Journal of Information Processing Vol.23 No.3 284–292 (May 2015)

[DOI: 10.2197/ipsjjip.23.284]

Regular Paper

Scrabble is PSPACE-Complete

Michael Lampis[1,a)] Valia Mitsou[2,b)] Karolina Sołtys[3,c)]

Received: July 31, 2014, Accepted: January 7, 2015

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

Scrabble® is a board game played by two to four players. In this game, each player takes turns drawing lettered tiles randomly out of an opaque bag and then attempting to place those tiles on a 15×15 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. For a fuller description of the board game of Scrabble see the rules on the official website by Hasbro: http://www.hasbro.com/scrabble/en_US/discover/rules.cfm.

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 been answered for several popular 2-player board games, such as Chess [5], Go [8], Hex [4], and Othello [6], typically classifying their complexity as either PSPACE- or EXP-complete. However, unlike these combinatorial 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. This practice is quite common and many games which involve chance or imperfect-information have already been analysed complexity-wise under a deterministic setting, for example UNO [2] and Tetris [1]. This very question regarding the complexity of a deterministic version of Scrabble was listed as an open problem by Demaine and Hearn [3]. In this paper, we tackle exactly this question, showing that a derandomized version of Scrabble is PSPACE-complete.

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 decision tasks: which word to form and where to place it on the board. Though these 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.

We present two different hardness proofs arguing that both of these tasks are hard. In one reduction, the players are essentially given appropriate tiles so that they only have one possible word to play in each round, with a choice of two locations to place it. In the other, players are essentially forced to play at a specific place on the board, but are able to choose between two different words. In both cases, the problem of deciding optimal play still turns out to be PSPACE-complete. 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. Along the way, we show that even a solitaire version of the game, where one player tries to place all available tiles on the board while forming proper words, is NP-complete.

The rest of the paper is divided as follows: In Section 2 we present the model of the game that we study and define it formally. In Section 3, we prove that Scrabble is hard due to the players' ability to place their formed word in more than one place. In Section 4 we prove the hardness of Scrabble due to the players' ability to form more than one word using the same letters.

[1] LAMSADE, Université Paris Dauphine
[2] JST Erato Minato Discrete Structure Manipulation System Project
[3] University College London
[a)] mlampis@kurims.kyoto-u.ac.jp
[b)] vmitsou@erato.ist.hokudai.ac.jp
[c)] karolina.soltys.14@ucl.ac.uk

Finally, in Section 5, we give some conclusions and present some interesting questions for further investigation.

2. Our Model of Scrabble - Definitions

Informally, the question we are trying to answer is the following: 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 at most a polynomial number of possible configurations, putting the problem trivially in P.

Beyond the size of the board and the number of letters in the bag, we need to define an alphabet, a dictionary (a set of acceptable words), and a rack size which will determine how many letters each player has in 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 4.1 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 at least as hard). We will assume that players are not allowed to exchange tiles. Pass moves are allowed and do not affect our proofs: at any point when it's a player's turn to play, that player is behind in the score. So if she chooses to pass, the other player may also decide to pass. Repeating this for three times in a row ends the game, according to standard Scrabble rules, with the player who passed first losing the game. Thus, if the current player has a winning strategy, it must be one where she never chooses to pass.

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

Definition 2.1 A *position* π in a scrabble game is an ordered septuple $(\mathcal{B}, \sigma, p, r^1, r^2, s^1, s^2)$, where $\mathcal{B}$ is the *board*, which is an $n \times n$ matrix of symbols from Σ, $\sigma \in \Sigma^*$ is a sequence of lettered tiles called the *bag*, $p \in \{1, 2\}$ is the number of the active player, r^1, r^2 are multisets with symbols from Σ denoting the contents of the rack of the first and the second player respectively and $s^1, s^2 \in \mathbb{N}$ are the scores of the first and the second player respectively.

Definition 2.2 We define a *Scrabble game* $\mathcal{S}$ to be an ordered quadruple $(\Sigma, \Delta, k, \pi_0)$ where: Σ is a finite *alphabet*, $\Delta \subset \Sigma^*$ is a finite *dictionary*, $k \in \mathbb{N}^+$ is the size of the rack and π_0 is the initial position of the game.

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 formed words is added to the active player's points, used letters are removed from the player's rack which is then refilled with an equal amount of new letters from the bag (or less, if $|\sigma_i| < k$). The new letters form a prefix of σ_i.

Definition 2.3 A *play* $\Pi = \pi_1 \dots \pi_l$ is a sequence of positions such that, for all i, π_{i+1} is attainable from π_i by the active player by forming a *proper play* on the board.

Definition 2.4 A play $\Pi = \pi_1 \dots \pi_l$ is *finished* if the player who is about to make a move is unable to form a proper play. The *winner* of a finished play is the player who gained more points during the game.

