Utilizing Heuristic Approach to Tackle NP-Hard Complexity in Tetris

Berto Richardo Togatorop - 13522118
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): bertotogatorop27@gmail.com

***Abstract*— Tetris, a widely popular puzzle game, presents a computational challenge classified as NP-hard, making it a formidable problem for exact algorithms. This paper explores the application of heuristic algorithms to efficiently manage the complexity inherent in Tetris. We propose and implement various heuristic strategies aimed at optimizing gameplay by minimizing gaps and maximizing completed rows. The effectiveness of these heuristics is evaluated through extensive simulations and performance metrics, demonstrating significant improvements in decision-making speed and game performance. The result suggest that heuristic approaches offer a practical solution to handling the computational demands of Tetris, providing insights that can be extended to other NP-hard problems in combinatorial optimization.**

Keywords—Tetris, Heuristic Approach, NP-hard Problems, Computational Complexity, Greedy Algorithm, Branch and Bound

I. INTRODUCTION (*HEADING 1*)

Tetris, a globally popular puzzle game, presents significant computational challenges due to its NP-hard classification. The game's complexity comes from the need to place randomly shaped tetrominoes in an optimal way to clear lines and prevent the stack from reaching the top of the playing field. With the large number of possible game states and the exponential growth of the search space with each new piece, solving Tetris optimally in real-time is computationally infeasible with exact algorithms.

To tackle this problem, heuristic approaches such as greedy algorithms and branch and bound methods can be used. Greedy algorithms make locally optimal decisions at each step, aiming for immediate benefits. These algorithms are efficient and easy to implement but may not always lead to the best overall solution because they focus only on short-term gains.

Branch and bound algorithms, on the other hand, explore all possible solutions more thoroughly by systematically branching out possible moves and using bounds to prune branches that are unlikely to lead to an optimal solution. This method balances the need to explore different possibilities with the need to find good solutions quickly.

In this paper, I explore the use of both greedy algorithms and branch and bound methods to address the NP-hard complexity of Tetris. I propose and implement various heuristic strategies to improve the performance of these algorithms, focusing on minimizing gaps, reducing the height of columns, and maximizing the number of lines cleared. The effectiveness of these strategies is evaluated through extensive simulations, demonstrating their strengths and weaknesses in handling the computational demands of Tetris.

By combining the strengths of greedy algorithms and branch and bound techniques, this paper aims to offer practical solutions for the Tetris problem.

II. FUNDAMENTAL THEOREM

A. Heuristic

In problem-solving, a heuristic is a technique or strategy that uses practical methods to find solutions, especially when faced with complex or difficult problems. Heuristics are often employed in situations where finding an optimal solution is computationally infeasible within a reasonable amount of time. In the context of Tetris, heuristics guide the decision-making process by providing rules of thumb to optimize gameplay. These rules prioritize certain actions, such as minimizing gaps, reducing the height of the stack, or maximizing the number of lines cleared, without guaranteeing an optimal solution. This heuristic approach will affect the behavior of the algorithm using it.

B. Greedy Algorithm

Greedy algorithms are a class of heuristic algorithms that make locally optimal choices at each step with the hope of finding a global optimum.

The Greedy Algorithm uses several elements to determine its steps:

1. Candidate set, C: Contains options that can be chosen at each step.
2. Solution set, S: Contains options that have been chosen.
3. Solution function: Determines whether the chosen option provides a solution.

4. Selection function: Selects options based on a specific strategy.
5. Feasibility function: Checks whether the chosen option is suitable for inclusion in the solution set.
6. Objective function: Maximizes or minimizes the outcome.

At each iteration, the solution found is the best at the local level. In the end, if any, the global solution will be found.

The greedy algorithm in Tetris uses heuristics to determine the selection function to make locally optimal decisions at each step. These decisions are influenced by heuristic functions that prioritize immediate benefits, such as clearing the most lines or minimizing the height of the tallest column. While the greedy algorithm is efficient and straightforward, its reliance on heuristics may lead to suboptimal solutions in the long run.

