AI Tetris - A CS3243 Project (Group 21)

Lau Kar Rui (A0155936U), Tan Chee Wee (A0155400W),
Poh Jie (A0158523A), Matilda (A0178867A)

1 Introduction

We create an utility-based Tetris-playing AI agent with utility weights derived through *Particle Swarm Optimization* (PSO) to maximise the number of cleared rows in a game of Tetris.

2 Utility Function

With every new piece, the agent determines the best possible state derivable from the piece and plays it, by simulating all possible states of the board the piece legally can reach (via its orientations and positions). The best state s will have the highest utility function value $F(s)$, defined as

$$F(s) = \sum_{i=1}^{n} w_i f_i(s),$$

with each feature $f_i \in Features$ mapped to a weight w_i, where $i = 1...n$, $n =$ number of features. Each f_i derives a real value from a game state s.

3 Features Used

Table 3.1 describes the features f_i used to calculate $w_i \in Weights$ when training the agent with the PSO algorithm described in Section 4.1.

Feature	**Description**
RowsCleared	Number of rows cleared
MaxHeightIncrease	Maximum height increase among all columns
AvgHeightIncrease	Average height increase among all columns
AbsoluteDiff	Absolute height difference between all columns
NumHoles	Number of holes; a hole is defined as an empty cell with a non-empty cell above it
ColumnTransition	Number of column transitions; defined as empty cells adjacent to filled cells in the same column or vice versa
RowTransition	Number of row transitions; defined as empty cells adjacent to filled cells on the same row or vice versa
WellSum	Number of empty cells above each columns' top heights which are adjacent to filled cells on both sides

Table 3.1: Features used

4 Implementation

4.1 Particle Swarm Optimization (PSO)

The PSO method is used with a population (a.k.a. swarm) of candidate solutions (a.k.a. particles), which in the search-space of F moves to find the best possible positions (a.k.a. $w_i \in Weights$) for $F(s)$.

The movement of each particle $\vec{x}$ is governed by the *velocity* $\vec{v}$, which is derived from each particle $\vec{x}$'s *personal best position* $\vec{p}$, and the *swarm's best position* $\vec{g}$ through the formulae below, where ω refers to the inertia, ϕ_p, ϕ_g the social and cognitive parameter, and r_p, r_g two random values $\in [0...1]$

$$\vec{v} \leftarrow \omega\vec{v} + \phi_p r_p(\vec{p} - \vec{x}) + \phi_g r_g(\vec{g} - \vec{x})$$
$$\vec{x} \leftarrow \vec{x} + \vec{v}$$

An *inertia* ω of lower magnitude allows the swarm to converge faster to a solution (optimality not guaranteed) while the opposite encourages exploring the search-space. The *velocity* of each particle is re-calculated after every iteration (where the Tetris game is played with $\vec{x}$ being the weights of the features) and serves as the swarm's willingness to move in the search-space.

Values for the constants (swarm size $N, \omega, \phi_p, \phi_g$) were first chosen based on values stated in [1] and later optimized with the Meta Optimization algorithm discussed in Section 4.2. Details of full algorithm is further elaborated on in Appendix A.1.

4.2 Meta Optimization of PSO

While literature exists on finding the ideal parameters of PSO [2], these parameters generally do not have a specific problem in mind. Hence, we wanted to investigate if there exist better parameters for our TetrisAI problem. We turn to the Local Unimodal Sampling (LUS) algorithm [3] and its use in the Meta Optimization (MO) Problem [4].

Details of the LUS algorithm can be found in Appendix A.2, but a brief of our usage of it is as follows: random values are generated for the 4 constant parameters (swarm size N, inertia ω, cognitive ϕ_p and social parameters ϕ_g). The PSO is then run using these parameters for 100 iterations. If the fitness value of PSO using this set of parameters is better than the best fitness value obtained so far, we do not reduce the search range for the parameters; else we do. In either, new parameters are generated.

The algorithm ran for 10 iterations, each iteration consisting of 100 PSO iterations. The best weights obtained so far was used as the best score seemed to be stuck at around 1 million rows cleared using only the PSO algorithm to see if we could get better results using other parameters.

The 5 best sets of parameters (details available in Appendix A.1) derived from LUS were compared with 5 sets of the parameters from [1] by running 10 sets of PSO, for up to 300 iterations, and comparing the performance of the LUS parameters against the set of parameters from [1]. Each run of PSO started with no learned weights. The parameters attained from MO are referred to as *MO1, MO2*, etc, in the order they are displayed in Appendix A.1. Similarly, the parameters suggested in [1] are referred to as *Suggested*.

To reduce the impact of outliers skewing the data, the mean of the performance of *Suggested* was calculated for every 50 iterations. The performance of the PSO run was also recorded using MO parameters for every 50 iterations.

