December 2025

Modular Pattern Detection and Structural Analysis in NIM Game Variants

Siddharth Phase

Abstract

This research examines mathematical patterns and artificial intelligence techniques in NIM, a basic game within the area of combinatorial game theory. NIM has simple rules but reveals rich structures based on binary arithmetic and modular repetition. Using a Java-based win table generator, repeated patterns in game play and win/loss outcomes across several NIM variants were identified, looking in particular at when and how these patterns occur in the data.

Similar modular structure of NIM variants was consistent across all variants analyzed; it became increasingly difficult to identify patterns for the longer move parameter strings as more quantity and size of parameters increased. Geometric regularities were created for the lengths of the modular patterns and starting point of these patterns based on a combination of several modular arithmetic properties. By using classic combinatorial game theory with modern computational methods, the results of this study illustrate that games of this type can be used to study how to identify optimality and pattern recognition of an algorithm, and therefore provide a platform to identify the principles of these algorithms

Background & Introduction

Combinatorial game theory is a branch of mathematics that studies competitive games where both players have full knowledge of the game state: no hidden information, no chance, just strategy. One of the most iconic examples in this field is the game of NIM. In its basic form, two players take turns removing objects from one or more piles, and the player forced to make the last move wins (or, in some versions, loses). While the rules are simple, NIM has fascinated mathematicians and computer scientists for over a century because it blends logic, number theory, and strategic reasoning.

The origins of NIM stretch back to ancient China, where a stone-picking game called jiǎn-shízǐ is thought to be its predecessor. European references to NIM-like games appear as early as the 16th century, but the version we know today was formally analyzed in 1901 by Harvard mathematician Charles L. Bouton. Bouton never confirmed where the name "NIM" came from, though it is widely believed to derive from the German verb nimm, meaning "to take", a description of the game's main function.

Bouton's work was groundbreaking. He was the first to provide a complete mathematical solution to NIM, introducing concepts like safe positions (later called P-positions) and unsafe positions (N-positions). Most importantly, he showed that the winning strategy depends on representing pile sizes in binary and computing their NIM-sum (the bitwise XOR of all pile sizes). From this, he concluded that players could always force a win by moving from one winning position to another whenever possible.

This research aims to detect patterns in winning tables- data structures that contain winning positions for players given the certain moves they can make, assuming each player plays perfectly. These winning tables have repeating patterns, with some of the patterns starting at index 0 of the data structures, and some starting later on in the set.

In this analysis, the MOD function is used repeatedly. It just means the remainder of an integer when divided by another integer. For example, 5 mod 2 = 1 because 5/2 = 2 remainder 1. 10 mod 6 = 4 because 10/6 = 1 remainder 4.

But NIM's significance extends far beyond being a simple game. Variants of NIM have been used to model problems in optimization, resource allocation, and coding theory. The mathematical principle of NIM-sums appears in computer science, particularly in the design of algorithms, cryptography, and error-detecting codes. Because of its well-defined structure, NIM has also become a valuable testing ground for reinforcement learning and decision-tree algorithms, making it a useful benchmark in artificial intelligence research. Ultimately, this paper aims to examine NIM from multiple perspectives, including its mathematical foundations, generalizations, and its applications in both theory and practice.

Methods

Win Table Generator and Pattern Detection

A win table generator was programmed in Java using Eclipse IDE. The function works recursively, building off previous winning and losing positions. It takes a parameter that defines how long the win table should be - a length of 500 was used for most of the analysis.

Two key functions were built to detect repeating patterns in the win tables. One function returns the actual pattern that repeats, while the other outputs how long the pattern is and where it starts.

For example, in NIM(1, 3, 4), the win table looks like: [0: L] [1: W] [2: L] [3: W] [4: W] [5: W] [6: W] [7: L] [8: W] [9: L] [10: W] [11: W] [12: W] [13: W] [14: L] [15: W] [16: L] [17: W] [18: W] [19: W] [20: W].

The functions return "The repeating pattern starts on index 0" and "Pattern length = 7, pattern = [L, W, L, W, W, W, W]".

Some cases are more complicated and don't have a pattern starting at index 0. Take NIM(2, 4, 7) with the win table: [0: L] [1: L] [2: W] [3: W] [4: W] [5: W] [6: L] [7: W] [8: W] [9: L] [10: W] [11: W] [12: L] [13: W] [14: W] [15: L] [16: W] [17: W] [18: L] [19: W] [20: W].

Here the functions return "The repeating pattern starts on index 4" and "Pattern length = 3, pattern = [W, W, L]".

These win tables go far past length 20 for deeper analysis.

