The Game of Tetris in Machine Learning

Article · August 2017

CITATIONS
7

READS
3,075

2 authors, including:

Simón Algorta
Max Planck Institute for Human Development
4 PUBLICATIONS **7** CITATIONS

The Game of Tetris in Machine Learning

Simón Algorta [1] **Özgür Şimşek** [2]

Abstract

The game of Tetris is an important benchmark for research in artificial intelligence and machine learning. This paper provides a historical account of the algorithmic developments in Tetris and discusses open challenges. Handcrafted controllers, genetic algorithms, and reinforcement learning have all contributed to good solutions. However, existing solutions fall far short of what can be achieved by expert players playing without time pressure. Further study of the game has the potential to contribute to important areas of research, including feature discovery, autonomous learning of action hierarchies, and sample-efficient reinforcement learning.

1. Introduction

The game of Tetris has been used for more than 20 years as a domain to study sequential decision making under uncertainty. It is generally considered a rather difficult domain. So far, various algorithms have yielded good strategies of play but they have not approached the level of performance reached by expert players playing without time pressure. Further work on the game has the potential to contribute to important topics in artificial intelligence (AI) and machine learning, including feature discovery, autonomous learning of action hierarchies, and sample-efficient reinforcement learning.

In this article, we first describe the game and provide a brief history. We then review the various algorithmic approaches taken in the literature. The most recent and successful player was developed using approximate dynamic programming. We conclude with a discussion of current challenges and how existing work on Tetris can inform approaches to other games and to real-world problems. In the Appendix we provide a table of the algorithms reviewed and a description of the features used.

[1]ABC Research Group at the Max Planck Institute for Human Development, Berlin, Germany [2]University of Bath, Bath, United Kingdom. Correspondence to: Simón Algorta <algorta@mpib-berlin.mpg.de>.

VGML Workshop at the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

2. The Game of Tetris

Tetris is one of the most well liked video games of all time. It was created by Alexey Pajitnov in the USSR in 1984. It quickly turned into a popular culture icon that ignited copyright battles amid the tensions of the final years of the Cold War (Temple, 2004).

The game is played on a two-dimensional grid, initially empty. The grid gradually fills up as pieces of different shapes, called *Tetriminos*, fall from the top, one at a time. The player can control how each Tetrimino lands by rotating it and moving it horizontally, to the left or to the right, any number of times, as it falls one row at a time until one of its cells sits directly on top of a full cell or on the grid floor. When an entire row becomes full, the whole row is deleted, creating additional space on the grid. The game ends when there is no space at the top of the grid for the next Tetrimino. Each game of Tetris ends with probability 1 because there are sequences of Tetriminos that terminate the game, no matter how well they are placed (Burgiel, 1997). Figure 1 shows a screenshot of the game along with the seven Tetriminos.

Despite its simple mechanics, Tetris is a complex game. Even if the full sequence of Tetriminos is known, maximizing the number of rows cleared is NP-complete (Demaine et al., 2003).

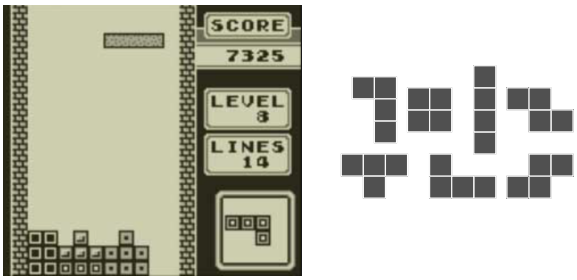

Figure 1. Screenshot of Tetris on Gameboy and the seven Tetriminos.

Tetris can be modeled as a Markov decision process. In the

most typical formulation, a state includes the current configuration of the grid as well as the identity of the falling Tetrimino. The available actions are the possible legal placements that can be achieved by rotating and translating the Tetrimino before dropping it.

This formulation ignores some pieces of information provided in some implementations of the game, for example, the identity of the *next* Tetrimino to fall after the current one is placed. It also excludes actions that are available in some implementations of the game. One example is sliding a Tetrimino under a cliff, which is known as an *overhang*. Another example is rotating the T-shaped Tetrimino to fill an otherwise unreachable slot at the very last moment. This maneuver is known as a *T-spin* and it gives extra points in some implementations of the game.

