High-Dimensional Function Approximation for Knowledge-Free Reinforcement Learning: a Case Study in SZ-Tetris

Wojciech Jaśkowski, Marcin Szubert, Paweł Liskowski and Krzysztof Krawiec
Institute of Computing Science, Poznan University of Technology
Piotrowo 2, 60965 Poznań, Poland
{wjaskowski,mszubert,pliskowski,kkrawiec}@cs.put.poznan.pl

ABSTRACT

SZ-Tetris, a restricted version of Tetris, is a difficult reinforcement learning task. Previous research showed that, similarly to the original Tetris, value function-based methods such as temporal difference learning, do not work well for SZ-Tetris. The best performance in this game was achieved by employing direct policy search techniques, in particular the cross-entropy method in combination with handcrafted features. Nonetheless, a simple heuristic hand-coded player scores even higher. Here we show that it is possible to equal its performance with CMA-ES (Covariance Matrix Adaptation Evolution Strategy). We demonstrate that further improvement is possible by employing *systematic n-tuple network*, a knowledge-free function approximator, and VD-CMA-ES, a linear variant of CMA-ES for high dimension optimization. Last but not least, we show that a large systematic n-tuple network (involving more than 4 million parameters) allows the classical temporal difference learning algorithm to obtain similar average performance to VD-CMA-ES, but at 20 times lower computational expense, leading to the best policy for SZ-Tetris known to date. These results enrich the current understanding of difficulty of SZ-Tetris, and shed new light on the capabilities of particular search paradigms when applied to representations of various characteristics and dimensionality.

CCS Concepts

•Mathematics of computing → Evolutionary algorithms; •Computing methodologies → Reinforcement learning; Temporal difference learning; *Neural networks;* **•Applied computing → Computer games;**

Keywords

Reinforcement learning; covariance matrix adaptation; CMA-ES; VD-CMA; function approximation; knowledge-free representations; video games; n-tuple system

GECCO '15, July 11 - 15, 2015, Madrid, Spain

ISBN 978-1-4503-3472-3/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2739480.2754783

1. INTRODUCTION

Evolutionary algorithms (EAs) implement a *generate-and-test* approach: candidate solutions are randomly perturbed and only then evaluated. This distinguishes them from the gradient-based approaches that modify candidate solutions in a directional way and so attempt to reduce the discrepancy between the current and the desired search state. By exploiting problems in such a way, the gradient-based methods may be more efficient at finding well-performing candidate solutions. On the other hand, by strictly following gradient when updating candidate solutions, they bias the search and may fail to exploit the uncharted parts of the search space. EAs are less biased in this sense and may so compensate for their inability to utilize the information on gradient.

The interplay between these aspects is particularly relevant for reinforcement learning (RL), the study of agents that issue actions that change the states of an environment and learn from the resulting delayed rewards. Within RL, EA-based techniques fall into the category of *direct policy search*, while examples of gradient-based learning techniques can be found among *value function-based methods* (which attempt to approximate the unknown underlying value of particular states). Value function-based methods prove spectacularly good in some problems (e.g., Backgammon [22]), while direct policy search fares very well in the others (e.g., Othello [8]).

Understanding the causes of these performance differences has been an important part of research agenda in RL [9]. There is partial evidence for many factors being involved here; Szita [16] suggested that policy representation (relying on function approximation), the presence of randomness, environment observability, and training regime are, among others, the critical factors. In this study, we focus on policy representation and, more specifically, on its dimensionality, meant as the number of variables/parameters that characterize candidate policies.

Value function-based methods derive an update rule for every variable of an agent's policy, and are by this token relatively insensitive to the their number. For instance, in some studies on Othello, the temporal difference learning algorithm, the classical value function-based RL method, proved very efficient when learning policies encoded by thousands of parameters [20]. At the same time, the results indicated inferior performance of EAs, which traversed the space of candidate solutions more slowly using conventional evolutionary search operators. However, we are currently witnessing the advent of EA techniques that are designed to handle high numbers of dimensions while capturing the interdependency between them. A notable representative of this trend is the CMA-ES (Covariance-Matrix Adaptation Evolution Strategy, [7]) approach; in particular, its recent variant, VD-CMA-ES [1], which avoids estimation of the entire covariance matrix and is thus particularly suitable to high-dimensional spaces. In this light, it is justified to ask: are these new generation EA techniques, when applied to RL problems, capable of equaling the performance of the value function-based methods for policy representations with a large number of parameters?

