Approximate Dynamic Programming Finally Performs Well in the Game of Tetris

Victor Gabillon, Mohammad Ghavamzadeh, Bruno Scherrer

Inria informatics mathematics

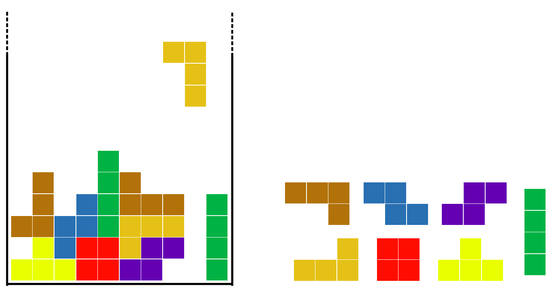
Adobe

Game Formulation

Tetris is a popular video game created by Alexey Pajitnov in 1985.

- played on a grid originally composed of 20 rows and 10 columns,
- 7 different tetrominos fall from the top,
- the player chooses the position and orientation of the falling piece, $|\mathcal{A}| \leq 32$,
- the goal is to remove as many rows as possible before the game is over, i.e., when there is no space available at the top of the grid,
- the player knows only the current falling piece, and not the next coming pieces.

Find a controller (policy) that maximizes the average (over multiple games) number of lines removed in a game (score) is computationally hard as it contains a huge number of board configurations (about $2^{200} \simeq 1.6 \times 10^{60}$), and even in the case that the sequence of pieces is known in advance, finding the strategy to maximize the score is an NP hard problem.



Controllers

A **state** $s =$ **board** $+$ **falling piece**.
All controllers rely on an **evaluation function** f that gives a value to each possible action at a given state. This function is usually a **linear combination** of features ϕ. Then, the controller, defined by the weights $\theta \in \Theta$ of the linear combination, chooses the action with the highest value:

$$\pi(s) = \operatorname*{argmax}_a f(s,a) = \phi(s,a)\theta$$

a weight $\theta \Leftrightarrow$ a controller

Different Approaches

- **ADP:** Most ADP algorithms tried to learn the evaluation function as a **value function** by **approximating** it in the function space spanned by the features ϕ.
  $\Longrightarrow$ Poor results: at most $20,000$ lines. **Tetris value functions are hard to represent.**
- **Policy Search:** directly optimize the weights θ of the controller, using black-box optimizers such as cross entropy method (Szita et. al. 2006)
  $\Longrightarrow$ Good results: $35,000,000$ lines.
  Tetris policies are easier to represent.
- **ADP that directly search in the policy space** : We conjecture that such methods should perform well in Tetris. We will use the CBMPI algorithm (Scherrer et. al. 2012)

Final Performances

We compare our best controllers DT-10 and DT-20 to the current state of the art controller.

Boards \ Policies	BDU	DT-10	DT-20
10×10	4200	5000	4300
10×20	$36M$	$29M$	$51M$

Classification-based MPI

CBMPI computes greedy policies as the output of a cost-sensitive classifier that minimizes the empirical error:

$$\widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu};\pi) = \frac{1}{N}\sum_{i=1}^{N}\left[\max_{a\in\mathcal{A}} \widehat{Q}_k(s^{(i)},a) - \widehat{Q}_k\big(s^{(i)},\pi(s^{(i)})\big)\right]$$

