HOW TO CONSTRUCT TETRIS CONFIGURATIONS

Hendrik Jan Hoogeboom and Walter A. Kosters
Leiden Institute of Advanced Computer Science (LIACS)
Universiteit Leiden
P.O. Box 9512, 2300 RA Leiden, The Netherlands
E-mail: {hoogeboo,kosters}@liacs.nl

KEYWORDS

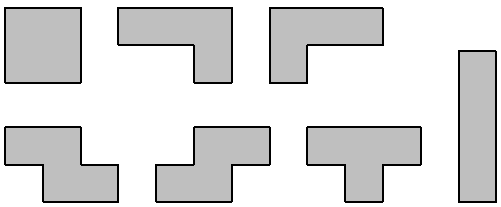
TETRIS, reachable configurations.

ABSTRACT

In this paper we show that every (reasonable) TETRIS configuration can be constructed from an initially empty game board using a suitable sequence of pieces. On game boards of even width a simple parity condition has to be fulfilled. The paper provides an explicit construction. This problem is connected to the NP-completeness of certain TETRIS related decision problems.

INTRODUCTION

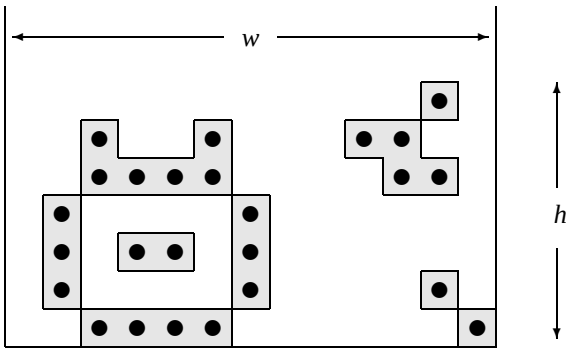
It has recently been shown that certain decision problems concerning the TETRIS game are NP-complete (Demaine et al. 2002; Breukelaar et al. 2003; Breukelaar et al. 2004) or (un)decidable (Hoogeboom and Kosters 2004). This fits in the larger research picture, where for many games these sorts of problems are addressed (Berlekamp et al. 1982; Demaine 2001). In the current paper we are interested in the possibility of generating a given TETRIS configuration by supplying suitable pieces and handling them in the appropriate way. This problem arose during the proof of the NP-completeness, where the constructibility of certain configurations used in the reduction was questioned. Note that similar questions for games as chess are very complicated.



A regular TETRIS game is played on a rectangular game board with squares; it uses pieces (all occupying 4 squares) in 7 shapes, as in the following picture. Their names and notations for this paper are *square* `Sq`, *left gun* `LG`, *right gun* `RG`, *left snake* `LS`, *right snake* `RS`, *T* `T`, and *I* `I`, respectively.

In a regular TETRIS game, random pieces are presented to the player, one at a time. Initially such a piece is located in the middle of the top row of the game board. The player can translate and/or rotate the current piece, while it is falling down. If the piece meets the floor or another piece, it is fixed. If a whole row (or even more than one) gets filled, it is *cleared*: it is removed from the game board, and the rows above it fall one row down. Note that (parts of) pieces do not fall further down, once rows below it are cleared. The game ends when the new piece cannot fall down anymore, because it is blocked by pieces below it. The more pieces the player has handled, and the more rows he or she cleared, the better it is. For a more precise definition of the game, consult (Demaine et al. 2002); for other results on TETRIS, see, e.g., (Burgiel 1997).

A TETRIS *configuration* is a game board, where some of the squares are already occupied. A configuration is called *constructible* if it is possible to reach it, from an initially empty board, with a suitable series of pieces using appropriate translations and rotations. In this paper we shall prove that essentially every reasonable configuration is constructible, see Theorem 3. The one non-trivial exception deals with boards of even width, where some simple parity condition should be fulfilled. The example configuration below, on a board of width 13, is constructible. Our construction requires 276 TETRIS pieces, and clears $(4 \cdot 276 - 25)/13 = 83$ intermediate rows. Note that the rules of TETRIS allow squares to float, as in the upper right hand part of the configuration.



OVERVIEW

The occupied squares from the configuration are called *pixels*; the example configuration above has 25 of them. The width w of the game board, 13 for the example, should be at least 4.

From now on we shall assume $w \geq 5$, the case $w = 4$ being an easy adaptation. Let h denote the height of the configuration, i.e., the number of non-empty rows; in the example configuration it is 7. All congruences (denoted by $\equiv$) are modulo 4; e.g., $p \equiv 2$ means that $p-2$ is divisible by 4.

We will assume that there is enough room above the configuration to perform the proper piece rotations and translations. The construction itself requires some extra room; 8 rows will suffice. Once a column and an orientation for a piece is fixed, it just falls down; it is neither translated nor rotated anymore. Hence, there is no need for any special rotation model (cf. (Demaine et al. 2002)).

