Enhancing Player Levels in the Nim Game Using Genetic Algorithms

Matheus Peres[1], Francisco Henrique de Freitas Viana[1], Kennedy M. Fernandes[2], Pedro Henrique Gonzalez[3], Leandro de M.B. Soares[1], Eduardo Bezerra[1], Joel Andre F. dos Santos[1], Diego Brandão[1]

[1] Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
Rio de Janeiro – RJ – Brasil

[2]Universidade Federal do Sul da Bahia (UFSB)
Teixeira de Freitas, BA – Brasil

[3]Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, RJ – Brasil

{francisco.viana,diego.brandao}@cefet-rj.br

Abstract. *The Nim game is a fundamental example in combinatorial game theory, exemplifying impartial games where moves depend solely on the current position. Creating difficulty levels in Nim for computer play presents challenges, as small changes in game element positioning can significantly alter the game's difficulty. Initial ideas, such as randomly applying the winning formula or ensuring constant computer wins, but it proved unsatisfactory due to erratic or discouraging gameplay. This article proposes using Genetic Algorithms (GAs) to develop varied difficulty levels in Nim. Our approach avoids the established winning formula, making defining an independent fitness function complex. Instead, we employ a competitive coevolutionary fitness function, where individuals are evaluated against each other, allowing the fitness criteria to evolve with the population. This method produces more nuanced and engaging difficulty levels, enhancing player experience through a balanced and evolving challenge.*

Keywords *Combinatorial Games, Nim game, Genetic Algorithms.*

1. Introduction

The Nim game is important in combinatorial game theory, serving as a primary example of an impartial game. An impartial game is one where the available moves depend only on the current position and not on which player is making the move. Analyzing Nim helped establish fundamental concepts and methodologies for studying more complex games [Mukhar 2013].

It is often used as an educational tool to teach mathematical reasoning and problem-solving. It helps students understand how to break down complex problems into simpler parts and how to use logic to devise strategies [Rougetet 2018, Stoffová et al. 2019].

Furthermore, Nim is well-known for its connection to binary numbers and the concept of Nim-sum, where the binary representation of pile sizes are XOR

together [Bouton 1901]. The first player has a winning strategy if the Nim-sum is non-zero, and the second player wins if the Nim-sum is zero.

A challenge in game development is balancing the interaction between the game's level design and the desired player experience. Changing the level's layout can affect the routes available and the interactions of various agents within the level [Berseth et al. 2014]. Even minor adjustments to the positioning of game elements can significantly alter the difficulty of the player's journey toward their goal. In the context of the NIM game, these adjustments are pretty delicate because, as mentioned, the strategies for winning a game are well-known.

This article aims to create difficulty levels in the game played against the computer. One initial idea would be randomly applying or not the formula. However, this would generate a characteristic far from a human being, toggling between very smart and weak plays that do not make sense together. Another possibility is to apply the formula so that the computer player always wins, which would end up causing human players to lose interest in the game.

The idea developed here is based on artificial intelligence techniques. Specifically, we have used Genetic algorithms to create different levels in the NIM game. A genetic algorithm (GA) is a good option for evolving an average human-like machine player. This player is neither a novice nor an expert but represents the typical player who understands the basic rules and strategies of the game. This is possible because the instances used to develop the genetic algorithm were generated from the plays of different human players, so in a way, the GA generates human behavior. In our approach, the algorithm does not consider the formula proposed by [Bouton 1901]. In this case, it becomes a complex task to establish an independent fitness function to evaluate the individuals. A competitive coevolutionary fitness function seems a more advantageous tool [Angeline e Pollack 1993]. In competitive fitness, each individual from the population is only evaluated against others. Thus, our fitness function evolves as the population proceeds with more qualified individuals in the GA generations.

This article is structured in six more sections. The Section 2 presents related work. Section 3 presents some definitions of the NIM game. Section 4 presents the genetic algorithm approach developed. The computational experiments are presented in Section 5. Finally, Section 6 presents the final considerations of the work.

2. Related Work

Competitive fitness functions have multiple advantages against the usual independent functions commonly applied in GA. Many complex problems require very little knowledge of the domain and the search space [Angeline e Pollack 1993, Takagi e Pallez 2009, Rosin e Belew 1996]. When the nature of the problem allows it, competition between individuals in the population can be used as a performance measure instead of an objective function [Vrajitoru 2007]. Empirical studies have analyzed the effects on the performance of the dependency between algorithm properties and problem properties [Popovici et al. 2012].

