1. *3DSAT is NP-complete and ASP-complete.*

Proof. We reduce from 3SAT to 3DSAT by converting a 3CNF formula ϕ into a 3D CNF formula ϕ'. For each variable x of ϕ, we create three variables $x^{(1)}, x^{(2)}, x^{(3)}$ in ϕ' (intended to be equal copies of x of the three different colors) and add six clauses to ϕ' to force $x^{(1)} = x^{(2)} = x^{(3)}$:

$$\left.\begin{aligned} \overline{x^{(1)}} \vee x^{(2)} \vee x^{(3)} &\iff (x^{(1)} \to x^{(2)}) \vee x^{(3)} \\ \overline{x^{(1)}} \vee x^{(2)} \vee \overline{x^{(3)}} &\iff (x^{(1)} \to x^{(2)}) \vee \overline{x^{(3)}} \end{aligned}\right\} \iff x^{(1)} \to x^{(2)}$$

$$\left.\begin{aligned} x^{(1)} \vee \overline{x^{(2)}} \vee x^{(3)} &\iff (x^{(2)} \to x^{(3)}) \vee x^{(1)} \\ \overline{x^{(1)}} \vee \overline{x^{(2)}} \vee x^{(3)} &\iff (x^{(2)} \to x^{(3)}) \vee \overline{x^{(1)}} \end{aligned}\right\} \iff x^{(2)} \to x^{(3)}$$

$$\left.\begin{aligned} x^{(1)} \vee x^{(2)} \vee \overline{x^{(3)}} &\iff (x^{(3)} \to x^{(1)}) \vee x^{(2)} \\ x^{(1)} \vee \overline{x^{(2)}} \vee \overline{x^{(3)}} &\iff (x^{(3)} \to x^{(1)}) \vee \overline{x^{(2)}} \end{aligned}\right\} \iff x^{(3)} \to x^{(1)}$$

Thus the clauses on the left are equivalent to the implication loop $x^{(1)} \implies x^{(2)} \implies x^{(3)} \implies x^{(1)}$, which is equivalent to $x^{(1)} = x^{(2)} = x^{(3)}$.

For each clause c of ϕ using variables x in the first literal, y in the second literal, and z in the third literal, we create a corresponding clause c' in ϕ' using

$x^{(1)}$, $y^{(2)}$, and $z^{(3)}$ (with the same negations as in c). All clauses in ϕ' (including the variable duplication clauses above) thus use a variable of the form $x^{(i)}$ in the ith literal for $i \in \{1, 2, 3\}$, so we can 3-color the variables accordingly.

This reduction is parsimonious: because we force $x_i^{(1)} = x_i^{(2)} = x_i^{(3)}$, any solution to ϕ' is just three identical copies of a solution to ϕ. Thus we also obtain an efficiently computable bijection between solutions to ϕ and ϕ'. Because 3SAT is ASP-complete [8], so is 3DSAT.

Theorem 1. *TGINGTLI is NP-hard, even with three colors and six sources of equal period T.*

Proof. Our reduction is from 3DSAT. Figure 2 gives a high-level sketch: variables of the same color are connected in a chain, with a source of the same color on the left and a corresponding sink on the right, which splits and merges for each variable. The split for variable x_i allows the player to choose where to route the packet stream of that color, into at most one of two possible literal paths x_i and $\overline{x}_i$. The literal path taken by the packet stream is viewed as *false*; an empty literal path is viewed as *true*. Literal paths pass through a clause gadget for each clause containing that literal.

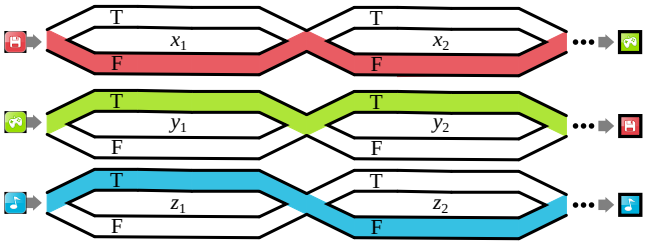
Each clause gadget allows at most two packet streams to successfully pass through its three literal paths of three different colors, corresponding to the constraint that at most two literals are false. If the player attempts to pass three packet streams (three false literals) through the same clause gadget, one of those streams becomes trapped in a cycle; the reduction is designed so that this results in a loss. Thus a clause containing paths labeled $\{\overline{x_i}, \overline{y_j}, z_k\}$ corresponds to the 3DSAT clause $x_i \vee y_j \vee \overline{z_k}$.

If all three packet streams reach their sinks, then all clauses must have been satisfied by the chosen literals, corresponding to a satisfying assignment, and the loading bar increases by 3. Conversely, if any packet stream gets stuck in a loop (e.g., because a clause was unsatisfied and blocked it), then the loading bar will increase by at most 2. We will arrange for a loss in this case.

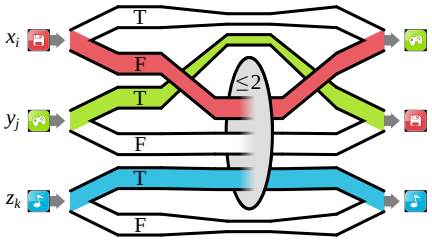
Now we describe the reduction in detail; refer to Figure 8 for a full example. Most cells of the game board will be prefilled, leaving only a few empty cells (denoted by question marks in our figures) that the player can fill. Any blank space in figures is filled arbitrary, say with a downward red arrow.

For each color, say red, we place a red source on the left edge of the construction and a corresponding red sink on the far right. Then, for each red variable x_i in sequence, we place a variable gadget of Figure 3. To prevent the packets from entering a loop, the player must choose between sending the stream upward or downward by placing a red arrow, which results in it following one of the two rightward literal paths, representing the literals x_i and $\overline{x}_i$ respectively. Recall that a literal whose path is followed by the packet stream is viewed as false. Then we recombine these two literal paths with the merge gadget of Figure 4, before splitting for the next red variable, or finally routing to the red sink.

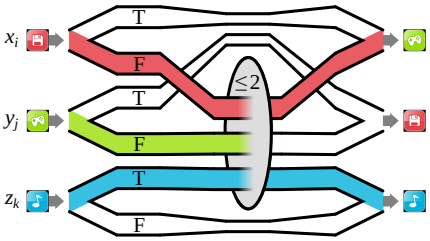
Next we route each literal path to sequentially visit every clause containing it. Figure 6 shows a crossover gadget to enable such routing, which works regardless



(a) Variables in three chains (red, green, blue), each with its own color source on left and sink on right



(b) Satisfied clause



(c) Unsatisfied clause

Fig. 2: Sketch of our NP-hardness reduction. The clause gadget in (b–c) corresponds to the 3DSAT clause $x_i \vee y_j \vee \overline{z_k}$.

of the colors of the paths; in particular, the central cell has a different color to allow for the case of two crossing paths of the same color.

Figure 5 shows the clause gadget, which has just two empty cells (question marks). If neither of these cells has an upward red arrow, then any red packets entering on the left will cycle, going right, left, or down and then bouncing back (given the red arrow placements in the gadget). By symmetry, the same holds for all three colors. Thus any incoming packet stream can reach its exit on the right only if the player places an upward arrow of that color. Given that only two such arrows can be placed, at most two packet streams can pass through from entrance on the left to exit on the right. Therefore, if all three literals are false, then at least one stream of data must enter a cycle. On the other hand, if the clause is satisfied (at least one literal is true), then the (at most two) literal paths carrying data can pass their data to the clause's exits (i.e., the next clause containing the corresponding variable).

After routing the literal paths for x_i and $\overline{x}_i$ to visit all clauses containing those literals (as illustrated in Figure 8), we lengthen these two paths to have the same length. This lengthening is not illustrated in Figure 8, but is easy to do as follows. The two literal paths for a variable connect the same two squares on the grid, so their length has the same parity. We can repeatedly lengthen any shorter path by 2 by replacing two consecutive arrows with a diversion around a 2×2 square.

Similarly, we lengthen the red, green, and blue streams to all have the same length ℓ. To achieve this property, we enforce that all three stream lengths have

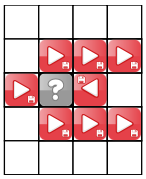


Fig. 3: Variable gadget lets the player route packets from a source to one of two literal paths.

Fig. 4: Merge gadget combines two literal paths back into a single stream.

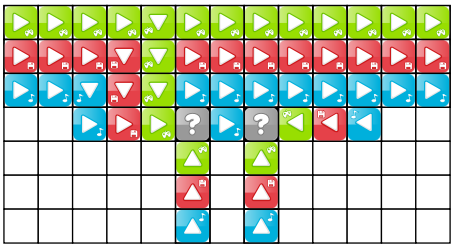


Fig. 5: Clause gadget. At most two streams of data, representing false literals, can pass through the gadget (by placing upwards arrows in the "?" cells) without entering a cycle. Placing any other direction of arrow also puts a stream into a cycle.