Definition 2.5 A *Scrabble solitaire* game is defined analogously to the normal game, but with only a single player. The player *solves* the solitaire if she manages to get rid of all the letters from the bag. We define (Σ, Δ, k, π)-*Scrabble solitaire*, with Σ, Δ and k as above and $\pi = \{\mathcal{B}, \sigma, r^1\}$, with $\mathcal{B}$, σ and r^1 also defined as above (in the solitaire version there is no score).

Definition 2.6 We define Scrabble to be the problem of determining the winner of a given Scrabble game and Scrabble-Solitaire to be the problem of determining if a given Scrabble solitaire game is solvable.

We will establish PSPACE-hardness via two reductions from 3-CNF-QBF, the problem of deciding whether a quantified boolean CNF formula is true. 3-CNF-QBF is a variation of satisfiability which is complete for the class PSPACE [7]. It can be viewed as a two player game, where players take turns setting truth values of the variables used in a formula ϕ interchangeably. If ϕ is satisfied then player 1 wins, else player 2 wins.

3-CNF-QBF:

Input: A first order formula $\exists x_1 \forall x_2 \exists x_3 \dots \forall x_n \; \phi(x_1, x_2, x_3, \dots, x_n)$, where ϕ is a propositional formula written in CNF which has m clauses, with each clause containing 3 literals.

Rules: Players 1 and 2 set truth values to variables of ϕ. Player 1 sets truth values to existentially quantified variables, whereas player 2 sets truth values to universally quantified variables.

Question: Does there exist a strategy for player 1 to make ϕ satisfiable?

We are also interested in a 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. The 3-CNF-SAT problem is defined as follows.

3-CNF-SAT:

Input: A propositional formula ϕ written in CNF that contains n variables and m clauses, where each clause contains at most 3 literals.

Question: Does there exist an assignment of truth values to the variables such that all the clauses of ϕ are satisfied?

3. Hardness due to Placement of the Words

In this section we present the first reduction, which shows that Scrabble is hard because players have a choice on where to position a formed word, despite that there is essentially a unique word to form [*1].

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

Lemma 3.1 Scrabble-Solitaire is NP-complete.

Proof. Proving that the problem is in NP is straightforward. To establish the NP-hardness of Scrabble-Solitaire, we will construct a reduction to this problem from 3-CNF-SAT. Given a 3-CNF propositional formula ϕ with n variables $x_1, x_2, \dots, x_n$ and m clauses, we construct in polynomial time a polynomial-sized Scrabble-Solitaire game $\mathcal{S}$, such that ϕ 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, when 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 gadget for variable x_i is shown in **Fig. 1**. The construction of the dictionary and the sequence in the bag will ensure that at some point during the value-assigning phase, the only way for the player to move on is to form a word like in **Fig. 2** (a) or to form a symmetrical arrangement (Fig. 2 (b)).

During the test phase, for each clause $c_i = (l_1 \vee l_2 \vee l_3)$ in every play there will be a position, where the player will be obligated 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 in the earlier phase, agrees with the literal (see **Fig. 3**).

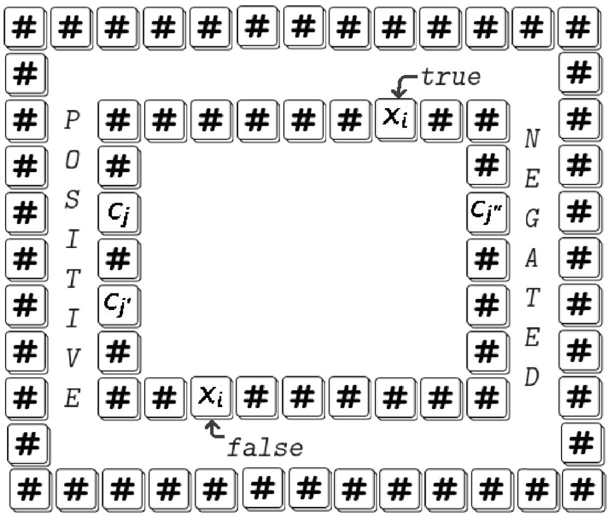

Fig. 1 A gadget corresponding to the variable x_i, which belongs to clauses $c_j, c_{j'}$ positive and to clause $c_{j''}$ negated.

[*1] 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.

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

- a symbol $\mathbf{x_i}$ for every variable x_i;
- a symbol $\mathbf{c_j}$, for every clause c_j;
- auxiliary symbols: *#*, *\$*, and $*$.

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

The dictionary Δ will contain the following words:

- the word $\mathbf{x_i}\$^{k-1}\mathbf{x_i}$ for every variable x_i;
- the word $\mathbf{c_j}*^{k-1}\mathbf{c_j}$, for every clause c_j;
- the dummy words appearing initially on the board due to the construction of the variable gadgets.

The sequence in the bag σ will be a concatenation of the following:

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

while the rack will initially contain: $r^1 = \{\mathbf{x_1}\} \cup \{\$\}^{k-1}$.

The phase of the game when at least one of the letters $\mathbf{x_i}$ are still on the rack is called the *value-assigning phase*. The following phase is called the *satisfaction phase*.