Computer Simulations and User Games

Computer vs computer simulations and computer vs user games were coded in Eclipse. The computer vs computer simulation includes a parameter p that sets the probability the computer makes a "smart move" when possible. A smart move uses the win tables - basically the computer starts at a winning position and moves to a losing position, forcing its opponent into another winning position when the computer's turn comes back around.

In the computer vs user game, there's also a float p that the user sets, which determines the probability the computer plays a perfect move (or a random one if p isn't satisfied). The user controls all the parameters, including the list of possible moves and starting number of rocks.

Percentage Analysis Methods

Methods called "twoPilePercent", "threePilePercent", "fourPilePercent", and "fivePilePercent" were written to calculate what percentage of all possible move combinations produce a win table with a detectable pattern starting at index 0.

For "twoPilePercent", a for loop runs through all combinations of NIM(a, b) where $a < b$ and $b <= N$. N is a parameter that sets the maximum value of the biggest move. The code tracks all combinations tested and how many had a pattern starting at index 0.

For "fivePilePercent", the code iterates through NIM(a, b, c, d, e) where $a < b < c < d < e <= N$.

Specific data for NIM(1, a, b), NIM(2, a, b), NIM(3, a, b), NIM(4, a, b), NIM(5, a, b), NIM(6, a, b), and NIM(7, a, b) was collected and put in a Google Sheet along with data from the percentage methods.

For analysis and creation of visualizations, the Google Sheet was exported as a csv, uploaded to python and combined with libraries such as matplotlib to make the data more comprehensible.

Results

Pattern Detection in NIM(1, a) Win Tables

Analysis of NIM(1, a) variants revealed highly consistent modular patterns across different maximum move values. For all tested values of the maximum move (a) from N=2 to N=49, 100% of combinations exhibited win tables with repeating patterns (MOD) starting at index 0. This demonstrates that single-parameter variants following the NIM(1, a) structure produce immediately detectable patterns without any offset.

However, at N=50, a deviation occurred: the percentage of combinations with patterns starting at index 0 dropped to 97.959% when analyzing win tables up to length 100. Interestingly, when the win table length was adjusted to 3×(biggest move), the pattern at N=50 returned to 100%. This suggests that the anomaly at N=50 may be due to insufficient win table length rather than a fundamental break in the pattern structure.

Multi-Parameter Pattern Analysis- NIM(1, a, b) Variants

The introduction of a second move parameter altered pattern behavior. As the maximum move value (b) increased from N=3 to N=50, the percentage of combinations with patterns starting at index 0 declined systematically:

Maximum N value	% of combinations starting at index 0
3-8	100
9	96.43
20	88.89
30	81.53
50	58.59

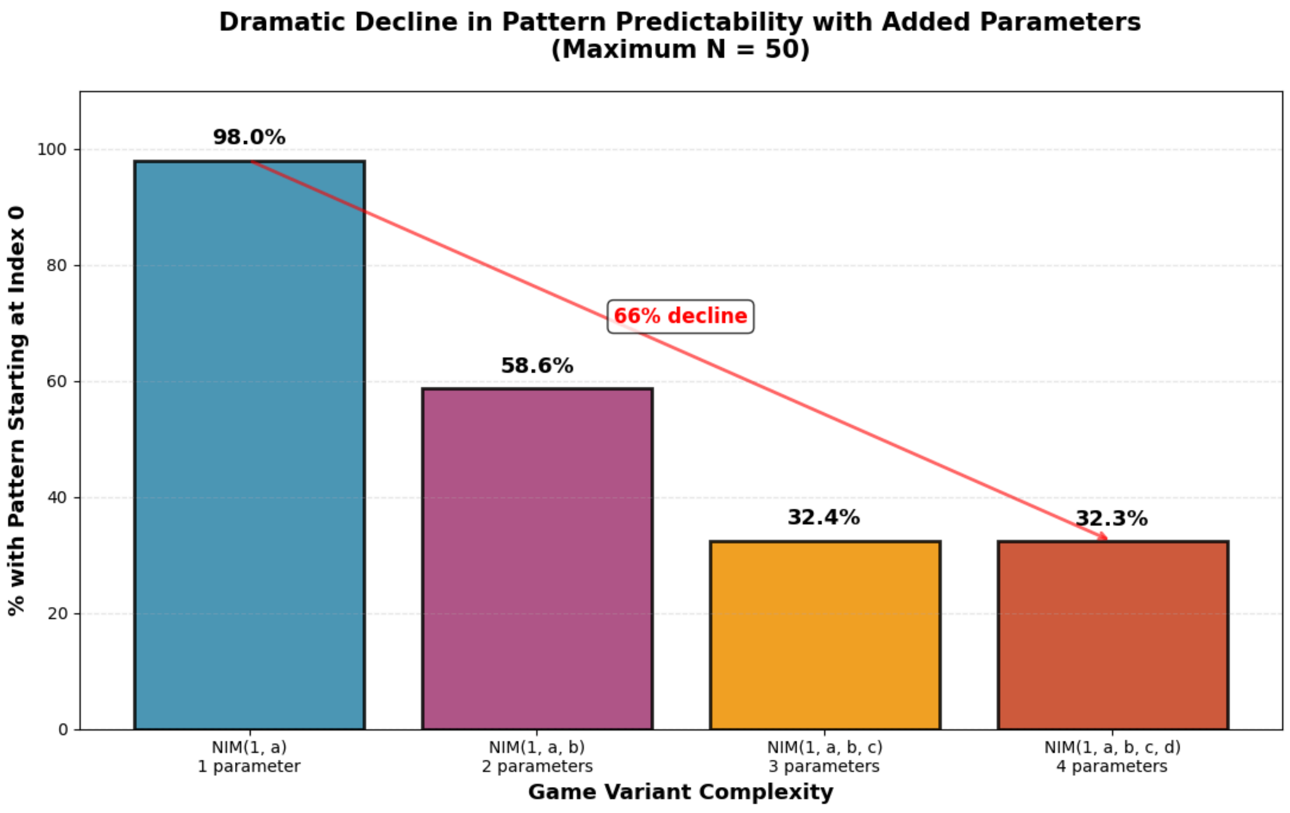
When analyzing shorter win tables (3× the biggest move), the decline was even more pronounced, dropping from 100% at N=7 to 68.55% at N=50. This indicates that increasing the parameter space introduces greater structural complexity, with more combinations producing patterns that begin at non-zero indices.

