Man, These New York Times Games Are Hard! A Computational Perspective

Alessandro Giovanni Alberti
Department of Mathematics, Sapienza University of Rome, Italy

Flavio Chierichetti
Department of Computer Science, Sapienza University of Rome, Italy

Mirko Giacchini
Department of Computer Science, Sapienza University of Rome, Italy

Daniele Muscillo
Department of Computer Science, Sapienza University of Rome, Italy

Alessandro Panconesi
Department of Computer Science, Sapienza University of Rome, Italy

Erasmo Tani
Department of Computer Science, Sapienza University of Rome, Italy

Abstract

The New York Times (NYT) games have found widespread popularity in recent years and reportedly account for an increasing fraction of the newspaper's readership. In this paper, we bring the computational lens to the study of New York Times games and consider four of them not previously studied: Letter Boxed, Pips, Strands and Tiles. We show that these games can be just as hard as they are fun. In particular, we characterize the hardness of several variants of computational problems related to these popular puzzle games. For Letter Boxed, we show that deciding whether an instance is solvable is in general NP-Complete, while in some parameter settings it can be done in polynomial time. Similarly, for Pips we prove that deciding whether a puzzle has a solution is NP-Complete even in some restricted classes of instances. We then show that one natural computational problem arising from Strands is NP-Complete in most parameter settings. Finally, we demonstrate that deciding whether a Tiles puzzle is solvable with a single, uninterrupted combo requires polynomial time.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases NP-Hardness, Puzzles, Games, New York Times, Pips, Letter Boxed, Strands, Tiles

Digital Object Identifier 10.4230/LIPIcs.FUN.2026.2

Related Version *Full Version*: `https://arxiv.org/abs/2509.10846`

Funding *Flavio Chierichetti*: Supported in part by BiCi – Bertinoro international Center for informatics, by a Google Focused Research Award and by the PRIN project 20229BCXNW (funded by the European Union – Next Generation EU, Mission 4 Component 1 CUP B53D23012910006). *Alessandro Panconesi*: Supported in part by BiCi – Bertinoro international Center for informatics and by a Google Focused Research Award.

Acknowledgements We wish to thank Joe Mitchell and the anonymous reviewers for their helpful feedback. We gratefully acknowledge the ELICSIR Foundation for its support and for fostering this collaboration. We also wish to thank the New York Times for making these truly entertaining (and hard!) games for the world to play. Finally, we wish to thank Gabriel Avram for useful suggestions on improving the images, Kyle MacMillan and Rachel B. Thomas for helpful advice and feedback on early versions of the paper.

© Alessandro Giovanni Alberti, Flavio Chierichetti, Mirko Giacchini, Daniele Muscillo, Alessandro Panconesi, and Erasmo Tani;
licensed under Creative Commons License CC-BY 4.0
13th International Conference on Fun with Algorithms (FUN 2026).
Editor: John Iacono; Article No. 2; pp. 2:1–2:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The New York Times is one of the most influential newspapers in the world. In its 173 years of history, it has reported on events as disparate as the American Civil War, the sinking of the Titanic, the publication of Einstein's theory of relativity and the discovery of the tomb of Tutankhamun.

In this time, the newspaper's reporting activities have won it over 130 Pulitzer prizes [16]. The news outlet was also vastly successful in transitioning to the digital era, and its online version boasts millions of subscribers [17], reportedly more than any other English language online newspaper in the world [9].

In spite of all its remarkable achievements, the New York Times' website is increasingly visited for a different reason: a carefully curated collection of puzzle games [5, 6, 15][1]. In February 1942, in a move said to have been caused by the attack on Pearl Harbor [14], the newspaper launched its famous crossword, which has since entertained millions of enthusiasts and become a household name in the crossword world. It is perhaps not surprising then, that the New York Times has leveraged its reputation as a one-stop shop for intellectuals and clever thinkers to break into the online puzzle game space.

Started in 2014 with the introduction of the Mini Crossword and later the word game *Spelling Bee*, the games section of the New York Times has added more and more titles to its collection over the years. In 2019 they introduced *Letter Boxed*, in which the players have to compose words using letters placed around a square, and *Tiles*, in which one is asked to find tiles sharing visual features. In 2022, they also acquired the popular game *Wordle*, which had already found a large following. In 2023, they introduced *Connections*, in which one is tasked with dividing sixteen words into four groups of words that are linked by some hidden connection. In 2024 they added *Strands* a word search-like game, and in 2025 they added the logic puzzle game *Pips*, in which a player has to position domino tiles on a board while satisfying given constraints.

According to a New York Times' spokesperson interviewed by The Verge, in 2024 alone the games were played over eleven billion times [12]. Despite the popularity of the New York Times games, so far, with the exception of Sudoku [19], Wordle [8, 13] and the traditional crossword [3], the games have not been studied and understood from a computational standpoint, and their true complexity has remained unknown to the world. This upsetting state of affairs has undoubtedly left many wondering just how hard these games are, as it brought daily players to question their own abilities in the face of these formidable challenges.

In this paper, we attempt to right this unspeakable wrong, and provide a computational perspective on several problems arising from the New York Times games. Our focus will be on four of the games, which are computationally rich and yet so far unstudied: Letter Boxed, Pips, Strands and Tiles. We consider various NYT games-inspired computational problems and analyze their computational complexity.

1.1 Overview of The New York Times Games

As mentioned in the introduction, the New York Times offers a collection of games beyond its traditional crossword puzzles (see Figure 1). In this paper, we will focus on four games: Letter Boxed, Pips, Strands and Tiles. In this section, we briefly review their rules.

[1] In March of 2024, ValueAct Capital Management, a hedge fund investing in the New York Times Company, filed a notice of exempt solicitation under Rule 14a-6(g) [10], in which they reported finding that in 2023 people spent more time playing NYT games than reading NYT news articles.

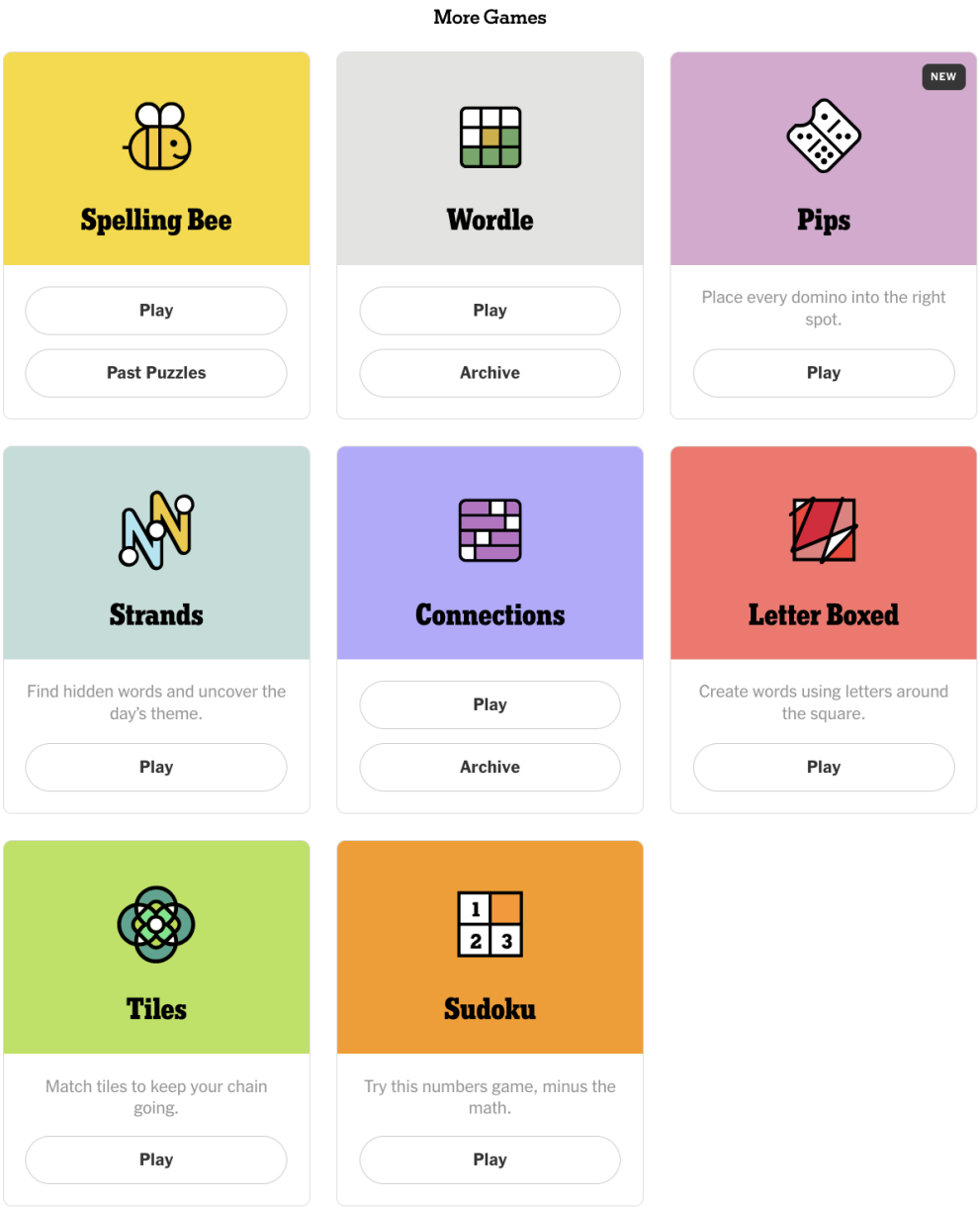

Figure 1 The "More Games" section of the `https://www.nytimes.com/crosswords` page. Accessed on 08/22/25. Used for research and illustrative purposes only.

