Reducing the Learning Time of Tetris in Evolution Strategies

Amine Boumaza

Univ. Lille Nord de France, F-59000 Lille, France,
ULCO, LISIC, F-62100 Calais, France
boumaza@lisic.univ-littoral.fr

Abstract. Designing artificial players for the game of Tetris is a challenging problem that many authors addressed using different methods. Very performing implementations using evolution strategies have also been proposed. However one drawback of using evolution strategies for this problem can be the cost of evaluations due to the stochastic nature of the fitness function. This paper describes the use of racing algorithms to reduce the amount of evaluations of the fitness function in order to reduce the learning time. Different experiments illustrate the benefits and the limitation of racing in evolution strategies for this problem. Among the benefits is designing artificial players at the level of the top ranked players at a third of the cost.

1 Introduction

Designing artificial game players has been, and still is, studied extensively in the artificial intelligence community. This is due to games being generally interesting problems that provide several challenges (difficult or large search spaces, important branching factors, randomness etc.) Many conferences organize special tracks every year where game playing algorithms are put in competition.

Among these games, Tetris [8] is a single player game where the goal is to place randomly falling pieces onto a 10×20 game board. Each completed horizontal line is cleared from the board and scores points to the player and the goal is to clear as much lines as possible before the board is filled.

Among the challenges in learning Tetris is the prohibitive size of the search space that approaches 10^{60}, and which can only be tackled using approximations. Interestingly enough one cannot play Tetris for ever, the game ends with probability one [7]. Furthermore, it has been shown that the problem of finding strategies to maximize the score in Tetris is NP-Complete [6].

Many authors proposed algorithms to design artificial Tetris player (see below), however to this day evolutionary algorithms outperform all other methods by far. And among these the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [10] and the noisy cross entropy[3] hold the record (several million lines on average).

In the present paper we will describe the use of racing procedures in the case of CMA-ES on the Tetris learning problem in order to reduce the learning time which as it will be clear shortly, can be very problematic. We will begin with a description of the problem of learning Tetris strategies and review some of the existing work. We present the algorithm with the racing methods used, and then present some experiments. Finally we will discuss the results and give some conclusions.

1.1 Learning Tetris Strategies

In the literature, artificial Tetris players use evaluation functions that evaluate the game state by assigning numerical values. The agent will decide which action to take based on the values of this function.

When a piece is to be placed, the agent simulates and evaluates all the possible boards that would result from all the possible moves of the piece, and then chooses the move that leads to the best valued game board. Note that since the agent follows a greedy strategy, the evaluation function is the only component that enters into the decision making process. Hence, designing a Tetris player amounts to designing an evaluation function. Such a function should synthesize the game state giving high values to "good boards" and low values to "bad" ones. The evaluation function for the game of Tetris may be considered as mimicking the evaluation a human player makes when deciding where to drop the piece and which orientation to give to it. It should rank higher game states that increase the chance to clear lines in subsequent moves. For example, such boards should be low in height and should contain as less holes as possible. These properties (the height, the number of holes) are called *features* and an evaluation function is a *weighted linear combination of these features.*

Let f_i denotes a feature function, f_i maps a state (a board configuration) to a real value. The value of a state s is then defined as:

$$V(s) = \sum_{i=1}^{N} w_i f_i(s) \tag{1}$$

where N is the number of features and w_i are weights which, without loss of generality can sum to one.

Let P be the set of all pieces and D the set of all possible decisions. We note $d_i(p) \in D$ the i^{th} decision applied on piece $p \in P$. A decision is a rotation (4 possibilities) and a translation (10 possibilities[1]) for the current piece. Furthermore, we note γ the function that maps a pair $(s, d_i(p))$, a state s and a decision $d_i(p)$, to a new state of the game after taking decision d_i for the piece p. Given all possible decisions $d_i(p) \in D$ for a current piece p in state s, the player will choose the highest valued one:

$$\hat{d}(s,p) = \arg \max_{d_i(p) \in D} V(\gamma(s, d_i(p))).$$

The function $\hat{d}(s,p)$ is the player's decision function. As stated above the only component that enter into the decision process is the value V, in other words

[1] The number of translations may be different depending on the game width.

the behavior of the player is conditioned by the value of the feature functions $f_{1...N}$. For a comprehensive review of the existing features the reader can see [19]. Learning tetris strategies amounts at (1) choosing a set of feature functions and (2) fixing their weights in eq. 1. In the present work we choose the set of eight features used in [19] and [5].

