# Scrabble is PSPACE-Complete

Michael Lampis[1], Valia Mitsou[2], and Karolina Sołtys[3]

[1] KTH Royal Institute of Technology, mlampis@kth.se
[2] Graduate Center, City University of New York, vmitsou@gc.cuny.edu
[3] Max Planck Institute für Informatik, ksoltys@mpi-inf.mpg.de

**Abstract.** In this paper we study the computational complexity of the game of Scrabble. We prove the PSPACE-completeness of a derandomized model of the game, answering an open question of Erik Demaine and Robert Hearn.
**Keywords:** *Scrabble, PSPACE-completeness, combinatorial games, computational complexity*

## 1 Introduction

In this paper we examine the computational complexity of optimal play in the game of Scrabble, a board game played by two to four players. In this game each player takes turns drawing lettered tiles randomly out of a bag and then attempting to place those tiles on a common $15 \times 15$ board, forming words. Points are awarded depending on the length of the formed words, the value of the letters used and various bonuses found on the board, with the winner being the player who has gathered the highest number of points at the end of the game.[4]

Having been invented in the US around the middle of the 20th century, Scrabble is now one of the most popular and well-known board games in the world. Besides the original english language version, Scrabble has been translated to dozens of other languages, while more than one hundred million Scrabble sets have been sold worldwide.

Since Scrabble is such a successful game, it becomes a natural question to determine the computational complexity of finding an optimal play. Similar questions have already been answered for several other popular board games, such as Othello, Chess and Checkers, typically classifying their complexity as either PSPACE or EXPTIME-complete ([3], [2], [5]). This is, however, complicated by the fact that, unlike those games, chance plays a non-negligible part in a match of Scrabble, as players don't know in advance the order in which tiles will be drawn. Still, much insight could be gained by investigating the complexity of a perfect-information version of Scrabble, where the order in which tiles will be drawn is known beforehand. In fact, this was listed as an open problem by Demaine and Hearn [1]. This is exactly the question we tackle in this paper by showing that this derandomized version of Scrabble is PSPACE-complete.

[4] For a fuller description of the board game of Scrabble see e.g. http://en.wikipedia.org/wiki/Scrabble/

This result on its own is probably not surprising, since most interesting board games are at least PSPACE-hard, and Scrabble is trivially in PSPACE from the fact that tiles cannot be removed from the board once they are placed. In addition to settling the complexity question though, we go about trying to understand what exactly makes the problem hard.

Informally, at any given round a Scrabble player is confronted with two tasks: deciding which word to form and deciding where to place it on the board. Though the tasks are not independent, since the formed word must be using some tiles already on the board, they are conceptually different and the hardness of the game could stem from either one. Put another way, it could be the case that deciding which word is best to play is easy if there is only one possible position where a word can be placed, or that deciding where to place the next word is easy if only one word can be made with the available tiles.

In fact, we will present two different hardness proofs arguing that both of these tasks are hard. In one reduction the players will be given appropriate tiles so that they will only have one possible word to play in each round, with a choice of two places to place it. In the other, players will be forced to play in a specific place on the board, but will be able to choose between two different words. In both cases, the problem of deciding optimal play will still turn out to be PSPACE-complete. Along the way, we can show that even a single-player version of the game, where one player tries to place all tiles, is NP-complete in both cases. Thus, we establish that during the course of a game, Scrabble players need to perform not one, but two computationally hard tasks, which is probably the reason why Scrabble is so much fun to play.

## 2 Our model of Scrabble - Definitions

Informally, the question we are trying to answer is, given a Scrabble position how hard is it to determine the best playing strategy? As mentioned, we will tackle this problem in a perfect information setting, where the contents of the bag and the order in which they are drawn are known in advance to both players (and therefore both players know each other's letters).

Moreover, since Scrabble is a finite game, in order to study its computational complexity we need to consider some unbounded generalization. The most natural way to go forward is to consider the game played on an $n \times n$ board. In addition, we assume that the bag initially contains a number of tiles that depends on $n$, since the restriction of the game where the bag contains a fixed number of tiles will yield a constant number of possible configurations, making the problem trivial.