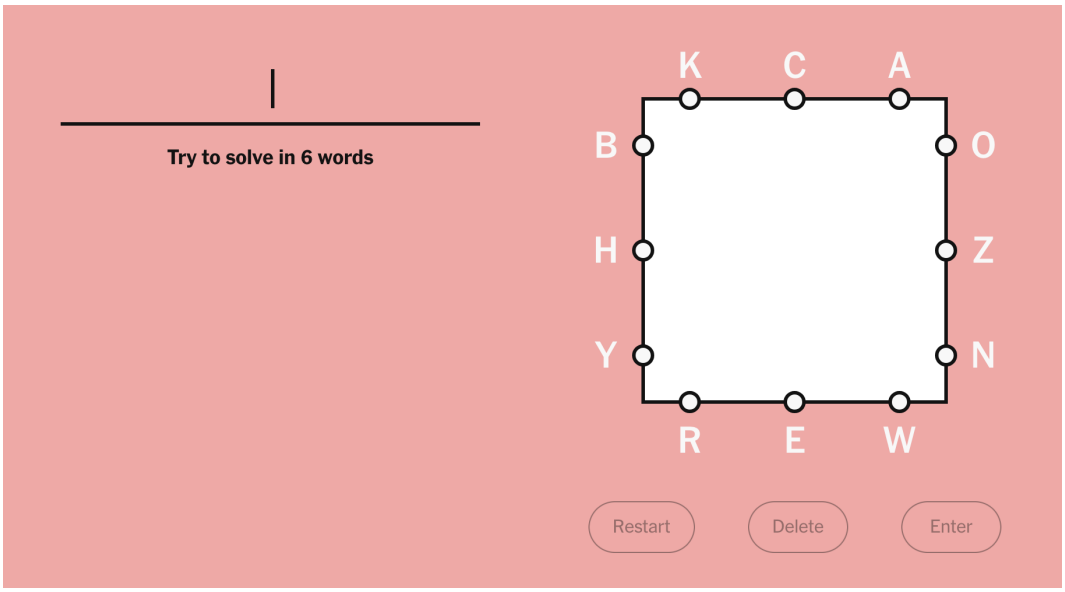

Figure 2 A typical instance of the Letter Boxed game. In this instance, the player is encouraged to complete the puzzle with at most 6 words.
Accessed from `https://www.nytimes.com/puzzles/letter-boxed` on 06/26/25. Used for research and illustrative purposes only.

Letter Boxed

In Letter Boxed (Figure 2), the player is given a square with three letters on each of its sides. The letters are all distinct. The player can then start to compose English words using the letters on the sides of the square, with the following set of rules. First, the player may never use two letters from the same side consecutively in a word. Second, each word formed must start with the same letter (on the same side) that ended the previous word. Finally, every letter must appear in at least one of the player's words. When the player completes a word that includes all the letters not previously included in any word, the puzzle is solved.

The more skilled a player is, the fewer words they will need to complete a puzzle. Next to the puzzle, one is given a suggestion on how many words may be needed for an average player to complete it. This is typically given in the form of a number between 4 and 6.

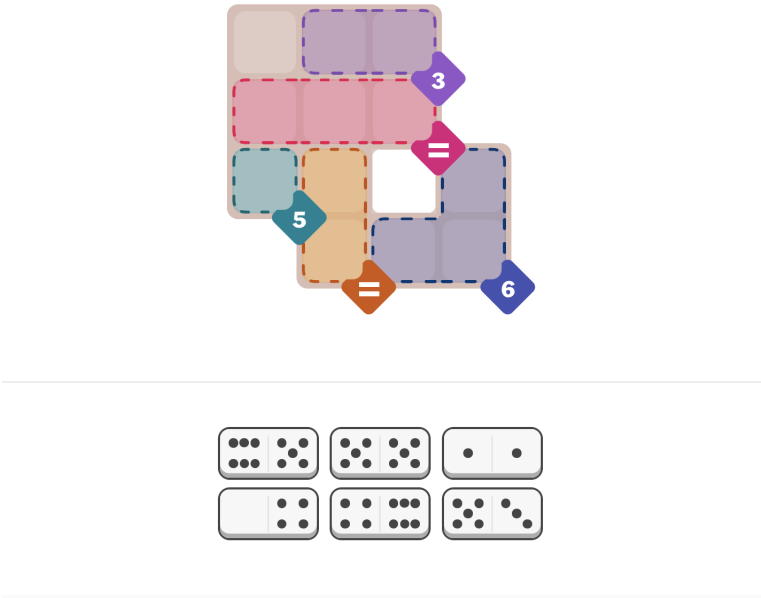

Figure 3 A medium difficulty instance of Pips. In this instance, the board contains five constraint regions. Three of them (labeled with the numbers 3, 5 and 6) specify a constraint on the sum of the domino squares placed on them, and two of them (labeled with the = symbol) specify that all the domino squares placed in the region must have the same value. The goal of the player is to tile the board with all the domino pieces displayed at the bottom, in a way that respects all the constraints. Each domino piece must be used exactly once in the solution.
Accessed from `https://www.nytimes.com/games/pips/medium` on 08/22/2025. Used for research and illustrative purposes only.

Pips

Pips (Figure 3) is the newest entry in the NYT games family. In this game, introduced in the summer of 2025 [18], the player is given a collection of domino tiles and they have to place them on a board in such a way that each square of the board is covered by exactly one half of a domino tile (i.e. the tiles tessellate the board). The board also displays a collection of connected pairwise disjoint regions, each of which specifies a constraint on the values of the domino pieces placed within the region. There are three types of constraints.

1. "$=$" (resp. "$\neq$") constraints specify that the numbers placed within the region must all be equal to (resp. distinct from) one another.
2. "n" constraints, for some number $n \in \mathbb{N}_0$, specify that the numbers within a specific region must sum to n.
3. "$<n$" (resp. "$>n$") constraints specify that the sum of the numbers inside the region must be smaller (resp. greater) than n.

The goal of the game is to place the domino tiles so as to fill the entire board in a way that satisfies all of the constraints specified.

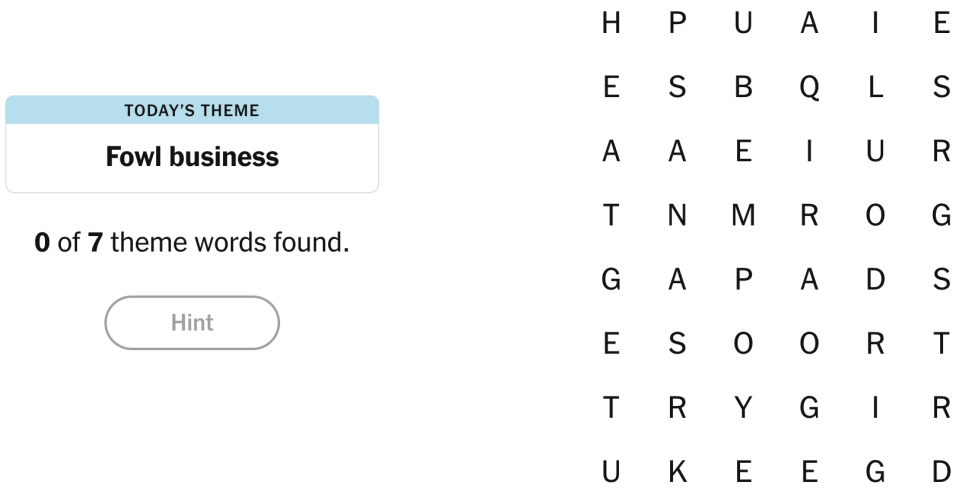

Figure 4 An instance of the Strands game. In this instance, the hint for the theme of the day is "Fowl business". This wordplay suggests that the theme of the day may be related to birds that are hunted. In fact, in the bottom-left corner one can easily find the word *turkey*, and the spangram is the phrase *game birds* which can be seen crossing the puzzle from left to right.
Accessed from `https://www.nytimes.com/games/strands` on 06/26/2025. Used for research and illustrative purposes only.