The original version of the game used a scoring function that awards one point for each cleared line. Subsequent versions allocate more points for clearing more than one line simultaneously. Clearing four lines at once, by placing an I-shaped Tetrimino in a deep well, is allocated the largest number of points. Most implementations of Tetris by researchers use the original scoring function, where a single point is allocated to each cleared line.

3. Algorithms and Features

Tetris is estimated to have 7×2^{200} states. Given this large number, the general approach has been to approximate a value function or learn a policy using a set of features that describe either the current state or the current state–action pair.

Tetris poses a number of difficulties for research. First, small changes in the implementation of the game cause very large differences in scores. This makes comparison of scores from different research articles difficult. Second, the score obtained in the game has a very large variance. Therefore, a large number of games need to be completed to accurately assess average performance. Furthermore, games can take a very long time to complete. Researchers who have developed algorithms that can play Tetris reasonably well have found themselves waiting for days for a single game to be over. For that reason, it is common to work with grids smaller than the standard grid size of 20×10. A reasonable way to make the game shorter, without compromising its nature, is to use a grid size of 10×10. The scores reported in this article are those achieved on the standard grid of size 20×10, unless otherwise noted.

The most common approach to Tetris has been to develop a linear evaluation function, where each possible placement of the Tetrimino is evaluated to select the placement with the highest value. In the next sections, we discuss the features used in these evaluation functions, as well as how the weights are tuned.

3.1. Early Attempts

Tsitsiklis & Van Roy (1996) used Tetris as a test bed for large-scale feature-based dynamic programming. They used two simple state features: the number of holes, and the height of the highest column. They achieved a score of around 30 cleared lines on a 16×10 grid.

Bertsekas & Tsitsiklis (1996) added two sets of features: height of each column, and the difference in height between consecutive columns. Using lambda-policy iteration, they achieved a score of around 2,800 lines. Note, however, that their implementation for ending the game effectively reduces the grid to a size of 19×10.

Lagoudakis et al. (2002) added further features, including mean column height and the sum of the differences in consecutive column heights. Using least-squares policy iteration, they achieved an average score of between 1,000 and 3,000 lines.

Kakade (2002) used a policy-gradient algorithm to clear 6,800 lines on average using the same features as Bertsekas & Tsitsiklis (1996).

Farias & Van Roy (2006) studied an algorithm that samples constraints in the form of Bellman equations for a linear programming solver. The solver finds a policy that clears around 4,500 lines using the features used by Bertsekas & Tsitsiklis (1996).

Ramon & Driessens (2004) used relational reinforcement learning with a Gaussian kernel and achieved a score of around 50 cleared lines. Romdhane & Lamontagne (2008) combined reinforcement learning and case-based reasoning using patterns of small parts of the grid. Their scores were also around 50 cleared lines.

3.2. Hand-Crafted Agent

Until 2008, the best artificial Tetris player was handcrafted, as reported by Fahey (2003). Pierre Dellacherie, a self-declared average Tetris player, identified six simple features and tuned the weights by trial and error. These features were number of *holes*, *landing height* of the piece, number of *row transitions* (transitions from a full square to an empty one, or vice versa, examining each row from end to end), number of *column transitions*, *cumulative number of wells*, and *eroded cells* (number of cleared lines multiplied by the number of holes in those lines filled by the present Tetrimino). The evaluation function was as follows:

$$-4 \times holes - cumulative\ wells$$
$$- row\ transitions - column\ transitions$$
$$- landing\ height + eroded\ cells$$

This linear evaluation function cleared an average of 660,000 lines on the full grid. The scores were reported on an implementation where the game was over if the falling Tetrimino had no space to appear on the grid (in the center of the top row). In the simplified implementation used by the approaches discussed earlier, the games would have continued further, until every placement would overflow the grid. Therefore, this report underrates this simple linear rule compared to other algorithms.

3.3. Genetic Algorithms

Böhm et al. (2005) used evolutionary algorithms to develop a Tetris controller. In their implementation, the agent knows not only the falling Tetrimino but also the next one. This makes their results incomparable to those achieved on versions with knowledge of only the current Tetrimino. They evolved both a linear and an exponential policy. They reported 480,000,000 lines cleared using the linear function and 34,000,000 using the exponential function, both on the standard grid. They introduced new features such as the number of connected holes, number of occupied cells, and the number of occupied cells weighted by its height. These additional features were not picked up in subsequent research.

