

Improvements on Learning Tetris with Cross Entropy

Christophe Thiery, Bruno Scherrer

▶ **To cite this version:**

Christophe Thiery, Bruno Scherrer. Improvements on Learning Tetris with Cross Entropy. International Computer Games Association Journal, 2009, 32. inria-00418930

HAL Id: inria-00418930
https://inria.hal.science/inria-00418930

Submitted on 22 Sep 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

NOTES

IMPROVEMENTS ON LEARNING TETRIS WITH CROSS-ENTROPY

Christophe Thiery[1] *and Bruno Scherrer*[2]

Vandoeuvre-lès-Nancy Cedex, France

ABSTRACT

For playing the game of Tetris well, training a controller by the cross-entropy method seems to be a viable way (Szita and Lőrincz, 2006; Thiery and Scherrer, 2009). We consider this method to tune an evaluation-based one-piece controller as suggested by Szita and Lőrincz and we introduce some improvements. In this context, we discuss the influence of the noise, and we perform experiments with several sets of features such as those introduced by Bertsekas and Tsitsiklis (1996), by Dellacherie (Fahey, 2003), and some original features. This approach leads to a controller that outperforms the previous known results. On the original game of Tetris, we show that with probability 0.95 it achieves at least $910,000 \pm 5\%$ lines per game on average. On a simplified version of Tetris considered by most research works, it achieves $35,000,000 \pm 20\%$ lines per game on average. We used this approach when we took part with the program BCTS in the 2008 Tetris domain Reinforcement Learning Competition and won the competition.

1. INTRODUCTION

The Tetris game (see Fahey (2003) for a detailed description) was chosen as a benchmark problem by many researchers because it is known to be computationally hard to solve. It contains a huge number (about $2^{200} \simeq 10^{59}$) of board configurations[3] and finding the strategy that maximizes the average score is an NP-complete problem (Demaine, Hohenberger, and Liben-Nowell, 2003). An overview of previous Tetris works such as hand-written controllers, reinforcement learning approaches and optimization algorithms, as well as a list of all Tetris features known from the literature, are provided in (Thiery and Scherrer, 2009). In this note, we explain how, based on ideas from successful works, we built a performant Tetris controller, which in particular won the Tetris domain of the 2008 Reinforcement Learning Competition.

The note is organised as follows. The rest of this section deals with the Tetris game rules usually considered by the research works. Section 2 provides some insight into Dellacherie's program, the best Tetris controller known, which is a hand-written controller. In Section 3, we consider the recent work of Szita and Lőrincz (2006) about the cross-entropy method for Tetris and deepen their empirical analysis. Eventually, Section 4 describes how we combined the key features of Dellacherie and the method of Szita and Lőrincz in order to obtain a one-piece Tetris controller that outperforms, to our knowledge, the currently known Tetris controllers. In Section 5, we provide our conclusions and indicate directions for future work.

As most researchers, we focus on a simplified version of Tetris. First, we only consider *one-piece controllers*, i.e., controllers that use knowledge of the current board and the current piece only. A controller that also uses knowledge of the next piece, as it is possible in the original Tetris setting, can make decisions that take advantage of the combination of pieces, and is called a *two-piece controller*. It is relatively straightforward to extend works on one-piece controllers in order to build two-piece controllers: experimental data suggests that the performance

[1] thierych@loria.fr
[2] LORIA - INRIA Lorraine, Campus Scientifique BP 239, 54506 Vandœeuvre-lès-Nancy CEDEX, FRANCE. Email: scherrer@loria.fr
[3] This number is an estimate because it includes a few impossible configurations (see Fahey (2003)).

of a controller is improved by several orders of magnitude (Fahey, 2003). In this note, for simplicity, we only consider one-piece controllers.

In the original game (as specified by Fahey (2003)), the current piece appears in the playing area and falls gradually. The game is over when the piece has not enough space to appear in the top of the area. As in most works (Tsitsiklis and van Roy, 1996; Bertsekas and Tsitsiklis, 1996; Kakade, 2001; Lagoudakis, Parr, and Littman, 2002; Ramon and Driessens, 2004; Farias and van Roy, 2006; Szita and Lőrincz, 2006), we consider a simplified Tetris setting: the controller only decides in which column and orientation he drops the piece. Doing so, the game is slightly simplified since the piece does not appear in the playing area until the controller has decided where to put it. This is as if there were always space above the playing area to set the orientation and the column where one drops the current piece. It makes an important difference because the entire space of the board becomes available, including the top most rows. Moreover, we avoid situations where a piece cannot move from one side of the board to the other because the pile is too high. The simplified Tetris game is easier than the original game, and a controller is likely to complete more rows.

For a human player, a difficulty of the Tetris game is the fact that the pieces fall quite fast from the top of the playing area: the small amount of time often makes the decision process hard. This dimension of the problem does not appear when dealing with artificial players. The controllers we study in this paper are able to play 50,000 to 100,000 moves per second on a nowadays desktop computer. As a consequence, the falling speed of the pieces is negligible compared to the decision speed of the controllers. Again, as in most works (Tsitsiklis and van Roy, 1996; Bertsekas and Tsitsiklis, 1996; Kakade, 2001; Lagoudakis *et al.*, 2002; Ramon and Driessens, 2004; Farias and van Roy, 2006; Szita and Lőrincz, 2006), we will ignore the fall of the pieces and only focus on the real problem, that is, decide where and how one should drop each coming piece.