C. *Branch and Bound Algorithm*

Branch and bound is an algorithmic technique used for solving combinatorial optimization problems. This method systematically explores the solution space by branching into subproblems and using bounds to eliminate suboptimal branches.

The Branch and Bound Algorithm (BnB) is a versatile algorithm used to solve optimization problems. These problems involve finding the minimum or maximum value of an objective function while adhering to certain constraints. Typically, problems tackled using the Branch and Bound Algorithm are represented as a graph or tree structure. The process of searching for the optimal value is conducted using a state space tree, where each node carries a specific weight.

In essence, the Branch and Bound Algorithm consists of two main components:

a. The algorithm generates child nodes in the state space tree from the node being examined. Each child node inherits certain variations from its parent node and also carries a weight representing the cost required to reach the child node from the original node. This process is known as branching.

b. The algorithm prunes branches or nodes deemed to no longer lead to the desired solution. This pruning is achieved using a bounding function, hence the term "bounding."

The Branch and Bound Algorithm combines elements from Breadth First Search (BFS) and Least Cost Search. BFS is a search algorithm in the state space tree that generates child nodes in a "breadth-first" manner, exploring all neighboring nodes in a sequential order. BFS uses a queue data structure to store nodes being generated with the First In First Out (FIFO) principle. Least Cost Search involves generating child nodes from a parent node with the smallest weight in the state space tree.

In the Branch and Bound Algorithm, node generation follows specific rules, with the best-first rule being commonly used. Each generated node is assigned a cost or weight, denoted as ĉ(i), representing the estimated cheapest path to the target node through the node i. The next node to be expanded is no longer based on the order of generation but rather the node with the minimum cost (for minimization cases).

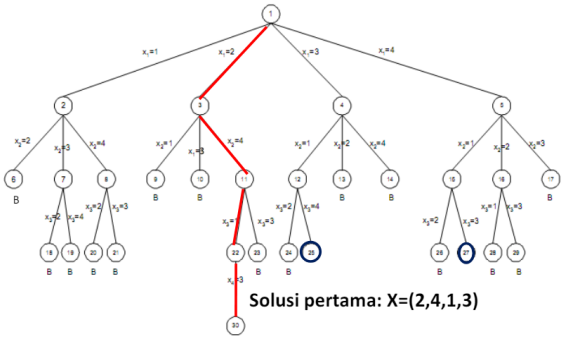

Fig 1.1 Branch and Bound Illustration for N-Queens Problem

Similarly to the greedy algorithm, the branch and bound algorithm can incorporates heuristics to guide its search through the solution space. Heuristic estimates are used to evaluate potential future outcomes and prune suboptimal branches. If a branch's heuristic score is worse than the current best-known solution, that branch is pruned. This method aiming to find better solutions than greedy algorithms alone by considering the long-term impact of each move.

D. *Computational Complexity*

The computational complexity of a problem refers to the amount of computational resources required to solve it. Problems are often classified based on their complexity class, with NP-hard problems being among the most challenging. A problem is considered NP-hard if it is at least as hard as the hardest problems in NP (nondeterministic polynomial time) in terms of computational resources required for their solution. Tetris is classified as NP-hard due to its inherent complexity and the difficulty of finding an optimal solution within a reasonable amount of time using exact algorithms.

E. *Tetris*

Tetris is a classic puzzle game where players manipulate falling tetrominoes—geometric shapes composed of four square blocks each. The goal is to place these tetrominoes in such a way that they form complete horizontal lines, which then disappear, preventing the stack of pieces from reaching the top of the playing field.

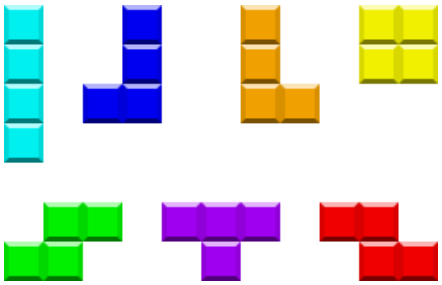

Fig 1.1 Tetriminoes

The tetrimino is as below :

1. I-Tetrimino: Consists of four squares arranged in a straight line.
2. J-Tetrimino: Comprises three squares forming an L-shape with one additional square separated and positioned below the L-shape.
3. L-Tetrimino: Comprises three squares forming an L-shape with one additional square separated and positioned above the L-shape.
4. O-Tetrimino: Comprises four squares arranged in a square shape.
5. S-Tetrimino: Consists of two pairs of squares forming a Z-shape when rotated.
6. T-Tetrimino: Consists of three squares forming a T-shape with one additional square positioned below the T-shape.
7. Z-Tetrimino: Consists of two pairs of squares forming an S-shape when rotated.