In general, the coevolutionary algorithm evaluates individuals based on their interactions with other individuals. Coevolutionary programming falls into two main

groups: Competitive and Cooperative Coevolutionary algorithms. In cooperative coevolution, individuals from each species temporarily enter collaborations with members of the other species and get rewarded according to the success of the collaborations in solving objective functions [Potter 1997]. In [Luke et al. 2011], a study is conducted regarding the parameter settings for moderately large problems and the use of large populations.

Back to competitive coevolution, we present some works that successfully applied competitive fitness functions in evolutionary programming. [Rosin e Belew 1995, Rosin e Belew 1997] have used competitive coevolution in three domains: Tic-Tac-Toe, Nim game, and a small version of Go. They have presented new methods for fitness evaluation with multi-population. The first one is called Competitive Fitness Sharing, in which an individual‘s fitness is divided by the sum of its similarities with the other individuals in the population. Second, they present Opponent Sampling, which tests each individual in the host population against only a limited sample of parasites from the other population. Third, they introduce the Hall of Fame, in which the best individual from every generation is retained for future testing [Rosin e Belew 1995, Rosin e Belew 1997].

Competitive coevolutionary fitness was used to evolve decision-making strategies in multiple other games. In [Hauptman e Sipper 2005], genetic competitive programming evolved a game-playing strategy for chess endgames. They have used K-Random opponents [Panait e Luke 2002] to evaluate each individual in the population. The evolved program finished second in the 2004 Computer Chess Championship. In [Fernández-Ares et al. 2017], genetic programming has been applied to create the behavioral engine of bots able to play a simple Real-Time Strategy (RTS) game. They conclude that so-called victory-based fitness generates better bots, on average, than other objective functions tested. Furthermore, battle strategy games [Wilisowski e Dreżewski 2015], 8-card poker [Oliehoek et al. 2006], Othello [Szubert et al. 2013], chess endgames [Sipper et al. 2007, Hauptman e Sipper 2005], backgammon, and tank-fight simulation [Sipper et al. 2007] were also used as a domain for competitive coevolutionary programming.

Besides games, real-world problems have also been solved using competitive coevolution. Competition-based fitness was used to find suppliers' optimal strategies in a deregulated electricity market [Ladjici et al. 2014]. Additionally, the so-called BrilliAnt won the Ant Wars contest at the GECCO-2007 Genetic and Evolutionary Computation Conference. They have used competitive one-population coevolution to evolve the ant to collect food in a square toroidal grid environment in the presence of competing ants [Jaśkowski et al. 2008]. Moreover, competitive fitness has outperformed objective fitness in evolving a multi-agent autonomous pilot for motorcycles. The objective function was based on average speed and time efficiency criteria, while the competitive function used the K-Random Opponents model [Vrajitoru 2007].

Some studies have also explored the consequences of intransitive superiority in coevolutionary programming [Funes e Pujals 2005, Watson e Pollack 2001, Luke e Wiegand 2002, Samothrakis et al. 2012]. In an intransitive game, if A beats B and B beats C, not necessarily A beats C. Intransitivity is one of the main reasons for cycles in the population dynamics [Watson e Pollack 2001]. By definition, the Nim game is an example of an intransitive domain.

The use of genetic algorithms for the NIM game is not new, as seen from the literature review. However, previous work has primarily focused on the genetic algorithm itself and the development of coevolutionary competitive approaches. This is the first work focused on developing a player-machine that behaves almost on the level of a player applying the victory pattern formula, allowing the human player to encounter significant difficulty and still have a chance of winning. It is also the first to compare three different fitness evaluation methods.

3. NIM Game

There are multiple different versions of the Nim game found in the literature. Here, we will present three: a Random Start Nim Game, a Single Heap Nim Game, and a Fixed Start Nim Game.

3.1. Random Start Nim Game (RSN)

In RSN, the initial configuration consists of several heaps of objects (we will treat these objects as matchsticks), as shown in Figure 1. Two players take turns removing an arbitrary number of matchsticks from a single heap at a time. The player who removes the last matchstick on the table loses the game.

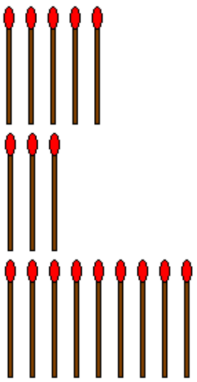

Figura 1. Example of Nim game

In our experiments, the simulations always start with four heaps. The number of matchsticks in each of the four heaps is randomly set for each game with a value ranging from 1 to N. Where N is the maximum number of matchsticks a heap can assume in the evolutionary run. The player who starts each game is also set at random.