2. DELLACHERIE'S CONTROLLER

In this section, we provide a brief overview of Dellacherie's controller. As we will see in the rest of the note, we used his efficient set of features to improve the results of Szita and Lőrincz (2006) regarding the cross-entropy method. Dellacherie's ideas are still valuable for the current researchers.

To our knowledge, the current best one-piece Tetris controller is due to Dellacherie (Fahey, 2003) and has been tuned by hand. As in most works, his controller relies on an evaluation function that computes a weighted sum of *feature functions* to select the best move. The feature functions are some basis functions that try to capture the relevant aspects of a game configuration and they are designed by an expert. Thus, the problem of building a Tetris controller consists in finding some feature functions and set their weights. Dellacherie created an efficient set of feature functions, and contrary to reinforcement learning approaches and optimisation algorithms, which attempt to set the feature weights automatically, he fixed the weights manually by trial and error (see Table 1). Surprisingly, this hand-written controller outperforms those works, including the high performance of Szita and Lőrincz (2006) that we describe in Section 3. On a total of 56 games, Dellacherie's algorithm completed an average score of about $660,000$ lines. As the source code of the algorithm is freely available, we reverse-engineered it and determined the features and their weights. Dellacherie's evaluation function is the following linear combination of features:

$$-l + e - \Delta r - \Delta c - 4L - W$$

where the features $l, e, \Delta r, \Delta c, L$ and W are detailed in Table 1.

Moreover, note that this measure of $660,000$ lines per game was not made on the usual simplified Tetris setting but on an implementation of the original Tetris game which, as mentioned in Section 1, is harder. On our implementation of the simplified Tetris as considered by most researchers, Dellacherie's controller reaches the average score of $5,200,000 \pm 20\%$ rows par game[4].

[4] In the rest of the note, we use the notation $m \pm c\%$ to represent confidence intervals valid 95% of the time, assuming Tetris scores follow a geometric law (see Thiery and Scherrer (2009)).

Feature	Id	Description	Comments
Landing height	l	Height where the last piece is added	Prevents from increasing the pile height
Eroded piece cells	e	(Number of rows eliminated in the last move) $\times$ (Number of bricks eliminated from the last piece added)	Encourages to complete rows
Row transitions	Δr	Number of horizontal full to empty or empty to full transitions between the cells on the board	Makes the board homogeneous
Column transitions	Δc	Same thing for vertical transitions	
Holes	L	Number of empty cells covered by at least one full cell	Prevents from making holes
Board wells	W	$\sum_{w \in wells}(1+2+\cdots+depth(w))$	Prevents from making wells[a]

Table 1: Features proposed by Dellacherie.

[a] A well is a succession of unoccupied cells in a column such as their left cells and right cells are both occupied. Deep wells are punished because they force to wait for a vertical bar.

Feature	Id	Description	Comments
Column height	h_p	Height of the pth column of the board	There are P such features where P is the board width
Column difference	Δh_p	Absolute difference $\lvert h_p - h_{p+1} \rvert$ between adjacent columns	There are $P-1$ such features where P is the board width
Maximum height	H	Maximum pile height: $\max_p h_p$	Prevents from having a big pile
Holes	L	Number of empty cells covered by at least one full cell	Prevents from making holes

Table 2: Features introduced by Bertsekas and Tsitsiklis (1996).

3. THE SZITA AND LŐRINCZ APPROACH

Even with hand-chosen weights, Dellacherie's controller has so far given the best results (Thiery and Scherrer, 2009). Then, a natural choice is to try to tune Dellacherie's weights automatically and this is what we do with the cross-entropy method.

The cross-entropy method currently seems to be one of the most efficient algorithms for tuning the weights of a Tetris evaluation-based controller. Szita and Lőrincz (2006) showed that it outperforms the previous reinforcement learning works by several orders of magnitude. In this section, we describe the cross-entropy method (in Subsection 3.1) and then explain how Szita and Lőrincz applied it to Tetris (in Subsection 3.2). We reproduce their experiment and make slightly different observations about the noise parameter, which is crucial for practical efficiency.

3.1 The Cross-Entropy Method

We begin by presenting the principles of the cross-entropy method. The following overview is inspired by the description made by Szita and Lőrincz (2006). A more in-depth description of this general purpose optimization algorithm can be found in De Boer *et al*., (2004) and Chaslot *et al*. (2008). Cross-entropy is a general stochastic iterative algorithm that tries to solve an optimization problem of the form:

$$w^* = \arg\max_w S(w)$$

where S is a function we want to maximize and w is a parameter to optimize (typically a vector).