In this study we attempt to answer this question for SZ-Tetris [17], a difficult single-player stochastic game, which is a constrained variant of Tetris, one of the most common yardsticks in RL. Our contribution consists in bringing the evidence that CMA-ES is indeed able to attain the performance of value function-based methods, albeit at a larger computational cost. We also demonstrate the superiority of high-dimensional, domain-independent policy representations (systematic n-tuple networks) to handcrafted board features. Finally, using such a high-dimensional representation, we produce the best SZ-Tetris player to date, solving in this way the challenge posed recently by Szita and Szepesvári [17]:

> *Challenge #1: Find a sufficiently good feature set (semi-automatically or fully automatically). A feature set is sufficiently good if CEM (or CMA-ES, or genetic algorithms, etc.) is able to learn a weight vector such that the resulting preference function reaches at least as good results as the hand-coded solution.*

2. SZ-TETRIS

2.1 Game description

Tetris is a popular single-player stochastic video game created by Alexey Pajitnov in 1984. The game is played on a 10 × 20 board. During the game random *tetrominoes* – shapes composed of four connected square blocks – appear one-by-one at the top edge of the board and fall down the board. The objective of the game is to keep the board possibly clear by rotating and moving tetrominoes sidewise and so forming horizontal lines of blocks without any gaps. When at least one such line is created, it disappears, and the blocks above (if any) move down by the number of lines cleared. A point is awarded for each cleared line and the game ends when a tetromino extends beyond the top of the board when set in place.

The original Tetris had been a popular benchmark in RL [16], but it became hard to study when the best policies started achieving the level of tens of millions cleared lines on average [23]. Learning and evaluating such controllers is computationally expensive.

This is why Szita and Szepesvári [17] proposed to study SZ-Tetris, a variant of the original Tetris constrained to tetrominoes 'S' and 'Z'. SZ-Tetris was originally introduced by Burgiel [4] to prove that every Tetris game eventually ends with probability 1, regardless of the players's actions. 'S' and 'Z' tetrominoes are the most difficult ones to handle, which makes SZ-Tetris much harder than the original game. As a result, the computational cost of a single simulation is significantly lower. This facilitates experimentation and allows to measure performance of complex strategies with a reasonable accuracy as Tetris games typically have huge variance.

There are $2^{10\times 20}$ states in both Tetris and SZ-Tetris. In the latter one, regardless of the type of tetromino available, there are 17 possible actions in each move (9 vertical + 8 horizontal).

2.2 A Hand-Coded Controller

Surprisingly, from the computational intelligence point of view, the best SZ-Tetris controller to date has been a hand-coded player proposed as a baseline for other approaches [17]. The controller first divides the board into 5 blocks, each consisting of two adjacent columns. Throughout the game, the 'S' tetrominoes are placed vertically in the first or second block, while the 'Z' tetrominoes in the fourth or fifth block. The type of tetrominoes to be dropped in the third, middle block alternates with game progress. Initially, the player drops there the tetrominoes of the same type as the first tetromino that appeared in the game. Once the highest column on the board exceeds the height of 16[1], the type of tetrominoes for the middle block is flipped. The policy always drops a tetromino into the least occupied block of the compatible tetromino type.

According to Szita and Szepesvári [17], this heuristics, referred to in the following as *Hand-coded*, clears 182 rows on average, which is consistent with our implementation that achieves a score of 183.61±1.4 (95% confidence interval), with the median score of 191.0.

2.3 State-Value Function and Action Selection

In every iteration of the game, given a state s of the board (a vector of $10 \times 20 = 200$ bits) and an incoming tetromino, a SZ-Tetris policy needs to choose the action (move) to be taken. As in many other reinforcement learning problems, one defines to this aim a *function approximator* f that captures the preference of board state in a real number. The direct *afterstates* of s are generated by simulating each of the 17 possible moves: 9 columns for a 2×3-tetromino dropped vertically, and 8 when rotated to 3×2 and dropped horizontally. Then, f is applied to each such *afterstate*, and the move leading to the state with the highest f is applied. Ties on f are resolved at random.

Note that, depending on the learning algorithm, f can be either a *state-value function*, which approximates the expected score from the given state or just a *state-preference function*, which absolute value has no interpretation.

Either way, the function f is the key determinant of policy performance and is usually based on a set of board *features*, either manually designed or defined in a generic, knowledge-free manner. The following subsections describe the classical expert-designed SZ-Tetris features and our proposal of knowledge-free features. Both these representations will be used in the experiments.

2.3.1 "Classical" Bertsekas & Ioffe (B&I) Features

In their early work on Tetris [2], Bertsekas *et al.* proposed the following set of features $\phi_i(s)$ of a board state (afterstate) s:

[1] This number was wrongly reported to be 15 in [17], but correct value of 16 appears in the code attached to the paper.

- The height h_k of the kth column of the board, $k = 1, \dots 10$,
- The absolute difference $d_k = |h_k - h_{k+1}|$ between the heights of the kth and the $(k+1)$ column, $k = 1, \dots 9$,
- The maximum column height max_h,
- The number of ‘holes‘ on the board.

In total, this gives rise to $10+9+1+1 = 21$ features, and the state-value function f is defined as their linear combination:

$$f(s) = \sum_{i=1}^{21} w_i \phi_i(s),$$

Thus, an algorithm that learns to play SZ-Tetris using the B&I features searches a 21-dimensional space of weights w_i.