Szita & Lörincz (2006) used the cross-entropy algorithm and achieved 350,000 lines cleared. The algorithm probes random parameter vectors in search of the linear policy that maximizes the score. For each parameter vector, a number of games are played. The mean and standard deviation of the best parameter vectors are used to generate a new generation of policies. A constantly decreasing noise allows for an efficient exploration of the parameter space. Later, Szita & Lõrincz (2007) also successfully applied a version of the cross-entropy algorithm to Ms. Pac-Man, another difficult domain.

Following this work, Thiery & Scherrer (2009a;b) added a couple of features (hole depth and rows with holes) and developed the BCTS controller using the cross-entropy algorithm, where BCTS stands for building controllers for Tetris. They achieved an average score of 35,000,000 cleared lines. With the addition of a new feature, pattern diversity, BCTS won the 2008 Reinforcement Learning Competition.

In 2009, Boumaza introduced another evolutionary algorithm to Tetris, the covariance matrix adaptation evolution strategy (CMA-ES). He saw that the resulting weights were very close to Dellacherie's and also cleared 35,000,000 lines on average.

The success of genetic algorithms deserves our attention given the recent resurgence of evolutionary strategies as strong competitors for reinforcement learning algorithms (Salimans et al., 2017). Notably, they are easier to parallelize than reinforcement learning algorithms. As discussed below, the latest reported reinforcement learning algorithm uses an evolutionary strategy inside the policy evaluation step (Gabillon et al., 2013; Scherrer et al., 2015).

3.4. Approximate Modified Policy Iteration

Gabillon et al. (2013) found a vector of weights that achieved 51,000,000 cleared lines using a classification-based policy iteration algorithm inspired by Lagoudakis & Parr (2003). This is the first reinforcement learning algorithm that has a performance comparable with that of the genetic algorithms. Lagoudakis & Parr's idea is to make use of sophisticated classifiers inside the loop of reinforcement learning algorithms to identify good actions. Gabillon et al. (2013) estimated values of state–action pairs using rollouts and then minimized a complex function of these rollouts using the CMA-ES algorithm (Hansen & Ostermeier, 2001). This is the most widely known evolutionary strategy (Salimans et al., 2017). Within this algorithm, CMA-ES performs a cost-sensitive classification task, hence the name of the algorithm: classification-based modified policy iteration (CBMPI).

The sample of states used by CBMPI was extracted from trajectories played by BCTS and then subsampled to make the grid height distribution more uniform. CBMPI takes at least as long as BCTS to achieve a good level of performance. Furthermore, how best to subsample to achieve good performance is not well understood (Scherrer et al., 2015, p. 27).

We next review an approach of a different nature: how the structure of the decision environment allows for domain knowledge to be integrated into the learning algorithm.

4. Structure of the Decision Environment

Real-world problems present regularities of various types. It is therefore reasonable to expect regularities in the decision problems encountered when playing video games such as Tetris. Recent work has identified some of these regularties and has shown that learning algorithms can exploit them to achieve better performance (Şimşek et al., 2016).

Specifically, Şimşek et al. (2016) showed that three types of regularities are prevalent in Tetris: simple dominance, cumulative dominance, and noncompensation. When these conditions hold, a large number of placements can be (correctly) eliminated from consideration even when the exact weights of the evaluation function are unknown. For example, when one placement simply dominates another placement (which means that one placement is superior to the other in at least one feature and inferior in none), the dominated placement can be eliminated.

In Tetris, the median number of possible placements of the falling Tetrimino is 17. Şimşek et al. (2016) reduced this number to 3 by using simple dominance to eliminate inferior placements, and to 1 by using cumulative dominance.

The filtering of actions based on a few indicative features is an ability that can potentially be useful in many unsolved problems. Simpler and more sensitive functions for making decisions can be found after filtering inferior alternatives from the action space, perhaps in a way similar to how people's attention is guided to a small set of alternatives that they consider worthy.

