CAPSTONE PROJECT 1
Planning Document

Evaluation of Nature-inspired Optimisation Algorithms in Solving Versus Tetris

by

Yap Wei Xiang
21067939

Bachelor of Science (Honours) in Computer Science

Supervisor: Dr Richard Wong Teck Ken

Semester: April 2024
Date:

Department of Computing and Information Systems
School of Engineering and Technology
Sunway University

Abstract

Contents

List of Tables

List of Figures

1 Introduction

Tetris is a popular video game created in 1984 by computer programmer Alexey Pajitnov [1]. It is a puzzle game that requires players to strategically place sequences of pieces known as "Tetriminos" into a rectangular Matrix (refer to Figure 1.1). In the classic game, players attempt to clear as many lines as possible by completely filling horizontal rows of blocks, but if the Tetriminos surpass the top of the Matrix, the game ends.

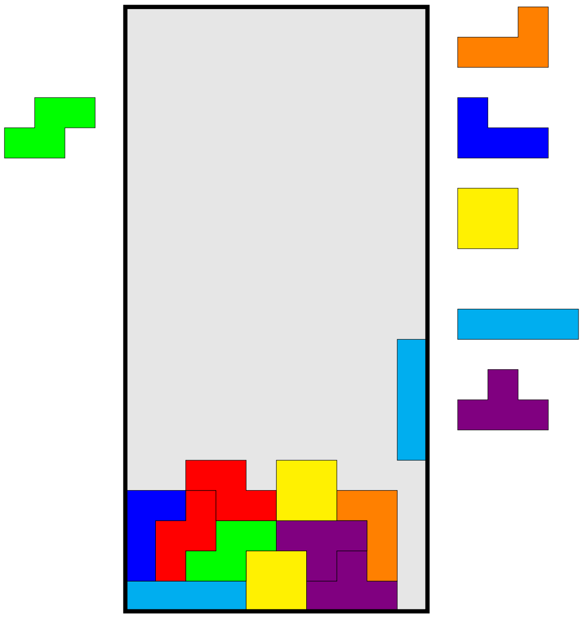

Figure 1.1: A typical modern Tetris game where four lines are about to be cleared. The Tetrimino on the left of the matrix is the *Hold* piece and the pieces to the right of the matrix are collectively known as the *Queue*.

Since its release, mathematicians and computer scientists have been intrigued by the game of Tetris, leading to a diverse array of research endeavours exploring the various facets of the game, including its computational complexity [2], and its possibility of being won [3] [4].

1.1 Motivation

In their paper, Demaine, Hohenberger, and Liben-Nowell showed that it is NP-complete to optimise several natural objective functions of Tetris [2]. NP-completeness poses a significant challenge in computational problem-solving, as it denotes the absence of polynomial-time algorithms for efficient solutions [5]. Moreover, the discovery of a polynomial-time algorithm for any NP-complete problem implies that any problem in the set of NP, encompassing efficiently verifiable but potentially difficult problems, could