Beyond the size of the board and the number of letters in the bag, we need to define an alphabet, a set of acceptable words and a rack size which will determine how many letters each player has on hand. All of these can be allowed to depend on the input, but since we are interested in proving hardness results we are happier when we can establish them even if those parameters are fixed constants. In fact, in Theorem 2 we prove that Scrabble is PSPACE-hard even with these restrictions, at the cost of making the reduction a little technical.

We will deal with a plain version of the game, where all letters have the same value and there are no premium positions on the board (clearly, the more general case with multiple values and possible premiums is harder). Also, for the most part we will assume that players are not allowed to exchange tiles or pass. Nevertheless, we will give arguments after Theorem 2 explaining why allowing players to pass does not affect our results.

Let us now give a more formal definition of the problem:

**Definition 1.** *We define a* Scrabble game $\mathcal{S}$ *to be an ordered quadruple* $(\Sigma, \Delta, k, \pi_0)$ *where:* $\Sigma$ *is a finite* alphabet*,* $\Delta \subset \Sigma^*$ *is a finite* dictionary*,* $k \in \mathbb{N}$ *is the size of the rack and* $\pi_0$ *is the initial position of the game, defined as below.*

**Definition 2.** *A* position $\pi$ *in a scrabble game is an ordered septuple* $(\mathcal{B}, \sigma, p, r^1, r^2, s^1, s^2)$*, where* $\mathcal{B} \in \mathbf{M}_{n\times n}(\Sigma)$ *is the* board*,* $\sigma \in \Sigma^*$ *is a sequence of lettered tiles called the* bag*,* $p \in \{1, 2\}$ *is the number of the active player,* $r^i$*, where* $i \in \{1, 2\}$*, are multisets with symbols from* $\Sigma$ *denoting the contents of the rack of the first and the second player respectively and* $s^i \in \mathbb{N}$*, where* $i \in \{1, 2\}$*, are the scores of the first and the second player respectively.*

**Definition 3.** *A* play $\Pi = \pi_1 \ldots \pi_l$ *is a sequence of positions such that, for all* $i$*,* $\pi_{i+1}$ *is attainable from* $\pi_i$ *by the active player by forming a* proper play *on the board.*

A proper play uses any number of the player's tiles from the rack to form a single continuous word (*main word*) on the board, reading either left-to-right or top-to-bottom. The main word must either use the letters of one or more previously played words, or else have at least one of its tiles horizontally or vertically adjacent to an already played word. If words other than the main word are newly formed by the play, they are scored as well, and are subject to the same criteria for acceptability. All the words thus formed must belong to the dictionary. After forming a proper play, the sum of the lengths of all words formed is added to the active player's points, letters used are removed from the player's rack and the rack is refilled up to $k$ letters (or less, if $|\sigma_i| < k$) with the appropriate number of letters forming the prefix of $\sigma_i$.

**Definition 4.** *A play* $\Pi = \pi_1 \ldots \pi_l$ *is* finished *if player* $l+1 \bmod 2$ *is unable to form a proper play, or if* $\sigma_l = \varepsilon$ *(i.e. the bag is empty). The* winner *of a finished play is the player with the greater number of points (draws are possible).*

We will establish PSPACE-hardness via two reductions from 3-CNF-QBF, the problem of deciding whether a quantified boolean formula is true. This is a well-known PSPACE-complete problem often used to establish hardness for games [4]. We are also interested in the variation of the game where there is only one player who tries to place all the tiles on the board, which we call Scrabble-Solitaire. Essentially the same constructions we present can also establish NP-hardness for Scrabble-Solitaire if one begins the reduction from 3-CNF-SAT.

## 3 Hardness due to placement of the words

In this section we prove that SCRABBLE is PSPACE-complete due to the ability of players to place their formed word in more than one places.[5]

We will first prove that the one-player version SCRABBLE-SOLITAIRE is NP-complete. PSPACE-completeness of SCRABBLE follows with slight modifications.

**Lemma 1.** SCRABBLE-SOLITAIRE *is NP-complete.*

Proving that the problem is in NP is straightforward. To estabilish the NP-hardness of SCRABBLE-SOLITAIRE, we will construct a reduction to this problem from 3-CNF-SAT. Given a 3-CNF propositional formula $\phi$ with $n$ variables $x_1, x_2, \ldots, x_n$ and $m$ clauses, we construct in polynomial time a polynomial-sized Scrabble-Solitaire game $\mathcal{S}$, such that $\phi$ is satisfiable iff $\mathcal{S}$ is solvable.