Results

If the width of the board equals $w = 12$ then every cleared row decreases the number of occupied squares by 12 and every new TETRIS piece increases it by 4. Hence, in every configuration the number of pixels will be a multiple of four. With the same reasoning, in the general case we have:

Lemma 1 *Consider a constructible* TETRIS *configuration with p pixels on a board of width w. Then there exists an integer u such that $u \cdot w + p \equiv 0$.*

Indeed, u can be taken equal to the number of cleared rows. An immediate consequence (for $w = 4, 8, 12, \ldots$, and $w = 6, 10, 14, \ldots$) is:

Corollary 2 *Consider a constructible* TETRIS *configuration with p pixels on a board of width w. If $w \equiv 0$, then $p \equiv 0$; if $w \equiv 2$, then $p \equiv 0$ or $p \equiv 2$.*

The main result of this paper is that these conditions are also sufficient:

Theorem 3 *A configuration of p pixels is constructible using suitable* TETRIS *pieces starting from the empty board of width w if and only if*

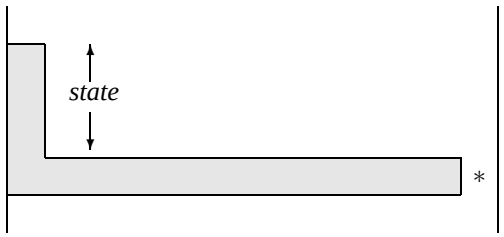
1. *no row is completely full, and*
2. *no row below the highest one containing pixels is completely empty, and*
3. *if $w \equiv 0$, then $p \equiv 0$; if $w \equiv 2$, then $p \equiv 0$ or $p \equiv 2$.*

The rest of this paper is dedicated to the proof of this theorem, giving an explicit construction (see (Hoogeboom and Kosters 2003) for an implementation in the form of a Java-applet). In the sequel we shall assume that all three conditions mentioned in the theorem are met.

The Construction

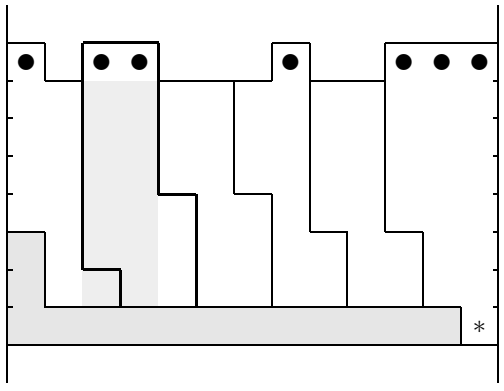
The configuration on the board is constructed row-by-row in a modular fashion. For each row the construction consists of two phases.

First we build a *platform* that serves as a scaffolding for the construction work (it prevents TETRIS pieces from falling down to lower rows). In general the platform looks as follows. The $*$ denotes the bottom right empty square of the platform; once it is filled, its whole row will be cleared. In some cases the platform is mirrored.



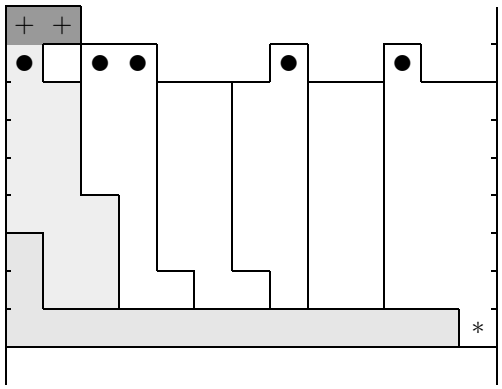
The number of squares sticking out vertically above the platform at the left end may vary between 0 and 3, and is referred to as the *state of the platform*. The state depends on properties of the part of the board that already has been built below it and on the number of rows cleared during its construction. We need such a state as the total number of squares presently occupied or cleared in the past must be a multiple of 4, cf. Lemma 1. The construction of the platform is given in a separate section.

In the second phase, sometimes referred to as the row construction phase, we build the pixels ● of the next row of the configuration on top of the platform, using six additional rows. Basically we construct consecutive blocks two columns wide and six rows high with the necessary pixels on top, proceeding from left to right. Again, however, we have the multiple of 4 restriction, and we carry a surplus of squares as *state* of the blocks. This state is visible as indent of up to three squares at the bottom in the left column of the block. The last block is designed to fill both the final square of the platform and the six rows of the blocks, clearing all additional squares that are not part of the final configuration.



As always, the number of squares occupied in the construction need not be a multiple of 4, and we have to take this into account. We solve this by allowing a group of up to three additional squares placed on top of one of the pixels. This *overflow* is indicated by $+$ in the figure below. The overflow is used as a starting point in the construction of the platform for the next row of pixels. If there is no overflow then we start the construction by putting a horizontal `I` on top of one of the pixels of the last row (artificially introducing an overflow of

four).



The precise form of the blocks (for each state, number of pixels and overflow) is explained in a separate section. Particular care has to be taken for the last block which has to clear the six intermediate rows and the platform.

The whole construction starts with a horizontal I. It is extended to a platform with state 1. In order to remove the last overflow — if any —, the construction ends with some final details, see below.

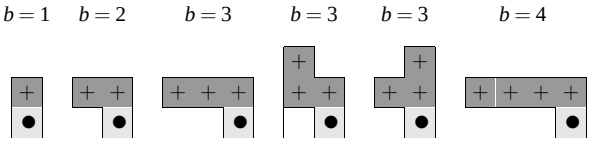
Note that in some cases there exist simpler constructions (e.g., for boards of odd width overflow can be handled differently), but we try to give a uniform approach. Indeed, some configurations are even extremely simple to reach (e.g., a single vertical I), whereas the construction from the sequel might use an abundance of pieces, clearing many rows on the way.

BUILDING THE PLATFORM

In this section we will erect a *platform* on the last row that was constructed so far. The platform will enable us to precisely construct the next row. After clearance of some additional rows the platform consists of a row which is completely filled, except for its rightmost square, plus up to three squares on top of the platform in the leftmost column, the *state* of the platform.

As we have seen, we have to pass an *overflow* of zero, one, two or three squares to the next row construction phase. Take a pixel ● that must be filled from the last row that was constructed. This pixel can be chosen at will, in our methods usually the leftmost one. On this single pixel we horizontally put the overflow of up to three squares. If there is no overflow, we add an extra horizontal I consisting of four squares.