The principle of the cross-entropy method is to iterate on a distribution of solutions instead of a single solution. We consider a family of parameterized distributions $\mathcal{F}$ (for example the gaussian distributions) and we want to determine a probability distribution $f \in \mathcal{F}$ which generates solutions w close to the optimal one w^*. At each iteration t, we consider a current distribution $f_t \in \mathcal{F}$ that generates random solutions w and we want the next distribution $f_{t+1} \in \mathcal{F}$ to produce better solutions. To do this, we consider that a solution w is a good solution if its

value is greater than a certain threshold γ_t, i.e., if $S(w) > \gamma_t$. Let g_{γ_t} be the uniform probability distribution that generates solutions with values greater than γ_t. g_{γ_t} does not belong to $\mathcal{F}$ in general, so we search the distribution $f_{t+1} \in \mathcal{F}$ being as close as possible to g_{γ_t}, with respect to the *cross-entropy measure*[5] (de Boer *et al.*, 2004). For many kinds of distribution families $\mathcal{F}$, this distribution f_{t+1} can be estimated from examples generated by the current distribution f_t. For example, if we consider the case where $\mathcal{F}$ is the family of gaussian distributions, the gaussian distribution that is the closest to g_{γ_t} is the one characterized by the mean and the variance of the distribution g_{γ_t}. We can estimate these parameters if we draw samples from the distribution f_t and select the ones that are above the threshold γ_t, that are the best ones.

The cross-entropy method we will consider in this note is detailed in Algorithm 1, and a graphical illustration is shown in Figure 1. Overall, it consists of repeating the following steps.

- We generate N samples with the current gaussian distribution f_t.
- We evaluate each of these N sampled vectors with respect to S.
- We select a proportion $\rho \in [0, 1]$ of the best solutions (this is equivalent to setting γ_t at a certain threshold).
- We set the parameters of the gaussian distribution f_{t+1} to the empirical mean and variance of the selected best solutions.

Algorithm 1 Noisy cross-entropy method with a gaussian distribution

Inputs:
$evaluate()$: a function that estimates the function to optimize S for some vector w
(μ, σ): the mean and variance of the initial distribution
N: the number of vectors that are generated at each iteration
ρ: the fraction of vectors that are selected
Z_t: the noise that is added at each iteration

loop
  Generate N vectors $w_1, w_2, \ldots, w_N$ from $\mathcal{N}(\mu, \sigma^2)$
  Evaluate each vector using $evaluate()$
  Select the $\lfloor \rho \times N \rfloor$ with highest evaluations
  $\mu \leftarrow$ (mean of the selected vectors)
  $\sigma^2 \leftarrow$ (variance of the selected vectors) $+ Z_t$
end loop

In the detailed description of Algorithm 1, the function $evaluate()$ that is used to evaluate each of the vectors can be set to S, or it can be an estimate of S (if S takes too much time to compute exactly). Also, at each iteration, a *noise term* Z_t is added to the variance update. When Z_t is not zero, the algorithm is called noisy cross-entropy (de Boer *et al.*, 2004). In practice, one can see this noise term as a way to avoid too fast convergence to a bad local optimum.

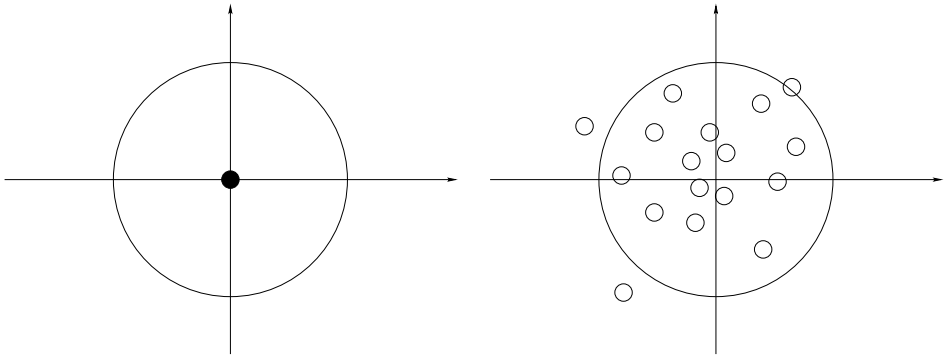
In spirit, cross-entropy is close to evolutionary algorithms. It is an iterative procedure that deals with a set of candidate solutions, or individuals. At each iteration, the best individuals are selected, and new solutions are generated from these ones. The main particularity of cross-entropy is the way the new individuals are generated.

3.2 Application to Tetris

Szita and Lőrincz (2006) applied the cross-entropy method with a gaussian distribution to the problem of Tetris. They consider a evaluation-based controller based on a set of features introduced by Bertsekas and Tsitsiklis (1996) since these features had already been used in several works (Bertsekas and Tsitsiklis, 1996; Kakade, 2001; Farias and van Roy, 2006). Such a controller bases its decisions on the following evaluation function:

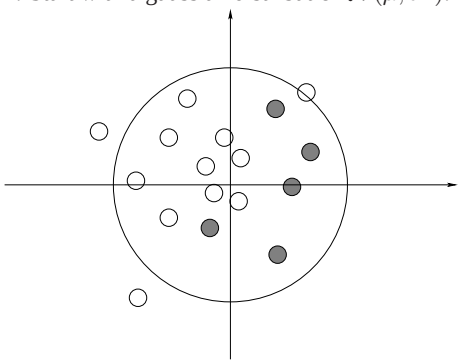
$$\sum_{i=1}^{10} w_i h_i + \sum_{i=1}^{9} w_{10+i} \Delta h_i + w_{20} H + w_{21} L,$$

[5] The cross-entropy measure (or *Kullback-Leibler distance*) defines a notion of distance between two probability distributions.