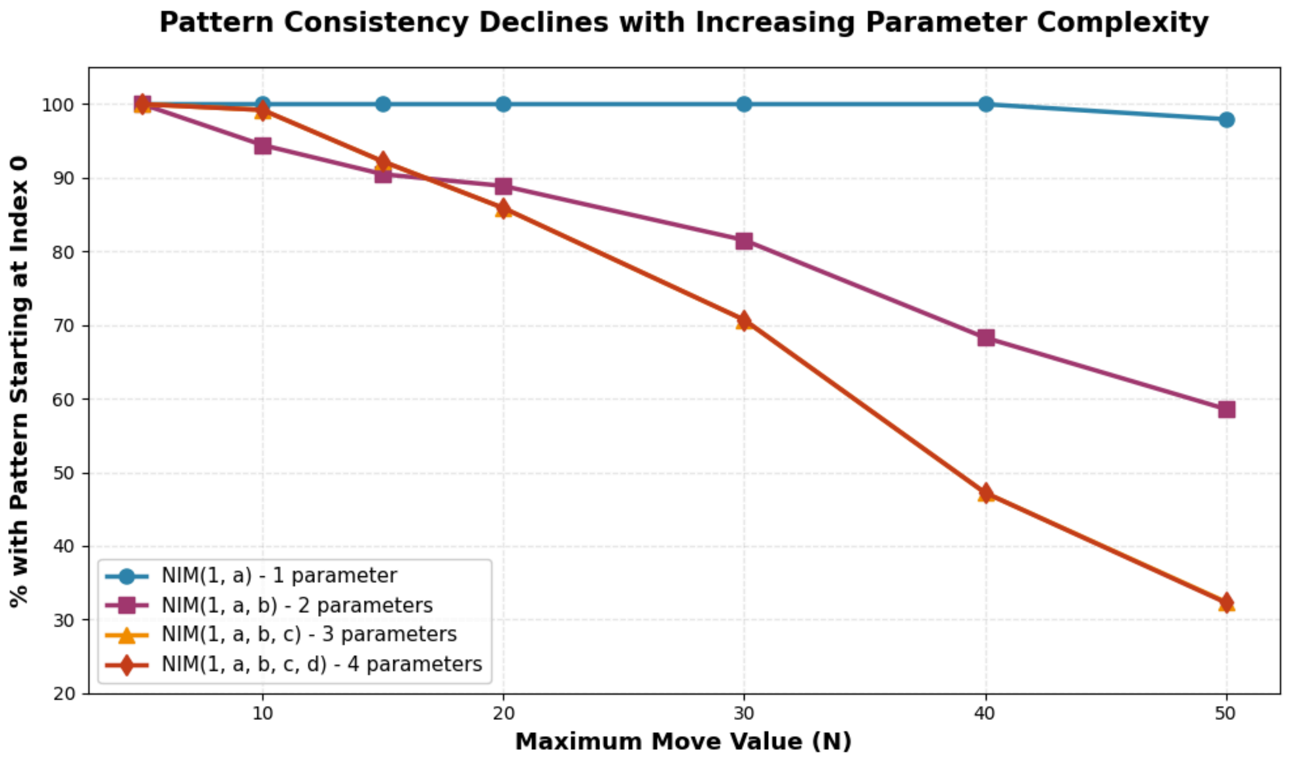
NIM(1, a, b, c) Variants [N indicates maximum c value]