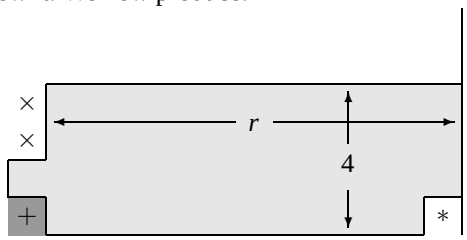
These b new squares together are referred to as the *overflow group*. The overflow group is situated on top of ● in some arbitrary horizontal position, for instance as far to the left as possible. These situations look like (with the b squares from the overflow group denoted by $+$'s):



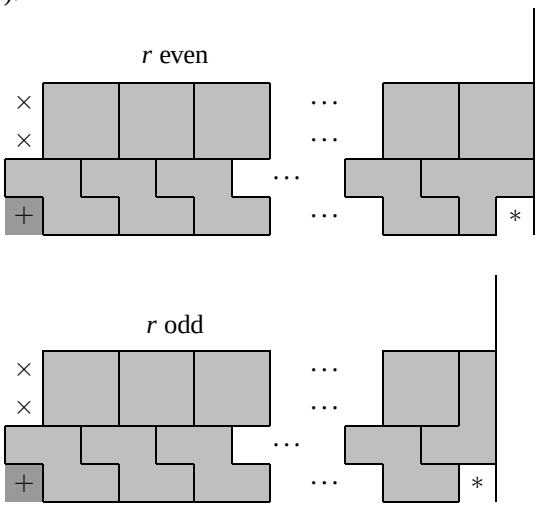
The second and third situation with $b = 3$ may only occur along the walls of the board. Note that the overflow group has to be built using appropriate pieces. For instance, the $b = 2$ case might be part of a RG. The precise construction is given below.

The value of b is such that the number of occupied squares so far (including these b) is a multiple of 4. This value is determined by the construction up till then, see below.

In the sequel we make frequent use of the following construction, which will be referred to as an *extension*. It starts from the rightmost square of an overflow group, denoted by $+$. Assume that there are $r > 0$ empty squares to its right. The $*$ again denotes the bottom right empty square of the platform to be, whereas the $\times$'s are squares intentionally left empty — for the moment. By the way, nothing is known about the row below it. We now produce:



This can be easily obtained with LS's starting from $+$, putting in one T or LG (dependent on the parity of r), and stacking Sq's into the two top rows (for $r = 1$ it consists of just a single LG):

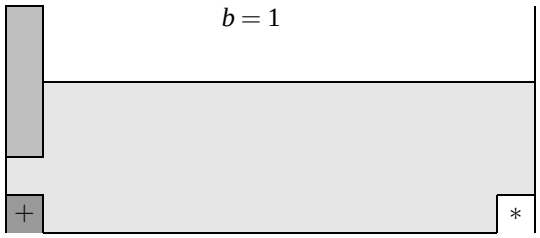


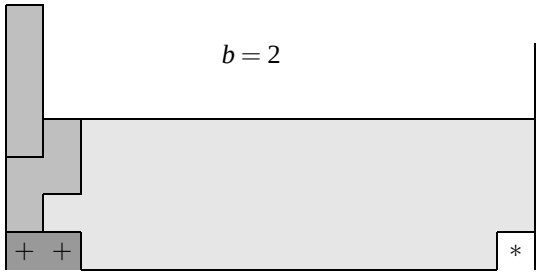
Notice that the final shape of the extension does not depend on the parity of r. Its construction requires exactly r TETRIS pieces.

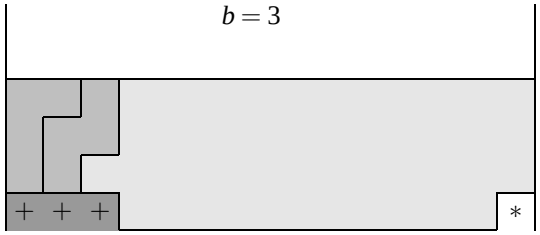
Now we can handle the platform construction. Let us say that there are $\ell \geq 0$ (still) empty squares to the left of the overflow group, and $r \geq 0$ (still) empty squares to its right. As the overflow group covers at most 4 columns of the board which we have assumed to be of width 5 or more, at least one of ℓ and r is nonzero.

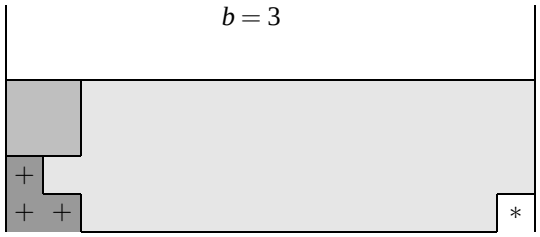
If $\ell = 0$, the overflow group is positioned against the left wall. Now first build an extension on the group, and then, depending

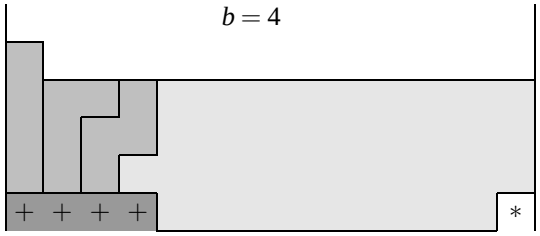
on b, proceed as follows:





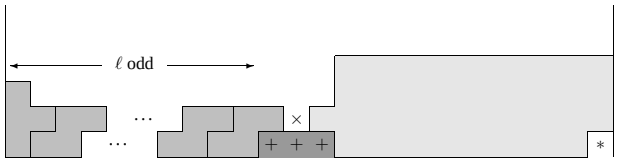


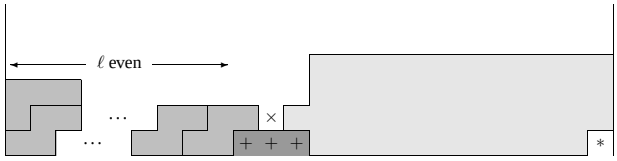




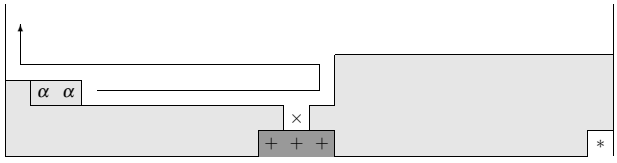
Note the two situations for $b = 3$. When $b > 2$ the final piece clears the top rows, in the other cases one or two rows are already cleared earlier (in some cases one row in the extension itself is cleared). In all cases the bottom row is never cleared due to the empty square to the right. If $r = 0$ we mirror this construction (and obtain a mirrored platform).

We now look at the case where both ℓ and r are nonzero. We first assume that $b > 1$, and start with an extension, followed by a series of RS's and a single T or RG. Depending on the parity of ℓ we have two cases, illustrated for $b = 3$:

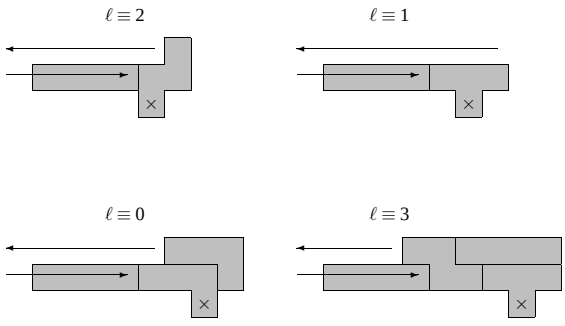




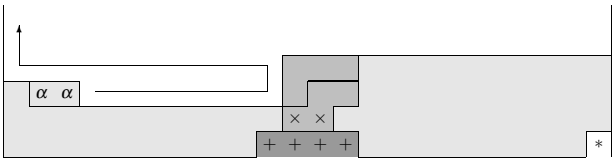
Remember that $\times$ denotes an empty square. The two cases can be combined into one, the α's denoting the difference:



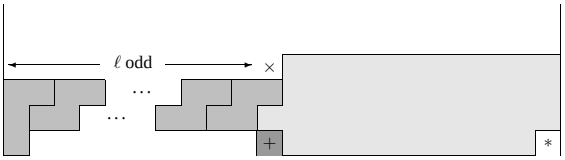
The case where $b = 2$ has its second row from below cleared. Now we can utilize *crawling*, which means the use of I's to fill long horizontal areas, starting from below, and using LG's, RG's and Sq's to go around edges in order to proceed to higher rows. In the picture the crawling is indicated by means of a long arrow, starting from the α-part. There are some necessary adaptations near $\times$ according to the value of b. If $b = 2$, there is no $\times$-square, and the crawling is easy. If $b = 3$ — as in the previous picture — there is one $\times$-square, and the crawling near this square proceeds like:



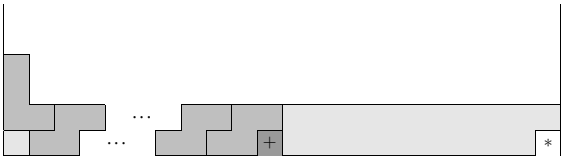
Finally we handle the crawling for $b = 4$, where there are two $\times$-squares, and where we give the overall picture:



It remains to consider the case $b = 1$ with both ℓ and r nonzero. If ℓ is odd, we first build an extension, followed by a series of RS's and a single LG:

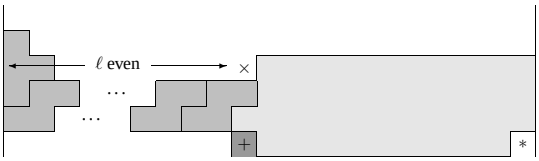


Now the two middle rows are cleared and we fill the remaining rows with RS's and a final RG, thereby clearing the top row:

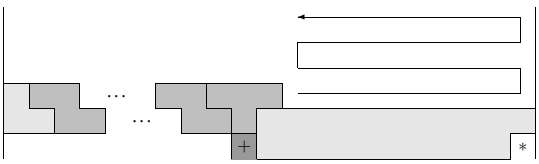


Up till this moment we always cleared three additional rows while building the platform. We now arrive at the last situation, where $b = 1$ and ℓ is even. In this case we were not able to find a solution with three additional rows, but we had to use four more. Note that by mirroring we could have assumed both ℓ and r to be even. We also have a construction with three additional rows for the case $b = 1$, ℓ even and $r = 2$, but unfortunately this did not generalize to larger r.

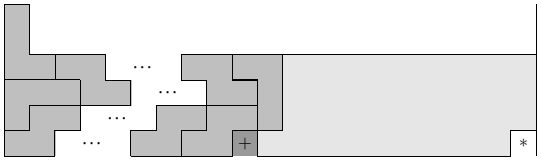
So we give a construction with seven extra cleared rows for the situation where $b = 1$ and $\ell \geq 2$ is even (there is no restriction on r except that it is nonzero). We start as usual with an extension and a series of RS's, this time followed by a T:



thereby clearing two rows. We then clear the two top rows by means of crawling (not necessary if $r = 1$; one can also use a sequence of vertical I's), continued for three more rows, followed by a series of LS's and one T:



And finally, using two series of snakes and three RG's, we get:



This completes the platform construction. We make the following observation.