In this paper, we evolve machine players that simulate a medium human being. For this, we set a limited time for GA to run and compare the performance of different techniques. We will test and compare three competitive fitness methods to select the best for generating a machine player: Single-Elimination Tournament, round-robin, or K-random [Angeline e Pollack 1993, Panait e Luke 2002].

3.2. Single Heap Nim Game (SHN)

In this version of Nim, one single heap is placed on the table, and players alternate between removing matchsticks. However, the number of matchsticks that a player can remove is bounded by a minimum and a maximum value. The player who removes the

last matchstick loses. GA is applied to the SHN domain with competitive fitness by [Hynek 2004] and [Panait e Luke 2002]. Both have used a single population to evolve a Nim game machine player. [Hynek 2004] also explores using the macromutation operator, the Headless Chicken Crossover. In macromutation, random individuals are inserted into the population during evolution. This technique is used to encourage diversity and thus avoid premature convergence.

3.3. Fixed Start Nim Game (FSN)

This version is very similar to the RSN that we are using here. Players take turns removing an arbitrary number of matchsticks from one of the heaps at a time. However, in this version, the player who removes the last matchstick wins. The four heaps always start with 3, 4, 5, and 4 matchsticks, respectively: (3, 4, 5, 4).

FSN is applied to competitive fitness in [Rosin e Belew 1995, Panait e Luke 2002]. [Panait e Luke 2002] compares five different methods for evaluating an individual. In Round Robin, individuals are tested against the entire population. Each individual plays only one game against an opponent in a random pairing. In the Single-Elimination Tournament, individuals are randomly paired to play one game; the losers are eliminated, and this continues until reaching the champion. In K-Random, each individual plays against K individuals randomly selected from the population. In the Hall of Fame, individuals are tested against the good individuals discovered so far in the evolution.

[Panait e Luke 2002] concludes from the experiments that Round Robin always appears to be a wrong choice against K-Random when both have a maximum time to run. In this work, we will compare the performance of Round Robin and K-Random again, now in the RSN Domain, and use another external function to evaluate the GA efficiency.

4. Our Approach

The main goal of this paper is to evolve machine players that simulate the behavior of an average human being playing the Nim Game. We aim to reach it in multiple search spaces, varying the maximum number of matchsticks per heap, which affects the total number of possible situations in the game. It is natural to imagine that increasing the search space creates stricter work for the GA to reach the objectives.

We will also test GA, considering its limited time to run. To compare the performance of different methods and operators, we will run GA multiple times under the same circumstances and during the same time, varying only one parameter per test. Thereby, we can determine which competitive fitness function presents the best results.

The GA will be applied with one-population evolutionary runs, high elitism, and competitive fitness functions. Individuals will be evaluated by one of the three fitness methods. The individuals who obtain the best scores in the fitness evaluations will proceed intact to the next population to decrease the probability of losing the best individual so far in the run (elitism). The missing spots in the next population will be filled with the crossover of the entire old population, where the chance of crossing is proportional to the individual's evaluation against the others.

Figure 2 shows the flowchart of the GA operators and how they were set.