Strands

Introduced in 2024 [7], Strands (Figure 4) is a modern spin on classic word search puzzles. In this game, the player is given a matrix of letters, partitioned into a hidden collection of words (the solution to the puzzle). Each word in the solution is related to the theme of the day, hinted at in the box on the left. The goal is to find all the words in the solution. Each of these is formed by a sequence of letters, each adjacent to the next vertically, horizontally or diagonally (the same cell cannot be repeated). Moreover, each puzzle solution contains a *spangram*, a word or phrase that connects two opposite sides of the matrix which explicitly reveals the theme of the day.

When the player queries a word, it is revealed whether it belongs to the solution or not. After finding three valid words that do not belong to the hidden collection, the user is given a hint on one of the hidden words. The hint highlights the set of letters in one of the words in the solution. If one were to obtain another hint right after, this would also reveal the word itself, by showing the exact order in which the previously highlighted letters appear in the word.

Figure 5 An instance of the Tiles game.
Accessed from `https://www.nytimes.com/puzzles/tiles` on 06/26/2025. Used for research and illustrative purposes only.

Tiles

In Tiles (Figure 5), the player is presented with a grid of tiles. Each tile in the grid displays some visual elements (*features*). The player begins the game by selecting an arbitrary tile. Each move consists of selecting a new tile to move to, distinct from the previous one. If the newly selected tile shares one or more non-deleted features with the previous one, then these features are deleted from both tiles, and the player's current combo increases by one. Otherwise, the player simply moves to the new tile. We refer to this latter case as a *teleport move*. Note that, when a player finds themselves on a tile on which all the features have been deleted, they are forced to make a teleport move. Teleport moves do not extend the current combo. Moreover, if the player ever makes a teleport move that is not forced, then the length of their current combo is set to zero. The goal of the game is to delete all the features from all the tiles.

1.2 Our Results

In this paper, we show several results on the hardness of New York Times puzzle games. All the missing proofs can be found in the full version of this article.

Our Results on Letter Boxed

For our first set of results, in Section 2, we consider the problem of deciding whether an instance of Letter Boxed is solvable with at most k words. We generalize the game by allowing n characters on each side of the square that we represent as explicit multisets (we allow repetitions of the same character), and we consider words coming from a dictionary D with alphabet Σ. We give a complete characterization of the complexity of the game for different regimes of the alphabet size $|\Sigma|$, the dictionary size $|D|$, the length L of the longest word in D, and the budget of words k.

▶ **Theorem 1.** *Consider the problem of deciding whether an instance of Letter Boxed is solvable with at most k words. Then the complexity of the problem for different parameter regimes is as described in Table 1.*

Table 1 The complexity of Letter Boxed. Asterisks (*) indicate that the result holds both if the parameter is constant and if it can grow with the input size. The settings solvable in polynomial time where parameter k is marked as "arbitrary" hold even if k is represented in binary notation, while the NP-Hardness results hold even when k is represented in unary notation (i.e. they are *strong* NP-*Hardness* results). The number of characters of each side, n, is always allowed to grow arbitrarily.

Alphabet Size $\lvert\Sigma\rvert$	Dictionary Size $\lvert D\rvert$	Maximum Word Length L	Budget of Words k	Complexity Class
Constant	*	*	*	P (Corollary 8)
Arbitrary	Constant	Constant	*	P (Corollary 8)
Arbitrary	Arbitrary	Constant	Constant	P (Corollary 8)
Arbitrary	*	Arbitrary	*	NP-Complete (Theorem 11)
Arbitrary	Arbitrary	Constant	Arbitrary	NP-Complete (Theorem 13)

Our Results on Pips

For Pips, we study the problem of deciding whether an instance of the puzzle admits a solution (Section 3). We show that this problem is NP-Complete, even in the special case in which all the domino tiles' faces have value either zero or one, and there are no "$\neq$", "$<n$" or "$>n$" constraints. We summarize the result in the following theorem.

▶ **Theorem 2.** *The problem of deciding whether a Pips puzzle is solvable is* NP-*Complete. This is true even in the restricted class of instances in which all the domino tiles' numerical values are either* 0 *or* 1 *and none of the constraints are of the form "$\neq$", "$<n$" or "$>n$".*

Moreover, we show that if instances can have arbitrarily large numerical values on the domino tiles, it is weakly NP-Hard to decide whether an instance with a single constraint can be solved.

Our Results on Strands

In Section 4 we consider the problem of deciding whether a Strands grid $M \in \Sigma^{n\times m}$ can be partitioned into valid words of a dictionary D, where Σ is an alphabet of characters. Ignoring the *spangram* that can always be added in the last row, we show that this problem is NP-Hard even if the instance contains only constantly many words of constant length.

▶ **Theorem 3.** *Given a Strands instance (Σ, D, M) determining whether M can be partitioned into valid words from D is* NP-*Complete, even if $|\Sigma|, |D|, L = O(1)$, where L denotes the maximum length of a word in D.*

Our Results on Tiles

In Section 5, we consider the problem of deciding whether a Tiles puzzle can be completed. We show that this condition is equivalent to each distinct feature appearing in an even number of tiles, and hence the problem can be solved in linear time. This result is summarized in the following theorem.

▶ **Theorem 4.** *Given any instance of Tiles, the following are all equivalent:*
1. *The instance is solvable,*
2. *The instance is solvable with a single, unbroken combo,*
3. *Each distinct feature in the instance appears in an even number of tiles.*

In particular, there exists an algorithm that decides whether an instance of Tiles is solvable in time linear in the size of the instance.

Here, the size of the puzzle is the number of features appearing in every tile (counted with multiplicity).

We also consider the problem of deciding whether a Tiles puzzle can be completed without making use of teleport moves. We show that this problem is in P for a restricted class of puzzles: those in which any two tiles share at most one feature. The complexity of this problem in the general setting is left as an open problem.

2 Letter Boxed

In this section, we discuss the complexity of some computational problems related to Letter Boxed. The central problem of interest will be determining the minimum number of words needed to complete a given Letter Boxed puzzle. We will show that this problem is NP-Hard in general, but becomes tractable for certain choices of parameters. The results in this section are summarized in Theorem 1 in Section 1.2.

Recall that a typical instance of Letter Boxed is given by a square, with three letters on each side. In order to obtain a parametrized problem, we allow our instances to have n characters on each side for a varying parameter n. Since we allow the set of distinct characters (alphabet) to be smaller than $4n$, we study a more general version of the problem in which characters could be repeated, even on the same side of the square.

Note that the specific position of each character on a side is immaterial to the game: the game is entirely isomorphic up to permuting characters on the same side of the square. We therefore define instances as being specified up to this isomorphism.

▶ **Definition 5** (Letter Boxed puzzle). *A* Letter Boxed puzzle *is a tuple* $(\Sigma, D, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4)$ *where* Σ *is a finite alphabet,* $D \subseteq \Sigma^*$ *is a finite set of strings of characters from* Σ *called the* dictionary*, and each* Γ_i *is a multiset of* n *elements (counted with multiplicity) from* Σ*, representing the characters appearing on the* i^{th} *side of the square.*

Given a Letter Boxed puzzle, we denote by L the length of the longest word in D. Note that we explicitly represent the multisets $\Gamma_1, \ldots, \Gamma_4$ so that the size of the puzzle is $4n + \sum_{w \in D} |w|$ characters.

In order to solve the problem, one needs to find a sequence of words that covers all the characters on every side,[2] and such that each word in the sequence ends with the same character with which the next word starts. Moreover, these words need to satisfy the constraint that no two subsequent characters are on the same side. We formalize these conditions in the following definition.

[2] If a character appears multiple times, all its occurrences must be covered.

▶ **Definition 6** (Letter Boxed Solution). *Given a Letter Boxed puzzle* $(\Sigma, D, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4)$*, a* solution of length k *to the puzzle is a pair* $(\texttt{words}, \texttt{sides})$*, where* $\texttt{words} = (w^{(1)}, \ldots, w^{(k)}) \in D^k$ *is an ordered sequence of* k *words from the dictionary, and* $\texttt{sides} \in [4]^{\sum_{i\in[k]} |w^{(i)}|}$ *specifies a side for every character in every word in* $\texttt{words}$*, with the following properties:*

1. *For every* $i = 1, \ldots, k-1$*, the last character of* $w^{(i)}$ *is the same as the first character of* $w^{(i+1)}$*.*

Moreover, letting $\sigma = w^{(1)} \circ w^{(2)} \circ \cdots \circ w^{(k)}$ *be the concatenation of the words in* $\texttt{words}$ *then:*

2. *For every* i*,* $\texttt{sides}_i \neq \texttt{sides}_{i+1}$*, unless* σ_i *is the end of some word* $w^{(j)}$*, in which case* $\texttt{sides}_i = \texttt{sides}_{i+1}$*,*
3. *For every* $i \in [|\sigma|]$*,* $\sigma_i \in \Gamma_{\texttt{sides}_i}$*,*
4. *For every* $i \in [4]$*, and for every character* $\gamma \in \Gamma_i$ *the number of indexes* j *such that: (i)* $\sigma_j = \gamma$*, (ii)* $\texttt{sides}_j = i$*, and (iii) either* $j = |\sigma|$ *or* σ_j *is not the end of some* w_ℓ*, is at least the number of occurrences of* γ *in* Γ_i*. Note that the third point ensures that we do not double count characters when starting a new word.*

If a Letter Boxed puzzle admits a solution of length k then we say it is *solvable with at most* k *words.*

We consider the following decision problem.