Maximum N value	% of combinations with patterns starting at index 0
5-9	100
10	99.21
30	70.70
50	32.39

NIM(1, a, b, c, d) Variants [N indicates maximum d value]

Maximum N value	% of combinations with patterns starting at index 0
5-8	100
15	92.21
30	70.70
50	32.29


Dramatic Decline in Pattern Predictability with Added Parameters
(Maximum N = 50)
% with Pattern Starting at Index 0
98.0%
58.6%
32.4%
32.3%
66% decline
NIM(1, a)
1 parameter
NIM(1, a, b)
2 parameters
NIM(1, a, b, c)
3 parameters
NIM(1, a, b, c, d)
4 parameters
Game Variant Complexity


Pattern Consistency Declines with Increasing Parameter Complexity
% with Pattern Starting at Index 0
Maximum Move Value (N)
NIM(1, a) - 1 parameter
NIM(1, a, b) - 2 parameters
NIM(1, a, b, c) - 3 parameters
NIM(1, a, b, c, d) - 4 parameters

Examination of specific NIM(1, a, b) combinations revealed periodicity in pattern length:

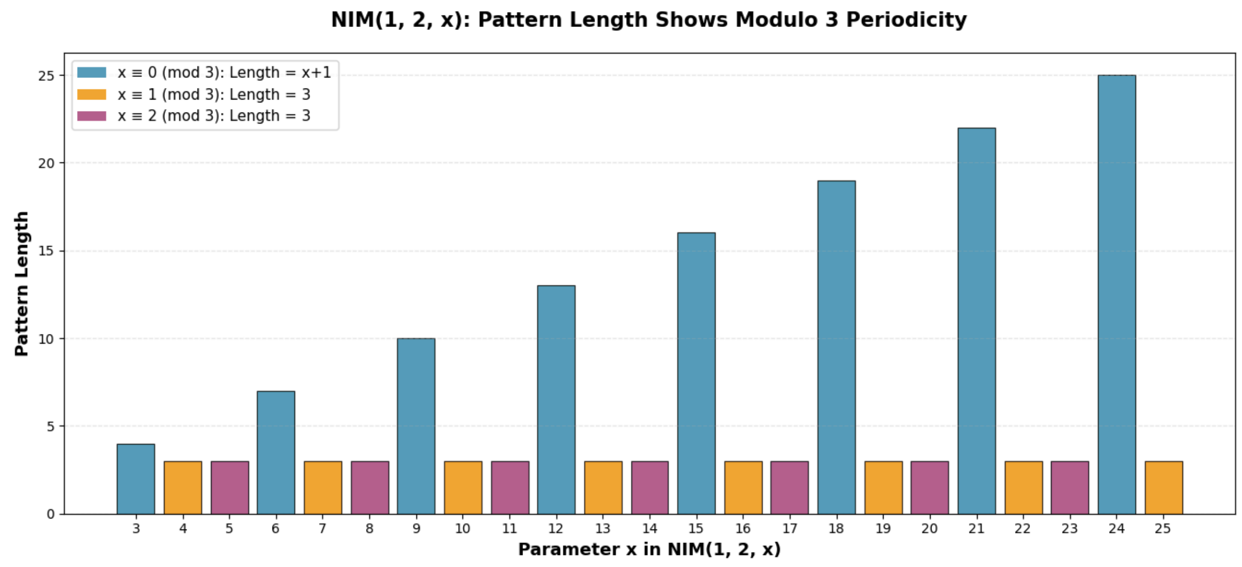
NIM(1, 2, x):

For x = 1 (MOD 3), pattern length consistently equals 3.

For example, NIM(1, 2, 4) has length=3, NIM(1, 2, 7) has length=3, and NIM(1, 2, 10) has length=3.