There are many papers that address the problem of learning Tetris strategies using different methods ranging from reinforcement learning to evolutionary computation. Dellacherie [8] fixed the weights by hand proposing good performing players. Different authors have used reinforcement learning some of which are : feature based value iteration [21], λ-policy iteration [2] and LS-λ-policy iteration [20], and approximate dynamic programming [9].

Other authors considered the problem differently and proposed to fix the feature weights using optimization techniques. The noisy cross entropy has been used by [18,19]. Evolutionary algorithms have also been used with some success. For instance [16] proposed to use genetic programming. [4] also used an evolutionary algorithm and different combinations of value function. Finally [5] proposed to use CMA-ES to optimize the weights.

At this stage the best performing player are the one proposed by [19] and [5]. They both score on average around 35 millions lines.

In all these studies, authors agree on the challenges that the problem of learning Tetris strategies introduces. First of all, the score distribution of a given strategy on a series of games follows a long tailed distribution[2] which requires to evaluate (compute the fitness) the performance of the players on a series of games rather that just one. Secondly, as the artificial player learns during the run of an algorithm and thus get better, the games it plays last longer and lengthen the evaluation time. The number of games that is required for an accurate assessment of the performance is usually a parameter of the algorithm. It is fixed empirically to obtain an enough sample to compute statistics of the performance and in the same time keep the evaluation time reasonable[3].

Different methods have been proposed to reduce the learning time in Tetris. On the one hand one can train a player on small instances of the game, in this case the possibilities to place the pieces are reduced and the player looses rapidly. On the other hand, one can increase the frequency of certain (hard to place) pieces. Here again the player looses more quickly since these pieces will disturb the games board. All these methods do not take into account the number of games played in order to estimate the score of the player which is usually a fixed number. In what follows we will describe the use of racing procedures to dynamically adapt the number of required games to evaluate the players.

1.2 Racing for Stochastic Optimization

On stochastic objective functions, the fitness values of each individual follow a distribution $\mathcal{D}$ usually unknown. We have thus $f(x) \sim \mathcal{D}_x\left(\mu_x, \sigma_x^2\right)$, where μ_x, σ_x^2

[2] Some authors argue that this distribution is exponential [8].

[3] Many authors reported weeks length and some times months of learning time.

are respectively the mean and the variance of the $\mathcal{D}_x$. On such problems, one cannot rely on a single evaluation of the offspring which is usually not a reliable measure, but needs to make several ones and compute statistics to estimate the true fitness.

In algorithms that rely only on the ranking of the population as it is the case for CMA-ES [10], ranking based on fitness values is not appropriate if the variability of the values is not taken into account. In such cases the ranking may not be correct and may lead to badly selected parents.

Different methods were proposed to overcome this problem. For example [11] proposes UH-CMA-ES which, among other things, increases the population variance when a change in the ranking of the population is detected. [17,15] proposed methods to reduce the number evaluations applicable for certain types distribution $\mathcal{D}$. Another way to overcome this problem is to take into account confidence bounds around the estimated fitness value using racing methods [13,1]. These methods adapts the number of evaluations dynamically until the algorithm reaches a reliable ranking in the population. [12] proposed to use such methods with CMA-ES and reported some success on different machine learning problems.

Basically these methods reevaluate only individuals where confidence intervals (CI) overlap. Reevaluation in this case is used to reduce the CI. Once enough individuals with non overlapping CI are found, the procedure stops. When the distribution $\mathcal{D}$ is unknown, the confidence intervals are computed using empirical methods using for example Hoeffding or Bernstein bounds which will be described below.

2 Learning Tetris with CMA-ES and Racing

We will begin this section by describing the CMA-ES algorithm and the racing method after which we describe their use in learning Tetris strategies. The covariance matrix adaptations evolution strategy [10] is an evolutionary algorithms that operates in continuous search spaces where the objective function f can be formulated as follows:

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} f(\mathbf{x}) \text{ with } f : \mathbb{R}^n \rightarrow \mathbb{R}$$

It can be seen as continuously updating a search point $m \in \mathbb{R}^n$ that is the centroid of a normally distributed population. The progress of the algorithm controls how the search distribution is updated to help the convergence to the optimum.