The complexity of Tetris arises from the randomness of incoming pieces and the need for strategic placement to maximize line clears and minimize gaps. Tetris is proven to be NP-hard, meaning that finding the optimal sequence of moves for an arbitrary sequence of tetrominoes is computationally infeasible in a reasonable amount of time. This complexity is due to the exponential growth of the search space with each additional piece, making it impossible to solve perfectly in real-time using exact algorithms.

III. Implementation

A. Tetris As NP-hard Problem

In Tetris, the objective is to manipulate a sequence of tetrominoes (Tetris pieces) falling down a grid to form complete rows without gaps. As the game progresses, the speed of falling tetrominoes increases, and players must make quick decisions to place them optimally.

Tetris can be modeled as a combinatorial optimization problem, where the goal is to maximize the number of lines cleared while minimizing the number of moves made. The complexity of Tetris arises from the vast number of possible configurations of tetrominoes and the constraints imposed by the game rules.

Tetris can be proven NP-hard by reducing it to the 3-partition problem he reduction from the 3-partition problem to Tetris involves encoding instances of the 3-partition problem into Tetris configurations.

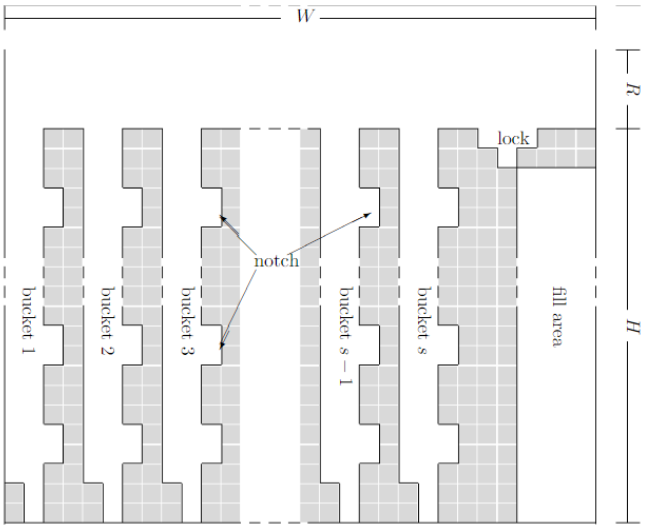

Fig 3.1 The reduced initial state of Tetris board

Its dimension is as follows :

- R is the space needed to rotate and translate the pieces. We consider Rto be big enough to rotate and translate Tetris pieces above the ‘buckets’ and therefore Ris of no consequence to the reduction.
- W is the width of the game board and is equal to 4s+ 6.
- H is the height of the bottom part of the game board that needs to be cleared and is equal to 5T+ 1

Tetris game board can be construct with special features that correspond to the properties of the 3-partition problem. By simulating the gameplay of Tetris on this specially constructed board, it is shown that finding an optimal solution for Tetris is at least as hard as solving the 3-partition problem.

The complexity of Tetris stems from the following factors:

1. The exponential number of possible configurations of tetrominoes.
2. The constraints imposed by the Tetris game rules, such as the requirement to form complete rows without gaps.
3. The need to make decisions quickly as the game progresses, adding a time dimension to the problem.

Due to its NP-hardness, finding an optimal solution to Tetris is computationally challenging and often requires heuristic algorithms or approximation methods to find near-optimal solutions efficiently.