5. Open Challenges

So far, the transformation of raw squares of the Tetris grid to a handful of useful features has been carried out by hand. Tetris is not yet part of the OpenAI universe or the Atari domain. No deep learning algorithm has learned to play well from raw inputs. Stevens & Pradhan and Lewis & Beswick (2015) have reported attempts that achieved at most a couple hundred lines cleared.

The scoring function where clearing multiple lines at once gives extra points makes a big difference in what policies score well. For this scoring function, it is likely that a linear evaluation function is not the best choice. Tetris would thus constitute a great test bed for learning hierarchies of actions (or options; Sutton et al. 1999), where a subgoal could be to set the stage for the I-shaped Tetrimino to clear four lines at once. T-spins are also performed this way: the player needs to set the stage and then wait for the T shape to be able to perform the maneuver. These subgoals are not defined by a unique state but by features of the grid that allow the desired action.

People enjoy playing Tetris and can learn to play reasonably well after a little practice. Some effort is being made to understand how people do this (Sibert et al., 2015). Progress in this research may help AI tackle the type of problems that gamers face every day.

Kirsh & Maglio (1994) gave a detailed account of how people perceive a Tetrimino and execute the actions necessary to place it. But an important question is not yet answered: How do people decide where to place the Tetrimino?

Finally, we have not yet developed an efficient learning algorithm for Tetris. Current best approaches require hundreds of thousands of samples, have a noisy learning process, and rely on a prepared sample of states extracted from games played by a policy that already plays well. The challenge is still to develop an algorithm that reliably learns to play using little experience.

6. Beyond Tetris

Can something be learned from Tetris that can be applied to bigger problems such as real-time strategy (RTS) or open-world games? Tetris may seem like a small game when compared to StarCraft or Minecraft. All of these games, however, share the difficulty that virtually no situation is encountered twice. Furthermore, fast learning should be possible by exploiting the regularities in the game.

Typically, AI bots facing RTS problems, such as StarCraft, deal with subtasks separately: high-level strategic reasoning is dealt with separately from tactics (Ontanón et al., 2013, p. 4). Tetris can be thought of as one of these subtasks, where a simple rule using an appropriate set of features can perform well. Different tasks may need different features. A high-level strategy, for instance, needs spatial features such as *trafficability*[1] (Forbus et al., 2002, p. 35) whereas tactics, where a unit decides to continue attacking or retreat, may use features such as the number of units. The unit-counting heuristic is reportedly used by a StarCraft bot to decide whether to retreat or keep attacking (Ontanón et al., 2013, p. 10).

Environmental structures such as those found in Tetris are likely to be present in other problems as well. In fact, simple dominance was also observed in backgammon, a complex game requiring multiple high-level strategies to play at expert level (Şimşek et al., 2016, p. 7). Similar approaches can inform the development of heuristics in problems such as StarCraft. Another example is based on the fact that units on higher ground always have an advantage over units on lower ground (Ontanón et al., 2013, p 3). Strategies where units move to higher ground will likely dominate other strategies. There may be no need to look further.

As AI research continues to deal with increasingly difficult real-world problems, inspiration may come from how people cope with the uncertainties in their lives. Video games, such as Tetris, provide controlled environments where people's decision making can be studied in depth. Game players have limited time and limited computational resources to consider every possible action open to them. Uncertainty about consequences, limited information about alternatives, and the sheer complexity of the problems they face make any exhaustive computation infeasible (Simon, 1972, p. 169). Yet they make hundreds of decisions in a minute and outperform every existing algorithm in a number of domains.

[1] Trafficability refers to "the ability of a vehicle or unit to move across a specified piece of terrain."

References

Bertsekas, Dimitri P and Tsitsiklis, John N. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

Böhm, N, Kóokai, G, and Mandl, S. An Evolutionary Approach to Tetris. *Proceedings of the 6th Metaheuristics International Conference (MIC2005)*, pp. 137–148, 2005.

Boumaza, Amine. On the evolution of artificial tetris players. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pp. 387–393. IEEE, 2009.

Burgiel, Heidi. How to Lose at Tetris. *The Mathematical Gazette*, 81(491):194–200, 1997.

Demaine, Erik D, Hohenberger, Susan, and Liben-Nowell, David. Tetris is hard, even to approximate. In *International Computing and Combinatorics Conference*, pp. 351–363. Springer, 2003.