LETTER BOXED *Input:* A Letter Boxed puzzle $(\Sigma, D, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4)$ and a positive integer k. *Question:* Can the Letter Boxed puzzle be solved with at most k words?

In this work we will also consider variants of the game with a number of sides different from four. Here, we think of instances as being formed by putting letters on the sides of a regular S-polygon. Instances of this version of the game are defined by extending the decision problem LETTER BOXED (and the corresponding Letter Boxed puzzle) in the natural way. We call this version S-SIDES LETTER BOXED. Note that under this definition LETTER BOXED is simply 4-SIDES LETTER BOXED.

The rest of this section proves Theorem 1. We will do so via a series of results. We start by providing a natural dynamic programming algorithm for the problem, which will be used to prove the cases solvable in polynomial time.

Specifically, we can prove the following result by defining a graph in which each vertex represents a "configuration" of the game, and there is a directed edge between two vertices if it is possible to move from one configuration to the other by typing a letter. Solving the game amounts to finding a path from a starting configuration to a winning configuration.

▶ **Theorem 7.** *There is an algorithm that finds the minimum number of words and the minimum number of letters required to solve an instance of* S-SIDES LETTER BOXED *with running time* $O(S^2 \cdot (n+1)^{S\cdot|\Sigma|} \cdot |D|^2 \cdot L)$*.*

By using this algorithm, we can show that LETTER BOXED is solvable in polynomial time in certain regimes.

▶ **Corollary 8.** LETTER BOXED *where either (i)* $|\Sigma| = O(1)$*, or (ii)* $|D| = O(1)$ *and* $L = O(1)$*, or (iii)* $L = O(1)$ *and* $k = O(1)$*, is solvable in polynomial time*

Sadly, these results do not imply that LETTER BOXED can be solved in polynomial time in general. In fact, we now show that various variants of LETTER BOXED are NP-Complete. We start by proving that the problem is in NP. The main observation to obtain this result is that any solvable instance of S-SIDES LETTER BOXED can also be solved with at most $\text{poly}(S, |\Sigma|, n)$ words. Therefore, we can non-deterministically guess a certificate of polynomial size.

▶ **Lemma 9.** *For any S, S-*SIDES LETTER BOXED *is in* NP*.*

We now prove that LETTER BOXED is NP-Hard even assuming that the dictionary consists of a single word ($|D| = 1$) and that the player is required to solve the puzzle with a single word ($k = 1$). Specifically, we show a polynomial-time reduction from POSITIVE NOT-ALL-EQUAL 3-SAT, defined as follows.

▶ **Definition 10** (POSITIVE NOT-ALL-EQUAL 3-SAT)**.** *In POSITIVE NOT-ALL-EQUAL 3-SAT, one is given a set V of boolean variables and a set $C \subseteq \binom{V}{3}$ of clauses each containing exactly three distinct positive literals (non-negated variables). The goal is to decide whether there exists a truth assignment of the variables such that each clause contains both a* true *and a* false *variable.*

POSITIVE NOT-ALL-EQUAL 3-SAT is known to be NP-Complete [2]. We use this fact to show the following.

▶ **Theorem 11.** LETTER BOXED *is* NP*-Complete even if $|D| = 1$ and $k = 1$.*

Proof. We show that the problem is NP-Hard by reducing from POSITIVE NOT-ALL-EQUAL 3-SAT. Let (V, C) be an instance of POSITIVE NOT-ALL-EQUAL 3-SAT. We will first build an instance of LETTER BOXED $(\Sigma, D, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4)$ in which the sets $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$ may have different cardinalities, and then show how to pad the instance so that each multiset has the same cardinality. We define the alphabet to be $\Sigma = V \cup C \cup \{\#\} \cup \{*_v \mid v \in V\}$, where $\#$ is a special character, and so is $*_v$ for each $v \in V$. We set $\Gamma_4 = \varnothing$ and Γ_1 contains only the character $\#$ with multiplicity 1 (recall that the same character can be used multiple times). For a variable $v \in V$ let $\eta(v)$ be the number of clauses containing v. We set $\Gamma_2 = \Gamma_3$ and for each $v \in V$ they contain the characters v with multiplicity $\eta(v)$ and $*_v$ with multiplicity $\eta(v) - 1$, while for each $c \in C$ they contain the character c with multiplicity 2. We are left with defining the only string in the dictionary. We call such string σ. The string σ starts with $\#$, then for each $v \in V$ we append the following string where v is repeated $\eta(v)$ times:

$$v \underbrace{*_v\ v\ \ldots\ *_v\ v}_{\substack{*_v\ v \text{ repeated} \\ \eta(v)-1 \text{ times}}} \#. \tag{1}$$

Note that only Γ_2 and Γ_3 contain v and $*_v$. Therefore, intuitively, this first string forces the player to choose on which side to collect all the v's and on which side to collect all the $*_v$'s, effectively assigning a truth value to the variable v. Then, for each $v \in V$ we append the following string, where $*_v$ is repeated $\eta(v) - 1$ times, required for cleanup:

$$\underbrace{*_v\ \#\ *_v\ \#\ \cdots\ *_v\ \#}_{*_v\# \text{ repeated } \eta(v)-1 \text{ times}}. \tag{2}$$

Finally, for each clause $c \in C$ that contains variables v_1, v_2, v_3 we append the following string to σ:

$$v_1\ c\ \#\ v_2\ c\ \#\ v_3\ c\ \#\ c\ \#. \tag{3}$$

Intuitively, if all of v_1, v_2, v_3 have been first selected from the same side (either Γ_2 or Γ_3), it will not be possible to pass through all the four characters c that are present in $\Gamma_2 \cup \Gamma_3$. Otherwise, if at least one of v_1, v_2, v_3 was selected from Γ_2 and at least one from Γ_3 it will be possible to pass though all the c's. Therefore the clause behaves as a not-all-equal clause.

We now provide a more detailed argument. We will show that (V, C) is solvable if and only if the instance $(\Sigma, \{\sigma\}, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4)$ is solvable with $k = 1$.

Suppose (V, C) is solvable and let $\phi : V \to \{T, F\}$ be an assignment that satisfies all the clauses (in the not-all-equal sense). We show how to follow the string σ while collecting all the characters on the square. The player starts on the side Γ_1 from #. Then, for each $v \in V$, if $\phi(v) = T$ the player moves first to Γ_2 and then alternates between Γ_3 and Γ_2 to process the characters of Equation (1) – if instead $\phi(v) = F$ it moves first to Γ_3 and then alternates. Similarly, to process the characters of Equation (2), if $\phi(v) = T$ (resp. $\phi(v) = F$) the player alternates between Γ_2 (resp. Γ_3) and Γ_1. At this point the characters we are missing from Γ_2 (resp. Γ_3) are those associated with the clauses and those associated with the variables v such that $\phi(v) = F$ (resp. $\phi(v) = T$). Consider the string associated with the clause $c = (v_1, v_2, v_3)$ (Equation (3)). We collect v_1, v_2, v_3 by going to the side where they still have to be collected (either Γ_2 or Γ_3) and collect the c by switching sides. Since the clause is satisfied by ϕ, at least one of v_1, v_2, v_3 will be collected on Γ_2 and at least one of them will be collected on Γ_3. Therefore, the first three c's will be collected without repetitions. Thus, thanks to the last portion of the string (# c #) all the four occurrences of c in $\Gamma_2 \cup \Gamma_3$ can be collected. Since this holds for all clauses, at the end of processing σ, we have collected all characters from $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3$.

Suppose now that the instance $(\Sigma, \{\sigma\}, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4)$ is solvable with a single word σ. Then, the player is forced to start from Γ_1 and, for each variable v, decide whether to start from Γ_2 or Γ_3: if the player starts from Γ_2 we set $\phi(v) = T$ otherwise we set $\phi(v) = F$. Consider now each portion associated with a clause $c = (v_1, v_2, v_3)$ (Equation (3)). Since there are exactly $2 \cdot \eta(v)$ characters equal to v in σ for each variable v, in order for the instance to be solved the player must have taken each of v_1, v_2, v_3 on the opposite side that was first used when processing the portion associated with Equation (1). Suppose that all of v_1, v_2, v_3 are taken on the same side. Suppose, without loss of generality, that this side is Γ_2. Then, the first three c's of Equation (3) must be taken on Γ_3. But then at most one of the two c characters can be covered from Γ_2 and the instance would not be solved. Therefore, under the hypothesis that the instance is solved, for each portion associated with a clause (Equation (3)), at least one of v_1, v_2, v_3 must be taken on side Γ_2 and at least one must be taken on side Γ_3. In other words, for each clause $c = (v_1, v_2, v_3)$, we have $\phi(x) = T$ for at least one $x \in \{v_1, v_2, v_3\}$ and $\phi(y) = F$ for at least one $x \in \{v_1, v_2, v_3\}$.