The general idea of the proof is as follows. We will create gadgets associated to variables, where the player will assign values to these variables. We will ensure that the state of the game after the value-assigning phase completes, will correspond to a consistent valuation. Then the player will proceed to the testing phase, where for each clause she will have to choose one literal from this clause, which should be true according to the gadget of the respective variable. If she cannot find such a literal, she will be unable to complete a move. Thus we will obtain an immediate correspondence between the satisfiability of the formula and the outcome of the game.

The construction of the dictionary and the sequence in the bag will ensure that at some point during the value-assigning, the only way for the player to move on is to form a word like in Figure 1a or to form a horizontally symmetrical arrangement (Fig. 1b).

During the test phase, for each clause $c_j = (l_{j_1} \vee l_{j_2} \vee l_{j_3})$ in every play there will be a position, where the player will be obliged to choose one of the literals from the clause in whose gadget she will try to play a word. She will be able to form a word there iff the value of the corresponding variable, which has been set to either true or false in the earlier phase, agrees with the literal.

Let us describe the game more formally. The alphabet $\Sigma$ of $\mathcal{S}$ will contain:

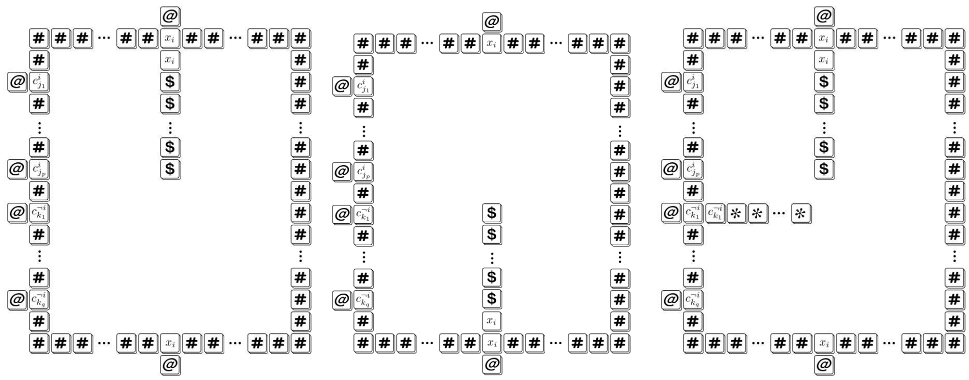
- a symbol $x_i$ for every variable $x_i$;
- a symbol $c_j$ for every clause $c_j$;
- auxilliary symbols: **\$**, $\#$, $*$ and **@**.

Let $r$ be such that no literal appears in more than $r$ clauses. The rack size will be $k = 2r$.

The dictionary $\Delta$ will contain the following words:

- the words **@**$x_i x_i$**\$**$^{2r-1}$ and **\$**$^{2r-1}x_i x_i$**@** for every variable $x_i$,
- the word **@**$c_j c_j *^{2r-1}$ for every clause $c_j$.

[5] In this section we prove hardness of a version of SCRABBLE with an unbounded size alphabet. In section 4 we prove the hardness of the natural variant of derandomized SCRABBLE, where the alphabet, word, rack and dictionary sizes are constants.



(a) $x_i$ set to false.

(b) $x_i$ set to true.

(c) A word played for a clause that $\neg x_i$ satisfies

Fig. 1: Variable $x_i$ with an assigned value.

– We also add all the dummy words that appear initially on the gadgets as it is described below.
  The gadget for variable $x_i$ is constructed as follows: It is a $(4r+3)\times(4r+3)$ square shape consisting mainly of the dummy symbol $\#$. The vertical words **@**$x_i$, $x_i$**@** are attached to the top and bottom walls respectively. Furthermore, for each clause $c_j$, we attach the horizontal word **@**$c_j$ to the left wall if $x_i$ appears in $c_j$ (upper part if it appears positive, lower part if it appears negative).