We find it important to note that the implementation used by Szita and Szepesvári [2] involved an additional end-game heuristic that was not explicitly described in the paper: the actions that immediately lead to terminal states are avoided unless there no other possibilities. Our preliminary experiments indicated that this seemingly minor modification has critical impact on performance when using B&I features[2], so we include it also in the implementation used in this study.

2.3.2 Systematic n-Tuple Network

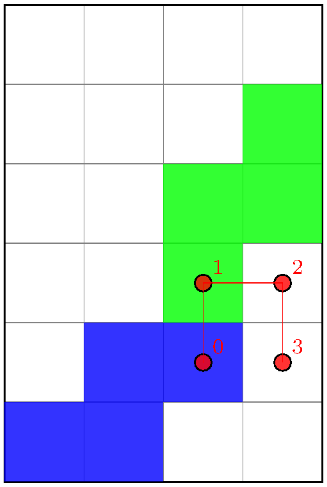
An *n-tuple network* is a combinatorial structure originally proposed by Bledsoe and Browning [3] for pattern recognition and recently successfully adopted for Othello [12], Connect-4 [25] and the puzzle game 2048 [18]. It consists of m *n-tuples*, each representing a single board feature. The ith n-tuple consists of a list of n distinct board locations $(loc_{ij})_{j=1\dots n}$ and an associated lookup table LUT_i that enumerates all possible states of the selected locations. For games with binary states of individual locations (like SZ-Tetris), the lookup table thus consists of 2^n entries, each holding a real number. Given a board state, the value of the feature is the entry of the lookup table addressed by the combined states of the board locations covered by the n-tuple. To keep the size of the lookup table within reasonable limits, n is typically low.

Formally, an n-tuple network implements a state-value function f as a sum of values returned by individual n-tuples f_i for a given state s:

$$\begin{aligned} f(s) = \sum_{i=1}^{m} f_i(s) &= \sum_{i=1}^{m} LUT_i \left[\operatorname{index}\left(s_{loc_{i1}}, \dots, s_{loc_{in}}\right)\right], \\ \operatorname{index}(\mathbf{v}) &= \sum_{j=1}^{|\mathbf{v}|} v_j c^{j-1}, \end{aligned}$$

where $s_{loc_{ij}}$ is a board value at location loc_{ij}, $\mathbf{v}$ is a sequence of board values at those locations (the observed pattern), and c is the number of possible values in a board location, so that $0 \leq v_j < c$ for $j = 1 \dots |\mathbf{v}|$. In the case of Tetris, $c = 2$ and values in $\mathbf{v}$ are 0 or 1 for empty and occupied locations respectively. See Fig. 1 for an illustration.

N-tuple networks mitigate the combinatorial explosion while capturing the utility of particular *combinations* of board states in a way that cannot be realized by the simpler linear representations.



LUT

0123	value
0000	3.04
0001	−3.90
0010	−2.14
⋮	⋮
1100	**−2.01**
⋮	⋮
1110	6.12
1111	3.21

Figure 1: A 2×2-tuple on a small Tetris board. According to the values in its lookup table, for the given board state it returns -2.01.

Given the overwhelming number of n-ary combinations that can be formed from a board in most games, the locations of n-tuple are often drawn at random [11]. However, recently, it was found out that systematically placing n-tuples on the board leads to better performance [8] for Othello and avoids introducing additional variation into policy performance (the primary source of variation being game stochasticity and, in multi-player games, the opponents). Therefore, in this study we systematically cover the SZ-Tetris board by placing a same-shaped square n-tuples in all possible locations while allowing for overlaps. We consider two n-tuple shapes: 3×3 and 4×4. The former case gives rise to $(10-3+1)(20-3+1)/2 = 72$ unique n-tuple locations[3]. The lookup table of each such n-tuple has $2^{3\times3} = 512$ entries, so the total number of weights of such a systematic n-tuple network amounts to 36 864. In the latter case, there are 136 unique 4×4-tuple locations, and the total number of weights is thus $136 \times 2^{16}/2 = 4\,456\,448$. These are the numbers of parameters that define the strategies to be learned in Section 3.

Note that, with the n-tuple network, we did not use the end-game heuristic mentioned in Section 2.3.1. The preliminary tests have shown that it is not necessary for this function approximator.

2.4 Other approaches to SZ-Tetris

As we mentioned in Section 2.2 the best controller for SZ-Tetris was hand-coded by Szita and Szepesvári [17] and its mean score is about 183. In the same work, the authors experimented with learning weights for, among others, B&I features (see Section 2.3.1) and its discretized variants, but their best learned agent scores only 133 on average. More recently, Faußer and Schwenker [5] approached the game with a voting committee of TD(λ) agents achieving the av-

[2] Without it, we were not able to exceed 50 points with B&I features.

[3] The division by 2 is because we take board symmetry into account: given any two vertically mirrored afterstates, there is no rationale to prefer any of them, so f should grant them with the same value.

erage score of 150. Importantly, all these studies relied on hand-coded knowledge-based representations.