Alg. 1 describes the general procedure in CMA-ES. At each iteration, the algorithm samples λ points (the offspring) from the current distribution (line 5) : $\mathbf{x}_i^{t+1} \sim \mathbf{m}^t + \sigma^t \mathbf{z}_i(t)$ and $\mathbf{z}_i^t \sim \mathcal{N}(0, \mathbf{C}^t)$ $(i = 1 \cdots \lambda)$, where $\mathbf{C}^t$ is the covariance matrix and $\sigma > 0$ is a global step-size. The $\mu \leq \lambda$ best of the sampled points (the parents) are recombined by a weighted average (line 8) where $w_1 \geq \cdots \geq w_\mu > 0$ and $\sum_{i=0}^{\mu} w_i = 1$ to produce the new centroid $\mathbf{m}^{t+1}$.

Algorithm 1. Procedure $(\mu/\mu, \lambda)$-CMA-ES

```
Input (λ, m, σ, f, n)
1: $\mu := \lfloor \lambda/2 \rfloor$, $\mathbf{m}^0 := \mathbf{m}$, $\sigma^0 := \sigma$, $\mathbf{C}^0 := \mathbf{I}$, $t := 1$, $\mathbf{p}_c^0 = \mathbf{p}_\sigma^0 = 0$
2: repeat
3:     for $i := 1$ to $\lambda$ do
4:         $\mathbf{z}_i^t \sim \mathcal{N}_i\left(0, \mathbf{C}^t\right)$        /* Sample and evaluate offspring */
5:         $\mathbf{x}_i := \mathbf{m}^t + \sigma^t \times \mathbf{z}_i^t$
6:     end for
7:     race($\mathbf{x}_1 \ldots \mathbf{x}_\lambda$)        /* Race the offspring */
8:     $\mathbf{m}^{t+1} := \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}$ /* Recombine $\mu$ best offspring, $f(\mathbf{x}_{1:\lambda}) \leq \ldots \leq f(\mathbf{x}_{\mu:\lambda})$ */
9:     $\mathbf{p}_\sigma^{t+1} := (1 - c_\sigma)\, \mathbf{p}_\sigma^t + \sqrt{c_\sigma (2 - c_\sigma)\, \mu_{eff}}\, \left(\mathbf{C}^t\right)^{\frac{1}{2}} \frac{\mathbf{m}^{t+1} - \mathbf{m}^t}{\sigma^t}$        /* Update $\sigma$ */
10:    $\sigma^{t+1} := \sigma^t \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma^{t+1}\|}{\mathbb{E}[\|\mathcal{N}(0,\mathbf{I})\|]} - 1\right)\right)$
11:    $\mathbf{p}_c^{t+1} := (1 - c_c)\mathbf{p}_c^t + \sqrt{c_c (2 - c_c)\, \mu_{co}} \frac{\mathbf{m}^{t+1} - \mathbf{m}^t}{\sigma^t}$        /* Update $\mathbf{C}$ */
12:    $\mathbf{C}^{t+1} := (1 - c_{co})\, \mathbf{C}^t + \frac{c_{co}}{\mu_{co}} \mathbf{p}_c^{t+1} \left(\mathbf{p}_c^{t+1}\right)^T + c_{co}\left(1 - \frac{1}{\mu_{co}}\right) \sum_{i=1}^{\mu} w_i \left(\mathbf{z}_{i:\lambda}^t\right) \left(\mathbf{z}_{i:\lambda}^t\right)^T$
13:    $t := t + 1$
14: until stopping_criterion
15: return $\mathbf{m}^t$
```

Adaptation of the search distribution in CMA-ES takes place in two steps, first updating the mutation step and then updating the covariance matrix. The step-size adaptation is performed using cumulative step-size adaptation [14], where the evolution path accumulates an exponentially fading path[4] of the mean m^t (line 9) where the backward time horizon is determined by c_σ^{-1}. The adaptation of the covariance matrix C^t takes into account the change of the mean (rank-1 update), and the successful variations in the last generation (rank-μ update) (respectively lines 11 and 12). An in depth discussion on setting the parameters c_σ, d_σ, c_c, μ_{eff}, μ_{co}, c_{co} and their default values can be found in [11].