Therefore, we have proved that the problem where $|\Gamma_1|, |\Gamma_2|, |\Gamma_3|, |\Gamma_4|$ can differ is NP-Hard. We now show how to pad the previous instance so that the multisets all have the same cardinality. Consider the previous instance. Add a new character τ. Note that $|\Gamma_2| = |\Gamma_3| > |\Gamma_1| = 1$. Add $|\Gamma_2| - 1$ copies of τ to Γ_1 and add $|\Gamma_2|$ copies of τ to Γ_4. Note that now all multisets have the same cardinality. Finally, append $2|\Gamma_2| - 1$ characters τ to the string σ. Observe that the previous instance is solvable if and only if this padded instance is solvable. This concludes the proof. ◀

Our construction can be easily generalized to show that S-SIDES LETTER BOXED with $k = 1$ and $|D| = 1$ is NP-Hard for each $S \geq 3$.

We now show that LETTER BOXED is NP-Complete even when all the words are assumed to have length 5. This rules out the possibility of designing a polynomial-time algorithm for the case in which $L = O(1)$.

To prove hardness we reduce from the classical 3D MATCHING problem, which has long been known to be NP-Hard [4]. We recall the problem definition below.

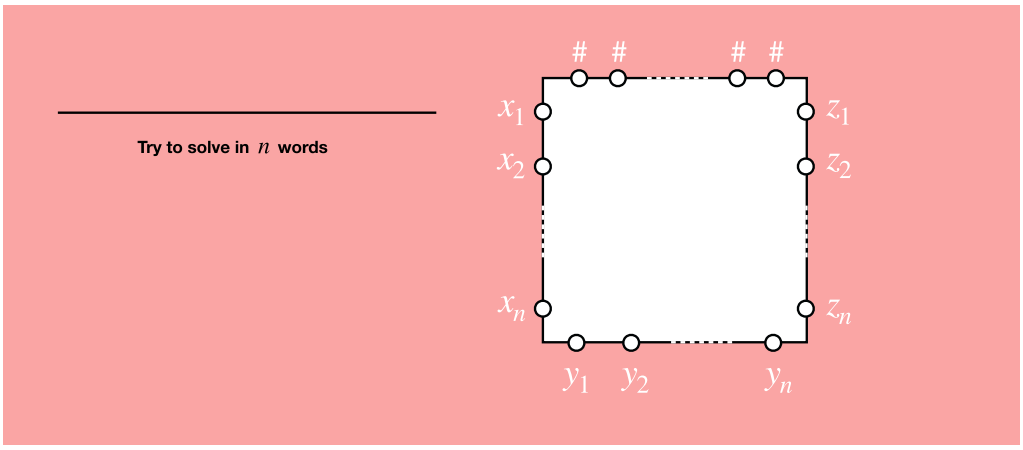

Figure 6 The LETTER BOXED instance constructed from the 3D MATCHING instance in the proof of Theorem 13.

▶ **Definition 12** (3D MATCHING). *In 3D MATCHING one is given as input three disjoint sets $X \cup Y \cup Z$ with $|X| = |Y| = |Z| = n$ and a collection $T \subseteq X \times Y \times Z$ of triples. The problem asks to decide whether there exists a subset $S \subseteq T$, with $|S| = n$, such that every element in $X \cup Y \cup Z$ appears in exactly one triple in S.*

▶ **Theorem 13.** LETTER BOXED *is* NP*-Complete even if all words in D have length* 5.

Proof. To show NP-Hardness, we now present a reduction from 3D MATCHING (Definition 12). Specifically, consider an instance of 3D MATCHING defined by three disjoint sets $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$ and $Z = \{z_1, \dots, z_n\}$ and a family $T \subseteq X \times Y \times Z$ of triples. We construct an instance of LETTER BOXED as follows. We take as alphabet $\Sigma = X \cup Y \cup Z \cup \{\#\}$, where $\#$ is a special character that does not belong to $X \cup Y \cup Z$, and as dictionary $D \subseteq \Sigma^5 \subseteq \Sigma^*$ given by $D = \{\#x_i y_j z_l \# \mid (x_i, y_j, z_l) \in T\}$. We set $k = n$ and take the four multisets to be:

$$\Gamma_1 = \underbrace{\{\#, \dots, \#\}}_{n \text{ times}}, \qquad \Gamma_2 = X, \qquad \Gamma_3 = Y, \qquad \Gamma_4 = Z.$$

The instance is illustrated in Figure 6.

We now show that the original 3D MATCHING instance is solvable if and only if the constructed Letter Boxed puzzle is solvable with at most k words.

Suppose that there exists a subset $S \subseteq T$ such that $|S| = n$ and, for each $v \in X \cup Y \cup Z$, v appears in exactly one triple in S. We construct a solution to the Letter Boxed puzzle as follows: order the triples in S arbitrarily, and let $w^{(i)}$ be the word $\#xyz\#$ where (x, y, z) is the ith triple in S in our arbitrary order. Set $\texttt{words} = (w^{(1)}, \dots, w^{(n)})$ and

$$\texttt{sides} = (\underbrace{1, 2, 3, 4, 1, \dots, 1, 2, 3, 4, 1}_{1,2,3,4,1 \text{ repeated } n \text{ times}}),$$

it is easy to see that $(\texttt{words}, \texttt{sides})$ is a valid solution of length k to the Letter Boxed puzzle, since S must have the property that it covers every element in $X \cup Y \cup Z$.

Now, suppose that the Letter Boxed puzzle is solvable with at most k words, as witnessed by a solution $(\texttt{words}, \texttt{sides})$, where $\texttt{words} = (w^{(1)}, \dots, w^{(k)})$. By construction, each $w^{(i)} = \#x^{(i)}y^{(i)}z^{(i)}\#$ for some $(x^{(i)}, y^{(i)}, z^{(i)}) \in T$.

Consider the set $S = \{(x^{(i)}, y^{(i)}, z^{(i)}) \mid i = 1, \ldots, n\}$. Since $\#x^{(i)}y^{(i)}z^{(i)}\#$ is in the dictionary, it follows that $S \subseteq T$. Furthermore, for every $x \in X$ there is one (and only one) i such that $x = x^{(i)}$ for otherwise the Letter Boxed solution would not be able to cover all the letters on every side with only n words. The same applies to Y and Z, concluding the proof. ◀

Our NP-Hardness proofs hold in the standard LETTER BOXED problem with four sides. However, increasing the number of sides can only make the problem harder. Indeed, we show:

▶ **Theorem 14.** *For every integer $S \geq 2$, we have:* S-SIDES LETTER BOXED $\leq_p$ $(S+1)$-SIDES LETTER BOXED, *where the starting instance has number of letters on each side $n \geq 2$. Moreover, in the instance with $S+1$ sides, the size of the dictionary $|D|$ increases by at most a constant factor, the maximum length L increases by one, the size of the alphabet $|\Sigma|$ increases by 3, the number of characters on each side increases by one, and the budget of words k increases by $O(S+n)$.*

3 Pips

In this section, we study the newest of the New York Times games: Pips. This game has a very rich structure, and its expressive set of constraints makes it computationally difficult to find the solution to a Pips puzzle. In particular, in this section, we show that the problem of deciding whether a Pips puzzle has a solution is NP-Complete.

Recall that a Pips puzzle is made up of a board (of arbitrary shape) and some domino tiles. Each tile is divided into two adjacent squares, each of which has a numerical value on it, represented by a certain number of black dots. The board also has constraints in the form of disjoint, connected regions with a constraint indicator signaling that the tile squares placed inside the region have to (i) have the same numerical value as one another ("=" constraints), or have pairwise distinct numerical values ("≠" constraints), (ii) sum up to a specific numerical value n ("n" constraints), or (iii) sum up to a value that is either greater than or smaller than a specific numerical value n ("$> n$" and "$< n$" constraints). The goal of the game is to place the domino tiles so as to fill the entire board in a way that satisfies all of the constraints specified.

We consider the following natural problem.

PIPS
Input: A Pips puzzle, specified by a collection of domino tiles, a board and a set of constraints.
Question: Does there exist a valid solution to the instance?

We show that PIPS is NP-Complete and, in particular, we will show a reduction from the POSITIVE PLANAR 1-IN-3-SAT problem, defined below.