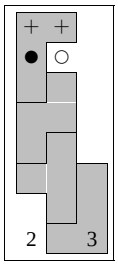
Lemma 4 *Starting from an overflow group with $b \in \{1,2,3,4\}$ squares we turn this into a platform with state $(b+1)$ (modulo 4), using w or 2w* TETRIS *pieces, meanwhile creating and clearing 3 or 7 additional rows.*

CREATING A ROW OF PIXELS

A row of pixels is built on top of the platform using building blocks of two columns wide and six additional rows in between platform and row of pixels. The six additional rows are cleared together with the platform during the construction. This leaves the rows below unchanged, builds the row under construction, and possibly places an overflow on the next row. Again, no further properties of the pixel row immediately below are used.

The number of additional rows, six, might seem arbitrary. It is indeed possible to use fewer rows, at the cost of a less uniform construction.

Each block has a basic shape of 2×6 with up to two pixels on top. When there is a pixel on top, the block cannot be formed by an integer number of TETRIS pieces. We compensate for this by introducing a *state transition* induced by the block: each block may leave up to three additional squares in the first column of the next block. If a block starts in state s (i.e., the s bottom squares of its first column are already occupied; $s = 0,1,2,3$), it carries q pixels ($q = 0,1,2$) and it has overflow f ($f = 0,1,2,3$), then it leaves state $t \in \{0,1,2,3\}$ determined by $q+t-s+f \equiv 0$ (so $q+t-s+f$ is divisible by 4).

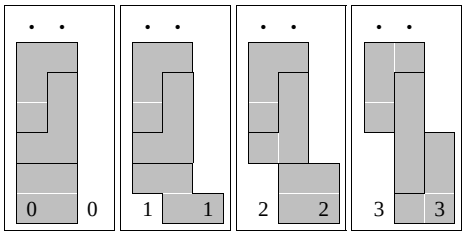


The figure illustrates a block from state $s = 2$ to state $t = 3$, putting a single pixel ($q = 1$), and an overflow of $f = 2$. Larger dots indicate the presence ● or absence ○ of a pixel. Indeed $q+t-s+f = 1+3-2+2$ is (a multiple of) 4.

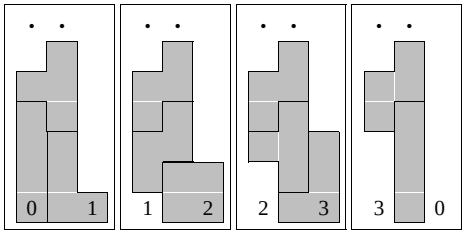
As there are four possible ways to put up to two pixels on a block, we have to design sixteen *basic blocks*, without overflow. In the sequel, small dots illustrate the position of the row of pixels under construction.

Transitions and Blocks

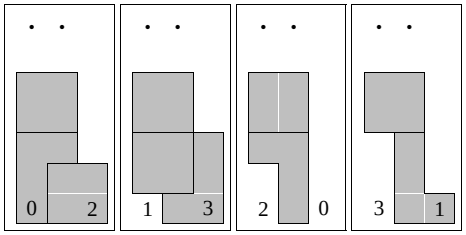
Transitions are incomplete blocks that consist of Tetris pieces changing one state into another. The transitions are characterized by their state change modulo four. Each type of transition has a characteristic shape in the two topmost rows.



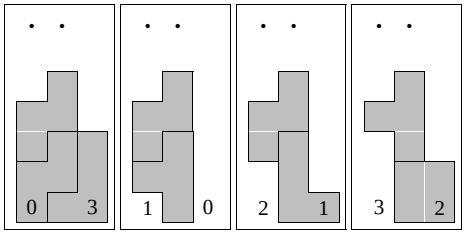
+0-transitions



+1-transitions



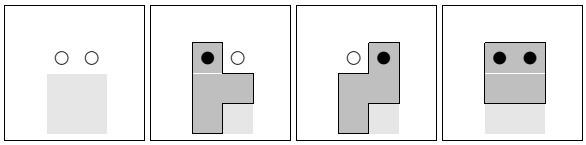
+2-transitions



+3-transitions

Basic blocks. Using these transitions as base, we now give the sixteen basic blocks, i.e., blocks carrying pixels, but no overflow.

- *No pixels.* A +0-transition will do the work.
- *Single pixel.* Put a T (left pixel) or a LS (right pixel) on top of a +3-transition.
- *Two pixels.* Put a Sq on top of a +2-transition.



Overflow blocks. As the number of squares occupied (including those cleared afterwards) during the construction is a multiple of four, we may have to put an overflow of up to three squares on top of one of the pixels in the row. This means that (at most) one of the blocks used to build the row of pixels is an *overflow block*. Here we show how to construct them. There is a large number of these blocks, depending on state, pixels and overflow.

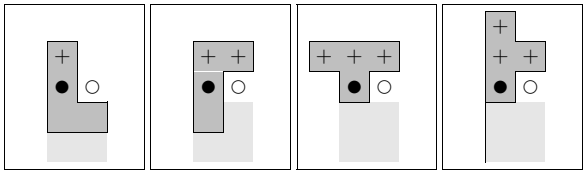
Note that we do not have to provide overflow blocks for the case where the block carries no pixels. Moreover, overflow zero is not considered as this is taken care of by the basic blocks above.