2.1 The Racing Procedure

The racing procedure we used is inspired from [12]. At each iteration of alg. 1, the λ offspring undergo multiple evaluations until either : 1) μ outstanding individuals get out of the race, in which case we are sure (with probability $(1 - \delta)$) that they are distinct. Or 2) the number of evaluations reaches some upper limit r_{limit}. Furthermore, if the actual number of evaluations necessary to distinguish μ individuals is less than the limit r_{limit}, this one is reduced:

$$r_{limit} := \max(1/\alpha\, r_{limit}, 3) \tag{2}$$

with $\alpha > 1$, and 3 is the minimum number of evaluations. On the other hand if r_{limit} evaluations where not enough to distinguish μ individuals, then r_{limit} is increased:

$$r_{limit} := \min(\alpha\, r_{limit}, r_{max}) \tag{3}$$

[4] The comparison of the travelled path with a path under random selection is used to adjust the step-size.

where r_{max} is a maximum number of evaluations. Initially $r_{limit} = 3$ and it is adapted at each iteration of alg. 1 using the above two rules.

After r re-evaluations, the empirical bounds $c_{i,r}$ around the mean fitness of each individual $\hat{X}_{i,r} = \frac{1}{r}\sum_{j=1}^{r} f^j(x_i)$ where $i = 1 \dots \lambda$, are computed with:

$$c_{i,r}^h = R\sqrt{\frac{\log(2n_b) - \log(\delta)}{2r}} \tag{4}$$

using the Hoeffding bound and

$$c_{i,r}^b = \hat{\sigma}_{i,r}\sqrt{2\frac{\log(3n_b) - \log(\delta)}{r}} + 3R\frac{\log(3n_b) - \log(\delta)}{r} \tag{5}$$

using Bernstein. Where $n_b \leq \lambda r_{limit}$ is the number of evaluations in the current race, $\hat{\sigma}_{i,r}^2 = \frac{1}{r}\sum_{j=1}^{r}\left(f^j(x_i) - \hat{X}_{i,r}\right)^2$ is the standard deviation of the fitness for individual x_i, and $(1-\delta)$ is the level of confidence we fix. $R = |a-b|$ such that the fitness values of the offspring are almost surely between a and b. These two constant are problem dependent.

After each evaluation in the race the lower bounds $lb_{i,r}$ and the upper bounds $ub_{i,r}$ around the mean of each offspring are updated to the tightest values: $lb_{i,r} = max\left(lb_{i,r-1}, \hat{X}_{i,r} - c_{i,r}^{h/b}\right)$ and $ub_{i,r} = min\left(lb_{i,r-1}, \hat{X}_{i,r} + c_{i,r}^{h/b}\right)$

Beside the above empirical bounds, there exists for the game of Tetris estimated bounds on the mean score of the player [19] which we also used in the racing procedure. We have:

$$|X - \hat{X}|/\hat{X} \leq 2/\sqrt{n} \tag{6}$$

with probability 0.95, where $\hat{X}$ is the average score of the player on n games and X is the expected score. In the remainder we will refer to this bound as the Tetris bound.

3 Experiments

In the following, we present few experiments we conducted in different settings to compare the effect of the racing procedures described above. In all experiments the initial search point x_0 was drawn randomly with $\|x_0\| = 1$ and the initial step-size $\sigma_0 = 0.5$. We let the algorithm run for 25 iterations (improvements for larger values were not noticeable), we set $r_{max} = 100$ and $\delta = 0.05$. There are eight feature functions in the evaluation function therefore the dimension of the search space is $n = 8$. CMA-ES parameters c_σ, d_σ, c_c, μ_{eff}, μ_{co} and c_{co} were set to their default values as presented in [11].

In order to reduce the learning time, we adopt the same scheme as in [5] and evaluate the offspring on harder games, in which “S” and “Z” appear four times more frequently than in the standard game. Experiments showed that learning on this setting does not impair the performance of the player on the standard game.

The algorithm maximizes the score of the game i.e. the fitness function is the average of lines scored by a search point on a number of games. When racing is applied, this number of games can go from 3 to r_{max}. When it is not applied it is r_{max}. The initial bounds a and b used to compute R (eq. 5) were fixed experimentally to $a = 150$ and $b = 2000$.

The games were simulated on the MDPTeris platform[5]. All the expriments were repeated 50 times and the curves represent median values.

An important issue (incidentally ignored by many authors) about learning Tetris players and in general about learning the weights of evaluation functions for games, is that the weight should be normalized. This fact is described in [5] where empirical evidence show how in CMA-ES the step-size and the principal axes of the covariance matrix diverge. In these experiments we follow the same steps and normalize the offspring after the sampling step.