However, when x = 0 (MOD 3) , length jumped to x+1

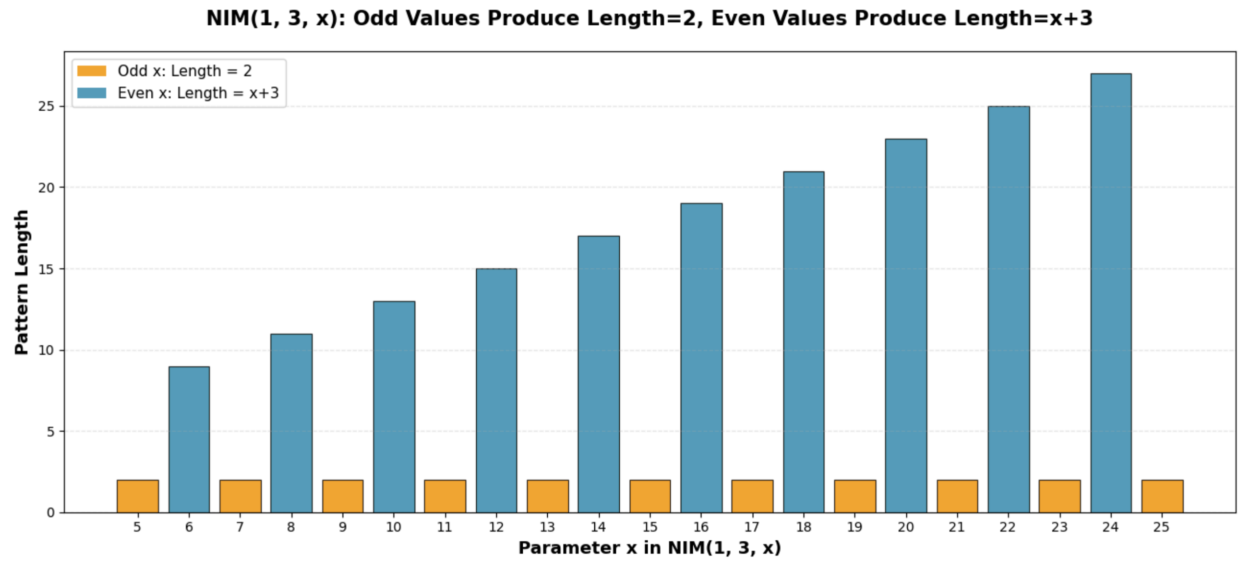
For example, NIM(1, 2, 6) has length=7, NIM(1, 2, 9) has length=10, and NIM(1, 2, 12) has length = 13.



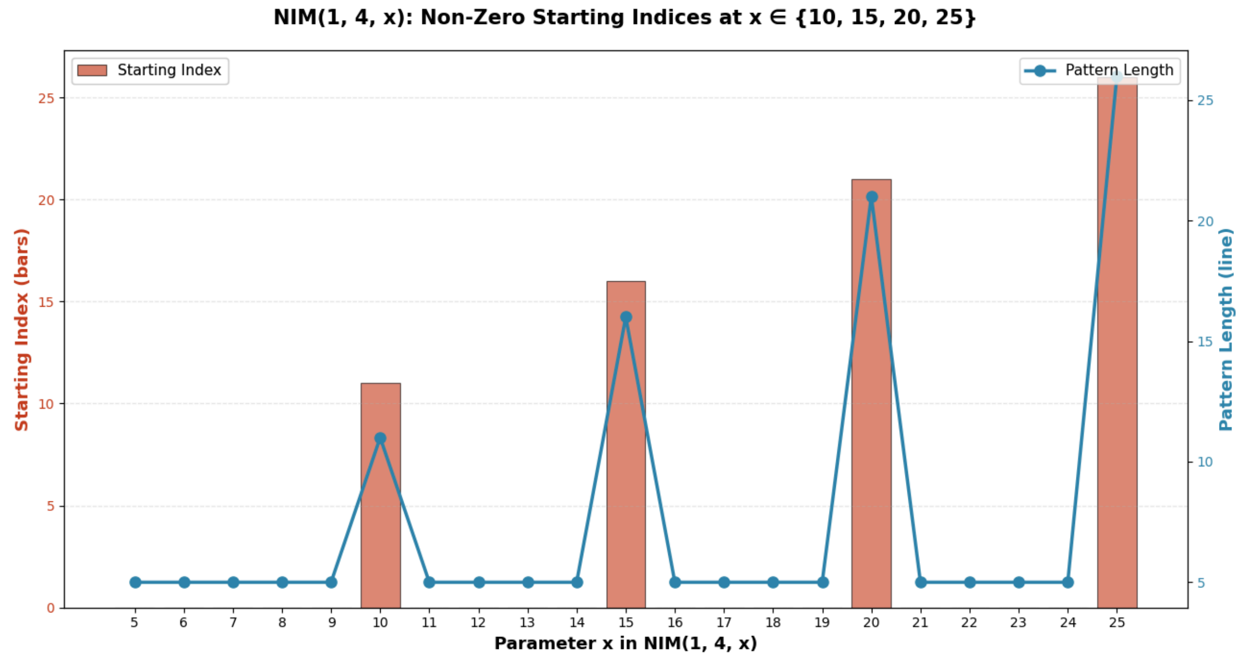
NIM(1, 3, x): Alternating pattern emerged where odd values of x produced length=2, while specific even values produced substantially longer pattern lengths of x+3

For example, NIM(1, 3, 5) through NIM(1, 3, 25) with odd x consistently showed length=2.