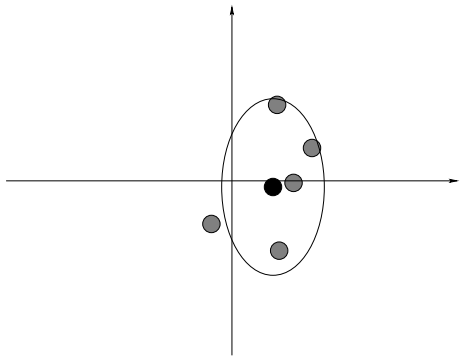


1. Start with a gaussian distribution $\mathcal{N}(\mu, \sigma^2)$.

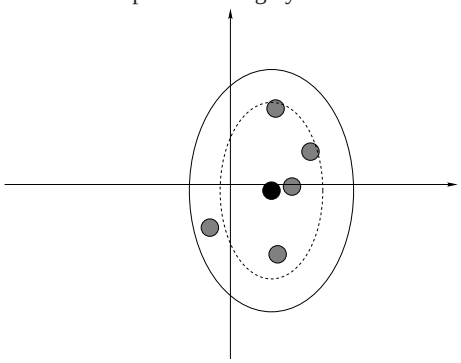
2. Generate N vectors with this distribution.



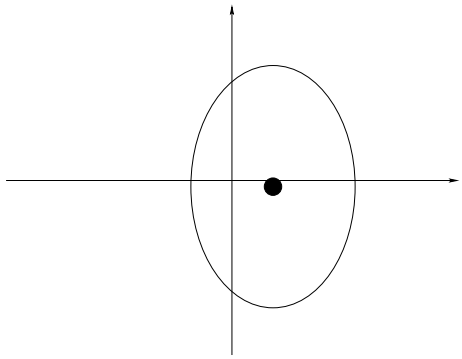
3. Evaluate each vector with $evaluate()$ and select a proportion ρ of the best vectors. These vectors are represented in gray.



4. Calculate the mean and the variance of the best vectors.



5. Add a noise term to the variance in order to avoid too fast convergence to a local optimum.



6. These mean and variance characterize a new gaussian distribution that will generate vectors for the next iteration.

Figure 1: A graphical representation of the noisy cross-entropy method for optimizing a two-dimensional vector. The gaussian distribution is represented with a black filled circle for the mean and an ellipse for the variance.

where the features $h_i, \Delta h_i, H$ and L are detailed in Table 2, and where $w = (w_1, \ldots, w_{21})$ is the vector of paramaters to tune. Naturally for this application, the function $w \mapsto S(w)$ to optimize is the expected mean score achieved by the corresponding controller. Szita and Lőrincz start with a gaussian centered at $\mu = (0, 0, \ldots, 0)$ with variance $\sigma^2 = (100, 100, \ldots, 100)$. At each iteration, they generate $N = 100$ vectors and evaluate each of them by playing one game. They select the 10 best vectors ($\rho = 10\%$) in order to generate a new gaussian distribution. After each iteration, they play 30 games with the mean weights of the new distribution in order to obtain a learning curve representing the evolution of performance.

In the experiment of Szita and Lőrincz, the $evaluate()$ function is the score of a *single* game. As stated in Thiery and Scherrer (2009), Tetris scores have a large deviation, so it is clear that this evaluation is not accurate. With our implementation, we tried to evaluate each vector by playing more games to see whether this was a crucial choice, and we concluded that it was not. Although we observed that the number of iterations needed to reach the maximal performance level is lower (which is natural since the selection process is more accurate), we also noticed that doing so did not lead to better controllers eventually.

Szita and Lőrincz ran the cross-entropy method in three conditions: without any noise ($Z_t = 0$), with a constant noise ($Z_t = 4$), and with a linearly decreasing noise ($Z_t = \max(5 - t/10, 0)$). The resulting experimental data suggest that the performances were significantly improved when using noise. Their best controller was obtained with the decreasing noise, reaching an average score of 350,000 $\pm 37\%$ rows. As the noise parameter seemed to be crucial for the performance, we conducted some extra experiments, which we discuss now.