The sequence in the bag $\sigma$ will be a concatenation of the following:

$$\sigma = \prod_{i=1}^{n}\left(x_i \$^{2r-1}\right)\prod_{j=1}^{m}\left(c_j *^{2r-1}\right)$$

The time period when at least one of the letters $x_i$ are still on the rack will be called the *value-assigning phase*. The following time period will be called the *satisfaction phase*.

We can now prove the following facts (proofs for facts 1 and 2 are omitted due to space limitations).

**Fact 1** *The player has always to empty her rack in order to perform a proper play.*

**Fact 2** *During the value-assigning phase, at each turn, the player performs an action that is in our setting equivalent to a correct valuation of a variable, as shown in Figure 1.*

**Fact 3** *During the satisfaction phase, at each turn, the player's actions are equivalent to checking whether a clause that had not been checked before is satisfied by a literal of the player's choice, as shown in Figure 1c.*

*Proof.* Based on the previous two facts, we know that during each round in the satisfaction phase, the contents of the player's rack are $\{c_j\} \cup \{*\}^{2r-1}$ for some clause $c_j$. Let $c_j = l_{j_1} \vee l_{j_2} \vee l_{j_3}$. One can easily see that the player can only form a legal word by extending one of the 3 words $@c_j$ that appear in the gadgets of variables $x_{j_1}, x_{j_2}$ and $x_{j_3}$ by creating the word $@c_j c_j *^{2r-1}$. A simple analysis shows that the player can play this word in gadget $x_{j_i}$ iff $l_{j_i}$ is true.

The above facts imply that the game correctly simulates assigning some valuation to a 3-CNF formula and checking whether it is satisfied. It is easy to check that the size instance of the Scrabble solitaire game obtained by the reduction is polynomial in terms of the size of the input formula and that the instance can be computed in polynomial time. We have thus shown that SCRABBLE-SOLITAIRE is NP-complete.

To prove the PSPACE-completeness of SCRABBLE it suffices to notice that the above reduction from 3-CNF-SAT to SCRABBLE-SOLITAIRE easily translates to the analogous reduction from 3-CNF-QBF(proof ommitted due to space limitations).

**Theorem 1.** SCRABBLE *is PSPACE-Complete.*

## 4 Hardness due to formation of the words

In this section we prove the hardness of Scrabble due to the ability of the players to form more than one words using the same letters. Furthermore, we will optimize this reduction so that it works even for constant-size $\Sigma, \Delta$ and $k$.

**Theorem 2.** SCRABBLE *is PSPACE-Complete even when restricted to instances with constant-size alphabet, dictionary and rack.*

*Proof.* We will proceed in steps. In section 4.1 we simply sketch the high-level idea, which consists of a board construction that divides play into two phases, the assignment and the satisfaction phase. Then, in sections 4.2, 4.3, 4.4 we present in full a slightly simplified version of our construction which uses a constant-size $\Sigma$ and $\Delta$ but unbounded $k$. Finally, in section 4.5 we give the necessary modifications to remove words of unbounded length from the dictionary and obtain a reduction where $k$ is also constant.

### 4.1 Construction Sketch

Our reduction is from 3-CNF-QBF. Suppose that we have a 3-CNF-QBF formula $\exists x_1 \forall x_2 \exists x_3 \ldots \phi$ with $n$ variables $x_1, x_2, \ldots, x_n$, where $\phi$ has $m$ clauses $c_1, c_2, \ldots, c_m$. We create an instance of $(\Sigma, \Delta, k, \pi)$-SCRABBLE, as follows.

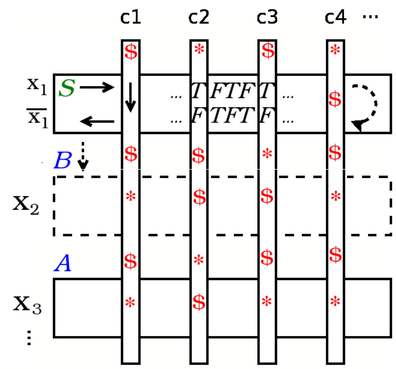



Fig. 2: A high level of the game

The board will be separated in $n$ roughly horizontal segments which correspond to variables and $m$ vertical segments which correspond to clauses (see figure 2).