	1	**50**	**100**	**150**	**200**	**250**	**300**
MO1	7887	462311	528863	528863	833335	833335	1736058
MO2	0	25967	48899	532314	532314	532314	532413
MO3	0	10222	111219	171810	324310	401939	407827
MO4	0	63688	203806	419134	419134	419134	419134
MO5	29	500	500	500	500	500	500
meanSuggested	81.8	572057.6	711666.6	785112.8	825297.4	830466.8	830466.8

Table 4.1: Rows cleared every 50 iterations

From Table 4.1, it is clear that the PSO run using *MO1*'s parameters performed the best, while other MO parameters did not perform as well. In particular, the PSO run using *MO1* managed to clear 1.7 million rows, approximately twice that of *meanSuggested*.

Hence, it is clear that *MO1* parameters are distinctly better for finding the optimal weights for PSO. It is further suggested that for this particular problem, the *MO1* parameters can be refined by running the LUS algorithm starting with *MO1*'s set of parameters, thus likely achieving a higher score.

5 Scaling PSO For Big Data: Multi-threading

Multi-threading was implemented for each particle via the *Java Callable* class and each thread was run concurrently. This made it possible to run the algorithm on the NUS SoC Compute Cluster, which contains 2x8 physical cores at max frequency of 3.00 GHz. Table 5.1 shows the difference when the algorithm was run iteratively (no multi-threading) compared to when multi-threading was implemented.

	Time for 20 iterations (in seconds)
No multithreading	15364.734
Multithreading each Particle	3308.58

Table 5.1: Time comparison

Both versions were run on the same node as two separate tasks on the Compute Cluster. From the result, the multi-threaded version of the algorithm performed **4.64x** faster than its single-threaded counterpart, demonstrating the effectiveness of parallelization to speed up the application.

6 Training Results

After 1000 iterations of training as described in Sections 4.1 and 4.2, the weights obtained were:

$$\begin{pmatrix} MaxHeightIncrease \\ RowsCleared \\ AvgHeightIncrease \\ NumHoles \\ ColumnTransition \\ AbsoluteDiff \\ RowTransition \\ WellSum \end{pmatrix} \Longleftarrow \begin{pmatrix} -5.1948140083947793 \\ 5.53478180043909, \\ -2.1920993360428604 \\ -5.9838083427614395 \\ -12.880189747449215 \\ -1.3227027198368309 \\ -3.7823922425956837 \\ -2.2176395274310137 \end{pmatrix}$$

7 Results

The results of 100 runs using the weights obtained in Section 6 can be seen in Fig. 7.1, and the statistical findings in Table 7.1.

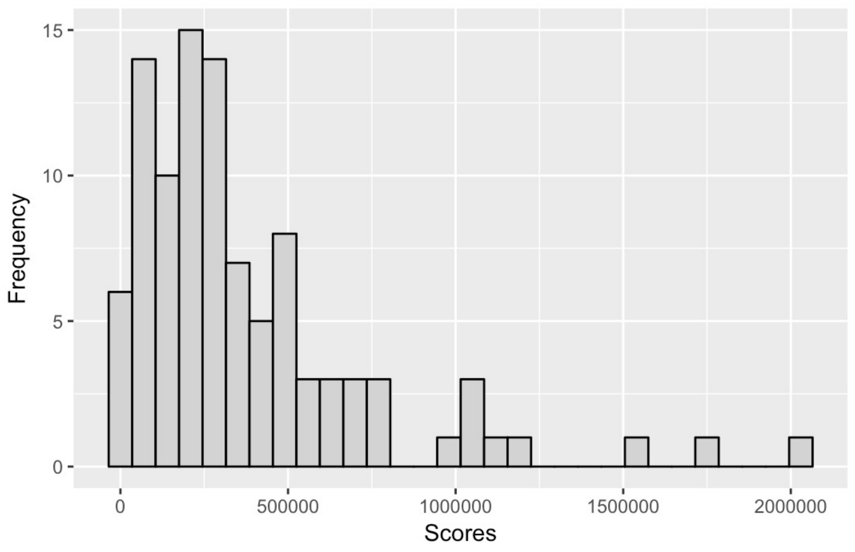


Figure 7.1: Graph of the results of 100 runs

Games Played	**Mean**	**Median**	**Std. Dev.**	**Min**	**Max**
100	375,299.22	287,109.5	366,742.303	6,667	**2,042,624**

Table 7.1: Statistics of 100 runs

As seen in Figure 7.1, there is a significant leftwards skewness of the scores, where on average the agent clears 375k lines, and its best cleared 2 million lines. There is also a significantly large standard deviation of 367k, meaning that the agent's performance is highly variable.

8 Conclusion

In conclusion, Particle Swarm Optimization is a good algorithm to apply to this problem, given that it converges to a relatively good position in a small number of iterations.