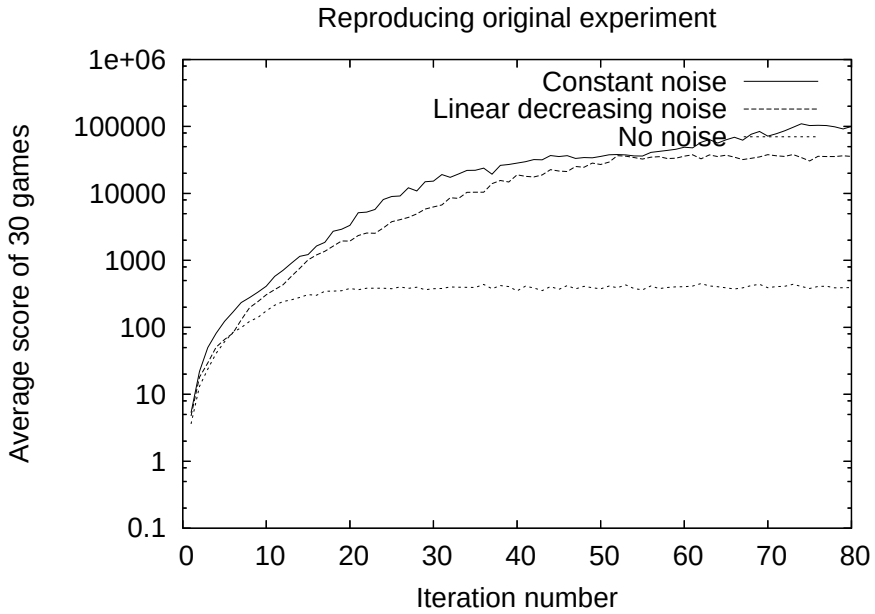


Figure 2: Our implementation of Szita and Lőrincz's experiment. Each curve represents the average learning curve of 10 executions with a given kind of noise (note the log scale). We observe that adding noise improves significantly the performances. The best average learning curve is obtained with the constant noise.

Szita and Lőrincz executed each of the three experiments (no noise, constant noise, and decreasing noise) only once because it took a very long time. They report a total execution time of one month. We took a great care in implementing our Tetris simulator, particularly in terms of optimization, so that we could reproduce their experiments several times. Indeed, our preliminary tests showed that several executions of the cross-entropy method with the same parameters could give very different results. We thus decided to run each of the three experiments by Szita and Lőrincz 10 times. With our Tetris implementation, this took about a week.

The results we obtained are shown in Figures 2 and 3. Figure 2 shows for each kind of noise the average learning curve of the 10 runs. Our experimental results confirm the observation by Szita and Lőrincz that adding noise significantly improves the performance. However, we observe that the average performance is the best for constant noise. Figure 3 shows, for each kind of noise, the detail of the 10 executions. The best performance is reached with linearly decreasing noise: one of the 10 executions obtains a controller which achieves an average

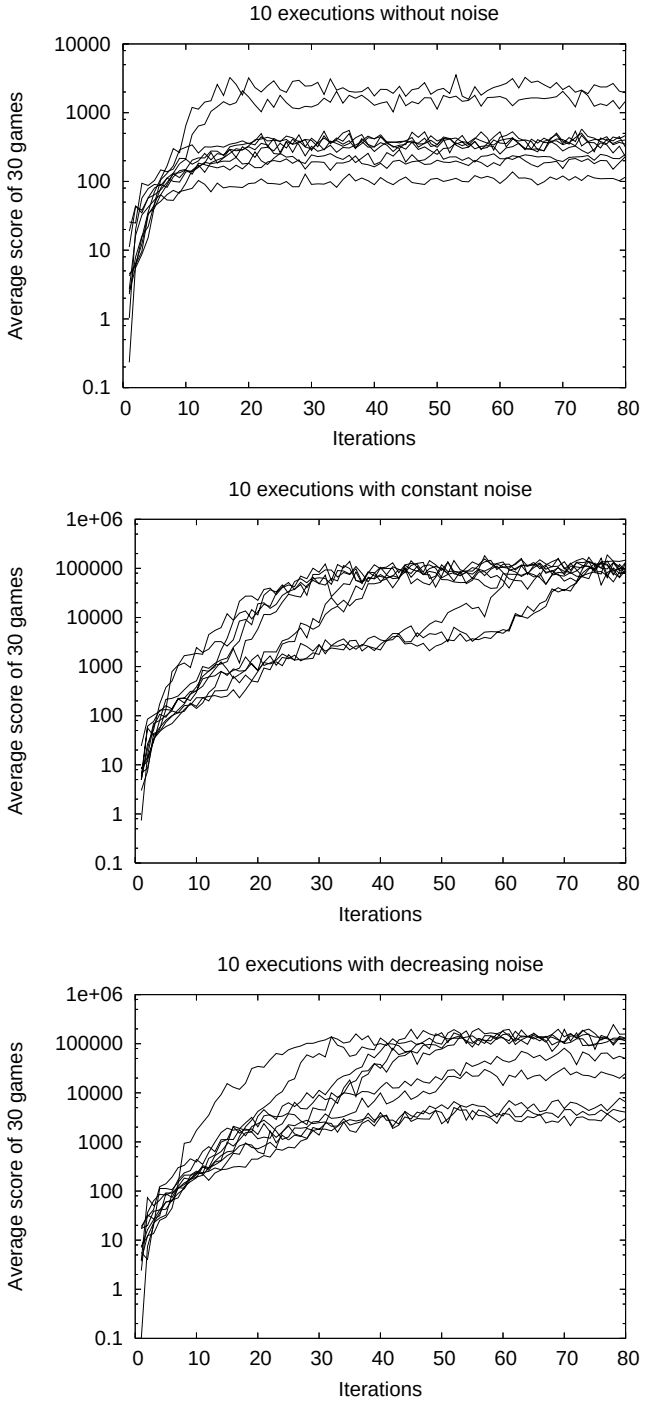


Figure 3: The 10 executions of each experiment from Figure 2 (note the log scale). **Without noise (top):** the learning curve stabilizes after iteration 20. The average score reached changes with the executions (100 to 3,000 rows completed). **Constant noise (middle):** the 10 executions reach equivalent performances eventually, between 100,000 and 200,000 rows. **Linearly decreasing noise (bottom):** the 10 executions reach very different values, from 5,000 to 250,000 rows.