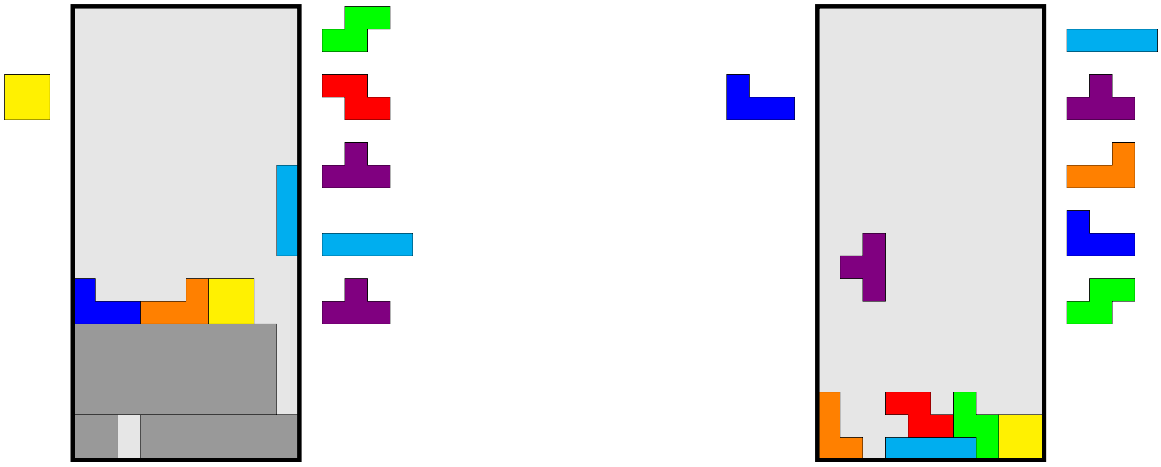

Figure 1.2: A typical game of Versus Tetris. Both players are trying to send lines to each other. The grey blocks are *Garbage Lines* sent from Player 2 (right) to Player 1 (left).

be solved in polynomial time [5]. NP-completeness extends beyond Tetris, with real-life instances of NP-complete arising in diverse fields such as route optimisation [6], job scheduling [7], and medicine [8].

To address these challenges, researchers have explored alternative approaches to tackle NP-complete problems, including the use of nature-inspired algorithms [9]. Although they might fail at finding optimal solutions, nature-inspired algorithms are able to return acceptable solutions in shorter running times [10]. In the context of optimising Tetris gameplay, studies have shown the effectiveness of using nature-inspired algorithms in playing the classic single-player game [11] [12]. However, there remains limited research on the effectiveness of nature-inspired optimisation algorithms in the multiplayer versus variant of the game.

1.2 Problem Statement

Versus Tetris (refer to Figure 1.2) presents a unique challenge in computational gaming due to its complex dynamics and real-time competitive nature. While previous research regarding the use of nature-inspired algorithms for Tetris optimisation have focused on single-player scenarios, the effectiveness of these algorithms in the multiplayer context remains largely unexplored. Despite the demonstrated success of these algorithms in improving single-player Tetris gameplay, their application to the multiplayer variant poses distinct challenges due to a different rule set and differing objectives that require further investigation.

1.3 Aim

The aim of this capstone project is to assess the effectiveness of nature-inspired optimisation algorithms in solving the game of Versus Tetris. By integrating insights from nature-inspired algorithms, the project seeks to create a robust and adaptable Tetris-playing software capable of competing against human players or other Tetris-playing programs. Through this endeavour, the project aims to contribute valuable insights into the application of nature-inspired algorithms in addressing computationally complex problems.

1.4 Objectives

The objectives of this project are as follows:

1. Formulate the problem of Versus Tetris for game AI.
2. Research and implement a variety of nature-inspired optimisation algorithms to determine their suitability for optimising gameplay strategies in Versus Tetris.
3. Design a comprehensive framework for objectively evaluating and comparing the performance of the algorithms.
4. Develop a playable game of Tetris that simulates gameplay and training.
5. Using the game, do comparative analyses with the designed framework to assess the effectiveness and efficiency of each algorithm.
6. Summarize findings from the comparative analyses.
7. Share the software with Tetris players of varying aptitudes to find the level of play for each algorithm.

1.5 Project Scope

This project will focus specifically on the evaluation of nature-inspired optimisation algorithms in the context of multiplayer versus Tetris. It will entail the development of a playable Tetris game capable of simulating gameplay and the training of algorithms. This simulation environment will facilitate in the analysis and evaluation of these algorithms' performances. The scope includes the exploring of a range of nature-inspired algorithms to address the unique challenges inherent in Versus Tetris.

2 Literature Review

2.1 The Difficulty of Tetris

In their article, Demaine, Hohenberger, and Liben-Nowell [2] proved that optimising several natural objectives of Tetris is NP-complete, even with a deterministic finite piece sequence. A deterministic finite piece sequence refers to a game at which the player knows every single piece in the sequence, and where there are finite pieces in the sequence, i.e. the game can end without the player losing. The authors defined the natural objectives of the game as follows [2]:

1. maximising the number of rows cleared;
2. maximising the number of piece placed;
3. maximising the number of Tetrises - clearing four lines on the same move;
4. minimising the height of the highest filled grid square.

In 2020, Asif, Coulombe, Demaine, *et al.* [13] demonstrated that playing any game of Tetris with a matrix of eight or more columns, or four or more rows, is NP-complete, further showcasing the difficulty of the game . Both of these papers aim to highlight the difficulty of the game, but what does difficulty actually entail?