3. EXPERIMENTS AND RESULTS

3.1 Direct Policy Search Methods

In the first series of experiments[4], following the existing body of research about effective learning methods for Tetris [16], we use only direct policy search methods. In particular, we employ Cross-Entropy Method (CEM, [13]) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES, [7]). Both methods can be classified as evolution strategies that work by maintaining, updating and sampling from a multivariate Gaussian probability distribution $\mathcal{N}(\mu, \Sigma)$. The simplest form of CEM, which we employ here, assumes that the covariance matrix Σ is a diagonal matrix. CMA-ES maintains and adapts the entire covariance matrix and features some additional adaptation mechanisms which CEM lacks. However, it was observed recently that CEM is just a special case of CMA-ES [14].

3.1.1 *Hand-Designed B&I Features*

In the first experiment we used the standard set of hand-designed Tetris B&I features, which implies learning taking place in a 21-dimensional parameter space (see Section 2.3.1). Following the initial study on SZ-Tetris [17], for CEM we used the selection rate 0.01[5] and population size of 1000. However, in contrast to that work, the fitness of an individual is an average number of cleared lines over 100 games, instead of just one game. In preliminary experiments, we have found that using such more precise evaluation is beneficial for the final performance.

For CMA-ES we used the standard version of the algorithm[6], also with the population size of 1000. What we, however, turned off was the step-size adaptation (step-size was set to 1), which we found harmful for the optimization process in the preliminary experiments.

We stop both algorithms after 200 generations (that is after $1000 \times 100 \times 200 = 20\,000\,000$ of games).

The performance obtained in 10 runs of the algorithms are shown in the left inset in Fig. 2. A single datapoint in the figure marks the performance of a best-of-generation solution. On average, CMA-ES performs slightly better than CEM (124.8 ± 13.1 vs. 117.0 ± 6.3 on average), but statistically there is no difference between these results. Interestingly, although most of the runs of CMA-ES cluster near the average of 124.8 points, one run found a policy that scores 180.6 ± 1.2 on average, equaling the SZ-Tetris record belonged to the hand-coded controller (cf. Section 2.2). To date, it was dubious whether this performance level can be achieved using the existing set of features (like the B&I ones) [17]. The achievement of this policy shows also that the employed direct policy search methods are prone to getting stuck in local minima, where significantly better optima exist. For reproducibility, below we show the weights of this best-performing controller:

h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}
70.8	-76.2	5.5	-2.4	-11.9	0.58	-15.7	60.3	72.2	-22.0
d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	
7.3	-0.9	-41.8	-2.3	-44.4	0.75	-28.3	55.7	-56.2	
max_h	holes								
93.1	-55.5								

3.1.2 *Knowledge-Free Systematic n-Tuple Network*

An agent using the handcrafted B&I features can equal the performance of the hand-made heuristic agent. Is it possible to achieve the same level of performance without manually designing the features? In order to answer this question, we employ the 3×3-tuple networks presented in Section 2.3.2.

Knowledge-free approaches require longer learning than the knowledge-based ones. n-tuple networks are no exception to this rule. While the function involving B&I features have only 21 parameters, the 3×3-tuple network consists of 36 864 of them. Such a high number of parameters precludes the use of CMA-ES, since its time complexity depends on the square of the number of optimized parameters. Instead we used VD-CMA-ES [1], a new linear variant of CMA-ES, which maintains the covariance matrix implicitly in a form of $D(I + vv^{\mathrm{T}})D$, where D is a diagonal matrix of dimension n and v is a vector in $\mathbb{R}^n$. This expression cannot represent all possible covariance matrices, but is able to capture some dependencies between the variables in a compressed way, allowing so to apply the concepts of covariance matrix adaptation to highly dimensional spaces.

The course of learning is shown in the right inset of Fig. 2. Due to the large number of parameters, the learning is much slower; however, while the learning with the B&I features saturates quickly, VD-CMA-ES with n-tuples does not exhibit such a tendency, and even after 1000 generations, when we stopped the experiment, there is an observable potential for further growth. Most importantly, however, VD-CMA-ES using the n-tuple network surpasses CMA-ES with B&I features by a large margin (219.7 ± 2.8 vs. 124.8 ± 13.2). The best controller obtained by VD-CMA-ES scores 223.0 ± 1.4 points on average.

The experiment also shows that, despite CEM is a viable method for a small number of parameters, it performs poor in these high-dimensional spaces, being not able to find any agent that clears more than 20 lines.

3.2 Temporal Difference Learning

3.2.1 *Systematic 3×3-Tuple Network*

Szita and Szepesvári stated: *"There are many RL algorithms for approximating the value functions. None of them really work on (SZ-)Tetris, they do not even come close to the performance of the evolutionary approaches."* [17]. Indeed, when we applied the classical temporal difference learning TD(0) [15] to B&I features, the agent performance after 1 million training episodes was still close to 0 regardless of the exploration and learning rates.