for $k = 1, 2, \ldots$ **do**

- ***Perform rollouts*:**

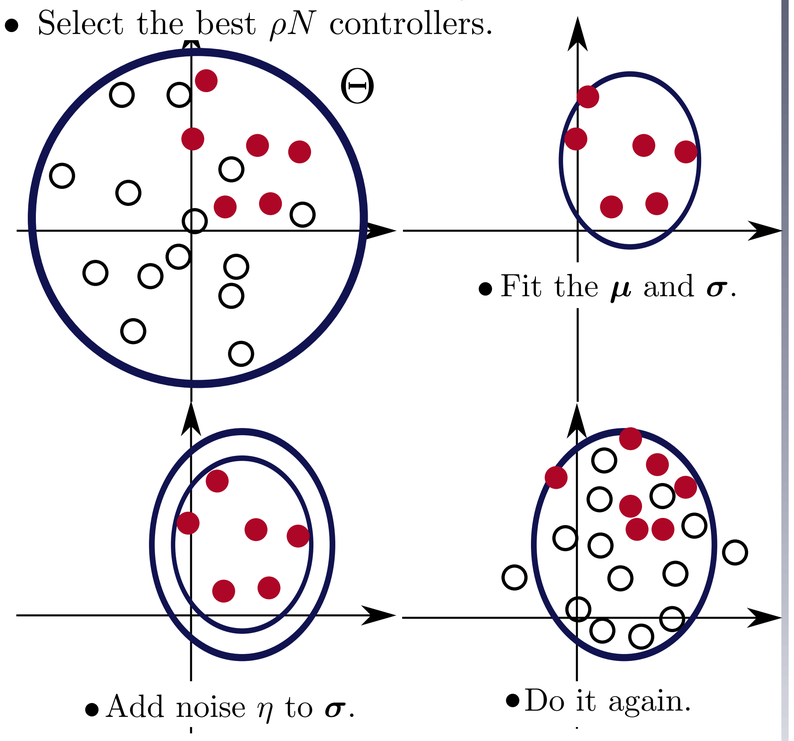
  Rollout set $\mathcal{D}_k = \{s^{(i)}\}_{i=1}^{N}$, $s^{(i)} \overset{\text{iid}}{\sim} \mu$
  for all states $s^{(i)} \in \mathcal{D}_k$ and actions $a \in \mathcal{A}$ **do**
  for $j = 1$ to M **do**
  Perform a rollout and return
  $R_k^j(s^{(i)},a) = \sum_{t=0}^{m} r_t^{(i,j)} + v_{k-1}(s_{m+1}^{(i,j)})$
  $\widehat{Q}_k(s^{(i)},a) = \frac{1}{M}\sum_{j=1}^{M} R_k^j(s^{(i)},a)$
  $\widehat{v}_k(s^{(i)}) = \widehat{Q}_k(s^{(i)},\pi(s^{(i)}))$
- ***Approximate value function:***
  $v_k \in \operatorname*{argmin}_{v\in\mathcal{F}} \widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu};v)$ **(regression)**
- ***Approximate greedy policy:***
  $\pi_{k+1} \in \operatorname*{argmin}_{\pi\in\Pi} \widehat{\mathcal{L}}_k^{\Pi}(\widehat{\mu};\pi)$ **(classification)**

with $\widehat{\mathcal{L}}_k^{\mathcal{F}}(\widehat{\mu};v) = \frac{1}{N}\sum_{i=1}^{N}\big(\widehat{v}_k(s^{(i)}) - v(s^{(i)})\big)^2$

- Let us study the parameter m.
- The cost sensitive lost (classifier) is minimized using CE. Contrary to the CE method for Tetris, no need to play any extra games!
- We use boards with different heights in the $\mathcal{D}_k \Rightarrow$ **significant improvement**
- If $v_k = 0$, CBMPI is **DPI** (Lazaric et. al. 2010)

Cross Entropy (CE)

- Sample N contollers $\theta \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$.
- Evaluate them with over L games.
- Select the best ρN controllers.



$\rho = 0.1, \eta = 4$, (Thiéry et. al. 2009) and $(n = 1000, L = 10)$ in the small board and $(n = 100, L = 1)$ in the large board.

Set of Features

Bertsekas: Mainly used in the ADP/RL community: the number of holes, heights of columns (including consecutive differences and max).

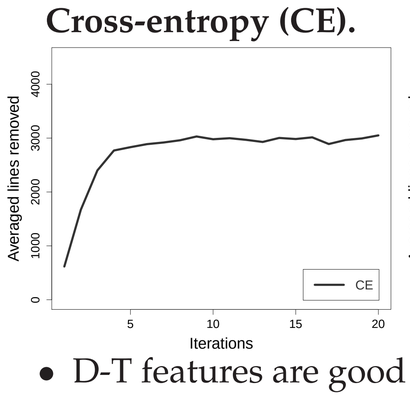
Dellacherie-Thiéry (D-T): landing height, eroded piece cells, row and column transitions, number of holes, board wells; plus hole depth, number of rows with holes, and diversity. The best reported controllers have been learned using these features.