Play will be divided into two phases: the assignment phase and the satisfaction phase. In the first phase the two players will play within the horizontal segments placing words that encode the truth values of the variables of the formula (hence, mostly the letters T and F are used in this phase). With appropriately placed walls we keep the players on track in this phase making sure that each player, during her turn, has only one available position to place a word (but possibly two availabe words to place if it is her turn to decide on a variable's truth value).

For the second phase, the players place words in the vertical segments. Here, we have encoded the structure of the formula by placing a different character on the intersection of two segments depending on whether the corresponding literal appears in that clause. The first player is always forced to play a word in these intersections and she is only able to do so till the end if and only if the chosen truth assignment satisfies all clauses.

### 4.2 The initial position

Let us now describe the game more formally. We create a $(\Sigma, \Delta, k, \pi)$ game of Scrabble, where the alphabet $\Sigma = \{$***#***, *******, ***$***, ***A***, ***B***, ***S***, ***T***, ***F***, ***0***, ***1***, ***2***, ***@***$\}$, the rack size $k$ is an odd number depending on $m$ (particularly $k = 10m + 5$), $\Delta$ is shown in table 1 and the initial position $\pi$ is described below.

For the following descriptions refer to figure 2.

The initial board $\mathcal{B}$ consists mainly of words containing the dummy symbol ***#***. We use these words to build walls inside the board that will restrict the players' available choices.

There is also a symbol ***S*** initially placed on the board. This indicates the starting point, where the first player is going to put her first word.

| Dictionary | |
|---|---|
| Words | Definition |
| $\boldsymbol{S(TF)^{\frac{k-1}{2}}S}$, $\boldsymbol{F(TF)^{\frac{k-1}{2}}S}$, $\boldsymbol{S(FT)^{\frac{k-1}{2}}F}$, $\boldsymbol{F(TF)^{\frac{k-3}{2}}STFTF}$, $\boldsymbol{F(TF)^{\frac{k-3}{2}}SFTFT}$ | *The literal played has value True.* |
| $\boldsymbol{S(FT)^{\frac{k-1}{2}}S}$, $\boldsymbol{T(FT)^{\frac{k-1}{2}}S}$, $\boldsymbol{S(TF)^{\frac{k-1}{2}}T}$, $\boldsymbol{T(FT)^{\frac{k-3}{2}}STFTF}$, $\boldsymbol{T(FT)^{\frac{k-3}{2}}SFTFT}$ | *The literal played has value False.* |
| **#AT**, **#AF** | *First player's turn to assign truth value*; |
| **#BS** | *Second player's turn to assign truth value*; |
| **$$**, ******, **#A**, **#B**, $\#^c$, for $c \leq 2k$ $\#^5Q\#^9Q\#^9Q\#^5$, for $Q \in \{$**$**, *****$\}$ | *Wall word* |
| **0****, **1****, **2****, **0$$**, **1$$**, **2$$** | *Word formed during satisfaction phase.* |
| **0**1T20**, **0$$1T20**, **0$$1F20** | *No unsatisfied literals in the clause so far.* |
| **1**2T01**, **1$$2T01**, **1$$2F01**, **0**2F01** | *One unsatisfied literal in the clause so far.* |
| **2**0T12**, **2$$0T12**, **2$$0F12**, **1**0F12** | *Two unsatisfied literals in the clause so far.* |
| **0120**, **1201**, **2012** | *Symbols'* **0**, **1**, **2** *order preserving words.* |

Table 1: The Dictionary $\Delta$. All valid words appear as regular expressions, together with their definitions. Synonyms are grouped together.

On the left side of the board, attached on the wall, there are several appearences of the symbols ***A*** and ***B***. These symbols indicate whether it is the first or the second player's turn to choose truth assignment (player 1 assigns values to the variables $x_{2i+1}$ whereas player 2 to the variables $x_{2i}$ for every $i = \lfloor \frac{n}{2} \rfloor$).