We can now prove the following facts.

Fact 3.2 The player must always empty her rack in order to perform a proper play.

Proof. Let us note that the *#* symbols that appear on the board can never be combined with the given letters in the bag and rack in order to form new words (this would require the use of an $\mathbf{x_i}$ or a $\mathbf{c_j}$ in the third position from the beginning or the end of the

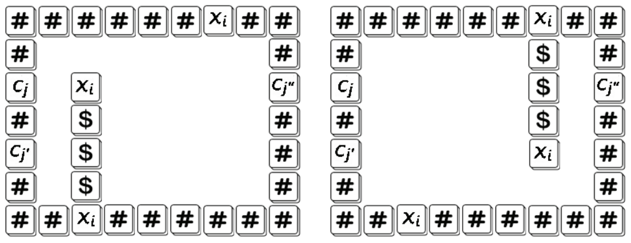

Fig. 2 Variable x_i with an assigned value.

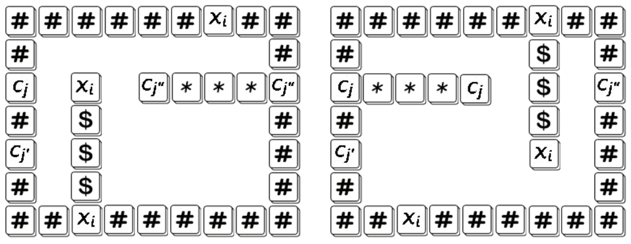
x_i
c_j
$c_{j'}$
$c_{j''}$
(a) $x_i = F$ satisfies $c_{j''}$.
(b) $x_i = T$ satisfies c_j.

Fig. 3 A clause that gets satisfied due to x_i.

[*2] 3-CNF-SAT remains NP-hard even in the restricted version where every literal appears at most 2 times [7], so k can be set to 4.

word, which is impossible in the current setting). So, the only words that the player can form have all length $k + 1$. □

Fact 3.3 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 Fig. 2.

Proof. From the previous fact we gather that during each round in the value-assigning phase, the contents of the player's rack are $\{\mathbf{x_i}\} \cup \{\$\}^{k-1}$ for some i. Observe that the player can form a word consisting of these letters only in one of two ways as shown in Fig. 2, since the wall surrounding the gadget for x_i forbids placing any words on the outside of the gadget. □

Fact 3.4 During the satisfaction phase, at each turn the player's actions are equivalent to checking whether a clause is being satisfied, as shown in Fig. 3.

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 $\{\mathbf{c_j}\} \cup \{*\}^{k-1}$ for some clause c_j. One can easily see that the player can form a legal word using these letters only by extending one of the appearences of c_j on the board in the gadgets. The player can pick any gadget x_i where c_i appears and for which the "assignment" word appears in the correct side (otherwise the "satisfaction" word would not fit). □

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 of the 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 show that the above reduction from 3-CNF-SAT to Scrabble-Solitaire can translate to the analogous reduction from 3-CNF-QBF. A detailed proof follows.

Theorem 3.1 Scrabble is PSPACE-Complete.

Proof. We are given a first order formula $\exists x_1 \forall x_2 \ldots \phi$, with n variables and m clauses. We can assume that n is even; if not, we just add in ϕ a new dummy clause in which a new variable x_{n+1} appears both positive and negated.

We first create a propositional formula ϕ' by duplicating all clauses from ϕ. Observe that the new instance of 3-CNF-QBF $\exists x_1 \forall x_2 \ldots \phi'$ is equivalent to the original.

It is easy to reduce the new instance of 3-CNF-QBF to a game of Scrabble $\mathcal{S}$. The alphabet Σ, the rack size k, and the board construction $\mathcal{B}$ are defined in the same way as in the proof of Lemma 3.1. The bag sequence σ and the dictionary Δ are again defined almost identically as before, apart from the addition of a 2-letter word *##* in the dictionary and the symbol *#* at the very end of the bag (this will give player 1 the chance to take the lead by forming a 2-letter word at the very end of the game if the formula is satisfiable). The scores are $s^1 = s^2 - 1$ (i.e., player 2 has a lead of 1 point) and it is the first player's turn.

The two players will be playing a normal game of Scrabble (starting by player 1) using a board obtained by applying the previous construction to the duplicated formula. Observe that, while the number of variable gadgets is the same, their sizes are doubled since each literal appears in twice as many clauses as in ϕ.

In the assignment phase, the two players will assign truth values to the variables $x_1, x_2, \ldots, x_n$ interchangeably. Since n is even, player 2 is the last player to put an "assignment" word on the board, leaving player 1 to begin phase 2.

For the satisfaction part, observe that, for every clause c_u there is an identical clause c'_u. If there is a literal l_i that satisfies c_u, then l_i also satisfies c'_u. That means that player 2 cannot be left without an available word to play since she can always match player 1's move.