- *Single pixel, left.*

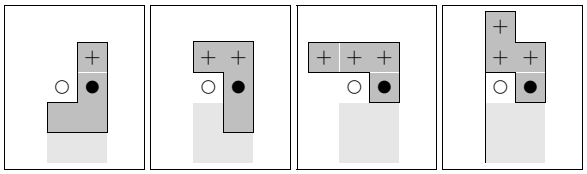
  One overflow. Place a RG on top of a +2-transition.

  Two overflow. Place a LG on top of a +1-transition.

  Three overflow. Place a T on top of a +0-transition. This will *not* work if the block under consideration is the leftmost block, as there is no room to the left to put the third overflow square. This is solved by a special form of the overflow. It is also possible to use a RG, but this might cause some (solvable) problems for the next block.



- *Single pixel, right.* These cases are symmetric to the ones above. For the second situation however, we need a variant of the +1-transitions, having an empty square top-right rather than top-left. These can be obtained by replacing the topmost piece (T or LS) by another one (RS or T, respectively).

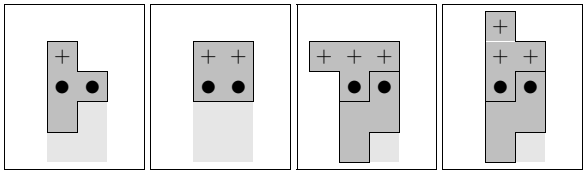


- *Two pixels.*

  One overflow. Place a T on top of a +1-transition.

  Two overflow. Place a Sq on top of a +0-transition.

  Three overflow. Place two pieces on top of a +3-transition.



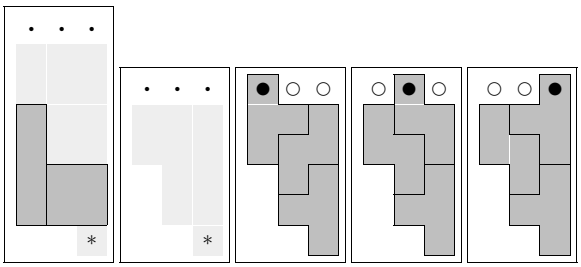
Final Block

The last pixel of the row that makes up the platform under the blocks is empty, and has to be filled to clear the platform. The process needs some attention. As lines are cleared we have to take care to leave support for the pieces that are dropped later. Usually the order of the pieces is immediately clear, but in some cases special care is necessary.

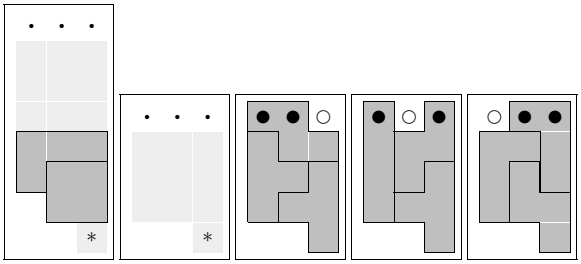
If the final block has width v ($v = 2,3$; v and w are equal modulo 2), starts in state s ($s = 0,1,2,3$), places q pixels ($q = 0,\ldots,v$), leaving overflow f ($f = 0,1,2,3$), these parameters must satisfy $6v+1-s+q+f \equiv 0$ (and $q = 0$ implies $f = 0$).

Odd width, no overflow. The final block has width $v = 3$, to complete an odd width row of pixels. We consider the cases that leave no overflow on the next row. The number of possibilities is limited by the formula above, simplifying to $s \equiv q + 3$.

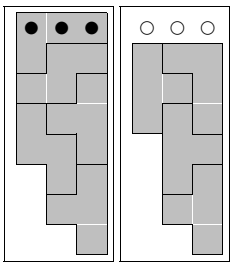
- *State* 0*, single pixel.* First place a `Sq` and `I`, clearing two lines. We then continue to clear the area while adding the final pixel.



- *State 1, two pixels.* First place a `Sq` and `RG`, clearing three lines. We then continue to clear the area while adding the final pixels; note that in the second case the `I` goes in first.



- *State 2, three pixels; state 3, no pixels.* Straightforward stacking.



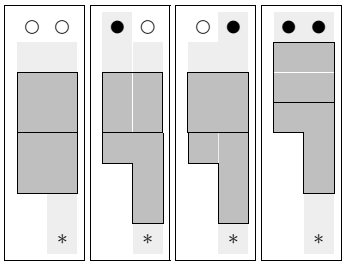
Even width, no overflow. The general construction resembles the previous case. The numerous possibilities are limited by $s \equiv q + 1$.

- *State 0.* This cannot occur without overflow.

- *State 1, no pixels.* Drop two `Sq`'s, clearing four rows. The remaining squares form the final `RG`.

- *State 2, one pixel.* Drop a `RG` and a `Sq`. The remaining squares form a final piece.

- *State 3, two pixels.* Again the same strategy: a `RG`, a `Sq` and a `RG` will work.

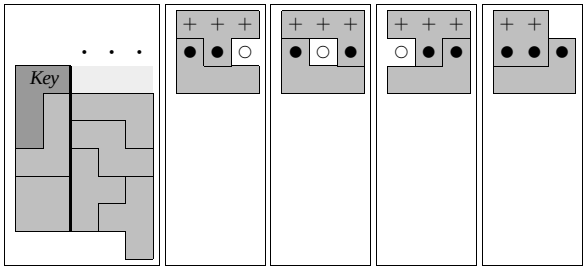


Final overflow. Placing overflow in the final block is a difficult task as we have to build a structure on a disappearing platform.