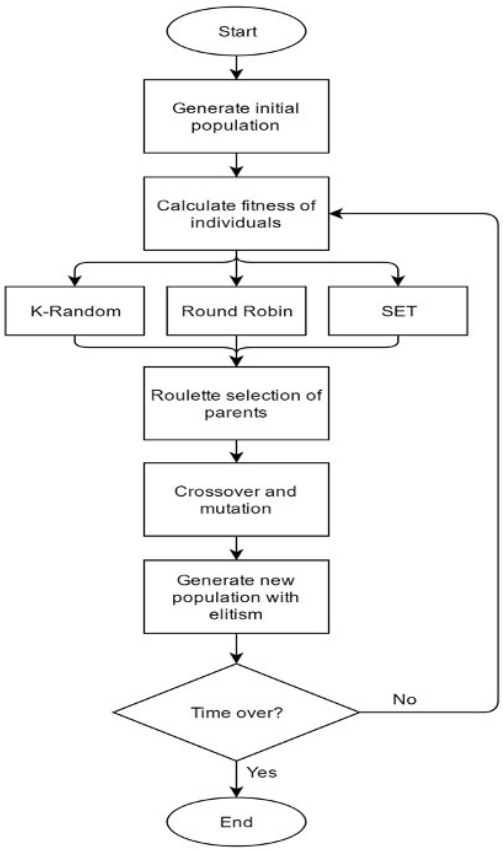

Figura 2. Flowchart shows an overview of how GA was set to operate

4.1. Solution Representation

We will treat each game situation in ascending order of matchsticks per heap, considering all permutations as the same situation. For example, if (4, 3, 2, 1) appears during a game, it will be converted to (1, 2, 3, 4) before any work over it. With this, we can shorten search space. Taking this, counting with (0, 0, 0, 0), the total number S of different situations, considering R heaps and a maximum of N matchsticks per heap, characterizes a combination with repetition allowed (multiset). The value of S is then given by the Equation 1.

$$S(N, R) = \frac{(N + R)!}{N! \times R!} \tag{1}$$

The chromosome from an individual will be represented as a set of ordered pairs (X, Y) with integers, one pair for each situation. X is the number of matchsticks in a heap before the play, and Y is the number of matchsticks left in the same heap after the play. It is important to notice that X should be one of the four values in the concerning situation. Also, Y should be less than X. For instance, if the problem is (1, 2, 2, 3), X=2, and Y=1, the situation left after the play should be (1, 1, 2, 3). We should also notice that it does not matter which heap containing 2 matchsticks will take the play because, as we have mentioned, the order in which the heaps are set does not count.

The population is generated based on this concept of chromosomes. Many

individuals are randomly created. Each individual is filled with one random (X, Y) pair for each possible situation. So, the chromosome size of an individual will change if we change the number of situations by changing the maximum number of matchsticks per heap.

4.2. Fitness Function

There are multiple methods to evaluate a population in a competitive environment. The choice of a fitness function may affect two main points in the GA run. First, varying the function can change the time the algorithm takes to evolve a population through each generation. Second, opting for a good fitness function may retard a convergence to a local optimum.

We will consider three different methods of competitive fitness function. We aim to determine which presents the best performance under the same conditions and during the same time to run.

4.2.1. Round Robin

In Round Robin (Full Competition), each individual once plays all the others in the population. Each game counts for the fitness of both individuals involved. The number of wins represents the individual's fitness. This is known to be a costly function when it comes to wasting time because of the large number of games required.

The number of games in each generation for Round Robin (GR), as a function of the population size (P), is given by the simple combination presented in Equation 2.

$$GR(P) = \binom{P}{2} = \frac{P \times (P-1)}{2} \tag{2}$$

4.2.2. K-random

In K-random, every individual plays k (in lower case) randomly selected individuals from the population. In our experiments, each game also counts for both players. However, once the opponents are randomly picked, individuals may have a different number of games played. An individual's fitness is determined by the percentage of wins instead of directly taking the number of wins. The round-robin method is similar to K-random when k tends to the population size. Playing only k individuals instead of the whole population is mostly used to decrease the time to evaluate the fitness of a population.

Once we vary the population size in different runs, we will consider K (in the upper case) equal to the percentage of the population size with which individuals will play, so $0 < K < 1$. So k stands for the number of individuals to play against, while K stands for the percentage of individuals to play against. The number of games in each generation for K-Random (GK), as a function of the population (P) and the number K, is given by the sum of $K * P$ games for each player (i), which leads to the formula presented in Equation 3.

$$GK(P,K) = \sum_{i=1}^{P} KP = P \times KP = KP^2 \tag{3}$$

4.2.3. Single-Elimination Tournament

Each individual plays one randomly paired game in a Single-Elimination Tournament (SET). The losers are eliminated, which goes over until a champion is reached. As in K-Random, SET is generally used to shorten the number of games played in each generation even more. To apply SET, the population size should be a power of two so that the tournament ends perfectly with two opponents to play.