▶ **Definition 15** (POSITIVE PLANAR 1-IN-3-SAT). *An instance of POSITIVE PLANAR 1-IN-3-SAT consists of a set of boolean variables and a set of clauses, each containing exactly three distinct positive variables. The objective is to determine a truth assignment to the variables such that each clause contains exactly one true variable.*

The graph associated to the formula is a graph with a vertex for each variable, a vertex for each clause, and an edge between a variable vertex and a clause vertex if the variable appears in the clause. We restrict to instances where such a graph is planar and can be embedded in a rectilinear *fashion (see Figure 7). That is, with the variables aligned along a straight line and three-legged clauses positioned above and below the variables. The edges between variables and clauses are embedded as straight segments.*

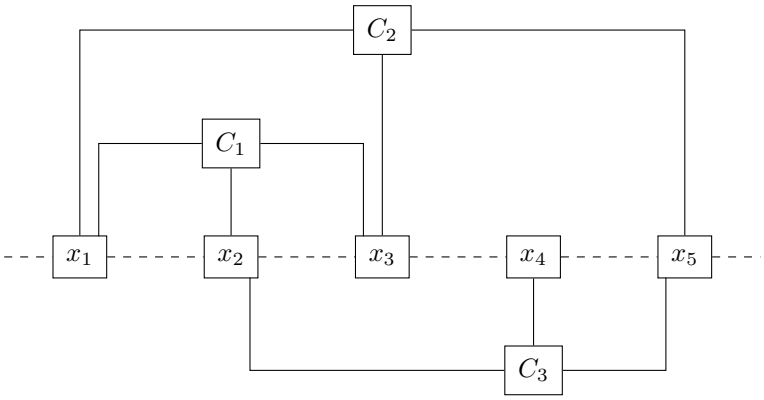

Figure 7 Rectilinear embedding of the graph associated with the boolean 1-in-3-SAT formula $(x_1, x_2, x_3) \wedge (x_1, x_3, x_5) \wedge (x_2, x_4, x_5)$.

POSITIVE PLANAR 1-IN-3-SAT is known to be NP-Hard even when restricted to rectilinear instances [11].

We now prove that PIPS is NP-Hard by a reduction from POSITIVE PLANAR 1-IN-3-SAT. The high level idea is to take the rectilinear embedding of the formula and replace each variable vertex with a variable gadget and each clause vertex with a clause gadget. Each variable gadget has a constraint to force all the tiles to be the same (either containing two zeroes or two ones), and each clause gadget has a constraint to ensure that the sum of the tiles in the clause is one: this will imply that exactly one variable is set to true.

▶ **Theorem 16** (Theorem 2 paraphrased). PIPS *is* NP-*Complete, even if there are only two types of domino tiles (one containing two zeroes, and one containing two ones) and only constraints of type "=" and "n".*

In the New York Times version, the only numbers that can appear on Pips tiles are numbers from 0 to 6. We note that if one can assign arbitrary non-negative integer values to the domino tiles, then Pips becomes (weakly) NP-Hard even restricted to instances with a single "n" constraint. Indeed, it contains the classical subset sum problem as a special case.

▶ **Theorem 17.** *PIPS, with arbitrary non-negative integer numbers on tiles, and a single "n" constraint is weakly* NP-*Hard.*

4 Strands

In this section we present our results on Strands. Recall that, in this game, one is given a grid of characters and the goal is to cover the grid with disjoint valid words (in the New York Times version, the player receives hints from the game in an interactive way). We will focus on the problem of finding a valid partition of the grid into words from the dictionary (according to the adjacency rules of the game) and hence deciding whether a puzzle is solvable. We consider the simplest instantiation of this problem, in which there is no interactivity between the player and the game, and the player simply needs to decide, given the dictionary and the grid, whether at least one possible solution exists.

We define an instance of Strands as follows.

▶ **Definition 18** (Strands Instance). *A* Strands instance *is a tuple* (Σ, D, M)*, where* Σ *denotes a finite alphabet,* $D \subseteq \Sigma^*$ *is a finite set of strings over* Σ*, referred to as the* dictionary*, and* $M \in \Sigma^{n \times m}$ *is a grid of characters from* Σ *with* n *rows and* m *columns.*

We consider the following decision problem. Given an instance (Σ, D, M), decide whether M can be partitioned into a sequence of words $w^{(1)}, \ldots, w^{(k)}$ such that $w^{(i)} \in D$ for each i. Here, each word is required to appear as a sequence of adjacent characters according to the Strands adjacency rule: when reading off a word, the next character can appear in any of the eight cells neighboring the current character (in particular, diagonal movements are allowed). Note that a contiguous sequence of characters on the grid M can be read in any direction, and that each word can only use each entry of the grid M at most once.

Our main result in this section is that this problem is NP-Complete even with a constant-size dictionary of constant-length words.

On Spangrams

Recall that, in the New York Times' version of the game, every puzzle contains a *spangram*: a word that connects two opposite sides of the grid. In general, requiring the game to have a spangram, rules out the possibility of having the maximum word length L bounded by a constant, since as the grid grows, the minimum length of a spangram would grow with it. So in order to prove our lower bound in the stronger setting of $L = O(1)$, we shall not impose that instances contain a spangram. We note, however, that adding a spangram, e.g. at the bottom of the instance, does not make the problem easier, and hence our hardness results will transfer to the set of instances that do contain a spangram via a straightforward reduction.

A simple reduction from Flow Free

The Strands problem, as defined above, shares some similarities with the *Flow Free* (or *Zig-zag Numberlink*) game, which was shown to be NP-Hard [1]. A Flow Free instance consists of k terminal pairs – each pair represented by the same color – placed in an $n \times m$ grid. A solution to the instance is defined as a set of k disjoint non-diagonal paths, each connecting the two terminals of a pair and collectively covering all vertices of the grid. (We refer the reader to [1] for more details on the game) One can prove NP-Hardness of Strands by giving a polynomial-time reduction from Flow Free: given a Flow Free instance with colors $C_1, \ldots, C_k$ and grid size $n \times m$, consider a $n \times m$ Strands grid with $\Sigma = \{C_1, \ldots, C_k, B, W\}$, where each cell containing a terminal C_i is filled with the letter C_i, while the remaining cells are filled with black B and white W, arranged in a checkerboard pattern. Define the dictionary D as $D = \bigcup_{i=1}^{k} D_{C_i}$ where each D_{C_i} corresponds to the possible paths connecting pair of terminals of color C_i and is given by

$$D_{C_i} = \begin{cases} \{C_i\,(B\,W)^l\,B\,C_i \mid 0 \le l \le m \cdot n\} & \text{if both terminals lie on white cells;} \\ \{C_i\,(W\,B)^l\,W\,C_i \mid 0 \le l \le m \cdot n\} & \text{if both terminals lie on black cells;} \\ \{C_i\,(B\,W)^l\,C_i \mid 0 \le l \le m \cdot n\} & \text{if the terminals lie on cells of different colors.} \end{cases}$$

It is then not hard to check (but also not all that important, given the stronger results below) that this reduction is correct.

However, observe this reduction from Flow Free requires a polynomially large alphabet, dictionary, and word lengths, and hence one needs a different reduction to prove the stronger statement we aim for.

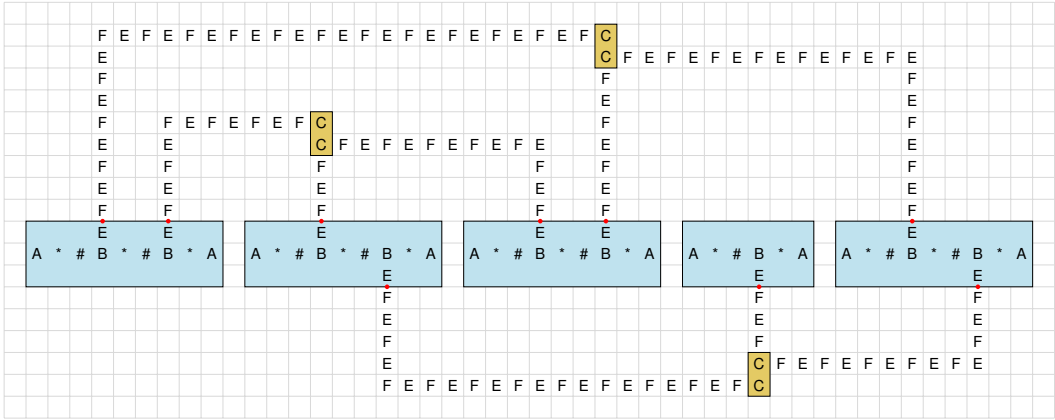

Figure 8 Strands instance corresponding to the POSITIVE PLANAR 1-IN-3-SAT instance given by the formula $(x_1, x_2, x_3) \wedge (x_1, x_3, x_5) \wedge (x_2, x_4, x_5)$. Variable gadgets are highlighted in blue, while clause gadgets are highlighted in yellow.

Main Result

We now prove the main theorem of this section.

▶ **Theorem 3.** *Given a Strands instance (Σ, D, M) determining whether M can be partitioned into valid words from D is* NP*-Complete, even if $|\Sigma|, |D|, L = O(1)$, where L denotes the maximum length of a word in D.*

Proof. We start by showing that the problem is in NP. A certificate consists of two matrices. The first is $V_1 \in \{S, E, C\}^{n \times m}$ and the second is $V_2 \in \{u, r, l, d, ur, ul, dr, dl\}^{n \times m}$. Intuitively, V_1 indicates for each cell whether it is the start, end, or continuation of a word. Instead, V_2 indicates the direction to move from the current cell to the next one to build a word. Given these two matrices, one can verify in polynomial time whether they encode a partition of M, and for each word in that partition, ordered according to the directions given by V_2, one can check whether the word is in D. Thus, the problem belongs to NP.