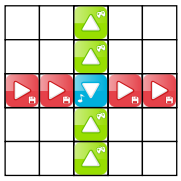


Fig. 6: Crossover gadget between two literal paths of same or different colors. (The center cell is colored different from both paths.)

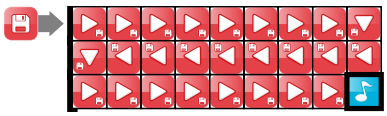


Fig. 7: Damage gadget forces damage at a unit rate after a desired start delay.

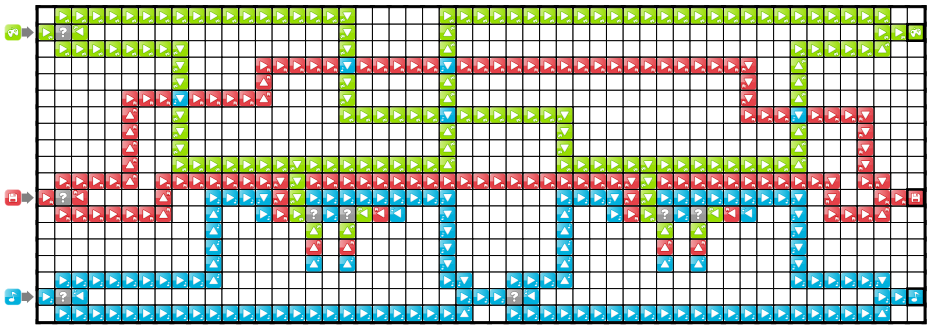


Fig. 8: An example of four variables x_1, y_1, z_1, z_2 and two clauses forming the 3DSAT formula $(x_1 \vee y_1 \vee \overline{z_1}) \wedge (\overline{x_1} \vee y_1 \vee \overline{z_2})$. Path length adjustments and damage gadgets are not depicted here.

the same parity, by placing all sources on the left edge to emit packets simultaneously into a common (leftmost) column, placing all sinks in a common (rightmost) column, and placing equally colored sources and sinks in a common row, as in Figure 8. Then we can lengthen the end of any shorter stream by a multi-

ple of 2 (in between splits into literal paths so they remain balanced) to reach a common length ℓ. These diversions may require additional rows or columns compared to Figure 8.

If the player successfully satisfies all clauses, then they route all three colors from source to sink, so the loading bar will increase by 3 units (1 per color) every t seconds after an initial delay of ℓ. We set the target and initial loading scores to satisfy $k^* - k_0 = 3$ so that the player wins in this case. If at least one color gets stuck in a loop before reaching its sink, then the loading bar increases by at most 2 units every t seconds after a delay of ℓ. To ensure that the player loses in this case, we add three copies of the damage gadget in Figure 7, adjusted in length to incur a total of 3 damage every t seconds after an initial delay of $\ell + 1$. Thus in total the loading bar decreases by 1 unit every t seconds, so the player eventually loses even if k_0 is large. Our reduction thus allows any $k_0 \geq 0$ and $k^* = k_0 + 3$. For the sake of fully specifying a reduction, we can set $k_0 = 0$ and $k^* = 3$.

Finally, we prove correctness of the reduction.

Let ϕ be a 3D CNF formula with satisfying assignment $(x_1, \ldots, x_n, y_1, \ldots, y_m, z_1, \ldots, z_k)$; we will show that there exists a solution to the TGINGTLI puzzle produced from ϕ by our reduction. First, place up or down arrows into each variable gadget to match the Boolean setting of the corresponding variable, so that the packets flow to the path of the false literal. Next, for each clause of ϕ, if it has a false literal x_i or $\overline{x_i}$, we place a red up arrow into one empty cell of the corresponding clause gadget. Similarly, for a false literal y_j or $\overline{y_j}$, place a green up arrow, and for a false literal z_k or $\overline{z_k}$, place a blue up arrow; since there are two empty cells in the gadget and at most two false literals in each clause, this is always possible.

With this arrow placement, the three sources send packets which enter a variable gadget of the given color, branching onto a path representing either the negated or unnegated literal through every clause gadget that the literal appears in, then merging into the next variable gadget of that color, repeating until the last variable. The arrow placements in the clause gadgets ensure that, if packets enter along a path corresponding to a false literal, then the packets will pass through the clause. The arrow placements in the variable gadgets ensure that, if x_i is true in the assignment, then any packets that enter are routed to the path for $\overline{x_i}$, and vice versa if false. By induction, we see that both types of placements ensure that, during the course of the game execution, packets will (1) reach every variable gadget, (2) traverse every false literal path, (3) pass through each clause gadget along the false literal paths without getting stuck, and thus (4) reach the end of the length-ℓ streams and enter the correctly colored sinks. As detailed above, the parameters are set such that when no packets get stuck, the player will win, and therefore this placement is a solution.

Conversely, suppose ϕ is a 3D CNF formula for which the TGINGTLI puzzle produced from ϕ by our reduction has an arrow placement layout that wins the game; we will show that ϕ must be satisfiable. Because there are only three sources in the damage gadgets (whose packets can never reach a matching sink)

and three sources pointing at the first variable gadgets of each color, in order for this layout to win the game, there must be a path for packets to traverse from the sources to the matching sink. Those packets must be of all three colors to offset the damage gadgets effect on the loading bar, as described above. By construction, these paths pass through every variable gadget of the same color, meaning that there must be a player-placed arrow pointing up or down in each.

From this analysis, we can derive an assignment to ϕ from the arrows placed in each variable gadget: set x_i to true if the arrow in the corresponding gadget routes packets into the path for the negated literal $\overline{x_i}$, otherwise set it to false, and similarly for each y_j and z_k. According to this assignment, we see that during the game execution each clause gadget will receive packets along each false literal path and no true literal paths. Because the packets must pass through the clause gadget to reach the end, there must be player-placed up arrows in the empty cells of each color of packet that enters along the false literal paths. Because there are only two empty cells in a clause gadget, we conclude that at most two of the literals in the corresponding clause of ϕ may be false. Thus one literal must be true in every clause of ϕ, meaning that ϕ is satisfied by this assignment.

If we allow more than six sources, we can duplicate some of the gadgets in the reduction to adjust how much the loading bar increases and decreases, which lets us set k_0 and k^* to any desired values with $0 \leq k_0 < k^*$. For example, we can match TGINGTLI as implemented: in level $i \geq 1$, $k_0 = \max\{i, 3\}$ and $k^* = 5 + 5i$.

We also consider the parsimony of the above reduction. The only actions a player can take in the TGINGTLI puzzle produced from a 3DSAT formula ϕ are to place arrows into the designated empty cells (marked with question marks in our figures) of the variable and clause gadgets. Given a satisfying assignment to ϕ, a solution to the TGINGTLI puzzle must place up or down arrows into each variable gadget to match the boolean setting of the corresponding variable. However, in a clause gadget, when other placements direct a packet stream of a given color into a clause gadget, there must be a placement of an up arrow into one of its two empty cells. Since the variable values satisfy ϕ, packets of at most two colors will pass into a clause so this is always possible, but there are multiple ways to accomplish this: if zero packets ever enter a clause, then each empty cell could be left empty or arbitrarilly filled with one of three colored arrows pointing in one of four directions, thus 13^2 possibilities; if only one color of packets pass through, then its up arrow can be in one of two locations and the other empty cell again has 13 possibilities; if two colors of packets pass through, then there only remains one binary choice of which empty cell gets which colored arrow. Thus we see that each solution to ϕ with c clauses maps to between 2^c and 13^{2c} possible layouts that will solve the corresponding TGINGTLI puzzle, so the above reduction is not parsimonious.

3 Membership in Δ_2^P/Σ_2^P and Hardness from Source Periodicity

In this section, we consider the effect of allowing each source to have a different period. We show that carefully setting periods together with the unbounded length of the game results in both NP- and coNP-hardness of determining the outcome of TGINGTLI, *even when the player is not making moves*. Conversely, we prove that the problem is in Δ_2^P when forbidding player input and in Σ_2^P when allowing player input.

3.1 Model and Problems

More precisely, we model each source s as emitting data packets of its color into the grid with its own period p_s, after a small warmup time w_s during which the source may emit a more specific pattern of packets. More generally, we can allow an arbitrary pattern of emitted packets during the period; we will allow this for our upper bounds, but use the simpler model of "one packet every p_s time units" for our lower bounds. We assume that periods and warmup times are integers encoded in unary, so that the entire behavior (at which integer times packets get emitted) during the warmup and periodic phases can be explicitly encoded. Equivalently, the emitting pattern of a source can be defined by a tally DFA (deterministic finite automaton over a unary alphabet [3]) where accepting states correspond to emitted packets.