Furthermore, in order to provide a coherent ranking of the offspring, we evaluate them on the same series of games. At each iteration, a series of r_{max} independent games (random seeds) are drawn randomly and each individual of the λ offspring plays the same games. This way we can assess of the offspring's performance on the same setting.

3.1 Results and Discussions

The effect of racing is clear on the learning time of the algorithm. The number of evaluations is indeed reduced without a loss of performance. Fig. 1 on the right shows the fitness function of the mean vector (the centroid of the distribution) for the different racing procedures and without racing. All algorithms reach the same performance at different evaluations costs. Hoeffding bounds perform the best reaching the performance for the smallest evaluation time, followed by Tetris bounds and Bernstein. Fig. 1 on the left shows the median rate of evaluations (r_{limit} being 100%) used in the races. With Tetris bounds this number of evaluation is reduced at the beginning of the learning process and increases towards the end reaching r_{limit}. For Hoeffding bounds the number of evaluations oscillates between r_{limit} and lower values. Finally, Bernstein bounds were not efficient compared to Hoeffding bounds, the number of evaluations is not reduced and at each race each individual is evaluated r_{limit} times. This is due to the fact that the standard deviation of the fitness is of the same order of its average. This is the case for Tetris, the standard deviation of the score distribution is almost equal to its mean. In other words $\hat{\sigma_{i,t}}$ is large compared to R in eq. 5. When this is the case Hoeffding bounds are tighter than Bernstein's, and this is the case in all our experiments.

Recall that r_{limit} is adapted using equations 2 and 3 and is initially set to 3, which explains why in the right of fig. 1, Bernstein races used less evaluations than in the case without racing.

We noticed also that there is a large variability in the racing procedure. Fig. 2 shows the rate of evaluation out of r_{limit} of all 50 runs for Hoeffding and Tetris bounds.

[5] MDPTetris simulator is available at `http://mdptetris.gforge.inria.fr`

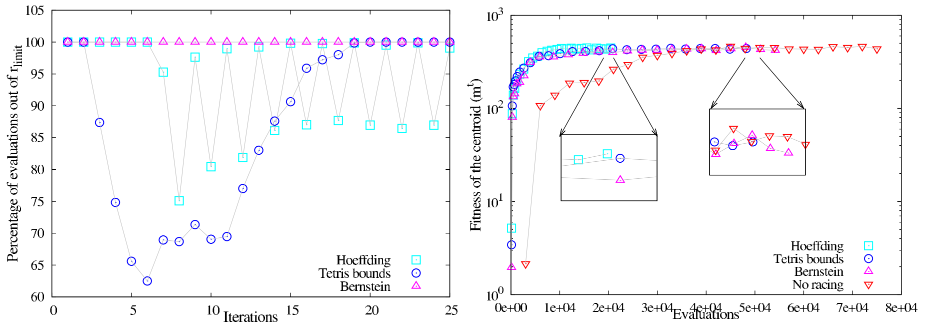


Fig. 1. On the left, the number of evaluations with racing using different bounds versus iterations. On the right, log of the fitness of the mean vector m^t versus the number of evaluations. Median values over 50 runs.

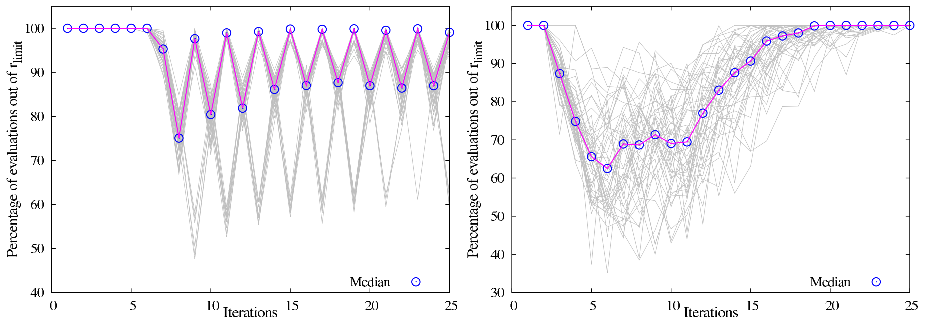


Fig. 2. The number of evaluations for 50 runs thick line represent the median value. With Hoeffding races (left). Races with Tetris bounds(right).