This section aims to elucidate some of the key concepts of computational complexity, which involves the study of intrinsic difficulties of computational problems [14], and attempt to justify the use of non-traditional algorithmic approaches to play the game.

2.1.1 Complexity Classes

The study of computational complexity asks questions about the intrinsic difficulty of computational problems [14]. Complexity classes are usually defined by referring to computation models and by putting suitable restrictions on them [15].

The class P encompasses all decision problems that are polynomial time solvable using a deterministic model of computation [16]. In this model, for any given input, the machine's computation follows a single predetermined path [5].

The complexity class NP, on the other hand is the class of all decision problems that can be solved in polynomial time by a nondeterministic algorithm [17]. The nondeterministic model of computation allows for guessing correct solutions out of polynomially many options in constant time [18], and solutions can be verified in deterministic polynomial

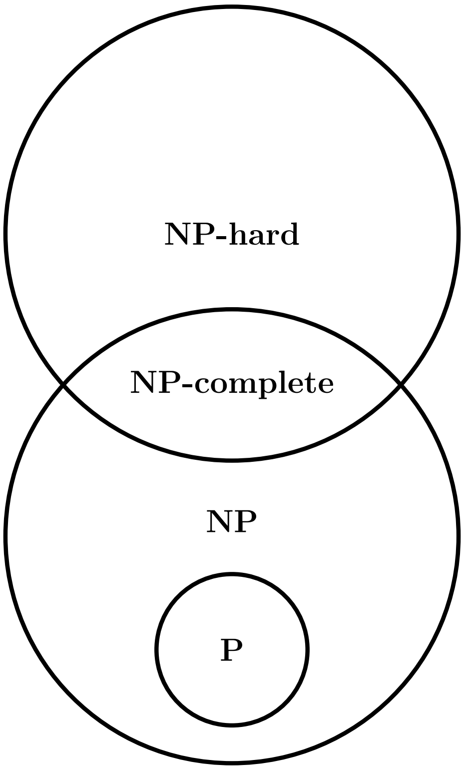

Figure 2.1: Visualisation of the sets P, NP, NP-hard and NP-complete.

time [5].

If all problems in *NP* can be reduced to some problem X, X is said to be *NP-hard* [5]. Reductions are useful in showing the relationship between computable problems, as a reduction from problem A to problem B tells us that problem B is at least as hard as problem A [18].

For a problem to be considered *NP-complete*, it must be a member of both the *NP* and *NP-hard* classes (refer to Figure 2.1) [18]. Therefore, *NP-complete* problems are some of the hardest problems in *NP* [16].

Since all problems in *NP* can be reduced to any *NP-complete* problem, if a deterministic polynomial time algorithm can be found for an *NP-complete* problem, it would also mean that the set *NP* is polynomial-time solvable, proving *NP* and *P* are equal sets. The question of whether $P = NP$ has famously been immortalised as one of the millennium prize problems from the Clay Institute of Mathematics [19]. Most researchers believe that $P \neq NP$ since years of effort have failed to yield efficient algorithms for *NP-complete* problems [16].

2.1.2 Justifying Non-traditional Algorithmic Approaches

It is important to note that even though Demaine, Hohenberger, and Liben-Nowell's [2] proof is based on a variant of Tetris that is quite different from Versus Tetris, the difficulty of Versus Tetris can be implied from the proof. Versus Tetris introduces additional gameplay factors, such as the competitive aspect where players can send "garbage" lines to their opponents, adding layers of complexity. As such, Versus Tetris is arguably more difficult than its classic variant, or Demaine, Hohenberger, and Liben-Nowell's variant [2].

Given the NP-completeness of Tetris [2] [13], it is evident that traditional deterministic algorithmic approaches would face significant challenges in efficiently solving the game [14]. The NP-completeness of the game implies that obtaining exact solutions are computationally intractable within a reasonable time frame.

To address this, we may relax the requirements for a solution. This can be done by either broadening the set of acceptable solutions or focusing on average-case scenarios instead of worst-case scenarios [14]. This shift in focus allows the use of more practical, non-traditional algorithms that provide acceptable solutions within a reasonable time, even if they are not optimal.