If the formula is satisfiable, then the bag will eventually empty. The last player to place a word will be player 1 using symbol *#* to create a two-letter word *##* anywhere on the board. In this case player 1 wins with $s^1 = s^2 + 1$.

On the other hand, if the formula is not satisfiable, then the last player to place a word will be player 2, leaving the score $s^1 = s^2 - 1$ and making player 2 the winner of the game. □

4. Hardness due to Formation of the Words

In this section we present the second reduction, where the hardness stems from the fact that there is more than one word to form (despite having essentially a unique place to position them on the board). Furthermore, we will optimize this reduction so that it works even for constant-size Σ, Δ and k.

Theorem 4.1 Scrabble is PSPACE-complete even when restricted to instances with a constant-size alphabet, dictionary and rack.

Proof. We will proceed in steps. In Section 4.1, we sketch the high-level idea, which consists of a board construction that divides play into two phases, the value-assigning and the satisfaction phase. In Sections 4.2, and 4.3, we sketch how the assignment and satisfaction works.

4.1 Construction Overview

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 ϕ has m clauses $c_1, c_2, \ldots, c_m$. We first double the formula by taking each clause twice (this will allow player 2 to copy player 1's moves throughout the game). Then, we create an instance of (Σ, Δ, k, π)-Scrabble, as follows.

The board will be separated in n roughly horizontal segments which correspond to variables and $2m$ vertical segments which correspond to clauses (see **Fig. 4**).

Play will be divided into two phases: the value-assigning phase and the satisfaction phase. In the value-assigning phase, the two players will play within the horizontal segments placing words that encode the truth values of the variables of the formula. 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 available words to place if it is her turn to decide on a variable's truth value).

For the satisfaction phase, the players place words in the vertical segments. The doubling of the clauses ensures that player 1 should be the one responsible for checking the satisfiability of each of the clauses (player 2 just repeats player 1's moves for the duplicate clauses). We have encoded the structure of the formula

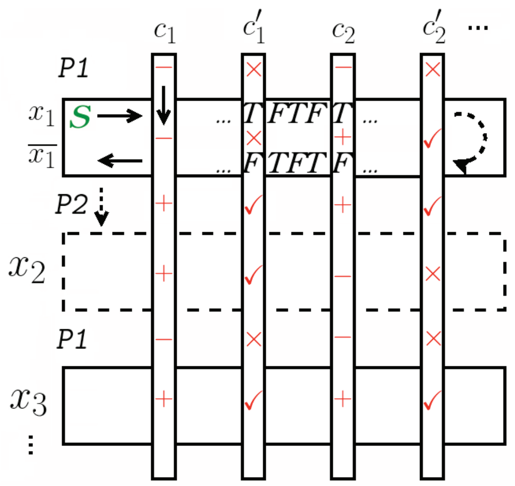

Fig. 4 A high level view of the game.

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

Dictionary	
Word	Definition
FTFTFTFTFTZFTF, ***TFTFTFTFTFZTFT***	*Value preserving words* *(true)* *(false)*
FTFZTTTTTFFFFF, ***TFTZTTTTTFFFFF***, ***FTFZFFFFFTTTTT***, ***TFTZFFFFFTTTTT***	*Value assigning words* *(true)* *(false)*
#PZ	*Turn indicating word during the value-assigning phase*
#ST	*Start indicating word*
#P, $\#^c$ ($c \leq 2k$) $\#^3$Q$\#^6$, $\#^7$Q$\#^{10}$, (Q ∈ {+, −, ✓, ×})	*Wall words*
0N−1T20, ***0N−1F20***, ***0N+1T20***, ***0M×1T20***, ***0M×1F20***, ***0M✓1T20***	*No unsatisfied literals in the clause so far*
1N−2T01, ***1N−2F01***, ***1N+2T01***, ***0N+2F01***, ***1M×2T01***, ***1M×2F01***, ***1M✓2T01***, ***0M✓2F01***	*One unsatisfied literal in the clause so far*
2N−0T12, ***2N−0F12***, ***2N+0T12***, ***1N+0F12***, ***2M×0T12***, ***2M×0F12***, ***2M✓0T12***, ***1M✓0F12***	*One unsatisfied literal in the clause so far*
0N120, ***1N201***, ***2N012***, ***0M120***, ***1M201***, ***2M012***	*Symbols'* **0**, **1**, **2** *order-preserving words*

by placing a different character on the intersections between a literal and a clause, depending on whether the corresponding literal appears in that particular clause. The word formation also depends on the chosen assignment. Being able to place a word in these intersections means that the clause has not been unsatisfied by the assignment yet. Player 1 shall be able to completely fill a clause segment if and only if the chosen truth assignment satisfies the clause.

Let us now describe the game in more detail. We create a game of Scrabble, where the alphabet is Σ = {***#***, **+**, **−**, ✓, **×**, ***P***, ***S***, ***T***, ***F***, ***Z***, ***N***, ***M***, ***0***, ***1***, ***2***, ***@***}. The rack size k shall be a constant odd number (in particular, $k = 13$). The dictionary Δ is shown in **Table 1** and the initial position π is described below.