NIM(1, 3, 8) produced length=11, NIM(1, 3, 18) produced length=21, and NIM(1, 3, 24) produced length=27.



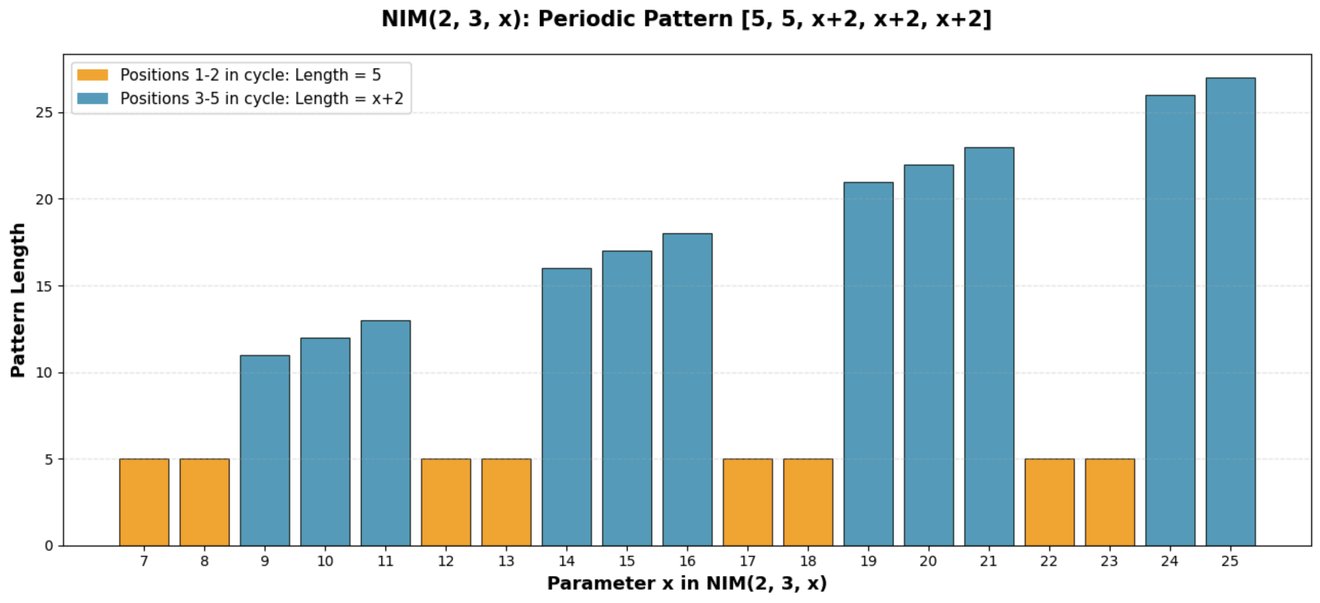
NIM(1, 4, x): More complex behavior with some combinations failing to start at index 0. For example, NIM(1, 4, 10) had a pattern starting at index 11 with pattern length=11. This pattern stayed consistent for x values of 10, 15, 20, and 25. Each of those values had a starting index and pattern length of x+1.



Specific Multi-Parameter Pattern Structures

NIM(2, a, b) variants demonstrated several notable regularities:

NIM(2, 3, x): from NIM(2, 3, 7) to NIM(2, 3, 25) there was a consistent pattern in length. It periodically repeated [5, 5, x+2, x+2, x+2].



NIM(2, 4, x): A distinct pattern appeared where x = 1 (MOD 6) produced patterns starting at index x-3. There were also several instances in which the pattern length was 6, 3 times in a row, followed by the pattern [x+2, x+2, 3]

NIM(2, 6, x): from NIM(2, 6, 7) to NIM(2, 6, 25) there was a repeated pattern in the lengths of [x+6, x+6, x+6, 4]

NIM(3, a, b) and NIM(4, a, b) Analysis

NIM(3, 4, x) showed extreme regularity: pattern length increased sequentially (steps of 1) with x, but exhibited periodic "gaps" where length=7 repeated for multiple consecutive values.