2.2 Some Approaches Taken to Solve Tetris

Now that the difficulty of Tetris has been established, we turn our attention to potential strategies that can be used to handle the game's complexity. In this section, we explore some of the innovative approaches that have been utilised to tackle the game of Tetris.

2.2.1 Learning by Imitation

In 2010, Zhang, Cai, and Nebel [20] employed a learning by imitation approach to the game of Tetris. Imitation learning aims to mimic human behaviour in some given task by learning a mapping between observations of demonstrations [21].

In their approach, played Tetris games were fed into a machine learning model as inputs, where the model would receive positive feedback when a decision matched that of the imitated system, and negative feedback otherwise [20]. The learning is deemed successful if the trained model maintains similarity even when faced with data outside of the training set.

The learning system consists of several components, including filters, pattern calculators, and support vector machines (SVMs) (refer to Figure 2.2) [20]. Because of the time consuming nature of training SVMs, each Tetrimino had its own dedicated filters,

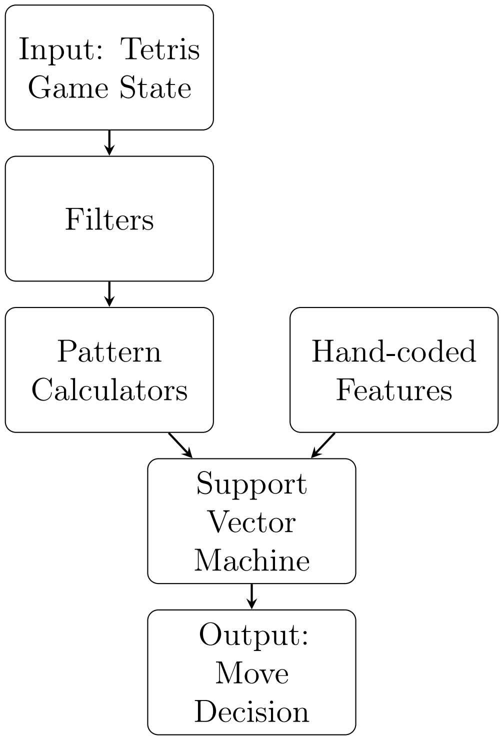

Figure 2.2: Abstraction of Zhang, Cai, and Nebel's System Components

pattern calculators and SVMs, i.e. seven SVMs [1] are working together in the artificial player. The output of any particular SVM describes how similar a candidate move is to the choice of the imitated player.

Filters are made up of a multitude of patterns, which can be thought of as specific configurations in the playing field that the model uses to recognise and evaluate different game states [20]. Patterns do not take into account the whole matrix, but only a small area in the playing field. Thus, the authors included a list of 19 hand-coded features that takes a more global look at the game, some notable features include the number of holes that will be created after the current placement, and the number of removed lines after the current placement. Both the activated patterns and the features are passed into the support vector machine, and the move that is calculated to be the most similar is played.

To evaluate their approach, Zhang, Cai, and Nebel used a human player as well as Fahey's artificial player [22] as systems for their model to imitate [20]. The outcome shows that their model can learn different styles of gameplay based on the imitated player, which is to be expected. When pitting the two learned models against each other in their variant of multiplayer Tetris [2], they found that the human imitated machine performed better than the one imitated from Fahey's AI. However, The system imitated from Fahey's AI

[1] Seven SVMs for the seven Tetriminos.

[2] In Zhang, Cai, and Nebel's variant of multiplayer Tetris, clearing n lines sends $n - 1$ attack rows to the opponent, with each attack row containing $n - 1$ empty cells [20]. Attack rows are added to the bottom of the opponent's matrix.

outperformed that of the human imitated system in single player games.

2.2.2 Upper Confidence Bounds for Trees

Monte-Carlo Tree Search (MCTS) is a best-first search method that is based on a randomised exploration of the search space [23]. The algorithm evaluates states to determine the most rewarding actions, with rewards being calculated from the leaf to the root [24]. MCTS has the added benefit of learning from previous explorations, gradually building a game tree in memory, resulting in more accurate estimation of the most promising moves [23]. In 2006, Kocsis and Szepesvári [25] proposed a method, which was deemed Upper Confidence Bound for Trees (UCT), to improve the performance of the algorithm by applying a bandit algorithm known as UCB1[3] to Monte-Carlo Planning.