This underperformance was puzzling, given that TD(0) has been found successful is so many various domains. We hypothesized that some peculiar properties of the B&I features could have prevented the algorithm from locating the well-performing policies. To verify this, we applied TD(0)

[4] The source code required to run the experiments along with the best obtained players is available at http://github.com/wjaskowski/gecco-2015-sztetris.

[5] Szita and Szepesvári [17] report to use the selection rate 0.1, but in the code repository referred from the paper selection rate was set to 0.01.

[6] https://code.google.com/p/cma-es/

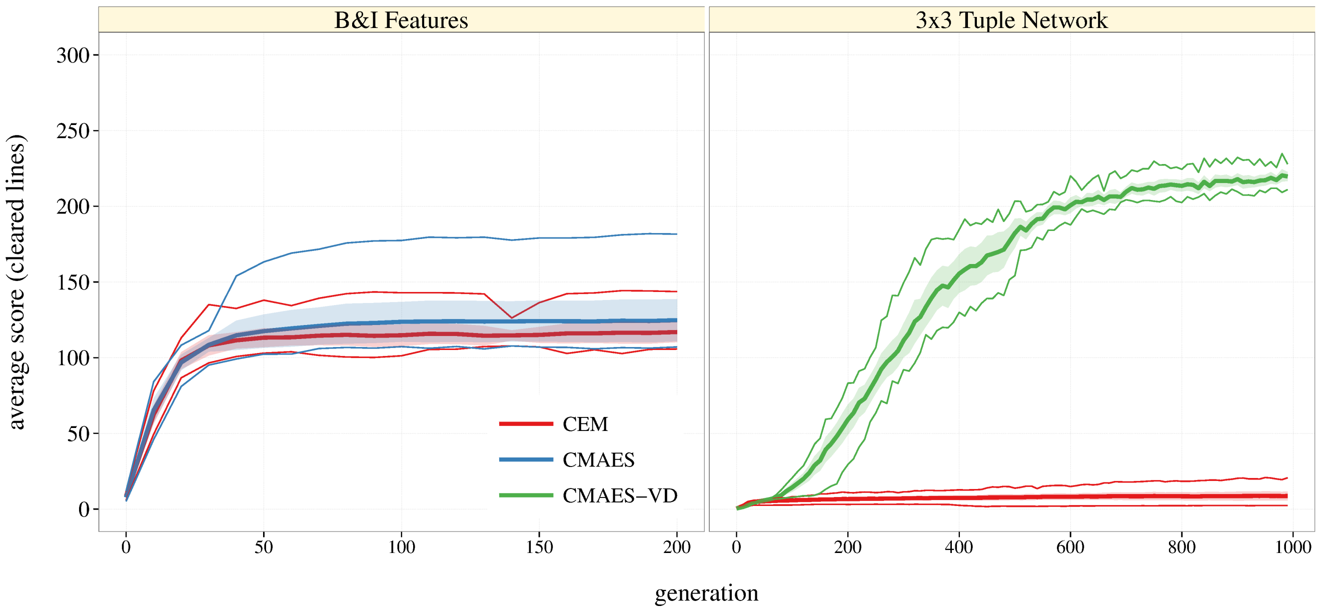


Figure 2: SZ-Tetris learning performance for different direct policy search methods and two state-value functions: the linear combination of Bertsekas & Ioffe features (left) and the 3×3 systematic n-tuple network (right). The thick curve plots the mean performance while the thin lines show the minimum and the maximum performance of 10 evolutionary runs. The 95% confidence interval around the mean has been marked as a shaded ribbon. The systematic n-tuple network was trained by VD-CMA-ES, which has linear time complexity in the function of the number of parameters (while CMA-ES's complexity is quadratic).