Fahey, Colin. Tetris ai, June 2003. URL http://www.colinfahey.com/tetris/tetris.html.

Farias, Vivek F and Van Roy, Benjamin. Tetris: A study of randomized constraint sampling. In *Probabilistic and Randomized Methods for Design Under Uncertainty*, pp. 189–201. Springer, 2006.

Forbus, Kenneth D, Mahoney, James V, and Dill, Kevin. How qualitative spatial reasoning can improve strategy game ais. *IEEE Intelligent Systems*, 17(4):25–30, 2002.

Gabillon, Victor, Ghavamzadeh, Mohammad, and Scherrer, Bruno. Approximate dynamic programming finally performs well in the game of tetris. In *Advances in neural information processing systems*, pp. 1754–1762, 2013.

Hansen, Nikolaus and Ostermeier, Andreas. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398.

Kakade, Sham. A natural policy gradient. *Advances in neural information processing systems*, 2:1531–1538, 2002.

Kirsh, David and Maglio, Paul. On distinguishing epistemic from pragmatic action. *Cognitive science*, 18(4): 513–549, 1994.

Lagoudakis, Michail G and Parr, Ronald. Reinforcement learning as classification: Leveraging modern classifiers. In *ICML*, volume 3, pp. 424–431, 2003.

Lagoudakis, Michail G, Parr, Ronald, and Littman, Michael L. Least-squares methods in reinforcement learning for control. In *Hellenic Conference on Artificial Intelligence*, pp. 249–260. Springer, 2002.

Lewis, Ian J and Beswick, Sebastian L. Generalisation over details: the unsuitability of supervised backpropagation networks for tetris. *Advances in Artificial Neural Systems*, 2015:4, 2015.

Ontanón, Santiago, Synnaeve, Gabriel, Uriarte, Alberto, Richoux, Florian, Churchill, David, and Preuss, Mike. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.

Ramon, Jan and Driessens, Kurt. On the numeric stability of gaussian processes regression for relational reinforcement learning. In *ICML-2004 Workshop on Relational Reinforcement Learning*, pp. 10–14, 2004.

Romdhane, Houcine and Lamontagne, Luc. Reinforcement of local pattern cases for playing tetris. In *FLAIRS Conference*, pp. 263–268, 2008.

Salimans, Tim, Ho, Jonathan, Chen, Xi, and Sutskever, Ilya. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Scherrer, Bruno, Ghavamzadeh, Mohammad, Gabillon, Victor, Lesner, Boris, and Geist, Matthieu. Approximate Modified Policy Iteration and its Application to the Game of Tetris. *The Journal of Machine Learning Research*, 16:47, 2015.

Sibert, Catherine, Gray, Wayne D., and Lindstedt, John K. Tetris: Exploring human performance via cross entropy reinforcement learning models. In *submitted for Proceedings of the 38th Annual Conference of the Cognitive Science Society*, 2015.

Simon, Herbert A. Theories of bounded rationality. *Decision and organization*, 1(1):161–176, 1972.

Şimşek, Ö., Algorta, S., and Kothiyal, A. Why most decisions are easy in tetris-and perhaps in other sequential decision problems, as well. In *33rd International Conference on Machine Learning, ICML 2016*, volume 4, 2016.

Stevens, Matt and Pradhan, Sabeek. Playing tetris with deep reinforcement learning. *Unpublished*.

Sutton, Richard S, Precup, Doina, and Singh, Satinder. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Szita, István and Lörincz, András. Learning Tetris using the noisy cross-entropy method. *Neural computation*, 18 (12):2936–2941, 2006.

Szita, István and Lõrincz, András. Learning to play using low-complexity rule-based policies: Illustrations through ms. pac-man. *Journal of Artificial Intelligence Research*, 30:659–684, 2007.

Temple, M. (Director). From russia with love. BBC, 2004.

Thiery, Christophe and Scherrer, Bruno. Improvements on learning tetris with cross entropy. *Icga Journal*, 32(1): 23–33, 2009a.

Thiery, Christophe and Scherrer, Bruno. Building controllers for tetris. *Icga Journal*, 32(1):3–11, 2009b.