Experiments

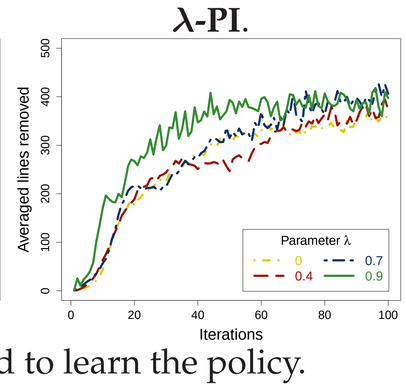
Small Board (10×10):

- **Dellacherie-Thiéry (D-T) features:**

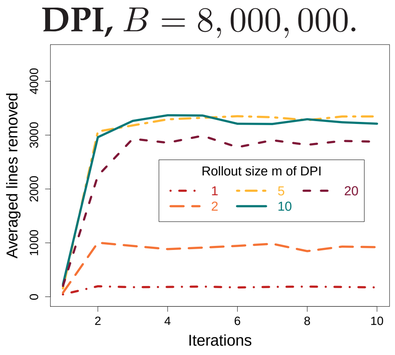
Cross-entropy (CE).



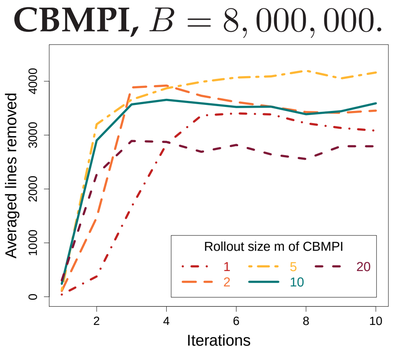
λ-PI.



DPI, $B = 8,000,000$.



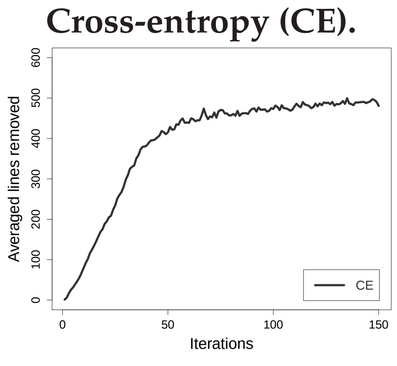
CBMPI, $B = 8,000,000$.



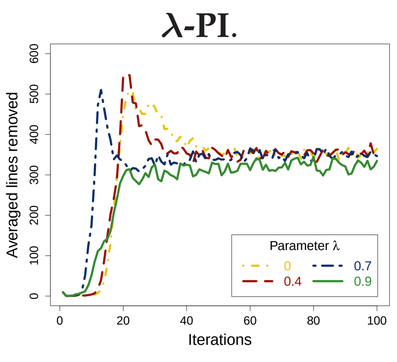
- D-T features are good to learn the policy.
- CBMPI outperforms CE with $\simeq$ number of samples.
- Approximating the value function helps.
- The classifier has a crucial role.

- **Bertsekas features:**

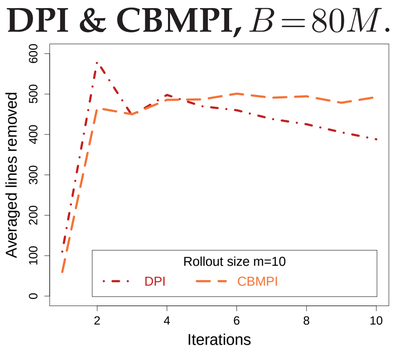
Cross-entropy (CE).



λ-PI.



DPI & CBMPI, $B = 80M$.



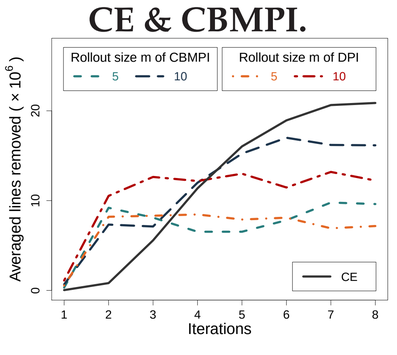
- Bertsekas features do not perform well in Tetris.
- CBMPI's classifier was unstable.

Large Board (10×20):

- **Dellacherie-Thiéry features:**

- CBMPI has similar performance as CE with 6 times less samples!

CE & CBMPI.



λ-PI.