Specifically:

x=10,11: length=7

x=17,18: length=7

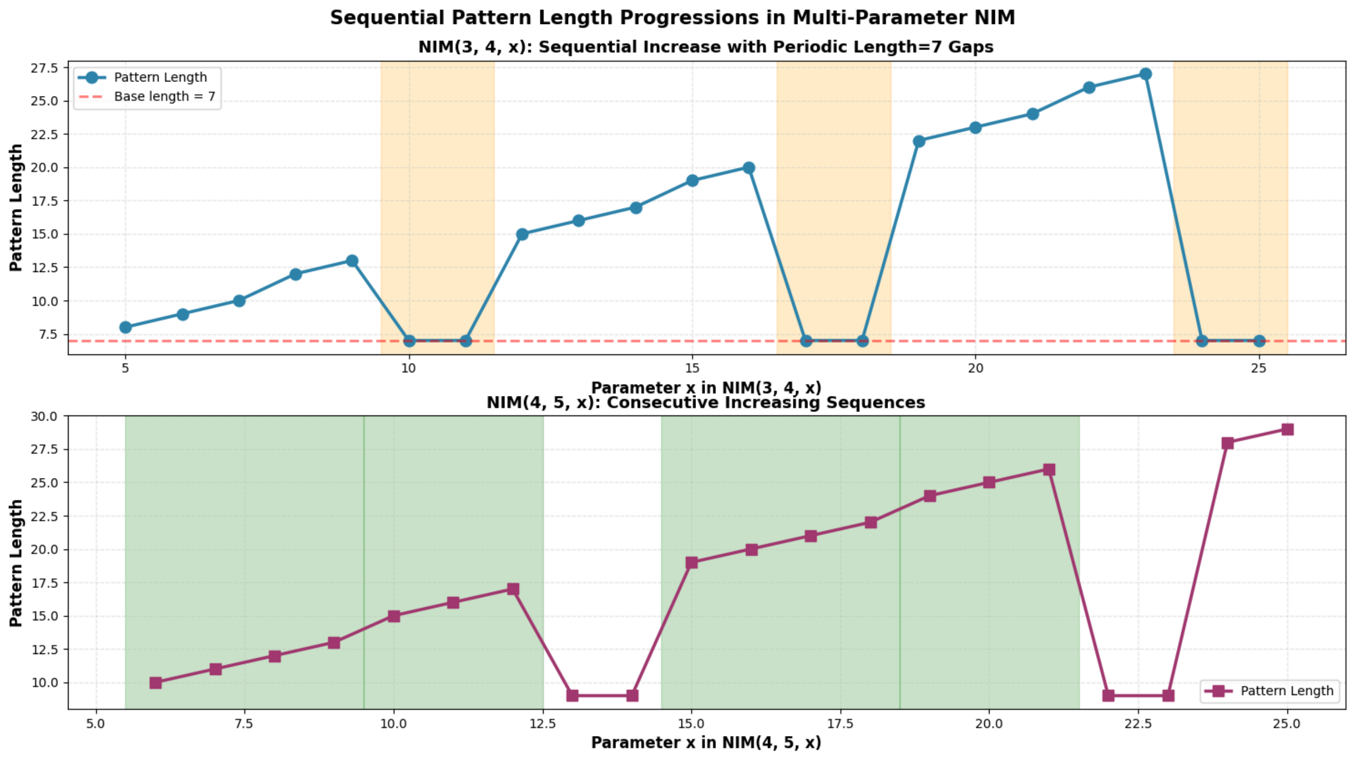
x=24,25: length=7

This suggests a base pattern length of 7 with additional structure layered on top.

NIM(4, 5, x) demonstrated similar consecutive increasing pattern lengths:

For values of x [6, 7, 8, 9, 10, 11, 12] and [15, 16, 17, 18, 19, 20, 21], The lengths were respectively [10, 11, 12, 13, 14, 15, 16, 17] and [19, 20, 21, 22, 23, 24, 25, 26, 27].

A "rule of +4" emerged: for many NIM(4, a, b) combinations, the MOD length approximately equaled 4 + b. This held particularly well for smaller values of a and b but there were exceptions as values of a and b increased.



Higher-Order Variants - NIM([5,6,7], a, b)

NIM(5, a, b), NIM(6, a, b), and NIM(7, a, b) exhibited increasingly irregular patterns with fewer combinations starting at index 0. Many entries showed "101 0", indicating no detectable pattern within the 100-length win table tested: a sign of either extremely long MOD lengths or aperiodic behavior that requires more advanced tools to analyze and detect patterns.