To establish that *Strands* is NP-hard, we provide a reduction from POSITIVE PLANAR 1-IN-3-SAT with the restrictions described above (see Definition 15). At a high level, given a boolean formula, our reduction substitutes each variable vertex in the corresponding rectilinear embedding of the graph with a vertex gadget, each clause vertex with a clause gadget, and each edge with an edge gadget (see Figure 8 for an example).

More precisely, the Strands instance of our reduction has alphabet and dictionary equal to: $\Sigma = \{A, B, C, *, \#, E, F, \sqcup\}$, and $D = \{\sqcup, A, A*, *\#BE, \#B*, EF, FCC\}$. We now specify how to construct the grid M of the Strands instance. It will be useful to separate the various gadgets with "blank" cells. Each such cell is represented by the character $\sqcup \in \Sigma$, which is also the only valid word in D containing this character. Consequently, the "blank" cells can always be collected in a unique way, and the character $\sqcup$ will not be displayed for simplicity.

Vertex gadget. Each variable x_i is depicted as a $3 \times (3k_i + 3)$ rectangle, where k_i denotes the number of clauses in which x_i appears, as displayed in Figure 9.

Specifically, the first and second columns of the gadget are identical for every variable, followed by a central 3×3 module, which is repeated as many times as the number of clauses in which the corresponding variable appears. In the kth module, the character E represents the starting point of the kth edge connecting the variable to a clause (the order is determined

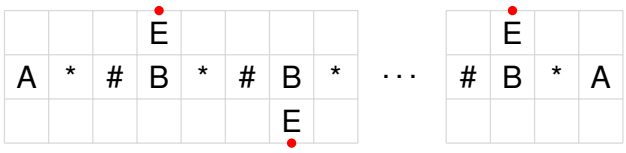

Figure 9 Variable gadget associated to a variable vertex. Each E character is the starting point of an edge (we highlight this with a red dot).

by scanning the edges of the rectangular embedding along the line of variables, from left to right). This character may appear either below or above the corresponding B, depending on whether the kth edge leaving the variable node starts from the bottom or the top of the square. Finally, the last column of the gadget is equal to the first one.

Note that the character $A \in \Sigma$ appears only in words $A \in D$ and in $A* \in D$, whereas the character $*$ appears only in $A* \in D$, $*\#BE \in D$, and $\#B* \in D$. Therefore, if the first column of the gadget associated with variable x_i is covered using word A, then the first occurrence of $*$ must be covered with $*\#BE$. Consequently, the remainder of the gadget must be covered exclusively using $*\#BE$ and $A*$. Conversely, if the first column is covered using $A*$, then the rest of the gadget must be covered only with $\#B*$ and word A. The first method covers all cells of the gadget and will be interpreted as the true assignment for the variable x_i, whereas the second method leaves the E's uncovered and corresponds to a false assignment.

Clause gadget. Each clause (x_i, x_j, x_k) is associated to a clause gadget of size 2×1 consisting of exactly two characters $C \in \Sigma$. Assuming that variables x_i, x_j and x_k are sorted according to their order in the embedding of the graph, the clause gadget will be placed on the same column of one of the E characters exiting from variable x_j, leaving an odd number of rows between the E of the variable gadget and the first C of this gadget. Thanks to the rectilinear embedding, the edge exiting from x_i will arrive from the left of the clause, the one exiting from x_k will arrive from the right, and that exiting from x_j will arrive from below or above depending whether the clause is above or below the line of variables.

Edge gadget. Each edge connecting a variable to a clause is represented by an alternating sequence of E's and F's of even length, starting from an appropriate E in the variable gadget. Thanks to the rectilinear embedding of the graph, each path is either vertical, or it consists first of a vertical segment and then of an horizontal segment (going left or right).

Depending on the position of the first E, the sequence will connect to one of the two C's of the clause gadget. Specifically, if the Manhattan distance between the E in the variable gadget and the closest C in the clause gadget is even, the even-length sequence will connect to the closest C, otherwise it will connect to the other C in the clause gadget.

Thus, every edge starts with the character E inside the variable gadget and ends with the character F immediately adjacent to the clause gadget. Since the graph is planar and every gadget is well-separated from the others, if the false assignment is chosen, then every edge gadget leaving x_i must be covered exclusively using EF, until reaching and covering the last occurrence of F along that edge path. Conversely, if the true assignment is chosen for the variable x_i, then all the E's of the gadget associated with x_i have already been covered, and the first F of an outgoing edge must be covered using the word EF together with the second occurrence of E. Therefore, the entire path must be covered using EF until reaching the last F, which remains uncovered this time. This last F must then be covered using the word $FCC \in D$, thereby covering the whole clause gadget.

Note that it could be possible to move diagonally from the last F of an edge gadget to the last F of another one; however, there is no word in D containing two F's. Therefore, the edge gadgets cannot be exploited to cheat.

Note that the width of the grid is linear in the sum of the degrees of the variable vertices, and it is thus linear in the number of clauses. The height is also linear in the number of clauses since each clause gadget has constant height. Thus, the grid has polynomial size and the reduction can be carried out in polynomial time.

We now aim to show that the original formula is satisfiable (in 1-in-3-SAT sense) if and only if the corresponding Strands instance is solvable. If the formula is satisfiable, consider a satisfying truth assignment for the variables and cover the corresponding gadget associated with each variable according to the truth assignment. Then, cover every edge gadget according to the unique covering determined by the truth assignment of the variable, as described previously. Let C_k be a clause of the formula. Since the truth assignment satisfies the formula, there exists exactly one true variable x_i appearing in C_k. Consequently, the corresponding clause gadget is fully covered by the covering of the edge connecting x_i and C_k, since it covers the last occurrence of F using FCC. Thus, every gadget associated with a clause is completely covered. Since no other true variable appears in C_k, all other edges entering C_k connect a false variable to C_k. Hence, they must have been fully covered using exclusively EF, without using the C's of C_k. Therefore, each edge gadget has been fully covered. In conclusion, every gadget is completely covered; hence, the Strands instance is solved.

If the corresponding Strands instance is solvable, consider a valid partition of the grid M. Since, as previously described, there are only two ways to cover the variable gadgets, this solution encodes a valid truth assignment to the variables. Let C_k be a clause. Since each gadget associated with a clause is fully covered, there exists an edge entering C_k which is fully covered using FCC; in other words, the edge connects C_k to a true variable. Therefore, there is at least one true variable appearing in C_k. By contrast, suppose that at least two true variables appear in C_k. Consider their respective edges connecting to C_k: both connect a true variable to C_k and both must be fully covered, therefore both would use the two C's of C_k, which is impossible. Therefore, there is exactly one true variable for each clause and the truth assignment satisfies the formula. ◀

While diagonal movements are allowed in Strands, this does not make any difference in our reduction, in fact:

▶ **Corollary 19.** *Given a Strands instance* (Σ, D, M) *with* $|\Sigma|, |D|, L = O(1)$ *determining whether* M *can be partitioned into valid words without using diagonal movements is* NP-*Complete.*

Proof. The construction of Theorem 3 is such that the Strands instance is either not solvable (not even with diagonal movements) or it is solvable without diagonal movements. ◀

Note that our main construction uses words of length 1 which are forbidden in the real version of the game. However, in our instances, this could easily be fixed by replacing each cell with a corresponding 3×3 block containing the same character 9 times. We now show that this result is more general. Indeed, it is possible to enlarge certain Strands instances so that the minimum word length is larger than some constant. Simply replacing each cell with a 3×3 block containing a single character is not enough in general as adjacent cells with the same character can be problematic since we cannot force the user to cover the blocks sequentially. This can be fixed by putting an appropriate perimeter around the blocks, as shown by the following result.

▶ **Theorem 20.** *Suppose that the Strands instance $I = (\Sigma, D, M)$ is either impossible to solve or it can be solved without diagonal movements. Suppose also that $\ell, L = O(1)$ are respectively the minimum and maximum length of a word in D. Then, there exists a Strands instance $I_2 = (\Sigma_2, D_2, M_2)$ that is solvable if and only if I is solvable and moreover: (i) if I_2 is solvable, then it is solvable without diagonal movements, and (ii) $|\Sigma_2| = O(|\Sigma|)$, $|D_2| = O(|D|)$ and the minimum and maximum word lengths are, respectively, $9 \cdot \ell$ and $9 \cdot L$.*

5 Tiles

New York Times games aficionados will have undoubtedly noticed that Tiles offers a more relaxing puzzling experience than some of the other titles in the catalog. A typical instance of Tiles can be solved with little effort, aside perhaps from a little strain to the eye. This is no accident. In this section, we show that any (solvable) instance of Tiles can be solved by simply repeatedly making arbitrary valid moves, while favoring non-teleport whenever possible. A consequence of this result is that the problem of deciding whether an instance of Tiles is solvable is in P, and in fact one can make such a decision in time linear in the size of the puzzle. Note that we deem a Tiles puzzle solvable if all of its features can be deleted. In general, we will show that this is true if and only if all of its features can be deleted in a single, unbroken combo. An instance is solvable with a single combo if the player can delete all the features in the instance from all of its tiles by starting at some tile and iteratively making one of two types of moves: (i) a standard move, in which the player selects a new tile that shares at least one non-deleted feature with the current tile and all the common non-deleted features between the two tiles are deleted, or (ii) a forced teleport move, in which the player, starting from a tile whose features have all been deleted, selects an arbitrary new tile.