TGINGTLI as implemented has a warmup behavior of each source initially (upon creation) waiting 5 seconds before the first emitted packet, then emitting a packet after 2 seconds, after 1.999 seconds, after 1.998 seconds, and so on, until reaching a fixed period of 0.5 seconds. This is technically a warmup period of 1881.25 seconds with 1500 emitted packets, followed by a period of 0.5 seconds.

In the ***simulation*** problem, we are given the initial state of the grid, a list of timestamped ***events*** for when each source emits a packet during its warmup period, when each source starts periodic behavior, when each source emits a packet during its periodic phase, and when and where the player places each arrow. We assume that event timestamps are integers encoded in binary. The problem then asks to predict whether the player wins (the loading bar reaches k^*) before a loss (the loading bar goes below zero).

In the ***game*** problem, we are given the same list of events related to sources, but are not given the player's arrow placements. Instead, the problem asks whether there exist timestamped arrow placements such that the player wins before a loss. If we allow nondeterministic algorithms, the game problem reduces to the simulation problem: just guess what arrows we place, where, and at what times.

A natural approach to solving the simulation problem is to simulate the game from the initial state to each successive event. Specifically, given a state of the game (a grid with sinks, sources of differing periods and offsets, placed arrows, and the number of in-flight packets at each location) and a future timestamp t,

we wish to determine the state of the game at time t. Using this computation, we can compute future states of the game quickly by "skipping ahead" over the time between events. On an $m \times n$ grid, there are $O(mn)$ events, so we can determine the state of the game at any time t by simulating polynomially many intervals between events.

This computation is easy to do. Given the time t, we can divide by each source's period and the period of each cycle of arrows to determine how many packets each source produces and where the arrows route them — either to a sink which affects loading, off the grid, stuck in a cycle, or in-flight outside a cycle — and then sum up the effects to obtain the new amount loaded and the number of packets at each location.

However, being able to compute future states does not suffice to solve the simulation and game problems because there might be an intermediate time where the loading amount drops below 0 or reaches the target amount k^*. Nonetheless, this suffices to show that the problems are in appropriate complexity classes in the polynomial hierarchy, by binary searching to find the earliest win time and verifying that there are no earlier loss times:

Lemma 2. *The simulation problem is in Δ_2^{P}, and the game problem is in Σ_2^{P}.*

Proof. To prove membership in $\Delta_2^{\mathrm{P}} = \mathrm{P}^{\mathrm{NP}}$, we give an algorithm that makes polynomially many queries to an NP oracle.

We will use the NP oracle to solve the following problems, which we claim are in NP.

- "**Win in $[t_1, t_2]$?**" Does the loading bar reach t^* at some time $t \in [t_1, t_2]$?
- "**Lose in $[t_1, t_2]$?**" Does the loading bar go below 0 at some time $t \in [t_1, t_2]$?

In both cases, an NP algorithm works as follows. Guess the time $t \in [t_1, t_2]$, which requires guessing as many bits as are in t_1 and t_2. Then compute the state of the game at time t in polynomial time, using the algorithm described above to quickly compute states at individual timestamps. Finally, accept if the loading bar has the desired value and reject otherwise.

Next we argue that, if the player ever wins, they do so within an exponential amount of time. After time $\max_s w_s$, all sources are periodic. All given event timestamps are encoded in binary, so their values are at most exponential. Afterward, all behavior is periodic, with period given by the least common multiple of the source periods and all arrow cycles; each such period is polynomial, so the lcm is at most exponential.

Now we use binary search to find the earliest time at which the player wins, between 0 and this exponential upper bound T. First we use an NP oracle to decide "Win in $[0, T]$?", and if not, return "no" immediately. Given an interval $[\ell, u]$ (initially $[0, T]$), we use an NP oracle to decide "Win in $[\ell, (\ell + u)/2]$?", and if so, recurse on this interval; otherwise, we recurse on $[(\ell + u)/2, u]$. Once we reach a unit interval, we obtain the earliest time t at which the player wins.

It remains to check that the player does not lose before time t. This can be done via a single oracle call to "Lose in $[0, t]$?"

Thus we solve the simulation problem in $\mathrm{P}^{\mathrm{NP}} = \Delta_2^{\mathrm{P}}$. To solve the game problem, we first existentially guess the details of the arrow placements, and then run the same algorithm. Thus we have an NP algorithm making calls to an NP oracle, which proves membership in $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^{\mathrm{P}}$.

An easier case is when the source periods are all the same after warmup, as implemented in the real game. Theorem 1 proved this version of the game NP-hard, and we can now show that it is NP-complete:

Lemma 3. *If all sources have the same polynomial-length period after a polynomial number of time steps, then the simulation problem is in P and the game problem is in NP.*

Proof. First we assume no player input (no additional placed arrows). In this case, we can afford to check for wins or losses in each interval between events by explicitly simulating the step-by-step motion of all packets, and checking for score underflow or overflow along the way. If we explicitly simulate for a duration longer than all packet paths, then the loading bar value becomes periodic with the common source period. (Cycles of arrows may have different periods but these do not affect the loading bar, and thus do not matter when checking for wins and losses.) If the game continues past the last event, then we explicitly simulate for one period, and (assuming no score overflow or underflow) measure the sign of the net score change over the period. If it is positive, then the player will eventually win; if it is negative, then the player will eventually lose; and if it is zero, then the game will go on forever.

Now we allow player input, either given as events in the simulation problem or guessed nondeterministically in the game problem. Because the timestamps are encoded in binary, these inputs can occur at exponentially large times. We use the algorithm above to simulate each interval between player inputs, explicitly simulating during the warmup phase until packet behavior becomes periodic, explicitly simulating the first period to determine the effect of one period, then multiplying this effect to fast-forward through most of the periodic phase until the last full period, and then explicitly simulating the last full period and the last partial period before the next player input. Given the periodic behavior, if the game did not end during the last full period, then it did not end during any previous period.

In the remainder of this section, we consider the case where each source can be assigned any integer period, and the period does not change over time.

3.2 Periodic Sum Threshold Problem

With differing source periods, the challenge is that the overall periodic behavior of the game can have an extremely large (exponential) period. For example, if we allow arbitrary behavior of n sources during the periodic phase,[1] then merely

[1] A general packet-emitting pattern during the periodic phase is relatively easy to simulate by stacking multiple sources, each with simple "emit every k time steps"

deciding whether all sources emit a packet at the same time is the intersection problem of n tally DFAs, which is NP-complete [3], even with no warmup periods where the problem is known as Periodic Full Character Alignment [5].

To model the TGINGTLI loading bar with winning and losing values, we define the closely related ***Periodic Sum Threshold Problem*** as follows. We are given a function $f(x) = \sum_i g_i(x)$ where each g_i has unary integer period T_i and unary maximum absolute value M_i. In addition, we are given a unary integer $\tau > 0$ and a binary integer time x^*. The goal is to determine whether there exists an integer x in $[0, x^*)$ such that $f(x) \geq \tau$. (Intuitively, reaching τ corresponds to winning.)

Theorem 2. *The Periodic Sum Threshold Problem is NP-complete, even under the following restrictions:*

1. *Each $|g_i|$ is a one-hot function, i.e., $g_i(x) = 0$ everywhere except for exactly one x in its period where $g_i(x) = \pm 1$.*
2. *We are given a unary integer $\lambda < 0$ such that $f(x) > \lambda$ for all $0 \leq x < x^*$ and $f(x^*) \leq \lambda$. (Intuitively, dipping down to λ corresponds to losing.)*

Proof. First, the problem is in NP: we can guess $x \in [0, x^*)$ and then evaluate whether $f(x) \leq c$ in polynomial time.

For NP-hardness, we reduce from 3SAT. We map each variable v_i to the ith prime number p_i excluding 2. Using the Chinese Remainder Theorem, we can represent a Boolean assignment ϕ as a single integer $0 \leq x < \prod_i p_i$ where $x \equiv 1 \mod p_i$ when ϕ sets v_i to true, and $x \equiv 0 \mod p_i$ when ϕ sets v_i to false. (This mapping does not use other values of x modulo p_i. In particular, it leaves $x \equiv -1 \mod p_i$ unused, because $p_i \geq 3$.)

Next we map each clause such as $C = (v_i \vee v_j \vee \overline{v_k})$ to the function

$$g_C(x) = \max\{[x \equiv 1 \mod p_i], [x \equiv 1 \mod p_j], [x \equiv 0 \mod p_k]\},$$

i.e., positive literals check for $x \equiv 1$ and negated literals check for $x \equiv 0$. This function is 1 exactly when x corresponds to a Boolean assignment that satisfies C. This function has period $p_i p_j p_k$, whose unary value is bounded by a polynomial. Setting τ to the number of clauses, there is a value x where the sum is τ if and only if there is a satisfying assignment for the 3SAT formula. (Setting τ smaller, we could reduce from Max 3SAT.)

To achieve Property 1, we split each g_C function into a sum of polynomially many one-hot functions (bounded by the period). In fact, seven functions per clause suffice, one for each satisfying assignment of the clause.