Conclusion

Summary of Key Findings

NIM variants exhibited mathematical structure of repeating modular patterns in their win tables. Pattern analysis revealed that simpler variants, particularly NIM(1, a) configurations, display 100% pattern detectability starting at index 0 for maximum move values up to N=49. However, as the parameter space expands, complexity increases and the percentage of combinations with patterns starting at index 0 declines. This trend intensified and generalized in higher-order variants, with NIM(1, a, b, c, d) showing only 32.29% pattern detectability at N=50 compared to 58.59% at NIM(1, a, b).

Specific regularities emerged across multiple variant families. NIM(1, 2, x) exhibited periodic behavior where pattern length equals 3 when $x \equiv 1 \pmod 3$, but jumps to x+1 when $x \equiv 0 \pmod 3$. NIM(1, 3, x) displayed alternating patterns with odd values consistently producing length=2, while certain even values generated longer patterns. More complex variants like NIM(3, 4, x) and NIM(4, 5, x) revealed sequential pattern length increases marked by periodic "gaps”. The "rule of +4" observed in NIM(4, a, b) variants, where pattern length approximately equals 4 + b, was also an interesting pattern yet no similar ones were found in this research for comparison or to find a mathematical emergence in the pattern.

Future Directions

An avenue of interest for this project was AI performance in NIM, specifically the testing of pre-existing Large Language Models on randomly generated NIM variants. Unfortunately, there was not enough time or guidance on this topic, so a statistical study could not be performed.

Systematically comparing large language models against both human players and a self-trained Q-learning agent (a reinforcement learning algorithm where the computer is rewarded for correct moves and penalized for losing moves) would show if general-purpose AI systems can develop mathematical intuition for combinatorial games or if they rely primarily on pattern matching from training data.

Finally, the practical applications of this research extend beyond pure mathematics and game theory. The pattern detection methods developed here could be adapted for sequence analysis in computational biology, cryptographic applications, or optimization problems with similar structures. The combination of mathematical analysis and AI implementation provides a model for research bridging classical theory with modern computational methods and machine learning.model for research bridging classical theory with modern computational methods and machine learning.

Acknowledgements

Thank you to professor William Gasarch at the University of Maryland’s Computer Science Department and Doctor Jonathan Fritz at Montgomery Blair High School for guiding me in my research and paper.

References

Bouton, Charles L. "Nim, A Game with a Complete Mathematical Theory." *Annals of Mathematics*, vol. 3, no. 1, 1901-1902, pp. 35-39, doi:10.2307/1967631.

Brookins, Philip, and Jason DeBacker. "Playing Games with GPT: What Can We Learn about a Large Language Model from Canonical Strategic Games?" *Economics Bulletin*, vol. 44, no. 2, 2024, pp. 457-466.

Fournier-Viger, Philippe, et al. "Efficient Algorithms to Identify Periodic Patterns in Multiple Sequences." *Information Sciences*, vol. 489, 2019, pp. 205-226.

Grundy, Patrick Michael. "Mathematics and Games." *Eureka*, vol. 2, 1939, pp. 6-8.

Jarleberg, Erik. *Reinforcement Learning on the Combinatorial Game of Nim*. Bachelor's thesis, KTH Royal Institute of Technology, 2011.

Renda, Alex, et al. "Comparing Humans, GPT-4, and GPT-4V on Abstraction and Reasoning Tasks." *arXiv*, 31 May 2023, arxiv.org/abs/2311.09247.

Rougetet, Lisa. "A Prehistory of Nim." *The College Mathematics Journal*, vol. 45, no. 5, 2014, pp. 358-363, doi:10.4169/college.math.j.45.5.358.

Siegel, Aaron N. *Combinatorial Game Theory*. American Mathematical Society, 2013.

Van den Herik, H. Jaap, et al. "Reinforcement Learning for Combinatorial Optimization: A Survey." *Computers & Operations Research*, vol. 134, 2021, article 105400, doi:10.1016/j.cor.2021.105400.

Xu, Jiawei, et al. "Playing Games with GPT: An Analysis of Strategic Reasoning in Large Language Models." *arXiv*, 8 Nov. 2024, arxiv.org/abs/2411.05990.

Ye, Karen. "NIM." *REU Papers*, University of Chicago, 2008, www.math.uchicago.edu/~may/VIGRE/VIGRE2008/REUPapers/Ye.pdf.

Zhou, Bei, and Søren Riis. "Impartial Games: A Challenge for Reinforcement Learning." *arXiv*, 25 May 2022, revised 3 Aug. 2025, arxiv.org/abs/2205.12787.