Tsitsiklis, John N and Van Roy, Benjamin. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1-3):59–94, 1996.

The Game of Tetris in Machine Learning—Appendix

Simón Algorta Özgür Şimşek

Algorithms

Table shows the algorithms that have been used to learn strategies for the game of Tetris, along with their reported scores and used feature sets.

Feature Sets

BERTSEKAS: Number of *holes*, height of the highest column (also known as *pile height*), *column height*, and *difference in height of consecutive columns*. Twenty-one features in total.

LAGOUDAKIS: Number of *holes*, pile height, *sum of differences in height of consecutive columns*, *mean height*, and the change in value of the mentioned features between current and next state. Finally, *cleared lines*. Nine features in total.

DELLACHERIE: number of *holes*, *landing height* of the Tetrimino, number of *row transitions* (transitions from a full square to an empty one, or vice versa, examining each row from end to end), number of *column transitions*, *cumulative wells*, and *eroded cells* (number of cleared lines multiplied by the number of holes in those lines filled by the present Tetrimino). A square that is part of a well is an empty square that can be reached from above with full squares on the sides. A well's depth is the number of vertically connected such squares. *Cumulative wells* is defined as $\sum_w \sum_{i=1}^{d(w)} i$, where w is a well and $d(w)$ is its depth.

BÖHM: *Pile height*, *connected holes* (same as holes but vertically connected holes count as one), *cleared lines*, *difference in height between the highest and lowest column*, *maximum well depth*, *sum of wells' depth*, *landing height* of the Tetrimino, *number of occupied squares*, *number of occupied squares weighted by their height*, *row transitions* and *column transitions*.

	ALGORITHM	GRID SIZE	LINES CLEARED	FEATURE SET USED
TSITSIKLIS & VAN ROY (1996)	APPROXIMATE VALUE ITERATION	16×10	30	HOLES AND PILE HEIGHT
BERTSEKAS & TSITSIKLIS (1996)	λ - PI	19×10	2,800	BERTSEKAS
LAGOUDAKIS ET AL. (2002)	LEAST-SQUARES PI	20×10	$\approx$ 2,000	LAGOUDAKIS
KAKADE (2002)	NATURAL POLICY GRADIENT	20×10	$\approx$ 5,000	BERTSEKAS
DELLACHERIE [REPORTED BY FAHEY (2003)]	HAND TUNED	20×10	660,000	DELLACHERIE
RAMON & DRIESSENS (2004)	RELATIONAL RL	20×10	$\approx$ 50	
BÖHM ET AL. (2005)	GENETIC ALGORITHM	20×10	480,000,000 (TWO PIECE)	BÖHM
FARIAS & VAN ROY (2006)	LINEAR PROGRAMMING	20×10	4,274	BERTSEKAS
SZITA & LÖRINCZ (2006)	CROSS ENTROPY	20×10	348,895	DELLACHERIE
ROMDHANE & LAMONTAGNE (2008)	CASE-BASED REASONING AND RL	20×10	$\approx$ 50	
BOUMAZA (2009)	CMA-ES	20×10	35,000,000	BCTS
THIERY & SCHERRER (2009A;B)	CROSS ENTROPY	20×10	35,000,000	DT
GABILLON ET AL. (2013)	CLASSIFICATION-BASED POLICY ITERATION	20×10	51,000,000	DT FOR POLICY DT + RBF FOR VALUE

Table 1. Scores and features used by reported algorithms in the game of Tetris. Only the highest score from each publication is reported.

BCTS (BUILDING CONTROLLERS FOR TETRIS): Dellacherie's feature set with the addition of *hole depth* (sum of full squares in the column above each hole) and *rows with holes*.

DT (DELLACHERIE PLUS THIERY): BCTS's feature set with the addition of *pattern diversity*, which is the number of patterns formed by the difference in height of two subsequent columns. For example, if a column has height 10 and the next one height 9, the pattern is 1. *Pattern diversity* is the number of distinct such patterns with a magnitude lower than 3.

RBF: Radial basis functions of the mean height of the columns. They are defined as $exp(\frac{-|c-ih/4|^2}{2(h/5)^2})$, i = 0, 1, 2, 3, 4, where c is the mean height of the columns and h is the total height of the grid.