It is interesting to notice that the racing procedures reduce the number of evaluations early in the process and do not perform as well at the end. At this stage the racing procedure cannot distinguish between the offspring and selection is made only based on the fitness values (averaged over the r_{limit} reevaluation). The CI around the offspring's fitness overlap indicating that they perform equally well. This also suggest (see below) that the ranking of the offspring might not be correct. Incidentally, this correspond to the time where the convergence of step-sizes start to "flatten" (figures 4 and 5).

The effect of the population size on the racing procedure can be seen on fig. 3 where is shown the number of evaluations out of r_{limit} for different population sizes. Apparently increasing the population size reduces the number of evaluations within the race using Hoeffding bounds. On the other hand the opposite can be seen when using Tetris bounds.

Furthermore the population size does not effect the performance of the algorithms, the median value of the mean vector fitness reach similar values for different population sizes. This is the case for both Tetris and Hoeffding bounds (figures 4 and 5).

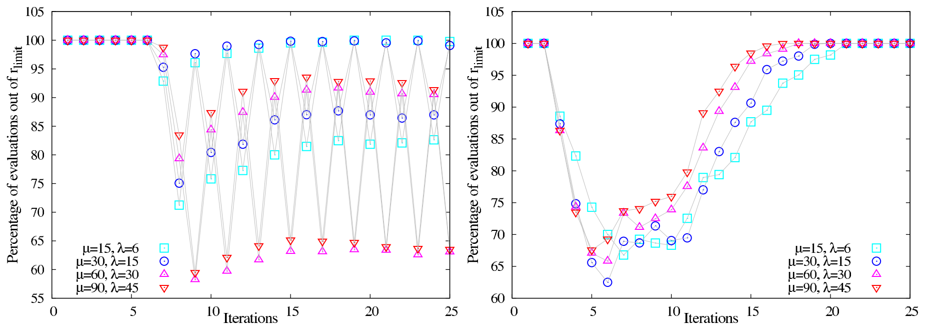


Fig. 3. The median number of evaluations over 50 runs for different population sizes. Using Hoeffding races (left). Using Tetris bounds (right)

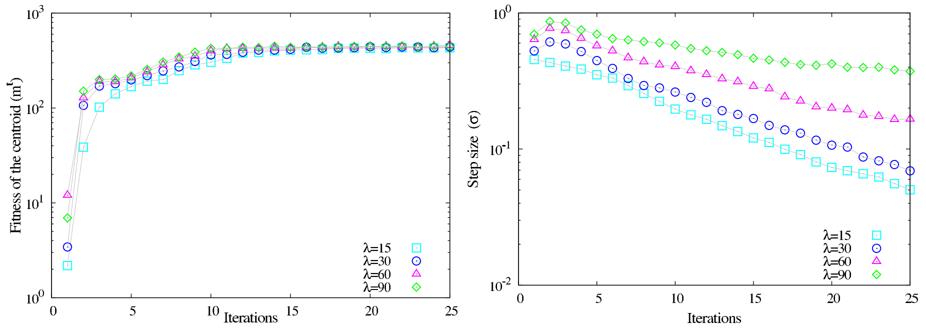


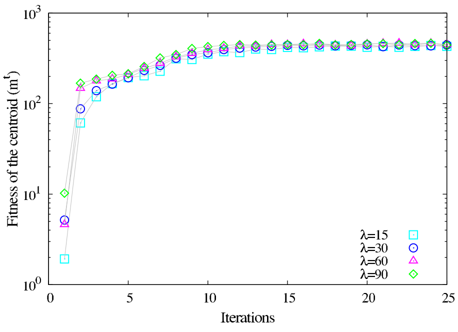
Fig. 4. CMA-ES and Tetris bounds with multiple population sizes. Log of the fitness of the mean vector m^t (left) and the step-size σ (right). Median values over 50 runs.

Fig. 6 shows the effect of the confidence value on the evaluation time. It presents four setting with different confidence $(1 - \delta)$ levels. Using Hoeffding bounds, if we reduce the confidence value the algorithm stops earlier. This is the expected behavior: reducing the confidence reduces the bounds and races are decided faster. On the other hand, Bernstein bounds are not affected at all. The races are not able to distinguish statistical differences between the offspring even with low confidence values.

In order to assess the results of the racing procedure, we test the learned players on a standard game[6] and compare the scores with previously published scores. We follow the same scheme as in [5] to construct the player. We record the last mean vector m^t of each run of the algorithm and compute the mean of all these vectors (in our setting 50). This is merely a convention and others are possible[7]. The score of a player is the average score on 100 games. Taking into

[6] A game of size 10×20 with all seven pieces equally probable.