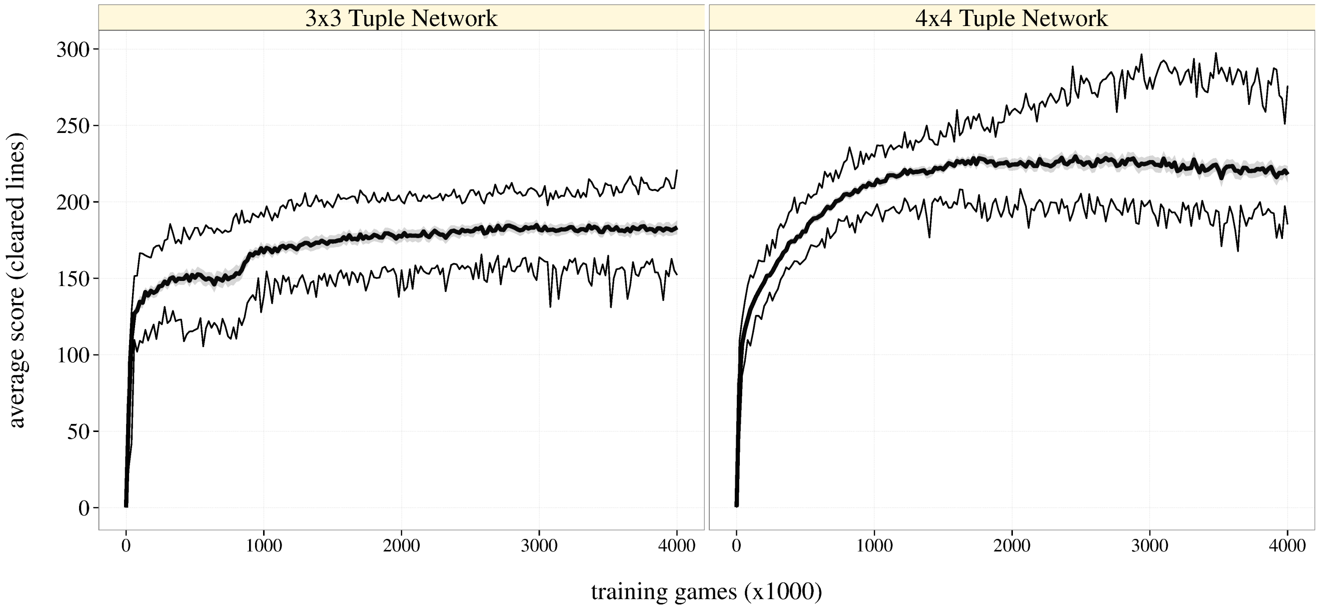


Figure 3: SZ-Tetris scores obtained by TD(0) agent in the function of the training games for 3×3 and 4×4-tuple network. The thick curve plots the average performance while the thin lines show the minimum and the maximum performance achieved in 50 runs. The 95% confidence interval around the mean has been marked in gray.

Table 1: Comparison of the methods for SZ-Tetris. Method's average has been estimated based on the number of samples in 'Runs' column. Best agent performance has been estimated on 10 000 episodes. ± precedes 95% confidence interval delta.

Algorithm	Evaluation Function	Features	Learning Games	Method's Average	Runs	Best Agent Score
Hand-coded [17]	-	-	-	-	-	183.6 ± 1.4
TD(0.5) committee [6]	MLP on B&I	10 × 1655	5 mln	-	-	ca. 150 [6]
CEM	B&I	21	20 mln	117.0 ± 6.3	10	146.4 ± 1.0
CMA-ES	B&I	21	20 mln	124.8 ± 13.1	10	180.6 ± 1.2
VD-CMA-ES	3×3-tuple network	36 864	100 mln	219.7±2.8	10	232.0 ± 1.4
TD(0)	3×3-tuple network	36 864	4 mln	183.3 ± 4.3	50	220.0 ± 1.3
TD(0)	4×4-tuple network	4 456 448	4 mln	218.0 ± 5.2	50	294.8 ± 1.4

to the 3 × 3-tuple network, using the learning rate $\alpha = 0.001$ and ϵ-greedy policy. ϵ was initialized by 0.1 and multiplied by 0.9 every 100 000 training episodes.

Figure 3 shows the results of 50 runs of TD(0). Its average score is significantly lower than the score obtained byVD-CMA-ES (183.3 ± 4.3 vs.219.7±2.8), but was achieved at a much lower computational expense (4 million vs. 100 million training episodes).

Nevertheless, n-tuples network is the first representation for which TD(0) works for this problem.

3.2.2 Systematic 4×4-Tuple Network

We treat the reasonably good performance achieved by TD(0) with the 3×3-tuple network as a sign of its capability to efficiently navigate in highly dimensional spaces. Given that the gradient-based update rule used by this algorithm operates on each parameter independently, there are no principal reasons why this could not hold in presence of even higher numbers of parameters.

In the right inset of Fig. 3, we present the performance of TD(0) applied to the systematic 4×4-tuple network. The learning proceeded here in a vast 4 456 448-dimensional search space, i.e., in a search space which dimensionality is of the same order as the number of learning episodes. Nevertheless, this apparently did not prevent TD(0) from excelling on this task. Although, its average performance of 218 ± 5.2 is similar to the one achieved by VD-CMA-ES using 3×3-tuple network 219.7±2.8, it was achieved using 25-times smaller computational budget (4 mln vs. 100 mln games). Short learning time made it possible to increase the number of independent runs to 50. This resulted in a strategy that clears an impressive 294.8±1.4 lines on average, i.e., reaching the level of play that has not been reported for SZ-Tetris to date.

4. DISCUSSION

To streamline our discussion, in Table 1 we summarize the experiments conducted in this paper and the relevant prior results on SZ-Tetris.

The arguably most important observation emerging from these results is the potential dwelling in high-dimensional knowledge-free representations. The systematic n-tuple networks proved versatile here in being not only good approximators of state-value function (when used with a gradient-based method of TD(0)), but also as state-preference functions (when evolved via direct policy search). In terms of average score, none of the configurations relying on handcrafted low-dimensional representations outperformed the policies equipped with n-tuple networks. We find this observation important, given that research on computer Tetris has been to date dominated by approaches that involve handcrafted features, sometimes sophisticated and deeply rooted in domain knowledge [24].