To achieve Property 2, for each prime p_i, we add the function $h_i(x) = -[x \equiv -1 \mod p_i]$. This function is -1 only for unused values of x which do not correspond to any assignment ϕ, so it does not affect the argument above. Setting $-\lambda$ to the number of primes (variables) and $x^* = \prod_i p_i - 1$, we have $\sum_i h_i(x^*) = \lambda$

periodic behavior, at the same location. But we will not need this extra functionality, beyond relating the problem to tally DFAs and Periodic Full Character Alignment.

because $h_i(x^*) \equiv -1 \mod p_i$ for all i, while $\sum_i h_i(x) > \lambda$ for all $0 \le x < x^*$. All used values x are smaller than x^*.

In total, $f(x)$ is the sum of the constructed functions and we obtain the desired properties.

3.3 Simulation Hardness for Two Colors

We can use our hardness of the Periodic Sum Threshold Problem to prove hardness of simulating TGINGTLI, even without player input.

Theorem 3. *Simulating TGINGTLI and determining whether the player wins is NP-hard, even with just two colors.*

Proof. We reduce from the Periodic Sum Threshold Problem proved NP-complete by Theorem 2.

For each function g_i with one-hot value $g_i(x_i) = 1$ and period T_i, we create a blue source b_i and a red sources r_i, of the same emitting period T_i, and route red and blue packets from these sources to the blue sink. By adjusting the path lengths and/or the warmup times of the sources, we arrange for a red packet to arrive one time unit after each blue packet which happens at times $\equiv x_i \mod T_i$. Thus the net effect on the loading bar value is $+1$ at time x_i but returns to 0 at time $x_i + 1$. Similarly, for each function g_i with one-hot value $g_i(x_i) = -1$, we perform the same construction but swapping the roles of red and blue.

Setting $k_0 = -\lambda - 1 \ge 0$, the loading bar goes negative (and the player loses) exactly when the sum of the functions g_i goes down to λ. Setting $k^* = k_0 + \tau$, the loading bar reaches k^* (and the player wins) exactly when the sum of the functions g_i goes up to τ.

This NP-hardness proof relies on completely different aspects of the game from the proof in Section 2: instead of using player input, it relies on differing (but small in unary) periods for different sources. More interesting is that we can also prove the same problem coNP-hard:

Theorem 4. *Simulating TGINGTLI and determining whether the player wins is coNP-hard, even with just two colors.*

Proof. We reduce from the complement of the Periodic Sum Threshold Problem, which is coNP-complete by Theorem 2. The goal in the complement problem is to determine whether there is *no* integer x in $[0, x^*)$ such that $f(x) \ge \tau$. The idea is to negate all the values to flip the roles of winning and losing.

For each function g_i, we construct two sources and wire them to a sink in the same way as Theorem 3, but negated: if $g_i(x_i) = \pm 1$, then we design the packets to have a net effect of ∓ 1 at time x_i and 0 otherwise.

Setting $k_0 = \tau - 1$, the loading bar goes negative (and the player loses) exactly when the sum of the functions g_i goes up to τ, i.e., the Periodic Sum Threshold Problem has a "yes" answer. Setting $k^* = k_0 - \lambda$, the loading bar reaches k^* (and the player wins) exactly when the sum of the functions g_i goes down to λ, i.e., the Periodic Sum Threshold Problem has a "no" answer.

4 Characterizing Perfect Layouts

Suppose we are given a board which is empty except for the location of the three data sinks. Is it possible to place arrows such that all possible input packets get routed to the correct sink? We call such a configuration of arrows a ***perfect layout***. In particular, such a layout guarantees victory, regardless of the data sources. In this section, we give a full characterization of boards and sink placements that admit a perfect layout. Some of our results work for a general number c of colors, but the full characterization relies on $c = 3$.

4.1 Colors Not Arrows

We begin by showing that we do not need to consider the directions of the arrows, only their colors and locations in the grid.

Let B be a board with specified locations of sinks, and let ∂B be the set of edges on the boundary of B. Suppose we are given an assignment of colors to the cells of B that agrees with the colors of the sinks; let C_i be the set of grid cells colored with color i. We call two cells of C_i, or a cell of C_i and a boundary edge $e \in \partial B$, ***visible*** to each other if and only if they are in the same row or the same column and no sink of a color other than i is between them. Let G_i be the graph whose vertex set is $C_i \cup \partial B$, with edges between pairs of vertices that are visible to each other.

Lemma 4. *Let B be a board with specified locations of sinks. Then B admits a perfect layout if and only if it is possible to choose colors for the remaining cells of the grid such that, for each color i, the graph G_i is connected.*

Proof. ($\Longrightarrow$) Without loss of generality, assume that the perfect layout has the minimum possible number of arrows. Color the cells of the board with the same colors as the sinks and arrows in the perfect layout. (If a cell is empty in the perfect layout, then give it the same color as an adjacent cell; this does not affect connectivity.) Fix a color i. Every boundary edge is connected to the sink of color i by the path a packet of color i follows when entering from that edge. (In particular, the path cannot go through a sink of a different color.) By minimality of the number of arrows in the perfect layout, every arrow of color i is included in such a path. Therefore G_i is connected.

($\Longleftarrow$) We will replace each cell by an arrow of the same color to form a perfect layout. Namely, for each color i, choose a spanning tree of G_i rooted at the sink of color i, and direct arrows from children to parents in this tree. By connectivity, any packet entering from a boundary edge will be routed to the correct sink, walking up the tree to its root.

4.2 Impossible Boards

Next we show that certain boards cannot have perfect layouts with c colors. First we give arguments about boards containing sinks too close to the boundary or each other. Then we give an area-based constraint on board size.

Lemma 5. *If there are fewer than $c-1$ blank cells in a row or column between a sink and a boundary of the grid, then there is no perfect layout.*

Proof. A perfect layout must prevent packets of the other $c-1$ colors entering at this boundary from reaching this sink; this requires enough space for $c-1$ arrows.

Lemma 6. *For $c = 3$, a board has no perfect layout if either (as shown in Figure 9)*

(a) a data sink is two cells away from three boundaries and adjacent to another sink;
(b) a data sink is two cells away from two incident boundaries and is adjacent to two other sinks;
(c) a data sink is two cells away from two opposite boundaries and is adjacent to two other sinks; or
(d) a data sink is two cells away from three boundaries and is one blank cell away from a pair of adjacent sinks.

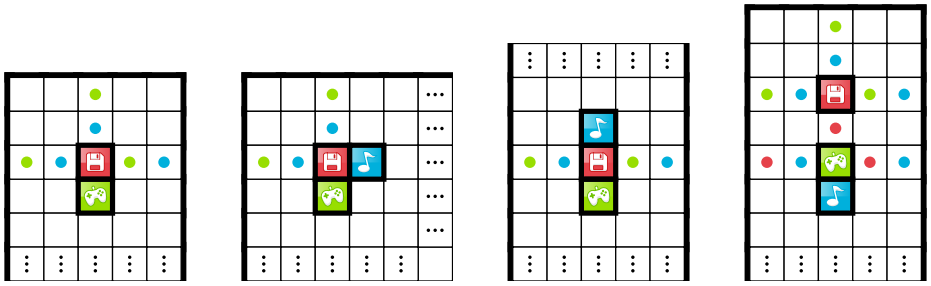


Fig. 9: Sink configurations with no perfect layout. Dots indicate arrows of forced colors (up to permutation within a row or column).

Proof. Assume by symmetry that, in each case, the first mentioned sink is red.

Cases (a), (b), and (c): The pairs of cells between the red sink and the boundary (marked with dots in the figure) must contain a green arrow and a blue arrow to ensure those packets do not reach the red sink. Thus there are no available places to place a red arrow in the same row or column as the red sink, so red packets from other rows or columns cannot reach the red sink.

Case (d): The pairs of cells between the red sink and the boundary (marked with green and blue dots in the figure) must contain a green arrow and a blue arrow to ensure those packets do not collide with the red sink. Thus the blank square between the red sink and the other pair of sinks must be a red arrow pointing toward the red sink, to allow packets from other rows and columns

to reach the red sink. Assume by symmetry that the sink nearest the red sink is green. As in the other cases, the pairs of cells between the green sink and the boundary must be filled with red and blue arrows. Thus there are no green arrows to route green packets from other rows or columns to the green sink.

We now prove a constraint on the sizes of boards that admit a perfect layout.

Lemma 7. *Let c be the number of colors. Suppose there is a perfect layout on a board where m and n are respectively the number of rows and columns, and p and q are respectively the number of rows and columns that contain at least one sink. Then*

$$c(m+n) + (c-2)(p+q) \leq mn - c. \tag{4.1}$$

Proof. Each of the $m-p$ unoccupied rows must contain c vertical arrows in order to redirect packets of each color out of the row. Each of the p occupied rows must contain $c-1$ vertical arrows to the left of the leftmost sink in order to redirect incorrectly colored packets from the left boundary edge away from that sink; similarly, there must be $c-1$ vertical arrows to the right of the rightmost sink. Thus we require $c(m-p)+2(c-1)p = cm+(c-2)p$ vertical arrows overall. By the same argument, we must have $cn+(c-2)q$ horizontal arrows, for a total of $c(m+n)+(c-2)(p+q)$ arrows. There are $mn-c$ cells available for arrows, which proves the claim.