score of $240,000 \pm 37\%$ rows. With the confidence interval, the results we obtained are consistent with the original result by Szita and Lőrincz ($350,000 \pm 37\%$). Examining Figure 3 gives more insight into the choice of noise: all of the 10 executions with constant noise reach similar performances eventually ($100,000$ to $200,000 \pm 37\%$ rows) while with the decreasing noise, the performances seem to vary considerably between several executions of the cross-entropy method. This suggests that if one only runs once the cross-entropy method (this was the case in the original work by Szita and Lőrincz and this will be the case in the next section because we build controllers that play very long games), the constant noise is more reliable.

4. TOWARDS AN EFFICIENT TETRIS CONTROLLER

We have seen that cross entropy is an efficient method for optimizing the relative weights of a set of features. As we saw when we mentioned Dellacherie's performance, it is also essential to choose a relevant set of features in order to capture the important aspects of the Tetris game (see Thiery and Scherrer, 2009). Thus, a natural approach, which we follow in this section, is to consider more elaborate Tetris features than the ones of Bertsekas and Tsitsiklis (Table 2).

We tried several combinations of features, including Dellacherie's (Table 1) since they are part of the best Tetris controller. We also introduced two original features: the *hole depth* and the *number of rows with holes*. The hole depth indicates how far holes are under the surface of the pile: it is the sum of the number of full cells above each hole. This feature aims at removing the holes faster, by punishing for putting a piece above a hole. Our second original feature counts the number of rows having at least one hole (two holes on the same row count for only one).

We executed the noisy cross-entropy method with a 10×20 board under the same conditions as Szita and Lőrincz (2006). We started with a gaussian centred at $\mu = (0, 0, \ldots, 0)$ with variance $\sigma^2 = (100, 100, \ldots, 100)$; we generated $N = 100$ vectors at each iteration, and we selected the 10 best ($\rho = 10\%$). In accordance with the discussion in Section 3, each vector was evaluated by playing only one game, and we used the constant noise ($Z_t = 4$). We launched the algorithm with 4 different feature sets: Dellacherie (D), Bertsekas + Dellacherie (BD), Dellacherie + Us (DU), and Bertsekas + Dellacherie + Us (BDU). As we expected, the performances achieved are much better than with Bertsekas and Tsitsiklis' features only. As the games are much longer, we could only make one execution for each feature set: even though our implementation is optimized and each of the 4 experiments was launched on a different machine, these experiments took about a month. Figure 4 provides the learning curves for the 4 feature sets. As in the work of Szita and Lőrincz, the curves represent the average score of 30 games played with the mean controller generated after each iteration. The first observation that we can make is that our original features have a significant impact on the scores: the corresponding curves (two dashed lines in Figure 4) are the ones that go the highest. We also observe that when we remove the features of Bertsekas and Tsitsiklis (the experiments without these features correspond to the two thick lines), the algorithm converges in fewer iterations (this is not surprising since there are fewer parameters to tune) and reaches similar best scores: this suggests that once we have Dellacherie's features, Bertsekas and Tsitsiklis' features are not informative any more.

Note that the curves of Figure 4 represent the average score of only 30 games, so that the corresponding confidence interval is quite large ($\pm 37\%$). To evaluate the controllers more precisely, we selected a few controllers for each feature set (we picked a few weight vectors corresponding to high spikes in the curves of Figure 4) and made them play more games. Table 3 reports, for the best controller of each feature set, the average score of 100 games in a 10×20 board. This corresponds to a confidence interval of $\pm 20\%$. We also made the same controllers play on a 10×16 board to obtain a lower bound on the score they would achieve on the original Tetris game[6]. On this reduced board, we played 1600 games, which leads to a confidence interval of 5%.

The use of the cross-entropy method for tuning the weight of Dellacherie's controller is relevant: with respect to the original hand-chosen parameters, the automatically tuned parameters improve the performance by about a factor 3. The best scores are reached with the BDU and DU features, which realize similar performances: $36,000,000 \pm 20\%$ or $35,000,000 \pm 20\%$ rows on the 10×20 board, and $910,000 \pm 5\%$ rows on the 10×16 board. The two evaluation functions we obtained are thus better than the previous best playing algorithm, Dellacherie's,

[6] Recall that we are playing on the usual simplified Tetris setting considered by most researchers (see Section 1). Playing on a 10×16 board with the simplified setting gives a lower bound on the score we would reach on a 10×20 board of the original Tetris game, because Tetris shapes do not exceed a height of 4 blocks.