The efficiency of the knowledge-free representations is also conceptually stunning, when one realizes that a n-tuple network-based SZ-Tetris agent has no explicit knowledge on *any* aspects of the game that are essential for designing the handcrafted representations: the constraints resulting from board dimensions, the shapes of the tetrominoes, the 'mechanics' of tetrominoes fitting to each other, etc. Its entire perception of a game state boils down to a sum of a few dozen of numbers selected from lookup tables by a 200-bit board state. Yet despite this appalling ignorance, n-tuples manage to break the current performance records.

Ultimately however, the performance of a given policy depends also on the training algorithm. In this respect, the capacities of temporal difference learning methods are impressive, particularly for the most extensive 4×4 tuples, where it manages to learn supreme policies from a number of episodes that is comparable to the number of policy parameters (4 million episodes vs. 4 456 448 weights). This suggests that Szita's and Szepesvári's diagnosis that value function-based RL algorithms do not work for SZ-Tetris ([17], p.4) does not hold anymore. Well-performing policies can be efficiently found by temporal difference learning for SZ-Tetris once the underlying representation of state-value function is rich enough.

This is however not to say that gradient-based learning of value functions is destined to be always the pinnacle of performance. The local nature of search performed by such techniques incurs a measurable risk of missing even better candidate solutions. Despite the high-dimensional search space, VD-CMA-ES, which is a direct search method, was not only found to surpass TD(0) for 3 × 3-tuple network, but also to match the average score of TD(0) for a larger 4 × 4-tuple network. There are two factors that might have aid the direct policy search: not only the global, exploratory character of CMA-ES, but also the fact that the direct policy search seeks a state *preference* function, and so cares only about the ordering of state values, while the methods like TD(0) approximate state-value function and so insist on finding the right absolute state values. Because for any given state-value function, there are infinitely many state preference functions that select the actions in exactly the same way, finding a function that is good only with respect to the above ordering may be easier.

The price to pay for the potential of achieving better performance is computational expense. Whether gaining a few extra cleared lines in SZ-Tetris is worth investing 25-times greater computation time is a matter of context. If one is ultimately interested in finding that one best-of-all strategy, this may be the case. In this context, it might be tempting to consider hybrid approaches that leverage TD's capability to operate in high-dimensional spaces while maintaining some features of parallel global search. The evolutionary temporal difference learning we proposed and studied in [19, 10, 21, 20] seems to be a viable option for such an endeavor.

We conducted also preliminary behavioral analysis of the well-performing policies[7], trying to identify the key features that might have determined their performance. All learning algorithms compared here managed to more or less rediscover the essence of the hand-coded policy, i.e., they learned to stack the 'S' and 'Z' tetrominoes vertically in five separate blocks of width two. The best 4×4-tuple TD policy is more sophisticated. First of all, it does not stick to a fixed division into five blocks of width two, and can drop a tetromino vertically so that it fills in a single gap at arbitrary abscissa. It can also thoughtfully drop tetrominoes in horizontal position. Most interestingly, sometimes it makes a move that, at first sight, looks suboptimal and leaves a hole buried beneath the tetrominoes. Our analysis revealed, however, that, in a longer run, this makes it easier to dig up ('reopen') the holes and fill them in. More behavioral patterns emerged in the policies learned here, but their thorough analysis is beyond the scope of this paper.

On a more technical level, this study is, to the best knowledge of the authors, the first practical demonstration of the efficiency of VD-CMA-ES [1], the new variant of CMA-ES designed with high-dimensional representation in mind. Apparently, given many dimensions, it may be not necessary to capture all pairwise dependencies between them by maintaining the complete covariance matrix. Modeling only a fraction of that information can be sufficient to attain superior results. On the other hand, taking *some* interdependencies into account is essential, and discarding them altogether has detrimental effect, as witnessed by the inferior performance of CEM.

5. CONCLUSIONS

In this study, we proposed a novel knowledge-free policy representation for the challenging reinforcement learning problem of SZ-Tetris and put it under extensive experimental scrutiny. The overall conclusion is that, by the virtue of high-dimensional systematic n-tuple network representation and appropriate learning algorithms, one can achieve at least the state-of-the-art level of performance in this game while being essentially completely ignorant about its nature. The peak demonstrator of validity of this stance is the best-to-date SZ-Tetris controller, which by clearing over 294 lines on average reaches a new quality of performance and conquers the hand-coded player, solving so the Challenge #1 posed by Szita and Szepesvári [17]. In authors' opinion, these results substantially enrich our current understanding of this game, and possibly similar RL tasks.

With this piece of work, we hope to pave the way towards a more routine use of knowledge-free representations in reinforcement learning tasks, whether approached with state-value-based function learning or with (evolutionary) direct policy search. As it follows from this study, both these approaches offer nowadays powerful algorithms that fare very well even when the number of variables to be simultaneously optimized is in the order of millions. By casting a learning task into a high-dimensional space, this project subscribes to the trend present in other branches of machine learning and computational intelligence, where, for instance, using thousands of features is nowadays common.