Up to insertion of empty rows or columns, rotations, reflections, and recolorings, there are six different configurations that $c=3$ sinks may have with respect to each other, shown in Figure 10. We define a board's ***type*** according to this configuration of its sinks (C, I, J, L, Y, or /).

(a) C **(b)** I **(c)** J **(d)** L **(e)** Y **(f)** /

Fig. 10: The six possible configurations of three sinks up to rotations, reflections, recolorings, and removal of empty rows.

A board's type determines the values of p and q and thus the minimal board sizes as follows. Define a board to have size ***at least*** $m \times n$ if it has at least m rows and at least n columns, or vice versa.

Lemma 8. *For a perfect layout to exist with $c=3$, it is necessary that:*

- *Boards of type Y or / have size at least 7×8.*
- *Boards of type C or J have size at least 7×8 or 6×9.*
- *Boards of type L have size at least 7×7 or 6×9.*

– *Boards of type I have size at least* 7×7, 6×9, *or* 5×11.

Proof. These bounds follow from Lemma 7 together with the requirement from Lemma 5 that it be possible to place sinks at least two cells away from the boundary.

4.3 Constructing Perfect Layouts

In this section, we complete our characterization of boards with perfect layouts for $c = 3$. We show that Lemmas 5, 6, and 8 are the only obstacles to a perfect layout:

Theorem 5. *A board with* $c = 3$ *sinks has a perfect layout if and only if the following conditions all hold:*

1. *All sinks are at least two cells away from the boundary (Lemma 5).*
2. *The board does not contain any of the four unsolvable configurations in Figure 9 (Lemma 6).*
3. *The board obeys the size bounds of Lemma 8.*

We call a board ***minimal*** if it has one of the minimal dimensions for its type as defined in Lemma 8. Our strategy for proving Theorem 5 will be to reduce the problem to the finite set of minimal boards, which we then verify by computer. We will accomplish this by removing empty rows and columns from non-minimal boards to reduce their size, which we show can always be done while preserving the above conditions.

Lemma 9. *All minimal boards satisfying the three conditions of Theorem 5 have a perfect layout.*

Proof. The proof is by exhaustive computer search of all such minimal boards. We wrote a Python program to generate all possible board patterns, reduce each perfect layout problem to Satisfiability Modulo Theories (SMT), and then solve it using Z3 [4]. The results of this search are in Appendix A.

If B_0 and B_1 are boards, then we define $\boldsymbol{B_0 \lessdot B_1}$ to mean that B_0 can be obtained by removing a single empty row or column from B_1.

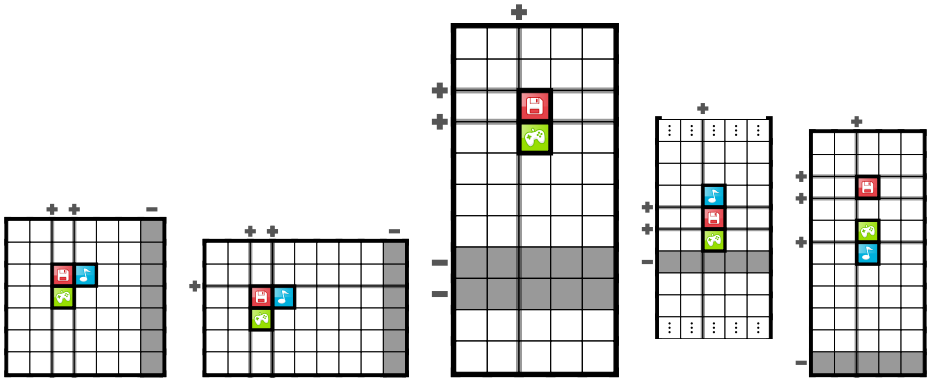
Lemma 10. *If* $B_0 \lessdot B_1$ *and* B_0 *has a perfect layout, then* B_1 *also has a perfect layout.*

Proof. By symmetry, consider the case where B_1 has an added empty row, say i. By Lemma 4, it suffices to show that we can color the cells of the new row i while preserving connectivity in each color. Given a connected coloring of B_0, we copy the coloring over to the corresponding rows of B_1, and then color the new row i to match one of the neighboring rows, $i \pm 1$ where the sign is chosen such that $i \pm 1$ is a row. In particular, if row $i \pm 1$ has a sink of color c, then we color the corresponding cell of row i with color c (but do not add a sink). Because row i of B_1 was empty of sinks, we are free to color it arbitrarily. Connectivity of the B_1 coloring follows from connectivity of the B_0 coloring.

Lemma 11. *Let B_1 be a non-minimal board satisfying the three conditions of Theorem 5. Then there exists a board B_0 that also satisfies all three conditions and such that $B_0 \lessdot B_1$.*

Proof. By symmetry, suppose B_1 is non-minimal in its number m of rows. By removing a row from B_1 that is not among the first or last two rows and does not contain a sink, we obtain a board B_0' satisfying conditions (1) and (3) such that $B_0' \lessdot B_1$. If B_0' also satisfies condition (2), then we are done, so we may assume that it does not.

Then B_0' must contain one of the four unsolvable configurations, and B_1 is obtained by inserting a single empty row or column to remove the unsolvable configuration. Figure 11 shows all possibilities for B_0', as well as the locations where rows or columns may be inserted to yield a corresponding possibility for B_1. (B_0' may have additional empty rows and columns beyond those shown, but this does not affect the proof.) For each such possibility, Figure 11 highlights another row or column which may be deleted from B_1 to yield $B_0 \lessdot B_1$ where B_0 satisfies all three conditions.



(a) L, 7×7 **(b)** L, 6×9 **(c)** I, 5×11 **(d)** I, 5×11 **(e)** I, 5×11

Fig. 11: All boards satisfying conditions (1) and (3) but not (2), up to rotations, reflections, and recolorings. An empty row or column may be inserted in any of the locations marked "+" to yield a board satisfying all three conditions. Removing the row or column marked "−" then preserves the conditions. In case (c), remove a row that does not contain the blue sink. In case (d), $\boxed{\vdots}$ denotes zero or more rows.

Proof (Proof of Theorem 5). It follows from Lemmas 5, 6, and 8 that all boards with perfect layouts must obey the three properties of the theorem. We prove that the properties are also sufficient by induction on the size of the board. As a base case, the claim holds for minimal boards by Lemma 9. For non-minimal boards B_1, Lemma 11 shows that there is a smaller board B_0 that satisfies all three conditions and such that $B_0 \lessdot B_1$. By the inductive hypothesis, B_0 has a perfect layout. Lemma 10 shows that B_1 also has a perfect layout.

5 Open Questions

The main complexity open question is whether TGINGTLI simulation is Δ_2^{P}-complete, and whether the game problem is Σ_2^{P}-complete. Given our NP- and coNP-hardness results, we suspect that both of these results hold.

One could also ask complexity questions of more restrictive versions of the game. For example, what if the board has a constant number of rows?

When characterizing perfect layouts, we saw many of our lemmas generalized to different numbers of colors. It may be interesting to further explore the game and try to characterize perfect layouts with more than three colors.

A related problem is which boards and configurations of sinks admit a ***damage-free*** layout, where any packet entering from the boundary either reaches the sink of the correct color or ends up in an infinite loop. Such a layout avoids losing, and in the game as implemented, such a layout actually wins the game (because the player wins if there is ever insufficient room for a new source to be placed). Can we characterize such layouts like we did for perfect layouts?

Perfect and damage-free layouts are robust to any possible sources. However, for those boards that do not admit a perfect or damage-free layout, it would be nice to have an algorithm that determines whether a given set of sources or sequence of packets still has a placement of arrows that will win on that board. Because the board starts empty except for the sinks, our hardness results do not apply.

Having a unique solution is often a desirable property of puzzles. Thus it is natural to ask about ASP-hardness [8] and whether counting the number of solutions is #P-hard.

Acknowledgments. This work was initiated during open problem solving in the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.892) taught by Erik Demaine in Spring 2019. We thank the other participants of that class for related discussions and providing an inspiring atmosphere. In particular, we thank Quanquan C. Liu for helpful discussions and contributions to early results.