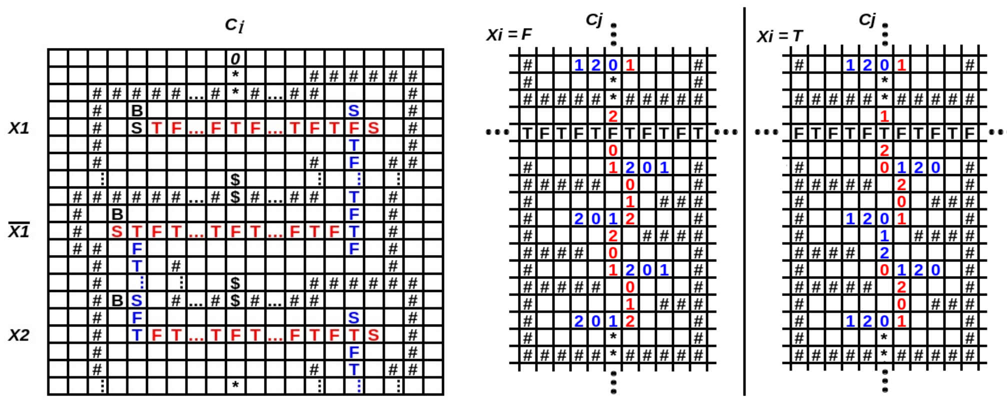
Last, we need to construct the clauses. For every clause there is a corresponding column as shown in the figure. We place the symbols ***$*** and ******* in the intersections with literals (horizontal lines) in order to indicate which literals appear in the particular clause. If a literal appears in the clause we put a ******* whereas if it doesn't we put a ***$*** (in figure 2, $c_4 = (x_1 \vee \neg x_2 \vee \neg x_3)$).

In the initial position $\pi$ of the game we also have:

- $r^1 = r^2 = r = \{\boldsymbol{T}, \boldsymbol{F}\}^{\frac{k-1}{2}} \cup \{\boldsymbol{S}\}$;
- $\sigma = r^a(\boldsymbol{012})^s @^{2k-6}\boldsymbol{A}$, where $a$ $(= 4n-2)$ is the number of turns played during the assignment phase and $s$ $(= \frac{40}{3}m^2n)$ the number of turns played during the satisfaction phase (see sections 4.3 and 4.4);
- Player 2 has a lead of 1 point and it is first player's turn.

### 4.3 Assignment Phase

In the first phase of the game (the assignment phase, see figure 3a), players will repeatedly draw the following letters: $\frac{k-1}{2}$ pairs ($\boldsymbol{T}$, $\boldsymbol{F}$) and a single symbol $\boldsymbol{S}$. The only words that they can form with these symbols are the assignment words from $\Delta$ (given in the first two lines in the dictionary of table 1). These words




(a) The assignment phase

(b) The satisfaction phase

Fig. 3: More detailed construction sketches.

have length $k + 1$, so in order to play them, one of the symbols should already appear on the board and the players have to empty their racks completely.

The major concern here is the assignment. We say that a word assigns the value True (resp. False) to a variable if the intersections of the positive literal's line with the clauses' columns contain the symbol $\boldsymbol{T}$ (resp. $\boldsymbol{F}$).

Player 1 plays first and has to choose among two possible proper plays, one that assigns the value True to $x_1$ and one the value False. Observe that player 1 is always forced to play horizontally whereas player 2 only plays vertically. To avoid having only player 1 choose the assignment, we use the symbols $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{S}$, as described in the appendix.

Once the assignment is fixed, players' unique choices are predetermined by the current position of the board and the dictionary. The amount of points that the two players gain after this phase is identical and equal to $2n(2k + 5)$ (there are $2n$ zigzags and each player constructs two $(k + 1)$-symbol long words and one 3-symbol long word in each).

### 4.4 Satisfaction Phase

For this section, refer to figure 3b.

After the assignment phase, the bag begins with a long string of the symbols ***0***, ***1***, ***2***. Satisfaction is realized by forming satisfaction words (the last four lines in the dictionary). A clause is considered satisfied when the corresponding vertical segment is fully filled with words.