A successful parallelization was employed during the training process, with each particle being played in a separate thread, achieving a speedup of 4.64x.

However, a bad feature of PSO is that it may be stuck in a local maxima relatively quickly if the parameters ω, ϕ_p, ϕ_g and *swarm size* were set too low or too aggressively. With this in mind, Meta Optimization was applied to find better parameter values, which were employed to hopefully get better positions for the particles $\vec{x}$. In the end a high score of **2,042,624** was achieved (from the old high score of 1,414,172 before parameters were tuned).

In addition, it is also important to note that this high score is not very high - others have managed to reach scores of more than 10 million [5]. This may be becausethe implementation is missing out on some crucial features or the implementation of PSO may have caused the agent to be stuck in a local maxima.

A Appendix

A.1 Particle Swarm Optimization (PSO) algorithm

```
For each particle $\vec{x_i}$ for $i = 1...N$, where $N$ is the number of particles in the swarm,
    Initialize $\vec{x_i}$'s position $\vec{p}$ with a uniformly distributed random vector: $\vec{x_i} \sim U(\vec{b_{lo}}, \vec{b_{up}})$
    Initialize the particle's best known position $pBest$ to its initial position: $pBest \leftarrow \vec{x_i}$
    If $globalBest > pBest$, update swarm's best positions $\vec{g} \leftarrow \vec{p_i}$
While termination criterion not met (for this case number of iterations),
    Play game with particle's position $\vec{p_i}$, obtain score $s_i$
    If $s_i > pBest$ update: $pBest \leftarrow s_i, \vec{p_i} \leftarrow \vec{x_i}$
    If $s_i > gBest$ update: $gBest \leftarrow s_i, \vec{g} \leftarrow \vec{p_i}$
    Update $\vec{x_i}$
```

The movement of each particle $\vec{x_i}$ is guided by the formulae

$$\vec{v} \leftarrow \omega\vec{v} + \phi_p r_p(\vec{p} - \vec{x}) + \phi_g r_g(\vec{g} - \vec{x})$$
$$\vec{x} \leftarrow \vec{x} + \vec{v}$$

where $\vec{p}$ is the particle's *personal best position* and $\vec{g}$ is the swarm's *global best position*, ϕ_p and ϕ_g being the *social* and *cognitive coefficient* respectively, emulating the swarm's willingness to move in the search-space.

ϕ_p acts as the particle's "memory", causing it to return to its individual best regions of the search space, while ϕ_g guides the particle to move to the localized search-space where the global best position was discovered. r_p and r_g are two random real values $\in [0..1]$ generated every assignment to $\vec{v}$.

ω refers to the *inertia* of the particle, and keeps the particle moving in the same direction it was originally heading. A lower value speeds up convergence of the swarm in the search space, and a higher value encourages exploring the search-space.

The values b_{lo} and b_{up} are respectively the lower and upper boundaries of the search-space. The parameters ω, ϕ_p, and ϕ_g are selected using the values stated in [1]

A.2 Local Unimodal Sampling (LUS) algorithm

- Initialise $\vec{x}$ to a random solution in the search space:
  $$\vec{x} \sim U(\vec{b_{lo}}, \vec{b_{up}})$$
  Where $\vec{b_{lo}}$ and $\vec{b_{up}}$ are the search-space boundaries.
- Set the initial sampling space $\vec{d}$ to cover the entire search-space:
  $$\vec{d} \leftarrow \vec{b_{up}} - \vec{b_{lo}}$$
- Until a termination criterion is met, repeat the following:
  - Pick a random vector $\vec{a} \sim U(-\vec{d}, \vec{d})$
  - Add this to the current solution $\vec{x}$, to create the new potential solution $\vec{y}$

– If $(f(\vec{y}) < f(\vec{x})$ then update the solution:

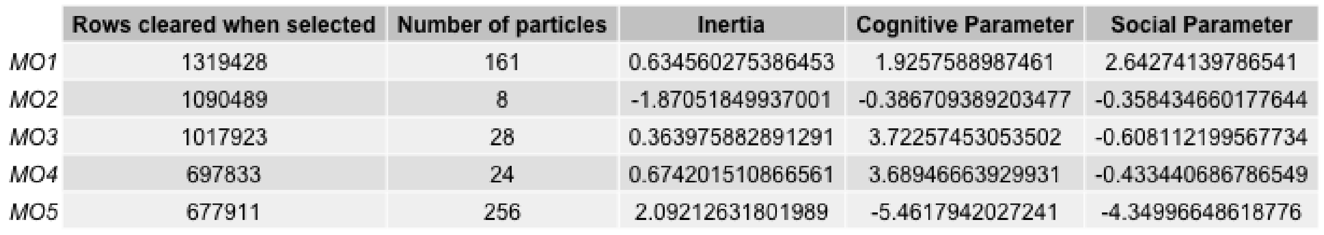
$$\vec{x} \leftarrow \vec{y}$$

Otherwise decrease the search-range by multiplication with the factor q, defined as $2^{-\beta/n}$, where β is arbitrarily defined and n is the number of times the search-range has been reduced.

$$\vec{d} \leftarrow q \cdot \vec{d}$$

Note that $f : \mathbb{R}^n \to \mathbb{R}$ is the maximum number of rows cleared by PSO algorithm using $\vec{x}$.