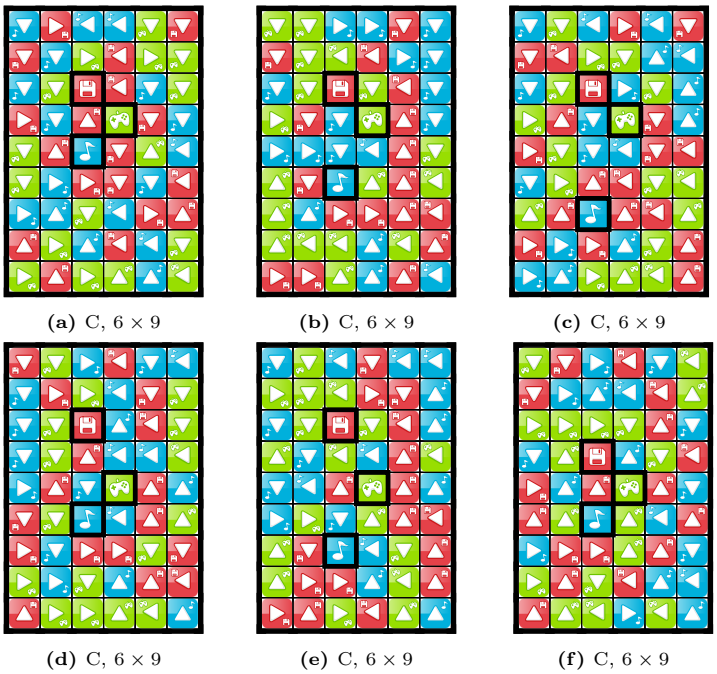
We also thank the anonymous referees for helpful suggestions, in particular for strengthening Lemma 2 from Σ_2^{P} to Δ_2^{P} membership.

Most figures of this paper were drawn using SVG Tiler [https://github.com/edemaine/svgtiler]. Icons (which match the game) are by looneybits and released in the public domain [https://opengameart.org/content/gui-buttons-vol1].

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

A Perfect Layouts from the Automated Solver

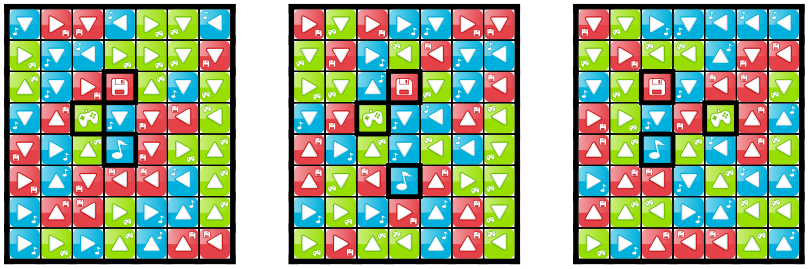
The following 13 figures show all cases found by the automated solver. Figures 12, 13, and 14 correspond to sinks in the C pattern. Figures 15, 16, and 17 correspond to sinks in the I pattern. Figures 18, 19, and 20 correspond to sinks in the J pattern. Figures 21 and 22 correspond to sinks in the L pattern. Figure 23 corresponds to sinks in the Y pattern. Finally, Figure 24 corresponds to sinks in the / pattern.



(a) C, 6 × 9 **(b)** C, 6 × 9 **(c)** C, 6 × 9

(d) C, 6 × 9 **(e)** C, 6 × 9 **(f)** C, 6 × 9

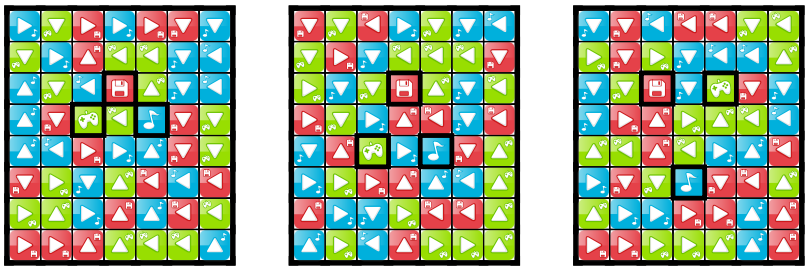
Fig. 12: Solutions from the automated solver for sinks in the C pattern of size 6 × 9.



(a) C, 7 × 8 **(b)** C, 7 × 8 **(c)** C, 7 × 8

(d) C, 7 × 8 **(e)** C, 7 × 8 **(f)** C, 7 × 8

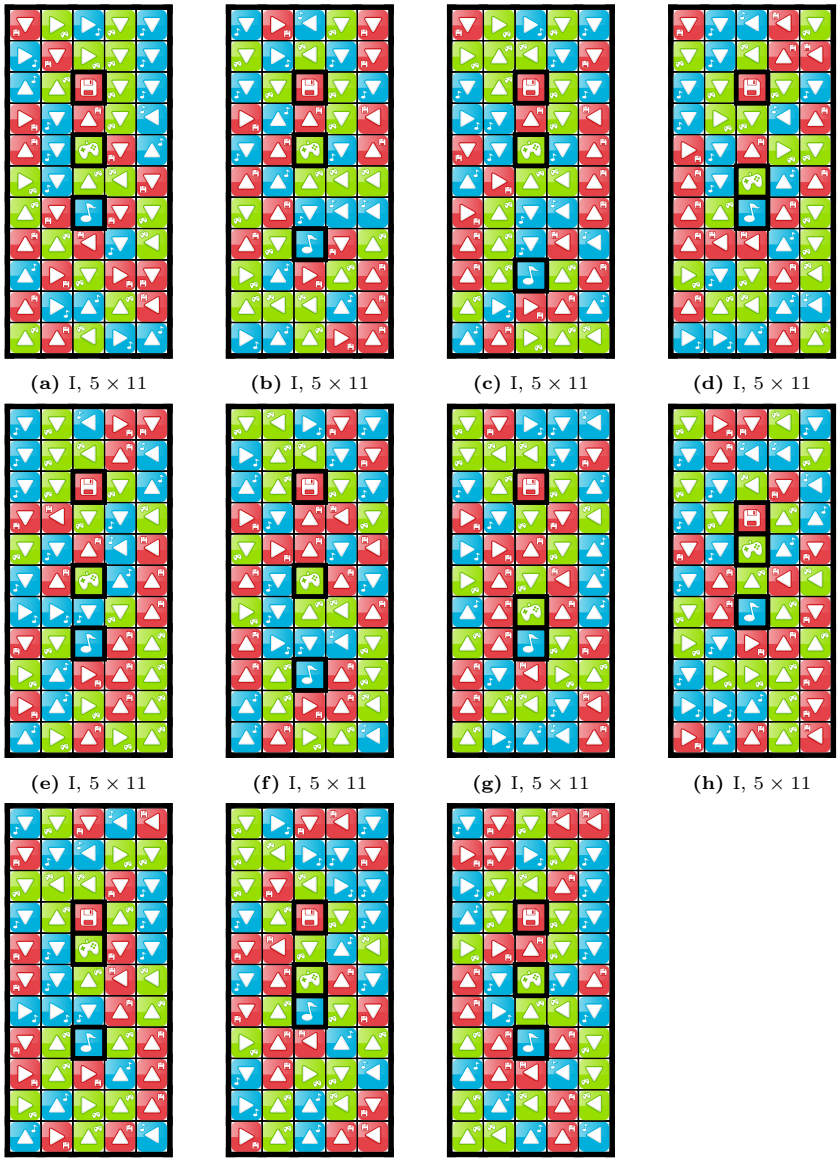
Fig. 13: Solutions from the automated solver for sinks in the C pattern of size 7 × 8.



(a) C, 8 × 7 **(b)** C, 8 × 7 **(c)** C, 8 × 7

(d) C, 8 × 7 **(e)** C, 8 × 7 **(f)** C, 8 × 7

Fig. 14: Solutions from the automated solver for sinks in the C pattern of size 8 × 7.

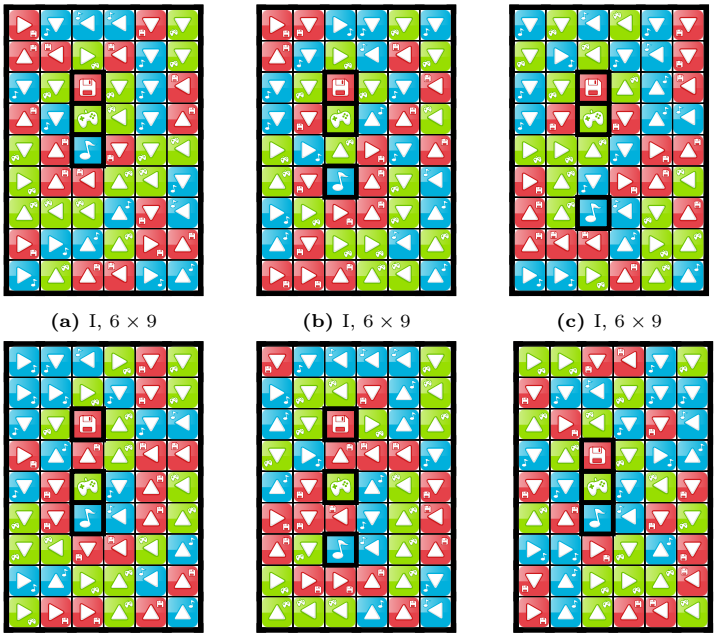


(a) I, 5×11 **(b)** I, 5×11 **(c)** I, 5×11 **(d)** I, 5×11

(e) I, 5×11 **(f)** I, 5×11 **(g)** I, 5×11 **(h)** I, 5×11

(i) I, 5×11 **(j)** I, 5×11 **(k)** I, 5×11

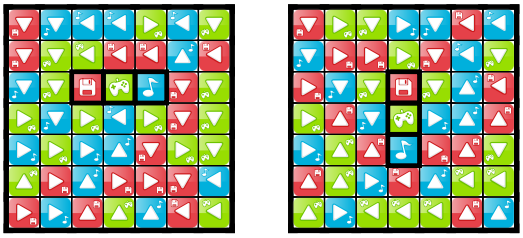
Fig. 15: Solutions from the automated solver for sinks in the I pattern of size 5×11.