For the following descriptions refer to Fig. 4 (or for a more detailed but still abstract preview to **Fig. 5**).

The initial board $\mathcal{B}$ consists mainly of words containing the dummy symbol ***#*** (we call them "wall words"). We use these words to build walls inside the board that will restrict the players'

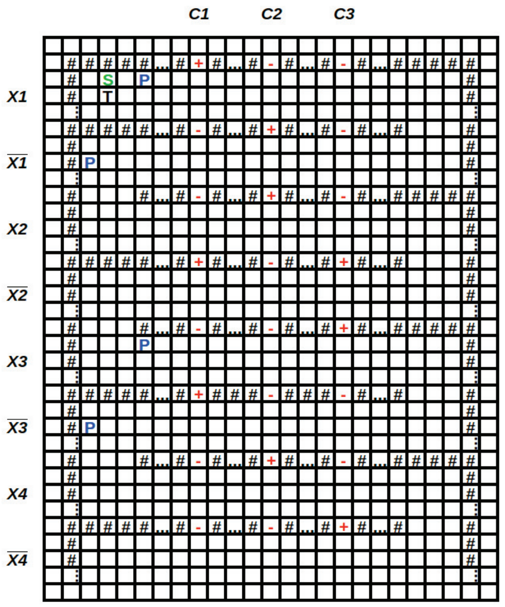

Fig. 5 An abstract view of the board. Duplicate clauses have been omitted.

available choices.

There is also a starting word ***#ST*** placed on the board, which indicates the starting point, where the first player is going to put her first word.

Attached on the wall, there are several appearances of the symbol ***P***. The position of ***P*** in the left side indicates 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 $i \leq \lfloor \frac{n}{2} \rfloor$).

Last, we need to construct the clauses. For every original clause in ϕ there is a corresponding column as shown in the figures. We place the symbols **+** and **−** at 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 Fig. 5, $c_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$. For the corresponding duplicate clauses, instead of **+** and **−** we use the symbols ✓ and **×** (as shown in Fig. 4).

In the initial position π of the game we also have:

- $r^1 = r^2 = r = \{\boldsymbol{T},\boldsymbol{F}\}^{\frac{k-1}{2}} \cup \{\boldsymbol{Z}\}$;
- $\sigma = \left((\boldsymbol{TF})^{\frac{k-1}{2}}\boldsymbol{Z}\right)^{a-1}\boldsymbol{012N@}^{k-4}\boldsymbol{012M@}^{k-4}(\boldsymbol{012N012M})^{\frac{s}{2}-1}\#$, where $a = O(mn)$ indicates the number of turns played during the value-assigning phase and $s = O(mn)$ the number of turns played during the satisfaction phase (see Sections 4.2 and 4.3);
- Player 2 has a lead of 1 point and it is first player's turn.

  In order for the proof to work for constant size words and rack, the walls should create a zig-zag pattern through the clauses (see **Fig. 9** at the end of the paper for a detailed view). The walls too should consist of constant size parts, as wall words are part of the dictionary.

4.2 Value-assigning Phase

In the first phase of the game (the value-assigning phase), players will repeatedly draw the following letters: $\frac{k-1}{2}$ pairs (***T***,***F***) and a single ***Z***. The only words that they can form with these symbols

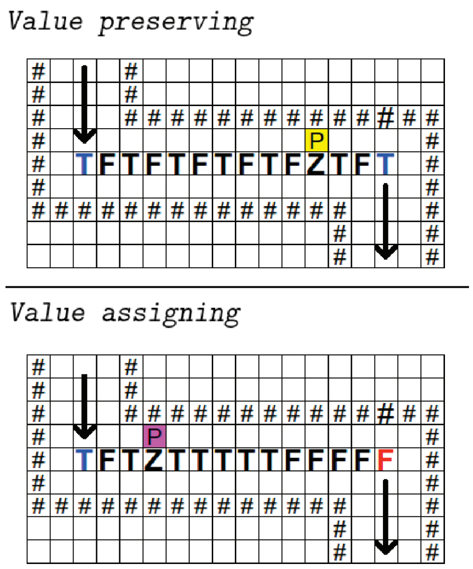

Fig. 6 The role of symbol ***P***. In the first case a value preserving word is played, whereas in the second a value assigning word is played, changing the assignment.