[7] One could also choose the best performing vector out of the whole set. However testing them all on a reasonable number of games might be long. For example testing one player on 10 games took 15 hours on a 2 Ghz CPU.



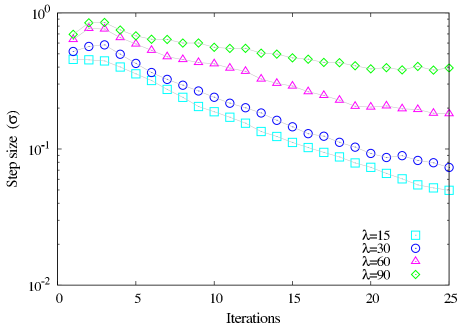
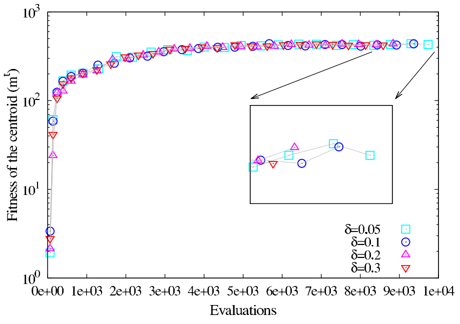


Fig. 5. CMA-ES and Hoeffding races with multiple population sizes. Log of the fitness of the mean vector m^t (left) and the step-size σ (right). Median values over 50 runs.



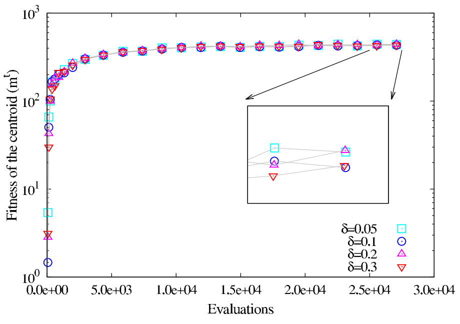


Fig. 6. Log of the fitness of the population centroid for different confidence values. Hoeffding races (left) and Bernstein races (right). Median values over 50 runs.

Table 1. Comparison of the scores of the learned strategies on two game sizes

Strategy	10×16	10×20	Evaluations
No racing [5]	8.7×10^5	36.3×10^6	100%
Tetris bounds	8.1×10^5	31.5×10^6	67%
Hoeffding	7.8×10^5	33.2×10^6	27%

account variability of the score distribution, a confidence bound of ±20% is to be taken into account (see eq.6).

Table 1 present the scores of the players learned on CMA-ES with racing compared to the score reported by [5]. We notice that for both games instances the scores are statistically equivalent. Therefore we can conclude that the racing procedures do not affect the performance of the player. Furthermore, the same performance was obtained using, in the case of Hoeffding bounds two thirds less evaluations and in the case of Tetris bounds one third less evaluations.

3.2 Further Thoughts

Even though there are benefits in using racing, there are also some disadvantages. First of all, the choice of a value for r_{max}, is crucial for time consuming problems.

A too low value leads to short races and thus to no statistical soundness. On the other hand giving a hight value may lead to too many evaluations once the algorithm converges.

As said earlier, when the algorithm converges, the confidence intervals overlap and the racing procedure cannot distinguish statistical differences. This could be considered as a limitation in problems where the cost of an evaluation is not the same at the beginning of the evaluation and towards the end, as it is the case for Tetris. If given the choice, one would prefer to have less costly evaluations rather than the converse. In problems where the cost of the evaluation is constant, this issue is not as severe.

Furthermore, in the event the offspring converge, they all have the same fitness (again statistically). This could raise problems in algorithms that select based on ranking. In the experiments we performed, we noticed that in some cases the ranking (based only on the fitness values) of the offspring changes at each reevaluation during the race. This indicates that at no time the ranking of the population is correct. Selection in such cases becomes random which, and this is purely speculative, might explain why the step-size increases at that stage (figures 4 and 5).

One way to circumvent this limitation, would be to use this convergence as an indication to restart procedure. Once the population converges and we cannot distinguish different individuals, start the algorithm over. It could also be used as a stopping criterion.

4 Conclusions

In the present work, we have described the use of racing methods to reduce the evaluation time in learning artificial Tetris players. Designing such players was done in the past by performing the learning on reduced instances of the game. The addition of racing methods can reduce significantly the learning time without loss of performance, as it has been shown in the experiments.