B, Greedy Algorithm

Tetromino placement problem in the game of Tetris into the framework of a greedy algorithm defined as follows:

1. Candidate Set (C): This set contains all possible positions available on the game board to place the Tetromino at each step.
2. Solution Set (S): This set consists of the potential solutions from Tetromino placements. Each solution is a complete arrangement of Tetrominoes that cannot be placed further.
3. Solution Function: A function that determines whether a Tetromino arrangement on the game board is the final solution or not. The solution is achieved when there are no more Tetrominoes that can be placed on the game board.
4. Selection Function: At each step, this function selects the best position to place the Tetromino based on certain criteria, such as trying to cover gaps or approaching the upper boundary.
5. Feasibility Function: This function checks if the selected position to place the Tetromino meets certain conditions, such as not causing unfilled gaps or exceeding the upper boundary of the game board.
6. Objective Function: This function aims to maximize the score gained.

With this, we can propose several greedy solutions. The solution is based on heuristic approaches. Every solution has its own drawbacks:

1. Height-based Placement: The greedy algorithm can choose positions to place the Tetromino based on the highest block height in the associated column. This aims to keep the game board relatively flat and minimize the likelihood of creating gaps. But, this solution ignore the combo score that can give much more big result.

2. Gap-based Placement: The algorithm can also select positions that have the potential to cover existing gaps on the game board. This way, we can reduce the likelihood of creating hard-to-fill gaps in subsequent steps.

3. Tetromino Priority Placement: The algorithm can prioritize placing certain Tetrominoes first, such as those that are more difficult to place or have more advantageous shapes.

With various possible greedy solutions, combinations of the above approaches or other variations can be used to develop an efficient Tetromino placement algorithm in the game of Tetris.

C. Branch and Bound Algorithm

Representation of Basic Tetris Gameplay System in Branch and Bound Algorithm

In the context of the Branch and Bound Algorithm, a set of characteristics is outlined as follows:

1. Live Node Identification (Simpul Hidup Sebagai Simpul-E): A live node refers to a node with the lowest cost value (least cost search).
2. Cost Evaluation for Each Node: The evaluation of cost for each node considers specific factors:
    - Lines cleared
    - Number of holes
    - Maximum height
    - Average height

In the Tetromino positioning challenge, lines cleared are maximized while the rest is minimized. Considering maximum height is really needed to keep low to keep the game and number of holes is not easy to clear, those factor have the most weight. Average weight sometimes is not good but sometimes needed to make combo. Same with lines cleared. Therefore, the estimated cost value for node iii is defined as:

$\hat{c}(i)$ = -0.5 * lines cleared + 1 * number of holes + 1 * maximum height + 0.5 * average height

Temporary Lowest Cost Valu**e** (c_{min}): The temporary lowest cost value encountered thus far is represented as $c_{min.}$

Bounding Function (B(i)): The bounding function at each node iii is specified as follows: B(i)= maximum block height $\leq$ 20.

Solution Node Identification: A solution node is identified post the placement of all available Tetrominos.

To implement the Branch and Bound Algorithm for Tetromino placement optimization, the following steps are undertaken:

1. Queue Initialization and Initial Node Insertion: Creation of a priority queue, such as Q, with the root node representing the initial Tetromino. If there's only one Tetromino, solution (goal node) attainment is concluded.

2. Queue Processing and Node Expansion: Looping through the queue Q until exhaustion. Selection of the node iii with the smallest estimated cost value $c\hat{}(i)$ from queue Q. If the queue is empty, designate the node with cost value c_{min} as the goal node.

3. Node Evaluation and Bounding Function Scrutiny: Evaluation of whether node iii is a solution node, along with scrutiny of the bounding function. If B(i) is true, mark node persistence as the temporary goal node while terminating other nodes with estimated cost value $\hat{c}(i) > c_{min}$

Through these steps, we indirectly create the problem status space tree for Tetromino placement using the branch and bound algorithm. An example is represented below :

IV. Conclusion

This paper explored the application of heuristic algorithms to address the NP-hard complexity inherent in the game of Tetris. By leveraging heuristic approaches such as greedy algorithms and branch and bound methods, we aimed to efficiently manage the computational demands of Tetris gameplay.