The most crucial step of the satisfaction phase is the placement of the words that intersect with literals. The numbers **0, 1, 2** indicate the number of false literals that the clause currently has. The combination of {*******, ***$***}, {***T***, ***F***} and {***0***, ***1***, ***2***} gives a unique vertical proper word to play in the intersection of a literal (horizontal) segment with the clause (vertical) segment. The ending symbol of the played word is the number of false literals we have seen in the clause so far. The combination $\{\boldsymbol{num}, \boldsymbol{*}, \boldsymbol{F}\}$ (where $\boldsymbol{num} = \boldsymbol{0}$, $\boldsymbol{1}$,or $\boldsymbol{2}$) is important, because it forms the word $\boldsymbol{num}\,\boldsymbol{*}\,\boldsymbol{*} \ldots \boldsymbol{F} \ldots \boldsymbol{num}{+}1$ which is the only one that increases $\boldsymbol{num}$ (the clause contains a false literal).

The words which contain only the numberic symbols $\boldsymbol{0}$, $\boldsymbol{1}$, $\boldsymbol{2}$ reserve the order of those symbols' appearence on the board and by doing so enforce that the appropriate numeric symbol begins the next intersection word.

Starting with literal $x_1$, the two players fill in words interchangeably, beginning with player 1 who plays vertically. Observe that the only way that a player won't be able to place a word is to be faced with the combination $\{\boldsymbol{2}, \boldsymbol{*}, \boldsymbol{*}, \boldsymbol{F}\}$ in an intersection (third false literal in the clause).

Notice that player 2 doesn't really have an incentive to play vertically because the number of points acquired if she plays vertically is equal to the number of points if she plays horizontally and equal to $\frac{4l}{2} + 3 = 2l + 3$, where $l = \frac{s}{2nm}$ is the number of turns played inside a literal segment (the additive term in the score comes in the vertical play case from the 7-letter long word played during the first turn and in the horizontal play case from the additional 3-letter long word which is formed during the last turn). Thus we can assume wlog that player 1 plays vertically and player 2 horizontally, and, despite that during the game there will be several possible proper plays, the final score after the satisfaction phase is independent of players' choices.

We argue now that if there is a satisfying assignment for the first order formula then player 1 wins, else player 2 wins.

The key point in this proof is that player 2 "matches" player 1's moves throughout the duration of the whole game. Since player 2 starts with a 1-point lead she will continue to have the lead after the end of the satisfaction phase.

If there is a satisfying assignment, then by the end of the game player 1 gets the last symbol in the bag which is an $\boldsymbol{A}$ and forms an additional 3-letter long word, which makes him the winner of the game with $s^1 = s^2 + 2$.

On the other hand, if there is no satisfying assignment, the two players will have at least one set of $\boldsymbol{0}$,$\boldsymbol{1}$,$\boldsymbol{2}$ on their hands and probably some copies of the useless symbol **@** which doesn't form any words, so player 1 is not going to get the symbol $\boldsymbol{A}$ from the bag. Player 2 is the last player to place a word on the board. This makes him the winner of the game with $s^2 = s^1 + 1$.

Let us also point out that the assumtion that players cannot pass does not affect our arguments so far. Indeed, observe that at any point when it's a player's turn to play, that player is behind in the score. If she chooses to pass, the other player may also pass. Repeating this a second time ends the game, according to standard Scrabble rules. Thus, if the current player has a winning strategy it must be one where she never chooses to pass.

### 4.5 Constant rack and word size

In order for the proof to work for constant size words and rack, we need to break the long assignment words into constant size ones and zig-zag through the clauses (see figure 4). Once we reduce the size of the words to a constant, an unbounded size rack is unnecessary. In fact, the rack has to be smaller than the maximum word size by one symbol.