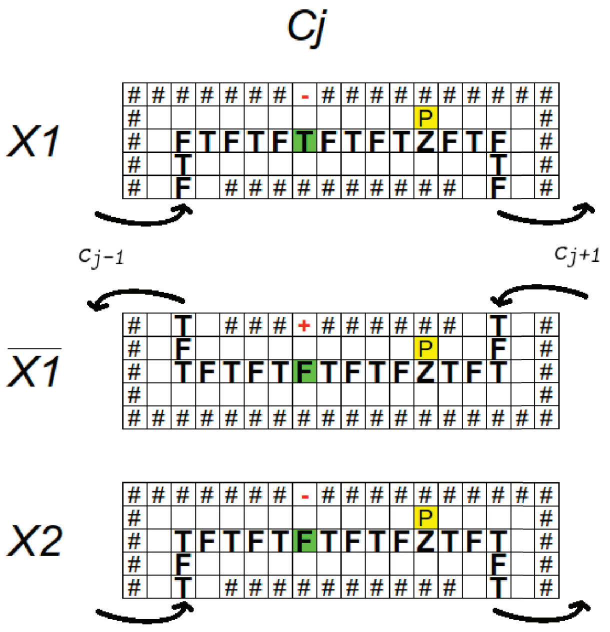

Fig. 7 The value-assigning phase. In this example $x_1 = T$, and $x_2 = F$.

are the "assignment" words from Δ (given in the first three lines of Table 1).

The main words that the players will be playing (first two lines) have length $k + 1$, so in every turn during this phase the players should be emptying their racks completely. The word ***#PZ*** is only an auxiliary word: no player would choose to play it as their main word. If some did, that player would refill her rack with a single ***T***, and her rack would now consist only of ***T***'s and ***F***'s; however, without a ***Z*** symbol the player would have no word to play in her next turn and would lose the game.

Observe that player 1 is always forced to play horizontally whereas player 2 only plays vertically. The assignment to the variables is performed depending on the position of the symbol ***P*** as follows (see **Fig. 6**): players are only allowed to play their ***Z*** symbol next to a ***P*** and form the auxiliary word ***#PZ***. Thus, depending on the position of ***P***, the players can either form a value preserving word (first line of the dictionary) or a value assigning word (second line of the dictionary).

Player 1 plays first and has to choose among two possible value assigning words, the one that assigns the value true to x_1 (***TFTZTTTTTFFFFF***), and the one that assigns the value false (***TFTZFFFFFTTTTT***). Once the assignment is fixed, players' unique choices (value preserving words) are predetermined by the board construction and the dictionary (***FTFTFTFTFTFTZFTF*** for true and ***TFTFTFTFTFZTFT*** for false). The crucial part in the assignment is the letter that will be placed at the intersection between the "assignment" word and the clauses columns (see **Fig. 7**). 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 ***T*** (resp. ***F***). Appropriate zig-zagging ensures that the value of $\neg x_1$ is negated (for more details see Fig. 9). When variable x_1 has been played completely, it is player 2's turn to play a value assigning word before entering the x_2 segment. Play continues in a similar manner and, after the end of this phase, the two players will have gained an equal amount

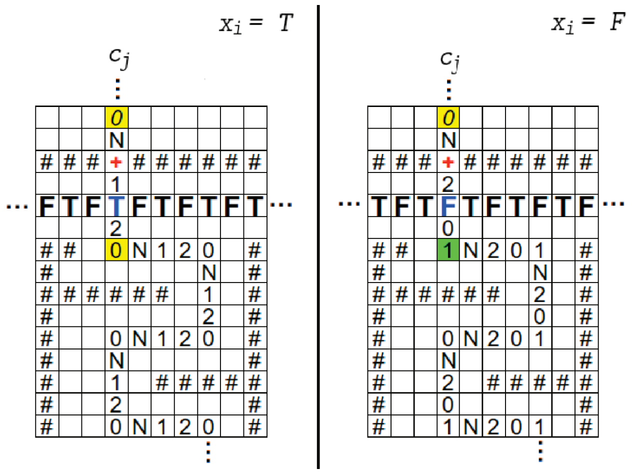

Fig. 8 The satisfaction phase.

of points, and player 2 will have the lead in the score.

4.3 Satisfaction Phase

After the value-assigning phase, the bag begins with a long string of the symbols ***0***, ***1***, ***2***, ***N***, ***0***, ***1***, ***2***, ***M***, padded in its first appearance with two $k - 4$-long sequences of @,one after the ***N*** and one after the ***M*** (@ is a garbage symbol not contained in any words). 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 in the original clauses that intersect with literals (see **Fig. 8**). The numbers ***0, 1, 2*** indicate the number of false literals the clause currently has. The possible combinations of {**+**, **−**}, {***T***, ***F***} and {***0***, ***1***, ***2***} give a unique vertical proper word to play at 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