The computational overhead incurred by evolutionary search in huge spaces of knowledge-free representations may seem prohibitive. Note however that the statistics shown in Table 1 reflect only the machine-part of the effort invested in finding a solution. In focusing on the time spent on computation, we tend to ignore the human intellectual investment put into designing and validating hand-coded heuristics or handcrafted features. What is, for that instance, the machine-equivalent of the effort devoted by the authors of [17] to design their SZ-Tetris heuristics? If we had the means to take this human investment into account, the overall effort would possibly be not so favorable for knowledge-based approaches anymore.

Acknowledgment

This work has been supported by the Polish National Science Centre grant no. DEC-2013/09/D/ST6/03932. M. Szubert has been supported by the Polish Ministry of Science and Higher Education, grant no. 09/91/DSMK/0568. K. Krawiec acknowledges support from grant 09/91/DSPB/0572. The computations have been performed in Poznan Supercomputing and Networking Center.

6. REFERENCES

[1] Y. Akimoto, A. Auger, N. Hansen, and Others. Comparison-Based Natural Gradient Optimization in High Dimension. In *Genetic and Evolutionary Computation Conference GECCO'14*, 2014.

[2] D. P. Bertsekas and S. Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming. *Lab. for Info. and Decision Systems Report LIDS-P-2349, MIT, Cambridge, MA*, 1996.

[7] Videos presenting exemplary games are available at http://github.com/wjaskowski/gecco-2015-sztetris.

[3] W. W. Bledsoe and I. Browning. Pattern recognition and reading by machine. In *Proc. Eastern Joint Comput. Conf.*, pages 225–232, 1959.

[4] H. Burgiel. How to lose at Tetris. *Mathematical Gazette*, 81:194–200, 1997.

[5] S. Faußer and F. Schwenker. Selective Neural Network Ensembles in Reinforcement Learning. In *European Symposium on Artificial Neural Networks*, pages 105–110, 2014.

[6] S. Faußer and F. Schwenker. Neural Network Ensembles in Reinforcement Learning. *Neural Processing Letters*, 41(1):55–69, 2015.

[7] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[8] W. Jaśkowski. Systematic n-tuple networks for othello position evaluation. *ICGA Journal*, 37(2):85–96, June 2014.

[9] S. Kalyanakrishnan and P. Stone. Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning*, 84(1-2):205–247, 2011.

[10] K. Krawiec, W. Jaśkowski, and M. Szubert. Evolving small-board go players using coevolutionary temporal difference learning with archive. *International Journal of Applied Mathematics and Computer Science*, 21(4):717–731, 2011.

[11] S. Lucas. Learning to play Othello with n-tuple systems. *Australian Journal of Intelligent Information Processing*, 4:1–20, 2008.

[12] S. M. Lucas. Learning to play Othello with N-tuple systems. *Australian Journal of Intelligent Information Processing Systems, Special Issue on Game Technology*, 9(4):01–20, 2007.

[13] R. Y. Rubinstein and D. P. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2004.

[14] F. Stulp and O. Sigaud. Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621*, 2012.

[15] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[16] I. Szita. Reinforcement learning in games. In *Reinforcement Learning*, pages 539–577. Springer, 2012.

[17] I. Szita and C. Szepesvári. SZ-Tetris as a benchmark for studying key problems of reinforcement learning. In *Proceedings of the ICML 2010 Workshop on Machine Learning and Games.*, 2010.

[18] M. Szubert and W. Jaśkowski. Temporal difference learning of n-tuple networks for the game 2048. In *IEEE Conference on Computational Intelligence and Games*, pages 1–8, Dortmund, Aug 2014. IEEE.

[19] M. Szubert, W. Jaśkowski, and K. Krawiec. Coevolutionary temporal difference learning for othello. In *IEEE Symposium on Computational Intelligence and Games*, pages 104–111, Milano, Italy, 2009.

[20] M. Szubert, W. Jaśkowski, and K. Krawiec. On scalability, generalization, and hybridization of coevolutionary learning: a case study for othello. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(3):214–226, 2013.

[21] M. Szubert, W. Wojciech Jaśkowski, and K. Krzysztof Krawiec. Learning Board Evaluation Function for Othello by Hybridizing Coevolution with Temporal Difference Learning. *Control and Cybernetics*, 2011.

[22] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[23] C. Thiery and B. Scherrer. Building controllers for Tetris. *International Computer Games Association Journal*, 32:3–11, 2009.

[24] C. Thiery and B. Scherrer. Improvements on learning Tetris with cross entropy. *International Computer Games Association Journal*, 32, 2009.

[25] M. Thill, P. Koch, and W. Konen. Reinforcement Learning with N-tuples on the Game Connect-4. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 184–194. Springer, 2012.