The UCB1 formula is employed to balance the exploration-exploitation trade-offs during the search [24]. It helps prioritise actions that are either promising or under-explored, which effectively prunes less promising branches of the search tree. This results in a more efficient search process and a smaller, more focused tree.

In 2011, Cai, Zhang, and Nebel [24] employed this approach to the game of Tetris. In their approach, the authors considered each state of the matrix, along with a given piece as a single node in the planning tree (refer to Figure 2.3). After an action is taken, if any number of rows are cleared, a predefined reward is given to the action and the most rewarding action is selected as the resulting move. To further improve efficiency, the authors pruned the search tree further, removing actions that resulted in disadvantageous holes in the field.

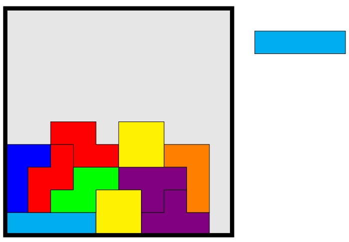

Figure 2.3: An example of a single node of the matrix state and piece in the planning tree.

Experimental results show that the artificial player shows promise, beating Fahey's AI [22] 91 times out of 100 games using the multiplayer Tetris rules mentioned in subsection 2.2.1 [24]. The results also show that the artificial player outperformed the SVM based pattern player from Cai, Zhang, and Nebel's previous work, where they approached the game with imitation learning [20].

[3]UCB stands for Upper Confidence Bound

2.2.3 Reinforcement Learning

In 2016, Stevens and Pradhan [26] adopted a deep reinforcement approach to the game of Tetris, using a convolutional neural network to approximate a Q function, which reflects the maximum predicted discounted sum of future rewards possible by following a given policy from states to actions (refer to Equation 2.1) .

$$Q^*(s, a) = \max \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi] \tag{2.1}$$

Where s_t is the state at time t, a_t is the action taken at time t, π is a policy function indicating what action to take in the current state, and is followed at every step from $t+1$ onwards, r_t is the reward obtained by taking action a_t at state s_t, and γ is a discount factor [26].

The authors defined a state as an image of the current game screen [26]. Actions were defined in two ways, single actions, where a single input used to move a piece is considered an action, and grouped actions, where an entire sequence of actions is considered that takes a piece from its initial point at the top of the matrix to the bottom. They also used two different types of rewards, the first being the scores given from the emulator used, which was defined as the number of lines cleared. The second type of reward was defined as a heuristic function proposed by Lee [27] (refer to Equation 2.2).

$$-0.51 \times Height + 0.76 \times Lines - 0.36 \times Holes - 0.18 \times Bumpiness \tag{2.2}$$

This heuristic takes into account the sum of the heights of every column, the number of cleared lines, the number of holes in the matrix, and the "bumpiness" - sum of the absolute differences in height between adjacent columns [27]. Interestingly, the weights of the heuristic were calculated by using a genetic algorithm.

The training process was further improved by using an epsilon-greedy policy, where the agent takes the best known move most of the time, but has a small probability ϵ to take a random action [26]. They also tried a novel approach where an agent that was already trained to play well could suggest moves to the algorithm. In this approach, a random action is selected with probability ϵ_{rand}, an action suggested by the transferred agent is selected with probability ϵ_{agent}, and the best known action to the machine is selected otherwise.

Other than that, a method known as experience replay was used to create a memory for the neural network to train on. This memory consisted of every "experience" that

Figure 2.4: Z and S pieces.

the network had encountered, and was sampled when training the network. Prioritised sweeping was also utilised, where the priorities were placed on experiences which had the greatest error, and therefore the most room to improve.

From the experimental results, one of the agents showed the most potential. This agent utilised a standard greedy-epsilon policy - no trained agent was used to suggest moves, grouped actions over single ones, heuristic rewards and random sampling over prioritised sweeping. It outperformed the other agents proposed in this paper, with an average score of 18. However, its capabilities were far inferior to that of Lee's bot, which had an average score of 200.