These experiments also showed that the population size does not affect the racing procedure and its performance. Using Hoeffding and Tetris bounds allowed to reduce the evaluation time, on the other hand Bernstein bounds were inefficient in all our problem instances due the properties of the Tetris fitness function. This also was the case when the confidence level was lowered.

Testing on the standard game instances allowed to show that players designed using CMA-ES with racing have the same performance as the best existing players. Racing also raises few questions that we leave for further investigations.

References

1. Audibert, J.-Y., Munos, R., Szepesvári, C.: Tuning Bandit Algorithms in Stochastic Environments. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 150–165. Springer, Heidelberg (2007)
2. Bertsekas, D., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific (1996)

3. de Boer, P., Kroese, D., Mannor, S., Rubinstein, R.: A tutorial on the cross-entropy method. Annals of Operations Research 1(134), 19–67 (2004)
4. Böhm, N., Kókai, G., Mandl, S.: An Evolutionary Approach to Tetris. In: University of Vienna Faculty of Business; Economics, Statistics (eds.) Proc. of the 6th Metaheuristics International Conference, CDROM (2005)
5. Boumaza, A.: On the evolution of artificial tetris players. In: Proc. of the IEEE Symp. on Comp. Intel. and Games, CIG 2009, pp. 387–393. IEEE (June 2009)
6. Burgiel, H.: How to lose at Tetris. Mathematical Gazette 81, 194–200 (1997)
7. Demaine, E.D., Hohenberger, S., Liben-Nowell, D.: Tetris is Hard, Even to Approximate. In: Warnow, T.J., Zhu, B. (eds.) COCOON 2003. LNCS, vol. 2697, pp. 351–363. Springer, Heidelberg (2003)
8. Fahey, C.P.: Tetris AI, Computer plays Tetris (2003), on the web `http://colinfahey.com/tetris/tetris_en.html`
9. Farias, V., van Roy, B.: Tetris: A study of randomized constraint sampling. Springer (2006)
10. Hansen, N., Müller, S., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation 11(1), 1–18 (2003)
11. Hansen, N., Niederberger, S., Guzzella, L., Koumoutsakos, P.: A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. IEEE Trans. Evol. Comp. 13(1), 180–197 (2009)
12. Heidrich-Meisner, V., Igel, C.: Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In: Proc. of the 26th ICML, pp. 401–408. ACM, New York (2009)
13. Maron, O., Moore, A.W.: Hoeffding races: Accelerating model selection search for classification and function approximation. In: Proc. Advances in Neural Information Processing Systems, pp. 59–66. Morgan Kaufmann (1994)
14. Ostermeier, A., Gawelczyk, A., Hansen, N.: A derandomized approach to self-adaptation of evolution strategies. Evolutionary Computation 2(4), 369–380 (1994)
15. Schmidt, C., Branke, J., Chick, S.E.: Integrating Techniques from Statistical Ranking into Evolutionary Algorithms. In: Rothlauf, F., Branke, J., Cagnoni, S., Costa, E., Cotta, C., Drechsler, R., Lutton, E., Machado, P., Moore, J.H., Romero, J., Smith, G.D., Squillero, G., Takagi, H. (eds.) EvoWorkshops 2006. LNCS, vol. 3907, pp. 752–763. Springer, Heidelberg (2006)
16. Siegel, E.V., Chaffee, A.D.: Genetically optimizing the speed of programs evolved to play tetris. In: Angeline, P.J., Kinnear Jr., K.E. (eds.) Advances in Genetic Programming 2, pp. 279–298. MIT Press, Cambridge (1996)
17. Stagge, P.: Averaging Efficiently in the Presence of Noise. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 188–197. Springer, Heidelberg (1998)
18. Szita, I., Lörincz, A.: Learning tetris using the noisy cross-entropy method. Neural Comput. 18(12), 2936–2941 (2006)
19. Thiery, C., Scherrer, B.: Building Controllers for Tetris. International Computer Games Association Journal 32, 3–11 (2009)
20. Thiery, C., Scherrer, B.: Least-Squares λ Policy Iteration: Bias-Variance Trade-off in Control Problems. In: Proc. ICML, Haifa (2010)
21. Tsitsiklis, J.N., van Roy, B.: Feature-based methods for large scale dynamic programming. Machine Learning 22, 59–94 (1996)