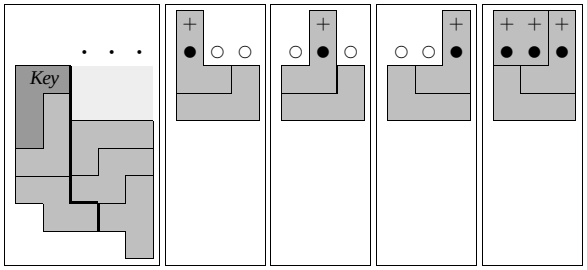
In general, we may assume that the necessary overflow is placed on top of the first (leftmost) pixel in the row. Hence, if we are forced to place the overflow in the final block, we may use the fact that the blocks to the left do not contain pixels, and that they are all in the same state. We will consider the previous block together with the final block, and do not fill that block using the final `LG` until the final block has been completed. In that way we postpone clearing the rows, allowing us to build a stable structure. In the pictures the final `LG` is indicated by *Key*, it is the last piece added to the construction.

Odd width, overflow. If the final block has width $v = 3$, then we are restricted to cases where $3 - s + q + f \equiv 0$, while additionally $f \neq 0$ and $q \in \{1, 2, 3\}$.

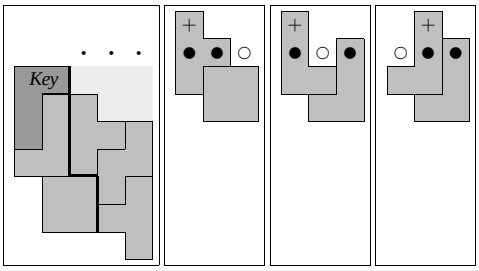
- *State 0, two pixels, three overflow; state 0, three pixels, two overflow.* First we build the last-but-one block except the last piece marked *Key*, then we complete the final block, and finally we use the `LG` as a key.

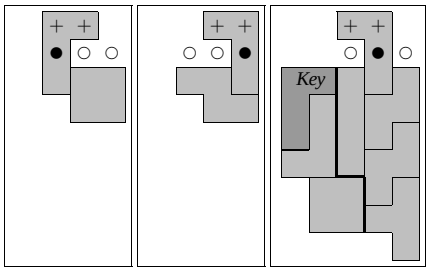


- *State 1, one pixel, one overflow; state 1, three pixels, three overflow.*

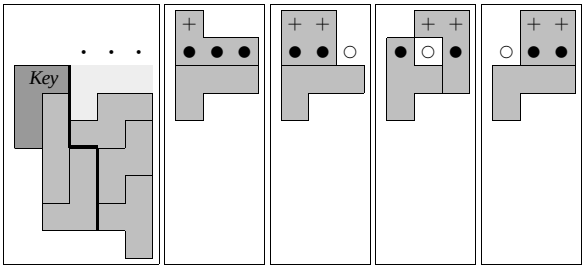


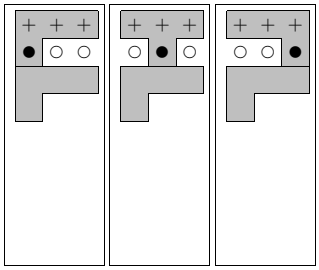
- *State 2, two pixels, one overflow; state 2, one pixel, two overflow.* The final variant has a separate solution; note that the `I` goes in before the `LS`'s.





- *State 3, three pixels, one overflow; state 3, two pixels, two overflow; state 3, one pixel, three overflow.*

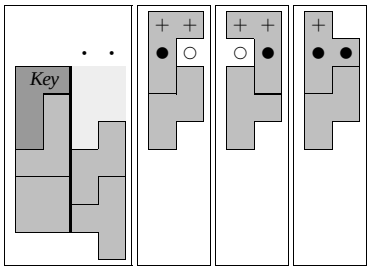




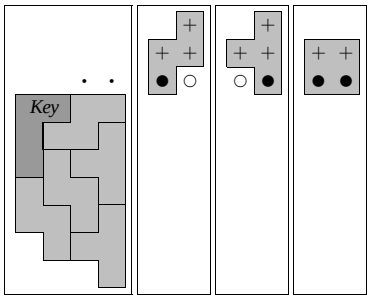
Even width, overflow. Here we consider the case when the last block has width $v = 2$ while it should carry overflow. This means that none of the previous blocks contain pixels. The possibilities we consider are limited by $1-s+q+f \equiv 0$, $f \neq 0$ and $q \in \{1,2\}$.

Again we include the previous block in the construction, either by postponing the completion of that block until the final block has been built, or by integrating the construction of the last block and the previous one, effectively creating a block of width four.

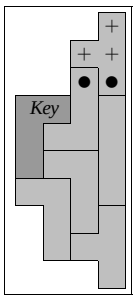
- *State 0, one pixel, two overflow; state 0, two pixels, one overflow.* Dropping the final `LG` of the last-but-one block is postponed.



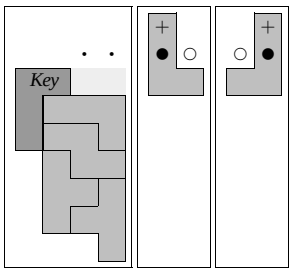
- *State 1, one pixel, three overflow; state 1, two pixels, two overflow.*



- *State 2, two pixels, three overflow.* The final two blocks are combined, as above. Again the `I` has to go in before the bottom rows are cleared.



- *State 3, one pixel, one overflow.*



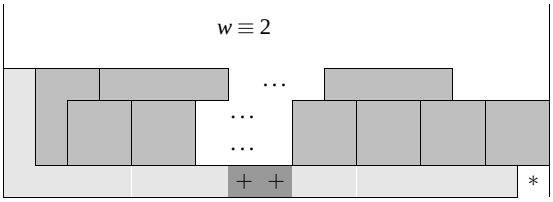
THE LAST DETAILS

The construction proceeds until the last pixels from the top row have been accomplished. We then have an overflow group of 0, 1, 2 or 3 squares we have to get rid of. If it is 0, we are already done. In this section we deal with the remaining cases.