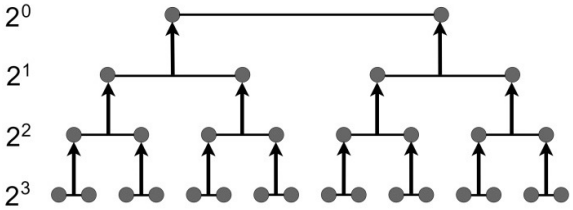

Figura 3. Number of games in each level of SET

For SET, the number of games per generation (GS), as a function of the population size (P), is given by the sum of the powers of two represented in Figure 3, for a population P=16. This leads to the formula presented in Equation 4. It represents the number of games per generation(GS) in a Single-Elimination Tournament as a function of the population (P).

$$GS(P) = \sum_{i=0}^{log(P)-1} 2^i = P - 1 \tag{4}$$

Round Robin, K-Random, and SET methods require a very different amount of games per generation. Plotting the three functions to compare their behavior regarding the population size is interesting. Figure 4 presents GR(P), GK(P, 0.2), and GS(P) with the population size ranging from 0 to 32. As we expected and see in the plot, the gap between the methods regarding the number of games required increases as the population increases.

4.3. Cross-over

As mentioned in 4.1, the chromosome comprises a set of ordered pairs (X, Y), one pair for each situation. In this work, we use a uniform crossover. Each ordered pair in a chromosome has a probability of 0.5 from each parent. Figure 5 represents a possible crossover of Parent 1 (in blue) and Parent 2 (in red), while the green color represents mutation as will be described in Section 4.4.

All individuals in the population have a chance to crossover. We use the roulette wheel selection to pick couples to reproduce. The probability of being selected to crossover is proportional to the individual's fitness. Thus, the best and worst individuals

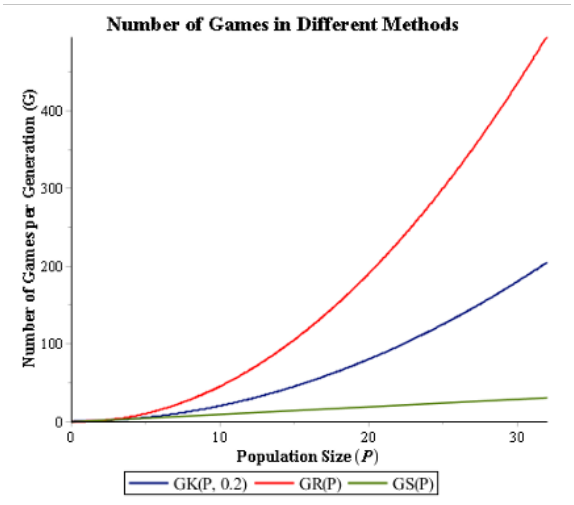

Figura 4. Number of games per generation needed in K-Random, Round Robin, and SET

in the generation may get to crossover, which should avoid uniformity among the competitors.

We allow elitism to guarantee that the quality of the solution is not reduced. Individuals with the best fitness are carried over unaltered to the next generation. We have experimentally noticed that a large elitism in this domain, around half of the population, seems beneficial to the attempt to accelerate the GA results.

4.4. Mutation

After newborn individuals are created, the mutation is added to their chromosomes to help maintain genetic diversity. There is a small probability that random pairs (X, Y) are added to modify the chromosome. The mutation is represented (in green) in Figure 5.

Figura 5. Mutation

5. Experiments

The experiments were performed using computational routines implemented in Java language on an Intel(R) Xeon(R) Gold 5120 CPU 2.20GHz, with 28 cores and 192*GB* of RAM.

We target to run two main experiments. First, K-Random, round-robin, and Single-Elimination Tournaments (SET) will be compared under the same circumstances. We will test the scalability of these methods concerning the search space size. Second, we will take qualitative tests with average human beings, i.e., those who have not played the game before and have just learned the rules.

We use an objective external fitness to determine the algorithm achievements. As mentioned in Section 1, winning the Nim game depends on a formula that uses binary numbers to determine if a play is not leading to a win [Bouton 1901]. Therefore, we will analyze all the plays composing the chromosome of each individual so we can plot the number of perfect plays of the best individual from each generation. It is important to emphasize that the GA does not know this formula. It is only used to measure the GA's performance. GA evaluates each individual by comparing its number of wins against other individuals.

5.1. Competitive Fitness Techniques

The purpose is to compare the performance of K-Random, Round Robin, and SET. These methods use different procedures to evaluate the fitness of the population. Because different procedures may require a different amount of time, each competitive fitness method will have the same maximum running time instead of a limited number of games.

Multiple experiments have used the FSN domain in the literature, where games always start with (3, 4, 5, 4) matchsticks in each heap. Even though this is an exciting set to start games, we aim to vary the search space dimension by changing the maximum number of matchsticks per heap (N) that games can start with.