Later in 2021, Chen [28] used a modified version of Stevens and Pradhan's algorithm as a baseline, and proposed several different approaches taken to further optimise the algorithm. The variant of the game used to test the author's algorithms has similar rules to that of modern Tetris games, where the system will generate bags containing a piece sequence with all 7 pieces, and bags are given to the player in sequence. This piece generation system ensures that the player will not need to experience a piece drought, since a piece is guaranteed to be given in the current or next bag. The game also has a hold feature, where the player has the option to hold on to a piece for future use, and swap it out for the piece in play.

Chen proposed that the quadratic rewards from the emulator scores in Stevens and Pradhan's paper might encourage the agent to take riskier moves, since clearing multiple lines would greatly increase the rewards given. However, this resulted in shorter survival times, as the agent waits for particular pieces. Knowing this, the author proposed utilising a linear reward function instead of a quadratic one. Comparing the results of both approaches, the authors found that the agent trained with a linear reward function outperformed that of a quadratic one, in both the survival time aspect, and the game score aspect. However, it should be noted that the quadratic models have higher points per piece dropped, meaning that they were much more efficient than the linear ones.

Chen also proposed the use of a "harder" Tetris game, which had a higher probability for Z and S pieces (refer to Figure 2.4). The author set the probability of each piece as shown in Table 2.1. Doing this significantly decreased training time, reducing it from 3 days to a mere 6 hours.

Finally, Chen proposed an update rule that takes the probability of each piece

Table 2.1: Probability of each piece in Chen's "hard" mode.

	Z	S	I	J	L	O	T
Probability	0.25	0.25	0.1	0.1	0.1	0.1	0.1

into account (refer to Equation 2.3). This idea emerged from the observation that the only randomness found in a Tetris game is from piece generation. The training process involved enumerating all possible next pieces and actions to find the best action, storing the expected reward and Q-value for each state to a replay buffer and randomly sampling from the replay buffer to update the Q-network. Result of this change showed that the models trained under these update rules performed consistently better than the original models.

$$Q(s) \leftarrow Q(s) + \alpha[\mathbb{E}_{allPossiblePieces}[r + \gamma \max Q(s') - Q(s)]] \tag{2.3}$$

Experimental results show that Stevens and Pradhan's model [26] scored an average of 704.68 with modern rules in place, but Chen's model significantly outperformed it, with an average score of 40163.58, increasing almost 57-fold.

2.2.4 Meta-heuristic Algorithms

According to Yang [29], the term "meta-heuristic" does not have any agreed upon definitions, with some scholars even using the term interchangeably with the word "heuristic". Even so, most definitions agree on the same things. Almufti, Shaban, Ali, *et al.* [30] consolidated some of the properties that characterise most meta-heuristics as follows:

1. Meta-heuristics are strategies that guide the search process.
2. The goal is to efficiently explore the search space in order to find near-optimal solutions.
3. Meta-heuristic algorithm techniques range from simple local search procedures to complex learning processes.
4. Meta-heuristic algorithms are non-deterministic and approximate in nature.
5. Meta-heuristic algorithms are not designed to solve a specific problem.

In this subsection, we will look into two works of literature that cover the use of meta-heuristic algorithms in optimising Tetris agents.

Harmony Search Algorithm

The harmony search algorithm is a meta-heuristic algorithm that mimics the improvisation of music players [31].

Most Valuable Player

2.3 Playing Tetris with Nature-inspired Algorithms

3 Technical Plan

3.1 Defining the Rules

3.2 Algorithm Selection

3.3

4 Work Plan

5 References

[1] Tetris Inc., *About Tetris*, `https://tetris.com/about-us`, [accessed Apr. 22, 2024].

[2] E. D. Demaine, S. Hohenberger, and D. Liben-Nowell, "Tetris is hard, even to approximate," in *Computing and Combinatorics*, T. Warnow and B. Zhu, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 351–363, ISBN: 978-3-540-45071-9.

[3] J. Brzustowski, "Can you win at tetris?" Master's Thesis, University of Waterloo, 200 University Ave W, Waterloo, ON N2L 3G1, Canada, 1988.

[4] H. Burgiel, "How to lose at tetris," *The Mathematical Gazette*, vol. 81, no. 491, pp. 194–200, 1997. DOI: `10.2307/3619195`.

[5] M. Sipser, in *Introduction to the Theory of Computation*, Cengage Learning, 2013.

[6] V. Lesch, M. König, S. Kounev, A. Stein, and C. Krupitzer, "A case study of vehicle route optimization," *CoRR*, vol. abs/2111.09087, 2021.