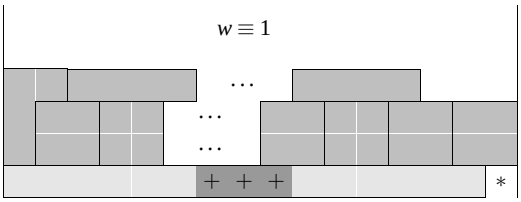
From below we are handed an overflow for every new pixel row, say b. In the current construction we use 10 (or sometimes 14; note that $14 \equiv 10$) intermediate rows that are cleared, containing $10w-b$ new squares. Let us say that the following row contains k pixels. We then choose the unique $b' \in \{0,1,2,3\}$ such that $10w-b+k+b' \equiv 0$ (remember that this means that $10w-b+k+b'$ is divisible by 4) as next overflow. This will guarantee that indeed the construction can be performed. Beginning with a first overflow group of size 4 and telescoping

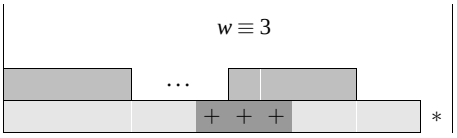
on this formula, for the last b we arrive at $b \equiv -10hw - p \equiv 2hw - p$, where p is the total number of pixels and h is the height of the configuration under construction.

From Corollary 2 we know that $w \equiv 0$ implies $p \equiv 0$, and therefore $b = 0$. If $w \equiv 2$, either $p \equiv 0$ or $p \equiv 2$; if $p \equiv 0$, $b = 0$; if $p \equiv 2$, $b = 2$. The latter case can be resolved by building a platform with state 3 on this overflow group, and clearing the whole thing with three extra rows (the middle two being cleared first; use a final LG):

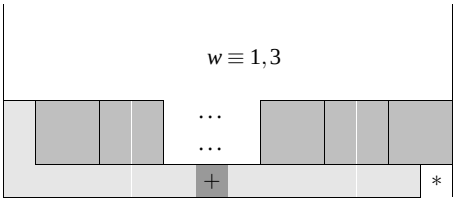


If $w \equiv 1$ or $w \equiv 3$, the final b can have any value. If $b = 2$, we first build a virtual next row containing zero pixels, in the usual way. We then get $b' \equiv b + 2w \equiv 2 + 2w \equiv 0$, and we are done. Next we deal with $b = 3$. As above, we build a platform with state 0 on this overflow group. This can be totally cleared with one or three extra rows, again using a final LG, the $w \equiv 1$ case being analogous to the $w \equiv 2$ situation above:





Finally we handle $b = 1$ by first clearing the two top rows:



and finish this as in the case $b = 3$. By the way, these constructions show that for odd w it is possible to reduce the overflow for each row to 0 — which is definitely not possible for even w.

CONCLUSION

We have shown that every reasonable TETRIS configuration is constructible, if a simple parity condition on the configuration is met.

To this problem we can attach a decision problem: given a configuration and an ordered series of TETRIS pieces, their sizes satisfying a suitable congruence modulo 4; is it possible to construct the configuration using this series? It would be interesting to understand the complexity of this problem, in particular concerning NP-completeness (see (Garey and Johnson 1979)). A similar problem is the quest for minimum length series of pieces that generate a given configuration.

We thank the referees for their comments.

REFERENCES

Berlekamp, E.R., J.H. Conway and R.K. Guy. 1982. *Winning Ways for Your Mathematical Plays, Volume 1 and 2*. Academic Press, New York.

Breukelaar, R., E.D. Demaine, S. Hohenberger, H.J. Hoogeboom, W.A. Kosters and D. Liben-Nowell. 2004. "Tetris is Hard, Even to Approximate." *International Journal of Computational Geometry and Applications*, 14: 41–68. doi:10.1142/S0218195904001354

Breukelaar, R., H.J. Hoogeboom and W.A. Kosters. 2003. "Tetris is hard, made easy." Technical Report, Leiden Institute of Advanced Computer Science, Universiteit Leiden.

Burgiel, H. 1997. "How to Lose at Tetris." *Mathematical Gazette*, 81: 194–200.

Demaine, E.D. 2001. "Playing Games with Algorithms: Algorithmic Combinatorial Game Theory." In J. Sgall, A. Pultr and P. Kolman, editors, *Proceedings 26th International Symposium on Mathematical Foundations of Computer Science MFCS2001*, volume 2136 of *Lecture Notes in Computer Science*, Springer-Verlag, 18–32. (Springerlink)

Demaine, E.D., S. Hohenberger and D. Liben-Nowell. 2002. "Tetris is Hard, Even to Approximate." In T. Warnow and B. Zhu, editors, *Proceedings Computing and Combinatorics, 9th Annual International Conference, COCOON 2003*, volume 2697 of *Lecture Notes in Computer Science*, Springer-Verlag, 351 – 363. (Springerlink)

Garey, M.R. and D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, New York.

Hoogeboom, H.J. and W.A. Kosters. 2003. Website — How to Construct Tetris Configurations. URL http://www.liacs.nl/~kosters/tetris/.

Hoogeboom, H.J. and W.A. Kosters. 2004. "Tetris and Decidability." *Information Processing Letters*, 89: 267–272. doi:10.1016/j.ipl.2003.12.006