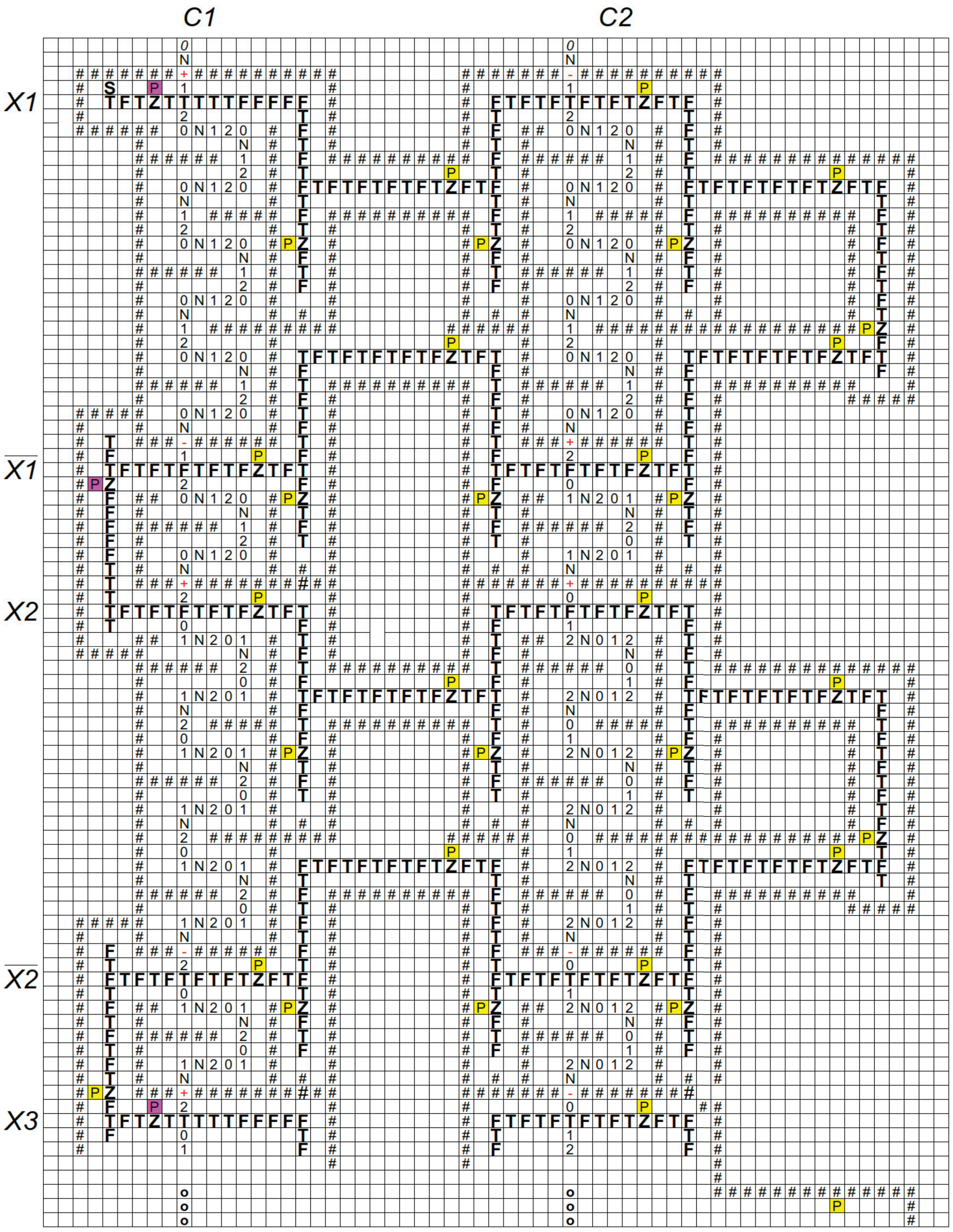

Fig. 9 A more detailed view of the board. Duplicate clauses have been omitted.

combination {***num***, **+**, ***F***} (where ***num*** = ***0***, ***1***) is important, because it forms the word ***num N*** **+** **** F **** (***num*** + 1), which is the only one that increases ***num*** (the clause contains a false literal). Additionally, in the gadget corresponding to the duplicate clause the combination {***num***, ✓, ***F***} will appear, which will accordingly cause an increase in ***num*** by one in the duplicate clause as well.

The words which contain only the symbols ***0***, ***1***, ***2***, and ***N*** (or ***M***) preserve the order of the numbers and by doing so enforce the appropriate number to begin the next intersection word.

The two players fill in words in turn, beginning with player 1. Because of the doubling of the clauses in ϕ, player 2 can always copy player 1 by playing at the duplicate clauses on the board. Observe that the order-preserving words are worth only 5 points whereas the rest are worth 7. However, the allowed combinations force player 2 to get no more than half of the 7-point words (she can only play these words in the sections corresponding to duplicate clauses). Furthermore, observe that the only way that a player won't be able to play a word is when faced with the combination {***2***,**+**, ***F***} (or {***2***,✓, ***F***} accordingly) at an intersection, which indicates a third false literal in the clause.

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

In each turn, a player uses all the useful symbols she has in hand {***0***, ***1***, ***2***, ***N***} or {***0***, ***1***, ***2***, ***M***} to form one of the "satisfaction" words and refills her rack with another copy of the same symbols. As it was argued above, player 2 always has an available move to perform (copy player 1), and since she starts with an 1-point advantage, she will continue to have the lead throughout the game until the end of the satisfaction phase.