[7] J. D. Ullman, "Np-complete scheduling problems," *Journal of Computer and System sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[8] J. Arle and K. Carlson, "Medical diagnosis and treatment is np-complete," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 33, pp. 1–16, Mar. 2020. DOI: `10.1080/0952813X.2020.1737581`.

[9] L. Davis, "Job shop scheduling with genetic algorithms," in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, 1985, pp. 136–140.

[10] W. Korani and M. Mouhoub, "Review on nature-inspired algorithms," *Operations Research Forum*, vol. 2, Jul. 2021. DOI: `10.1007/s43069-021-00068-x`.

[11] J. Lewis, "Playing tetris with genetic algorithms," 2015. [Online]. Available: `https://api.semanticscholar.org/CorpusID:17416568`.

[12] L. Langenhoven, W. S. van Heerden, and A. P. Engelbrecht, "Swarm tetris: Applying particle swarm optimization to tetris," in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8. DOI: `10.1109/CEC.2010.5586033`.

[13] S. Asif, M. Coulombe, E. Demaine, *et al.*, "Tetris is np-hard even with o(1) rows or columns," *Journal of Information Processing*, vol. 28, pp. 942–958, Jan. 2020. DOI: `10.2197/ipsjjip.28.942`.

[14] O. Goldreich, "Computational complexity: A conceptual perspective," *SIGACT News*, vol. 39, no. 3, pp. 35–39, Sep. 2008, ISSN: 0163-5700. DOI: 10.1145/1412700.1412710. [Online]. Available: https://doi.org/10.1145/1412700.1412710.

[15] D. P. Bovet, P. Crescenzi, and R. Silvestri, "A uniform approach to define complexity classes," *Theoretical Computer Science*, vol. 104, no. 2, pp. 263–283, 1992.

[16] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009, ISBN: 9780521424264.

[17] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, First Edition. W. H. Freeman, 1979, ISBN: 0716710455.

[18] MIT OpenCourseWare. "Np-completeness." 6.046J: Design and Analysis on Algorithms, Massachusetts Institute of Technology. (2015), [Online]. Available: https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/resources/mit6_046js15_lec16/.

[19] *The Millennium Prize Problems - Clay Mathematics Institute — claymath.org*, https://www.claymath.org/millennium-problems/, [Accessed 15-05-2024].

[20] D. Zhang, Z. Cai, and B. Nebel, "Playing tetris using learning by imitation," 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:10668262.

[21] A. Hussein, M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," English, *ACM Computing Surveys*, vol. 50, no. 2, Apr. 2017, ISSN: 0360-0300. DOI: 10.1145/3071073.

[22] *Tetris AI*, [accessed Jun. 23, 2024], 2003. [Online]. Available: http://www.colinfahey.com/tetris/.

[23] G. Chaslot, "Monte-carlo tree search," English, Ph.D. dissertation, Maastricht University, 2010, ISBN: 9789085590996. DOI: 10.26481/dis.20100930gc.

[24] Z. Cai, D. Zhang, and B. Nebel, "Playing tetris using bandit-based monte-carlo planning," Jan. 2011.

[25] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293, ISBN: 978-3-540-46056-5.

[26] M. Stevens and S. Pradhan, "Playing tetris with deep reinforcement learning," *Convolutional Neural Networks for Visual Recognition CS23, Stanford Univ., Stanford, CA, USA, Tech. Rep*, 2016.

[27] Y. Lee, *Tetris AI - The (Near) Perfect Bot*, https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/, [accessed Jul. 17, 2024].

[28] Z. Chen, "Playing tetris with deep reinforcement learning," Master's Thesis, University of Illinois at Urbana-Champaign, 2021.

[29] X.-S. Yang, *Nature-inspired optimization algorithms*. Academic Press, 2020.

[30] S. Almufti, A. Shaban, R. Ali, and J. Fuente, "Overview of metaheuristic algorithms," *Polaris Global Journal of Scholarly Research and Trends*, vol. 2, pp. 10–32, Apr. 2023. DOI: 10.58429/pgjsrt.v2n2a144.

[31] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony search," *simulation*, vol. 76, no. 2, pp. 60–68, 2001.