(a) I, 6 × 9 **(b)** I, 6 × 9 **(c)** I, 6 × 9

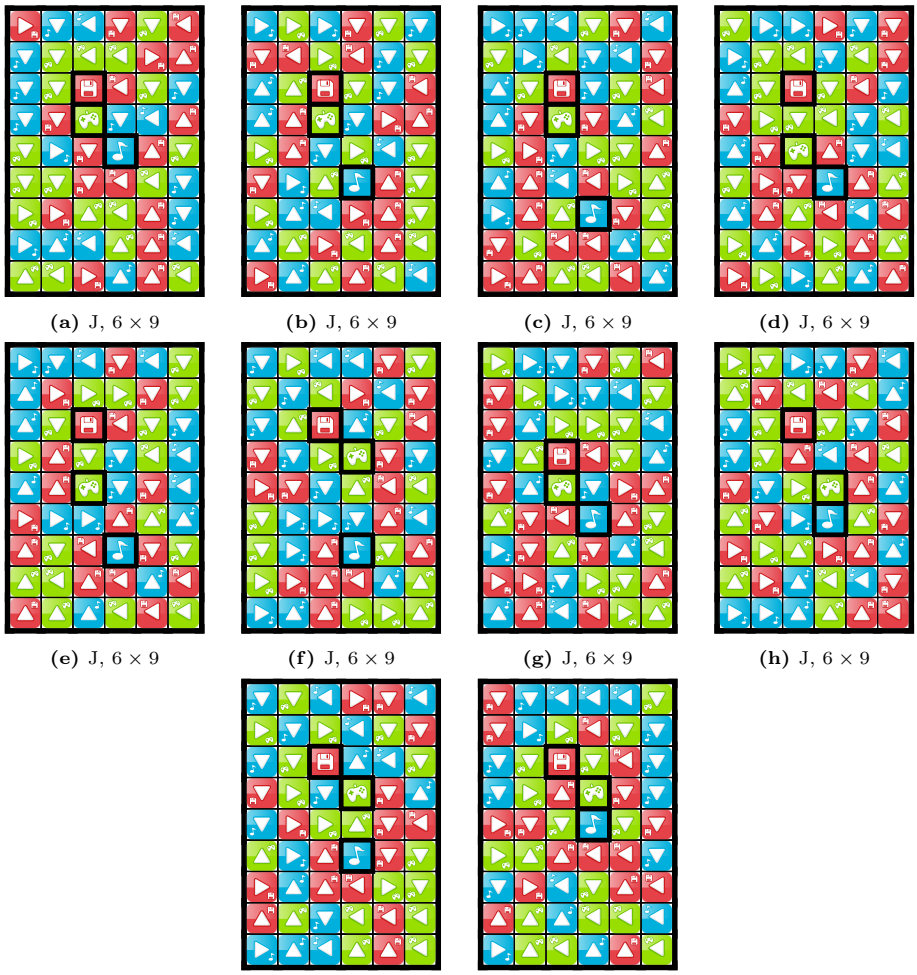
(d) I, 6 × 9 **(e)** I, 6 × 9 **(f)** I, 6 × 9

Fig. 16: Solutions from the automated solver for sinks in the I pattern of size 6 × 9.



(a) I, 7 × 7 **(b)** I, 7 × 7

Fig. 17: Solutions from the automated solver for sinks in the I pattern of size 7 × 7.

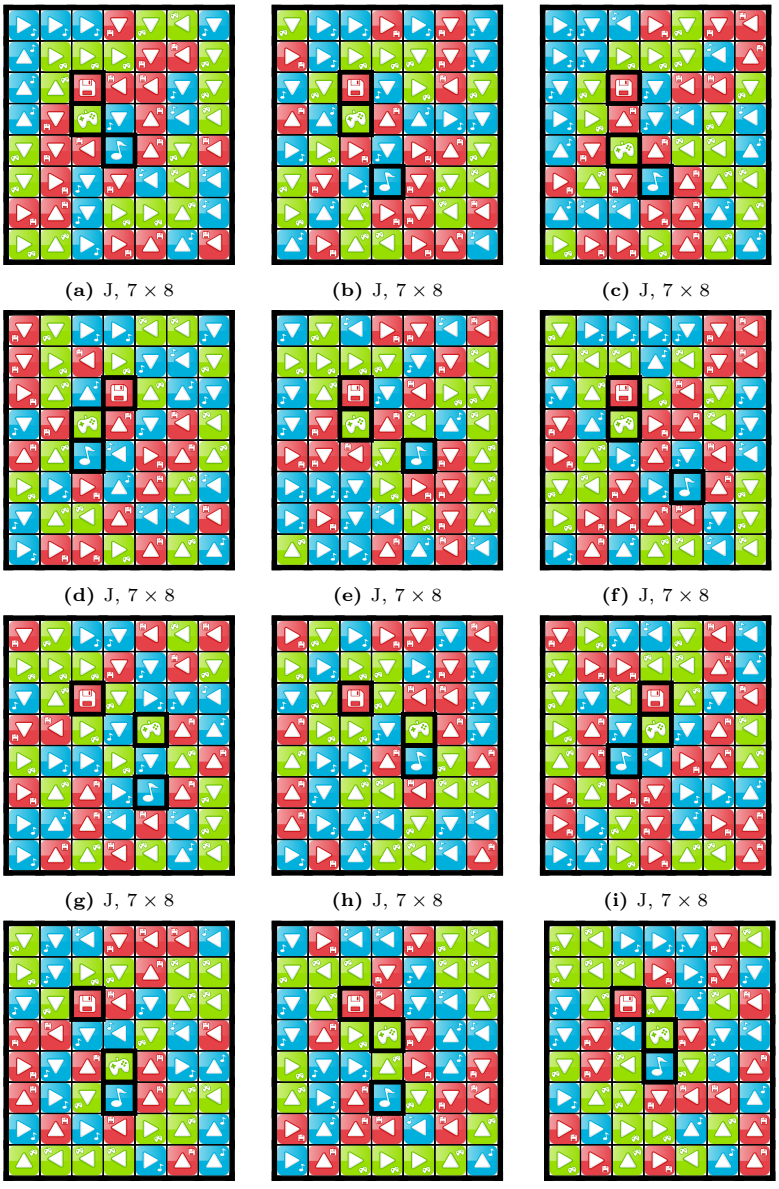


(a) J, 6 × 9 **(b)** J, 6 × 9 **(c)** J, 6 × 9 **(d)** J, 6 × 9

(e) J, 6 × 9 **(f)** J, 6 × 9 **(g)** J, 6 × 9 **(h)** J, 6 × 9

(i) J, 6 × 9 **(j)** J, 6 × 9

Fig. 18: Solutions from the automated solver for sinks in the J pattern of size 6 × 9.