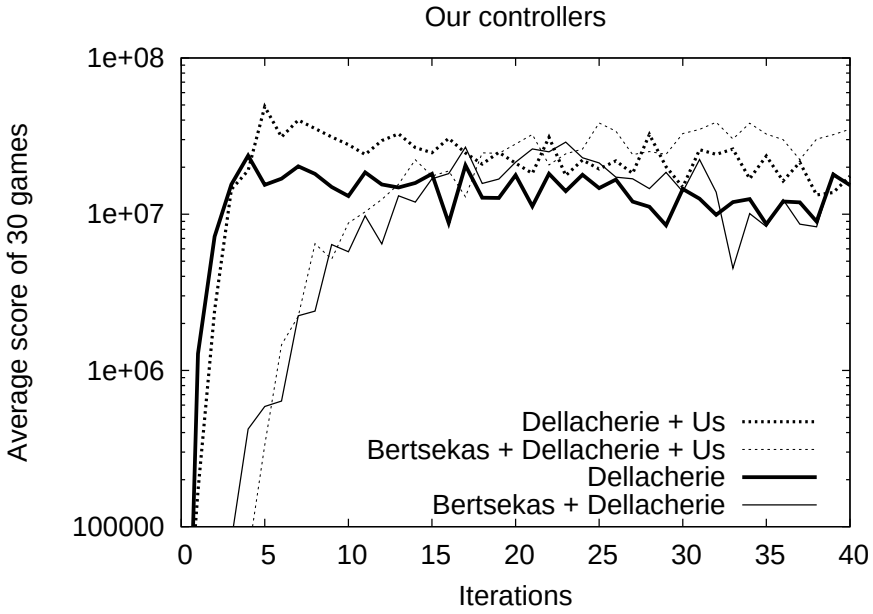


Figure 4: Evolution of the mean score of 30 games with the noisy cross-entropy method with 4 feature sets: Dellacherie, Dellacherie + Bertsekas, Dellacherie + Us, Dellacherie + Bertsekas + Us. Note the log scale. When we add our original features (the two dashed lines), the performances are significantly increased. When the features from Bertsekas are not present (this corresponds to the two thick lines), the algorithm converges much faster and the (best) performances reached are close.

Features	**DU**	**BDU**	**D**	**BD**
10×20 board	$35,000,000$	$36,000,000$	$17,000,000$	$20,000,000$
10×16 board	$910,000$	$910,000$	$530,000$	$660,000$

Table 3: Average score of the best controller obtained with the noisy cross-entropy method for each feature set. 100 games were played on a 10×20 board (the confidence interval is $\pm 20\%$) and 1600 games were played on a 10×16 board (the confidence interval is $\pm 5\%$). The features are represented by their first letter: D for Dellacherie, B for Bertsekas, U for Us. The best performances are obtained with the DU and BDU features.

Feature	**Symbol**	**Weight**
Landing height	l	-12.63
Eroded Piece Cells	e	6.60
Row transitions	Δr	-9.22
Column transitions	Δc	-19.77
Holes	L	-13.08
Board wells	W	-10.49
Hole depth	D	-1.61
Rows with holes	R	-24.04

Table 4: The weights of our DU controller (Dellacherie + Us). Average score: $35,000,000 \pm 20\%$ rows. See Section 4 and Table 1 for the feature definitions.

which makes an average score of $5,200,000 \pm 20\%$ rows on the 10×20 board. While the BDU controller contains 28 features, the DU controller only contains 8 features; it is simpler and faster, and thus to be considered as better. For completeness, we give its weights in Table 4.

5. CONCLUSION AND FUTURE WORK

We have revisited the application of the cross-entropy method to the Tetris game as proposed by Szita and Lőrincz. By reproducing 10 times their original experiment, we have deepened their experimental analysis: in particular, we observed that the constant noise seems more reliable than the linear decreasing noise. Using the expert knowledge (the feature functions) designed by Dellacherie (Fahey, 2003) and two original features, we built a Tetris controller that outperforms the previous works.

We used this approach when we took part with our program BCTS in the 2008 Tetris domain Reinforcement Learning Competition. Our team ("LORIA INRIA - Maia") won the competition[7] with BCTS (Building controllers for Tetris Systems).

Further improving the performance of our controller may be achieved through the design of more complex or more expressive evaluation functions. A first natural direction could be to exploit other feature functions of the literature (for instance those of Xtris (Llima, 2005) or Fahey (2003)) or to design new ones. A particularly interesting research direction is to consider the problem of automatically selecting and combining "basic features" in order to build "high level" efficient features. For instance, one could make such possible combinations part of the search space, as in the recent genetic programming approach of Girgin and Preux (2007).

In general, an important issue regarding the game of Tetris is the long running time for playing a game (which is part of the inner loop of the algorithms that tunes the weights). This is more important when the performance improves, as the games last longer. We see three ways to deal with this problem by reducing the duration of a game or the number of games played. The first two points may be generalized to other games that Tetris.