At the end of this section, we turn to the question of whether an instance can be solved without teleport moves. We show that this can be decided in linear time for instances in which any two tiles share at most one feature.

We begin by giving a formal definition of a Tiles instance.

▶ **Definition 21** (Tiles Instance)**.** *An instance of the Tiles problem $(F, \mathcal{T})$ is defined by a set $F = \{f_1, \ldots, f_m\}$, the* features *of the instance, and a multiset $\mathcal{T} = \{T_1, \ldots, T_n\}$ of subsets of F, the* tiles*.*

Intuitively, we think of every $T \in \mathcal{T}$ as the collection of all features that appear on tile T (each feature can appear at most once). The size of a Tiles instance is the quantity $\sum_{T \in \mathcal{T}} |T|$.

We now prove that solving a Tiles instance can be done in polynomial time. The main idea is that, for any feature, the parity of the number of tiles in which the feature appears is an invariant: it does not change as the player makes moves. This observation allows us to prove Theorem 4.

▶ **Theorem 4.** *Given any instance of Tiles, the following are all equivalent:*

1. *The instance is solvable,*
2. *The instance is solvable with a single, unbroken combo,*
3. *Each distinct feature in the instance appears in an even number of tiles.*

In particular, there exists an algorithm that decides whether an instance of Tiles is solvable in time linear in the size of the instance.

We conclude this section by showing that, given an instance of Tiles in which any two tiles share at most one feature, we can decide in linear time whether this is solvable without teleport moves. Note that solving the puzzle without teleporting, implies solving the puzzle with a single combo.

Each instance of Tiles naturally induces a bipartite graph, having features on one side, and tiles on the other. Adopting this perspective will be beneficial to the result that follows.

▶ **Definition 22** (Structure Graph of a Tiles Instance). *Given an instance $(F, \mathcal{T})$ of Tiles, its* structure graph *is the bipartite undirected graph $G = (F \cup \mathcal{T}, E)$, where:*

$$E = \{\{T, f\} \mid T \in \mathcal{T} \text{ and } f \in T\}.$$

The elements of F are the feature vertices*, and the elements of $\mathcal{T}$ are the* tile vertices*.*

Note that the structure graph of a Tiles instance is computable in linear time in the size of the instance.

▶ **Definition 23.** *The* sharing number *of a Tiles instance $(F, \mathcal{T})$ where $\mathcal{T} = \{T_1, \ldots, T_n\}$ is defined as:*

$$\text{share}(F, \mathcal{T}) = \max_{i,j \in [n], i \neq j} |T_i \cap T_j|.$$

It can be proved that solving an instance with sharing number 1, is equivalent to finding a Eulerian trail in the structure graph starting and ending on Tile vertices. Specifically, we have the following.

▶ **Theorem 24.** *An instance of Tiles with sharing number 1 is solvable without teleport moves if and only if its structure graph admits a Eulerian trail that starts and ends on Tile vertices (potentially not distinct from one another). In particular, there is an algorithm that decides whether such an instance is solvable without teleport move in time linear in the size of the instance.*

6 Conclusions

In this work, we considered a variety of natural computational problems arising from the New York Times games. For **Letter Boxed**, we give a complete characterization of the complexity of the game, across a host of different parameter settings. We then showed that deciding whether an arbitrary **Pips** puzzle is solvable is NP-Hard, even if all the tiles that occur are of the form $1 - 1$ or $0 - 0$, and all the constraints are "=" constraints or "n" constraints. Moreover we show that this problem is (at least) weakly NP-Hard even if every instance contains only one constraint, and all the tiles have to be distinct from one another. We also showed that deciding whether an instance of **Strands** can be covered with disjoint words from the dictionary is NP-Hard, even in the restricted case where the dictionary contains only a constant number of words, each of which is of constant length. Finally, we showed that deciding whether an instance of **Tiles** is solvable is in P, and in fact one can make such determination in time linear in the size of the instance. The main unsolved problem raised in this paper is determining the complexity of deciding whether a Tiles instance with arbitrary sharing number is solvable without teleport moves. We leave that as a challenge to the many fans of the New York Times games.

References

1 Aaron Adcock, Erik D. Demaine, Martin L. Demaine, Michael P. O'Brien, Felix Reidl, Fernando Sánchez Villaamil, and Blair D. Sullivan. Zig-Zag Numberlink is NP-Complete. *Journal of Information Processing*, 23(3):239–245, 2015. doi:10.2197/ipsjjip.23.239.

2 Andreas Darmann and Janosch Döcker. On a simple hard variant of Not-All-Equal 3-Sat. *Theoretical Computer Science*, 815:147–152, 2020. doi:10.1016/j.tcs.2020.02.010.

3 Laurent Gourvès, Ararat Harutyunyan, Michael Lampis, and Nikolaos Melissinos. Filling crosswords is very hard. *Theoretical Computer Science*, 982:114275, 2024. doi:10.1016/j.tcs.2023.114275.

4 Richard M Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2009. doi:10.1007/978-3-540-68279-0_8.

5 Charlotte Klein. Inside The New York Times' Big Bet on Games. Vanity Fair. https://www.vanityfair.com/news/inside-the-new-york-times-big-bet-on-games, 2023. Accessed: 2025-06-26.

6 Jonathan Knight. The Times' New Word-Search Game is Genius. The Atlantic. https://www.theatlantic.com/technology/archive/2024/03/strands-word-search-game/677656/, 2024. Accessed: 2025-06-26.

7 Elie Levine. Putting a new twist on a classic puzzle. The New York Times. https://www.nytimes.com/2024/03/04/crosswords/strands-word-search-game.html, 2024. Accessed: 2025-06-26.

8 Daniel Lokshtanov and Bernardo Subercaseaux. Wordle is NP-hard. In Pierre Fraigniaud and Yushi Uno, editors, *11th International Conference on Fun with Algorithms, FUN 2022, Island of Favignana, Sicily, Italy, May 30 - June 3, 2022*, volume 226 of *LIPIcs*, pages 19:1–19:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.FUN.2022.19.

9 Aisha Majid. 100k club: Exclusive ranking of world's top paywalled news publishers. Press Gazette https://pressgazette.co.uk/paywalls/100k-club-exclusive-ranking-of-worlds-top-paywalled-news-publishers/, 2023. Accessed: 2025-09-12.

10 ValueAct Capital Management. Notice of exempt solicitation submitted pursuant to rule 14a-6(g). https://www.sec.gov/Archives/edgar/data/1351069/000119312524062156/d809296dpx14a6g.htm. Accessed: 2025-06-26.

11 Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is NP-hard. *J. ACM*, 55(2), 2008. doi:10.1145/1346330.1346336.

12 Jay Peters. The New York Times is starting to test its take on scrabble. The Verge. https://www.theverge.com/games/682063/the-new-york-times-nyt-games-crossplay-scrabble-pips, June 2025. Accessed: 2025-09-11.

13 Will Rosenbaum. Finding a winning strategy for wordle is NP-complete. *arXiv*, 2204.04104, 2022. doi:10.48550/arXiv.2204.04104.

14 Richard F. Shepard. Bambi Is a Stag, and Tubas Don't Go 'PahPah': The Ins and Outs of 'Across'. The New York Times Magazine https://www.nytimes.com/1992/02/16/magazine/bambi-is-a-stag-and-tubas-dont-go-pahpah-the-ins-and-outs-of-across.html, February 1992. Accessed: 2025-07-16.

15 Eric Switzer. New York Times Hot New Word Game Just Made Everyone Freak Out. TheGamer. https://www.thegamer.com/new-york-times-hot-new-word-game-just-made-everyone-freak-out/, 2024. Accessed: 2025-06-26.

16 The New York Times Company. Pulitzer prize winners. https://www.nytco.com/award-collection/2019-pulitzer-prize-winners/. Accessed: 2025-07-17. URL: https://www.nytco.com/award-collection/2019-pulitzer-prize-winners/.

17 The New York Times Company. The New York Times Passes 2m International Digital Subscribers. https://www.nytco.com/press/the-new-york-times-passes-2m-international-digital-subscribers/, June 2024.

18 The New York Times Company. Introducing Pips, a New Logic Game from New York Times Games. https://www.nytco.com/press/introducing-pips-a-new-logic-game-from-new-york-times-games/, August 2025. Press release.

19 Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003. URL: http://search.ieice.org/bin/summary.php?id=e86-a_5_1052.