Through the implementation of various heuristic strategies focused on minimizing gaps and maximizing completed rows, we evaluated the effectiveness of these approaches through extensive simulations and performance metrics. Our findings demonstrated significant improvements in decision-making speed and game performance, highlighting the practical utility of heuristic algorithms in handling the computational complexity of Tetris.

The results suggest that heuristic approaches offer a practical solution for managing the computational demands of Tetris gameplay, providing insights that can be extended to other NP-hard problems in combinatorial optimization. By combining insights from computational complexity theory with heuristic algorithm design, we bridge theoretical concepts with practical applications, paving the way for advancements in both computer science and gaming.

In conclusion, the utilization of heuristic approaches, including greedy algorithms and branch and bound methods, represents a promising avenue for addressing the NP-hard complexity of Tetris and similar combinatorial optimization problems. Moving forward, further research and development in heuristic algorithms hold the potential to unlock new strategies and insights for enhancing gameplay experiences and tackling computational challenges in gaming and beyond.

V. Recommendations

The author recommend the creation of more sophisticated heuristic algorithms, focusing on the development of hybrid approaches that combine multiple heuristic techniques. Through such an approach, we believe that the efficiency and performance of Tetris gameplay can be significantly enhanced. Additionally, we suggest implementing heuristic algorithms in real-time Tetris game scenarios to unveil deeper insights into the effectiveness of heuristic algorithms in facing dynamic and unpredictable gaming environments. Furthermore, an analysis of user experience could provide a better understanding of the impact of heuristic-based optimization on player enjoyment and satisfaction. Further study of alternative heuristic criteria, such as Tetromino shape complexity or future piece prediction, is also advised to identify additional opportunities for enhancing Tetris gameplay optimization. Lastly, we encourage extending this research to other puzzle games or problem-solving domains with NP-hard complexity, which could broaden the scope of study and provide new insights into heuristic optimization techniques.

ACKNOWLEDGMENT

The author extends heartfelt gratitude to the Almighty for His grace and blessings, without which the completion of this paper would not have been possible. The author acknowledges that divine guidance and blessings were essential for achieving the desired outcomes of this endeavor.

Furthermore, the author expresses deepest appreciation to their beloved parents for their unwavering support throughout the process of writing this paper. Their moral, material, and spiritual encouragement have been invaluable and deeply cherished.

The author also wishes to convey profound gratitude to the instructors of the IF2211 course and other academic mentors for their guidance, knowledge, and direction throughout the academic journey. Their expertise and insights have significantly contributed to the development and understanding of this paper.

Lastly, the author thanks all individuals, friends, and colleagues who have directly or indirectly supported this work through discussions, feedback, or encouragement. Their contributions have been instrumental and greatly appreciated.

REFERENCES

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] R. Munir, “Algoritma Greedy Bagian 1,” IF2211 Strategi Algoritma. Retrieved:May 30, 2024, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

[2] R. Munir, “Algoritma Greedy Bagian 2,” IF2211 Strategi Algoritma. Retrieved:May 30, 2024, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

[3] R. Munir, “Algoritma Greedy Bagian 3,” IF2211 Strategi Algoritma. Retrieved:May 30, 2024, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf

[4] R. Munir, “Algoritma Branch and Bound Bagian 1,” IF2211 Strategi Algoritma. Retrieved:May 30, 2024, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

[5] R. Munir, “Algoritma Branch and Bound Bagian 2,” IF2211 Strategi Algoritma. Retrieved:May 30, 2024, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

[6] R. Munir, “Algoritma Branch and Bound Bagian 3,” IF2211 Strategi Algoritma. Retrieved:May 30, 2024, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf

[7] Demaine, Erik D.; Hohenberger, Susan; Liben-Nowell, David (July 25–28, 2003). Tetris is Hard, Even to Approximate.

[8] R. Breukelaar, H. J. Hoogeboom, and W. A. Kosters. Tetris is hard, made easy. Technical report, Leiden Institute of Advanced Computer Science, 2003.

[9] J. Brzustowski. Can you win at Tetris? Master’s thesis, U. British Columbia, 1992

Bandung, 12 Juni 2024

Berto Richardo Togatorop - 13522118

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.