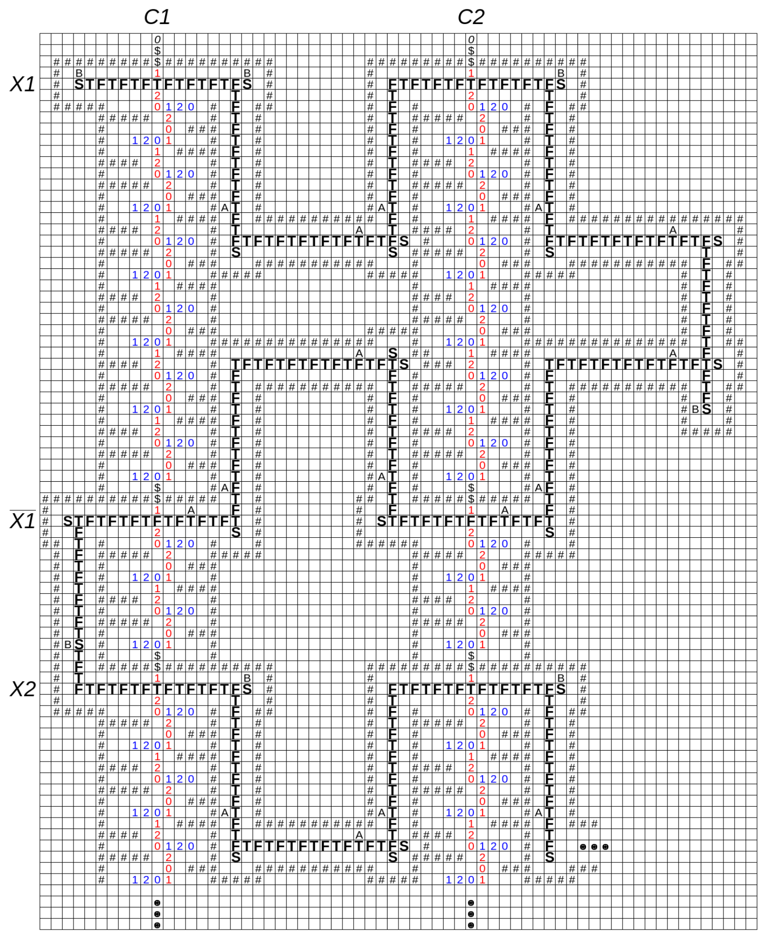



Fig. 4: Modifications for fixed size words and rack

Observe that the length of the assignment word should be equal to the height of the clause segments between a negative literal and its next positive. This distance is 4 mod 6. Also, the word has to be longer than the width of the clause segments (which is 11). Setting the word size equal to 16 ($k = 15$), satisfies both requirements. Careful counting arguments fix the zig-zaging between a positive and a negative literal (see figure 4).

We change the board construction to adopt the modifications:

– We build walls all around the board to force the aforementiond zig-zaging pattern. The walls too have to consist of constant size parts (the wall is part of the dictionary).
– Last, we need to place one ***A*** or ***B*** symbol in every horizontal or vertical section, so that force that players put their ***S*** symbol in the beginning or the end of their played word (forcing thus the assignment throughout variable segments) and also to make sure that the players will gain an equal amount of points ($= k + 3$) on every turn.

The rest of the proof follows the ideas of the proof for arbitrary size rack and words.

## 5 Conclusions

We have established the PSPACE-hardness of (deterministic) Scrabble in two different ways. The main ingredients for our two proofs are the possibility of placing words in many places in the first, and the possibility of forming several different words in the second. We have also established that hardness remains even when all relevant parameters are small constants.

Several interesting further questions can be posed in the same vein. Are the constants we have used optimal? What is the minimum-size alphabet or dictionary for which the problem is still PSPACE-hard? In particular, does the problem become tractable when the alphabet contains just one letter, or is the complexity of placing the tiles on the board enough to make the problem hard?

Another interesting question was posed by Demaine and Hearn [1]: is there a polynomial-time algorithm to determine the move that would maximize the score achieved in this round? Of course, in the case of a bounded-size rack the problem is immediately in P, but deciding how to place $n$ letters on the board optimally could be a much harder problem.

## References

1. E.D. Demaine and R.A. Hearn. Playing games with algorithms: algorithmic combinatorial game theory. In *Games of No Chance III, Proc. BIRS Workshop on Combinatorial Games, July*, pages 3–56. Citeseer, 2005.
2. A.S. Fraenkel and D. Lichtenstein. Computing a perfect strategy for n × n chess requires time exponential in n. *Journal of Combinatorial Theory, Series A*, 31(2):199–214, 1981.
3. S. Iwata and T. Kasai. The othello game on an n × n board is pspace-complete. *Theoretical Computer Science*, 123(2):329–340, 1994.
4. C.H. Papadimitriou. Computational complexity. 1994.
5. JM Robson. N by n checkers is exptime complete. *SIAM J. Comput.*, 13(2):252–267, 1984.