If there is no satisfying assignment, the two players will eventually be left with one set of {***0***, ***1***, ***2***, ***N***} and {***0***, ***1***, ***2***, ***M***} in hand (padded with $k-4$ useless copies of the symbol @). Player 2 is the last player to place a word on the board and player 1 will be unable to perform a proper play. So player 2 wins the game with a score $s^2 = s^1 + 1$.

On the other hand, if there is a satisfying assignment, we already argued that player 2 can never lead with more than 1 point (the two players are forced to play the same number of 7-symbol words). However, during the very last turn, player 1 shall get the last symbol in the bag (a ***#***), and form some wall word anywhere on the board, which will make her the winner of the game. □

5. Conclusions and open Questions

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 formed words in more than one places on the board in the first, and the possibility of forming more than one possible words to play in the second. We have also established that hardness remains even when all relevant parameters are small constants.

Several interesting questions can be posed. First observe that, although the proofs are not affected by the pass move, they cannot incorporate exchanging tiles. It would be interesting to see a proof where this additional rule of the game can be applied, although in reality it is uncommon to use (from the official website of Hasbro: "SCRABBLE players don't ever exchange their tiles, but it can be to your benefit to refresh your rack").

Second, Theorem 4.1 proves the hardness of the game even when the alphabet size is constant. It would be interesting, though, to find the minimum value so that the problem remains PSPACE-hard. In particular, what happens when the alphabet contains just one letter? Does the problem become tractable in this case, 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 [3]: is there a polynomial-time algorithm to determine the move that maximizes the score achieved in some given 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.

Last, in this paper we study an artificial perfect-information model with made-up alphabet and dictionary. It would be interesting to study some model where words are taken from the English (or some other existing) language. Proving hardness for such a model will probably be quite harder, unless the dictionary contains only a subset of the words of the language, otherwise it might prove difficult to verify that no words are formed by accident while placing tiles on the board. It will also be interesting to study a model which allows imperfect-information.

Acknowledgments The authors would like to thank the anonymous reviewers for their very constructive comments and suggestions.

References

[1] Breukelaar, R., Demaine, E.D., Hohenberger, S., Hoogeboom, H.J., Kosters, W.A. and Liben-Nowell, D.: Tetris is hard, even to approximate, *International Journal of Computational Geometry & Applications*, Vol.14, No.1-2, pp.41–68 (2004).

[2] Demaine, E.D., Demaine, M.L., Harvey, N.J., Uehara, R., Uno, T. and Uno, Y.: Uno is hard, even for a single player, *Theoretical Computer Science*, Vol.521, pp.51–61 (2014).

[3] Demaine, E.D. and Hearn, R.A.: Playing games with algorithms: Algorithmic combinatorial game theory, *Games of No Chance III, Proc. BIRS Workshop on Combinatorial Games, July 2005*, Vol.56, pp.3–56 (2009).

[4] Even, S. and Tarjan, R.E.: A combinatorial problem which is complete in polynomial space, *J. ACM* (*JACM*), Vol.23, No.4, pp.710–719 (1976).

[5] Fraenkel, A. and Lichtenstein, D.: Computing a perfect strategy for $n \times n$ chess requires time exponential in n, *J. Combinatorial Theory, Series A*, Vol.31, No.2, pp.199–214 (1981).

[6] Iwata, S. and Kasai, T.: The Othello game on an $n \times n$ board is PSPACE-complete, *Theoretical Computer Science*, Vol.123, No.2, pp.329–340 (1994).

[7] Papadimitriou, C.M.: *Computational complexity*, Addison-Wesley, Reading, Massachusetts (1994).

[8] Robson, J.M.: The Complexity of Go, *IFIP Congress*, pp.413–417 (1983).

Michael Lampis is a maître de conférences at the Universite Paris Dauphine. Before that he worked as a postdoctoral researcher at RIMS, Kyoto University and in the Royal Institute of Technology in Stockholm, Sweden. He obtained his Ph.D. from the City University of New York is 2011. His main research interests are Parameterized Complexity, Approximation Algorithms, and their intersection. He has (co)authored 14 journal and 16 conference publications.

Valia Mitsou obtained her Ph.D. from the City University of New York in 2014. She worked as a postdoctoral researcher for the JST Erato Minato project from June 2014 until March 2015. In May 2015, she will start a postdoc at the Hungarian Academy of sciences in Budapest, Hungary. Her main research interests include complexity theory, graph algorithms, and the complexity of games and puzzles. She has co-authored 1 book chapter, 6 articles in international scientific journals, and 8 publications in international conference proceedings.

Karolina Sołtys is a Ph.D. candidate at University College of London since September 2014. She obtained her bachelor's degree in Computer Science from Warsaw University in 2010 and her master's degree from Ecole Polytechnique in 2011. From August 2011 until December 2011 she was a Ph.D. student at UC Berkeley, then she moved to Max Planck Institute fur Informatik where she stayed until 2014. She has co-authored 3 journal and 2 conference articles.