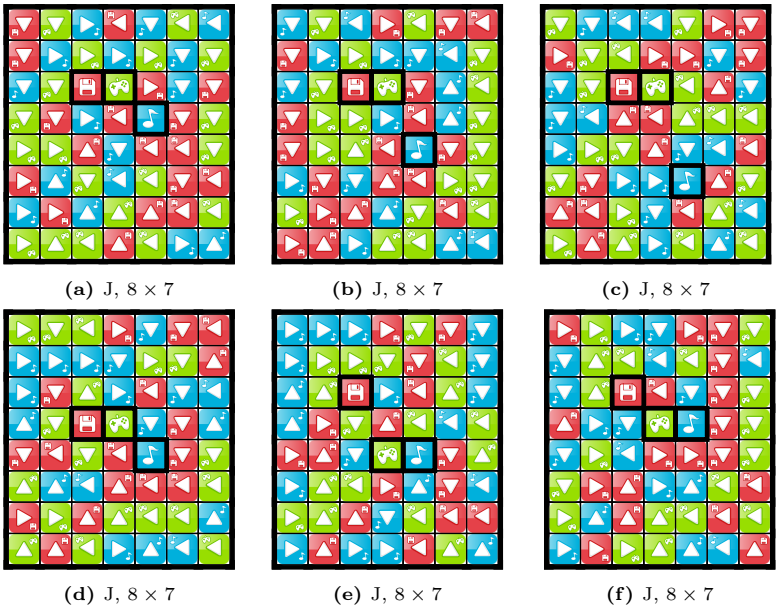
(a) J, 7×8 **(b)** J, 7×8 **(c)** J, 7×8

(d) J, 7×8 **(e)** J, 7×8 **(f)** J, 7×8

(g) J, 7×8 **(h)** J, 7×8 **(i)** J, 7×8

(j) J, 7×8 **(k)** J, 7×8 **(l)** J, 7×8

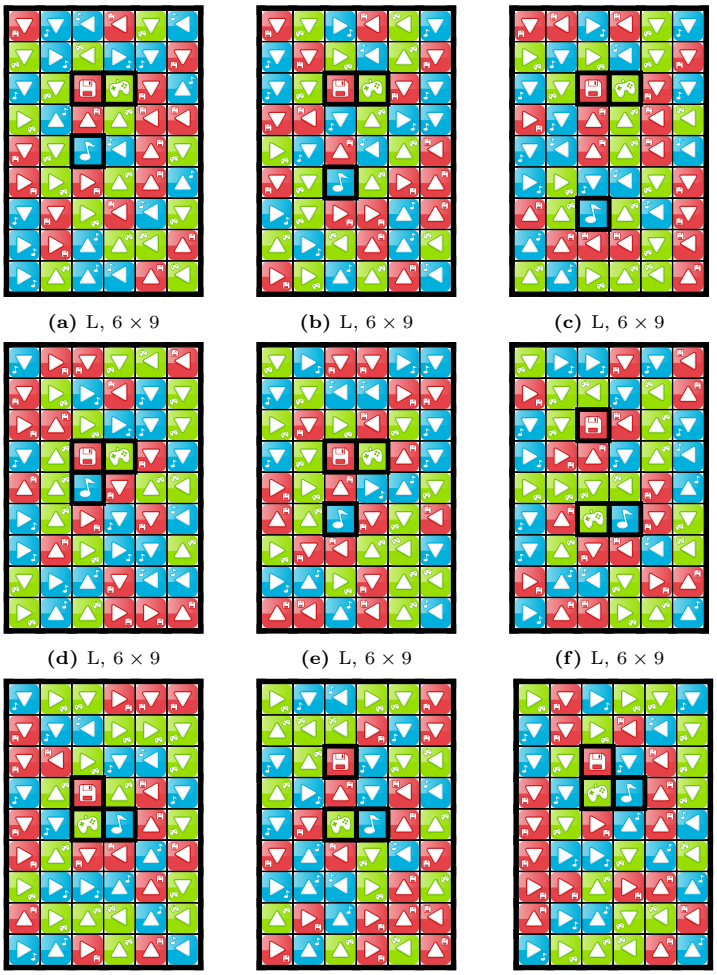
Fig. 19: Solutions from the automated solver for sinks in the J pattern of size 7×8.



(a) J, 8×7 **(b)** J, 8×7 **(c)** J, 8×7

(d) J, 8×7 **(e)** J, 8×7 **(f)** J, 8×7

Fig. 20: Solutions from the automated solver for sinks in the J pattern of size 8×7.

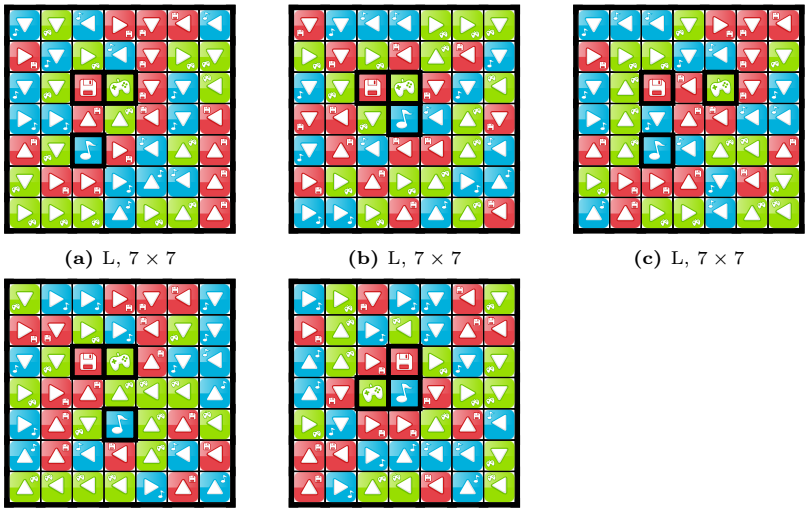


(a) L, 6 × 9 **(b)** L, 6 × 9 **(c)** L, 6 × 9

(d) L, 6 × 9 **(e)** L, 6 × 9 **(f)** L, 6 × 9

(g) L, 6 × 9 **(h)** L, 6 × 9 **(i)** L, 6 × 9

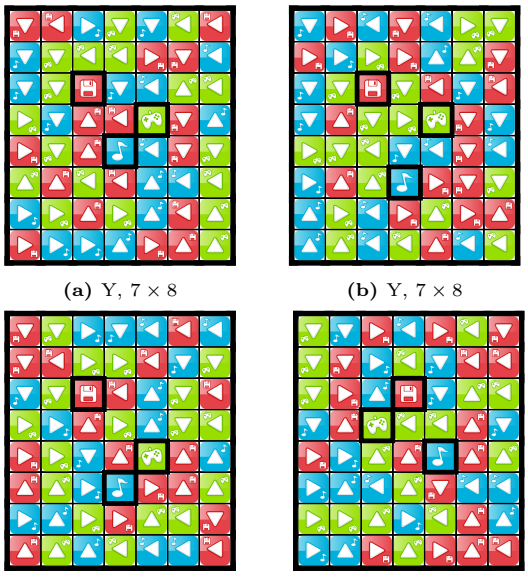
Fig. 21: Solutions from the automated solver for sinks in the L pattern of size 6 × 9.



(a) L, 7×7 **(b)** L, 7×7 **(c)** L, 7×7

(d) L, 7×7 **(e)** L, 7×7

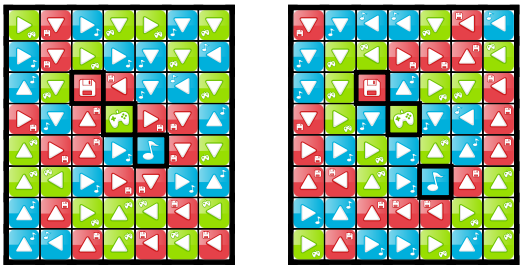
Fig. 22: Solutions from the automated solver for sinks in the L pattern of size 7×7.



(a) Y, 7×8 **(b)** Y, 7×8

(c) Y, 7×8 **(d)** Y, 7×8

Fig. 23: Solutions from the automated solver for sinks in the Y pattern of size 7×8.



(a) /, 7×8 **(b)** /, 7×8

Fig. 24: Solutions from the automated solver for sinks in the / pattern of size 7×8.

References

1. Goergen, K., Fernau, H., Oest, E., Wolf, P.: All paths lead to rome. arXiv:2207.09439 (2022). https://doi.org/10.48550/ARXIV.2207.09439, https://doi.org/10.48550/arXiv.2207.09439, https://arXiv.org/abs/2207.09439
2. Hayashi, Y.: Recording medium, method of loading games program code means, and games machine. U.S. Patent 5,718,632 (issued February 17, 1998)
3. Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata—A survey. Information and Computation **209**(3), 456–470 (2011). https://doi.org/https://doi.org/10.1016/j.ic.2010.11.013, https://www.sciencedirect.com/science/article/pii/S0890540110001999
4. Microsoft Research: The Z3 theorem prover. https://github.com/Z3Prover/z3
5. Morawietz, N., Rehs, C., Weller, M.: A timecop's work is harder than you think. In: Esparza, J., Král', D. (eds.) 45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020). LIPIcs, vol. 170, pp. 71:1–71:14 (2020). https://doi.org/10.4230/LIPIcs.MFCS.2020.71, https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.MFCS.2020.71
6. Ostrander, R.a.: This game is not going to load itself (December 2015), https://atiaxi.itch.io/this-game-is-not-going-to-load-itself, https://atiaxi.itch.io/this-game-is-not-going-to-load-itself
7. Trahtman, A.N.: The road coloring problem. Israel Journal of Mathematics **172**(1), 51–60 (Aug 2009). https://doi.org/10.1007/s11856-009-0062-5, https://doi.org/10.1007/s11856-009-0062-5
8. Yato, T., Seta, T.: Complexity and completeness of finding another solution and its application to puzzles. IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences **E86-A**(5), 1052–1060 (2003), http://ci.nii.ac.jp/naid/110003221178/en/, also IPSJ SIG Notes 2002-AL-87-2, 2002.