- A first idea would be to launch the algorithm on a reduced-size board in order to learn the weights. Doing so, the games are shorter, we can generate more vectors, evaluate them more carefully and the iterations can be faster. However, it is not clear whether a controller built on a small board will perform well on the standard board (10×20). We have made a few preliminary experiments, and the performances of controllers built by playing on smaller boards (like 10×16) seem to be slightly lower than the scores achieved by controllers directly built on large boards (10×20).
- We have seen that to evaluate how good a controller is, it is better to play many games. Instead of playing random games, we could play a small set of predetermined games, with some sequences of pieces previously generated. Thus, with this method, we would use the same sequences of pieces to compare different controllers. In the selection phase of the cross-entropy method, this might select the best samples in a more reliable way, because we would use the same training set. This training set would have to be representative of all possible games, so that a controller tuned this way can be efficient on tests sets.
- To reduce the time needed to evaluate a controller, a promising idea comes from a conjecture by Fahey (2003). He claims that the length of a Tetris game (and, consequently, the score) can be estimated from the very first moves. Indeed, consider for each height h (i.e. $h = 0$ to 20) the *frequency* of h, that is the proportion of time when the pile height was exactly h, during the n first moves. With a good controller, when h is big, the frequency of h is low since high piles are less likely to occur. Fahey observed experimentally that with his controller this decrease is geometric, and we made experiments that confirm his observation for other evaluation-based controllers. Consequently, it should be possible to estimate the parameters of this geometric distribution (they differ for each controller) by computing a regression, and deduce the frequency for $h = 21$, which is related to the average length of the game. For a given controller, we could thus estimate the mean duration of a game only by playing n moves instead of playing one or several games. However, our preliminary experiments suggest that such a method has a large variance, even if we play a large number of moves (such as $n = 1,000,000$ moves).

Further investigations remain to be done in these directions in order to reduce the execution time of the algorithm.

This paper confirms the efficiency of the cross-entropy method for Tetris. Then, a natural question is to determine in what circumstances the cross-entropy method can perform better than other optimization algorithms, for the Tetris game or other problems as well. We are currently considering another optimization approach, the CMA-ES

[7] See `http://2008.rl-competition.org/content/view/51/79/` for a description of the competition and `http://2008.rl-competition.org/content/view/52/80/` for the results.

algorithm (Covariance Matrix Adaptation Evolution Strategy) (Hansen and Ostermeier, 2001), which can be seen as an alternative to the cross-entropy method. The main difference is that, with CMA-ES, the gaussian distribution which generates weight vectors uses a covariance matrix. The gaussian distribution used in the cross-entropy method corresponds to the case of a diagonal covariance matrix: on Figure 1, the axes of the ellipses represented are always colinear to an axis of the coordinates system. With the CMA-ES algorithm, the ellipse axes may be in any direction, so the search space of the gaussian distribution is more expressive and the search process may be more efficient. The interest of using CMA-ES instead of cross-entropy is currently being investigated.

6. REFERENCES

Bertsekas, D. and Tsitsiklis, J. (1996). *Neurodynamic Programming*. Athena Scientific.

Boer, P. de, Kroese, D., Mannor, S., and Rubinstein, R. (2004). A tutorial on the cross-entropy method. *Annals of Operations Research*, Vol. 1, No. 134, pp. 19–67.

Chaslot, G., Winands, M., Szita, I., and Herik, H. J. van den (2008). Cross-Entropy for Monte-Carlo Tree Search. *ICGA Journal*, Vol. 31, No. 3, pp. 145–157.

Demaine, E. D., Hohenberger, S., and Liben-Nowell, D. (2003). Tetris is hard, even to approximate. *Proc. 9th International Computing and Combinatorics Conference (COCOON 2003)*, pp. 351–363.

Fahey, C. P. (2003). Tetris AI, Computer plays Tetris. `http://colinfahey.com/tetris/tetris.html`.

Farias, V. and Roy, B. van (2006). *Tetris: A study of randomized constraint sampling*. Springer-Verlag, Heidelberg, Germany.

Girgin, S. and Preux, P. (2007). Feature Discovery in Reinforcement Learning using Genetic Programming. Technical Report RR-6358, INRIA. http://hal.inria.fr/inria-00187997/fr/.

Hansen, N. and Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, Vol. 9, No. 2, pp. 159–195.

Kakade, S. (2001). A natural policy gradient. *Advances in Neural Information Processing Systems (NIPS 14)*, pp. 1531–1538.

Lagoudakis, M. G., Parr, R., and Littman, M. L. (2002). Least-squares methods in reinforcement learning for control. *SETN '02: Proceedings of the Second Hellenic Conference on AI*, pp. 249–260, Springer-Verlag, London, UK.

Llima, R. E. (2005). Xtris readme. http://www.iagora.com/~espel/xtris/README.

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

Szita, I. and Lőrincz, A. (2006). Learning Tetris Using the Noisy Cross-Entropy Method. *Neural Computation*, Vol. 18, No. 12, pp. 2936–2941.

Thiery, C. and Scherrer, B. (2009). Building Controllers for Tetris. *ICGA Journal*, Vol. 32, No. 1, pp. 3–11.

Tsitsiklis, J. N. and Roy, B. van (1996). Feature-Based Methods for Large Scale Dynamic Programming. *Machine Learning*, Vol. 22, pp. 59–94.