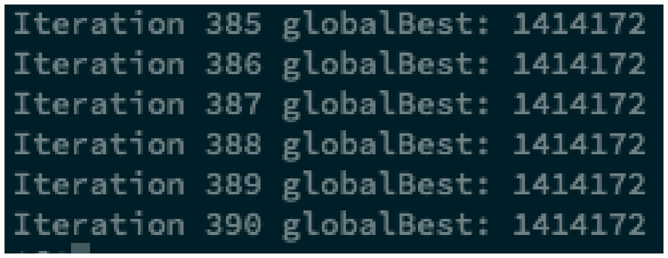
A.3 Parameters obtained from LUS Algorithm



	Rows cleared when selected	Number of particles	Inertia	Cognitive Parameter	Social Parameter
MO1	1319428	161	0.634560275386453	1.9257588987461	2.64274139786541
MO2	1090489	8	-1.87051849937001	-0.386709389203477	-0.358434660177644
MO3	1017923	28	0.363975882891291	3.72257453053502	-0.608112199567734
MO4	697833	24	0.674201510866561	3.68946663929931	-0.433440686786549
MO5	677911	256	2.09212631801989	-5.4617942027241	-4.34996648618776

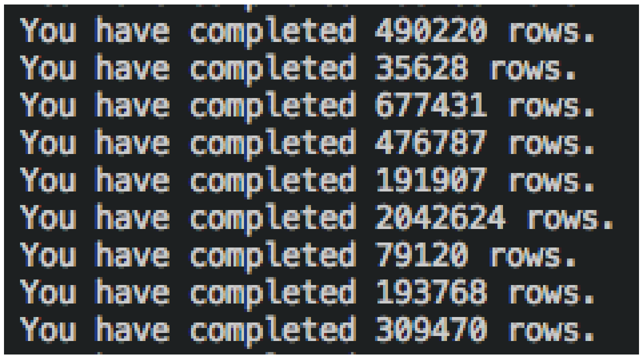
Figure A.1: Parameters information

A.4 Various training results



```
Iteration 385 globalBest: 1414172
Iteration 386 globalBest: 1414172
Iteration 387 globalBest: 1414172
Iteration 388 globalBest: 1414172
Iteration 389 globalBest: 1414172
Iteration 390 globalBest: 1414172
```

Figure A.2: Highscore of 1,414,172 before MO



```
You have completed 490220 rows.
You have completed 35628 rows.
You have completed 677431 rows.
You have completed 476787 rows.
You have completed 191907 rows.
You have completed 2042624 rows.
You have completed 79120 rows.
You have completed 193768 rows.
You have completed 309470 rows.
```

Figure A.3: Highscore of 2,042,624 after MO

References

[1] F Van Den Bergh and A P Engelbrecht. “A study of particle swarm optimization particle trajectories”. In: *Information Sciences* 176 (2006), pp. 937–971. DOI: `10.1016/j.ins.2005.02.003`. URL: `https://pdfs.semanticscholar.org/270d/24da13010c775f3b09b9fc53b2e612fc97aa.pdf`.

[2] Yuhui Shi and Russell C. Eberhart. “Parameter selection in particle swarm optimization”. In: Springer, Berlin, Heidelberg, 1998, pp. 591–600. DOI: `10.1007/BFb0040810`. URL: `http://link.springer.com/10.1007/BFb0040810`.

[3] Magnus Erik, Hvass Pedersen, and Andrew John Chipperfield. “Local Unimodal Sampling”. In: (). URL: `http://www.hvass-labs.org/people/magnus/publications/pedersen08local.pdf`.

[4] M.E.H. Pedersen and A.J. Chipperfield. “Simplifying Particle Swarm Optimization”. In: *Applied Soft Computing* 10.2 (Mar. 2010), pp. 618–628. ISSN: 15684946. DOI: `10.1016/j.asoc.2009.08.029`. URL: `http://linkinghub.elsevier.com/retrieve/pii/S1568494609001549`.

[5] Christophe Thiery and Bruno Scherrer. “Improvements on learning Tetris with the Cross-Entropy method NOTES IMPROVEMENTS ON LEARNING TETRIS WITH CROSS-ENTROPY”. In: (). URL: `https://hal.inria.fr/inria-00418930/file/article.pdf`.