This work will compare the three competitive fitness techniques in search spaces. First, games will start with heaps containing a random number of matchsticks ranging from 1 to 5. This limit (N=5) includes all the possible situations in the Fixed Start Nim Game used in other works. Second, we will set N=10. The number of potential situations, such as the chromosome size, S(N, 4), grows exponentially as we increase N. Games will have at most five matchsticks per heap in the first set of runs. K-Random, Round Robin, and Single-Elimination Tournament will have 15 seconds to run.

Parameters were empirically set. In the experiments, the best half of the population passes identically to the next population (elitism). The other half is completed using the roulette wheel crossover technique. The entire population can crossover, but the probability is proportional to the individual´s fitness.

Table 1 shows the parameters used in the first set of tests. With these settings, we ran the three fitness methods for 15 seconds. For this test, we set K=0.2 (20% of the population) in the K-Random method. Figure 6 shows the number of perfect plays of the best individual so far reached by each method through the generations. The results are an average of 30 runs for each method. For N=5, there is a maximum of 103 situations where a perfect play can be made.

As shown in Figure 6, K-Random evolves a better individual for N=5, followed by Round Robin. SET comes last. In the second set of runs, we use N equals 10. The fitness methods will now have 2 hours to run. The intention is to determine whether the search space dimension influences the performance of the methods.

We need a larger population to obtain good results as the search space increases. With an exponential growth of search space and number of games, the GA requires much more time when we double the N. Table 2 presents the parameters used in the second set of tests. For this comparison, we test different values of K in K-Random. Figure 7 shows the outcome of the fitness methods to evolve Nim game players. For N=10, a maximum

Tabela 1. Parameters for N=5

Parameters	Values
N	5
S(5,4)	126
Max. Number Perfect Plays	103
Runtime	15s
Population	200
Mutation	0.02
Elitism	0.5
Macromutation	0
Fitness Method	Variable

of 910 situations allow a perfect play.

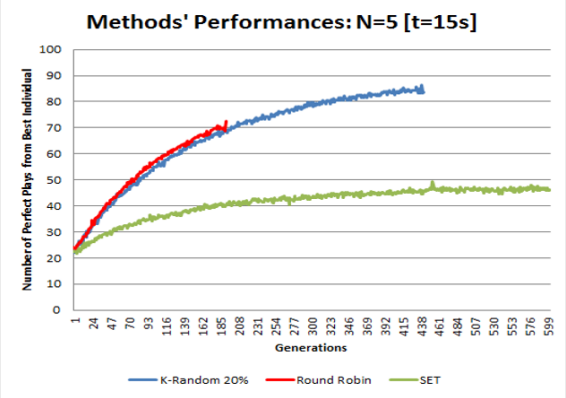

Figura 6. Performance of K-Random, Round Robin, and Single-Elimination Tournament to evolve game strategies with at most 5 matchsticks per heap

Unlike the case with N=5, when we increase N to 10, Round Robin performs a better result than SET and K-Random for any value of K less than 100%.

Tabela 2. Parameters for N=10

Parameters	Values
N	10
S(10,4)	1001
Max. Number Perfect Plays	910
Runtime	2h
Population	600
Mutation	0.02
Elitism	0.5
Macromutation	0
Fitness Method	Variable

With these two tests involving K-Random, Round Robin, and SET, we are tempted to affirm that the performance of each method in comparison to the others depends on

search space. For RSN, as we have seen in Figures 6 and 7, as we increase the maximum number of matchsticks per heap in the games, it becomes a better option to use a method that requires more games to evaluate the population.

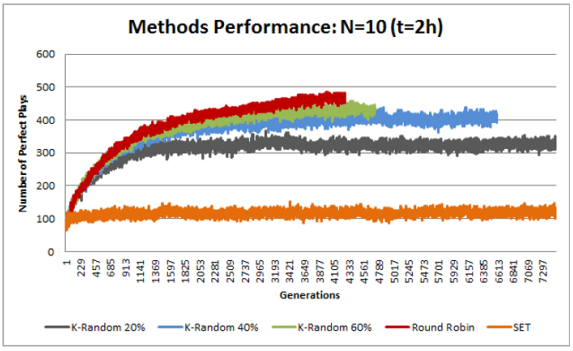

Figura 7. Performance of K-Random, Round Robin, and Single-Elimination Tournament to evolve game strategies with at most 10 matchsticks per heap

5.2. Discussion and Limitations

This study proposes using genetic algorithms (GAs) to develop varying difficulty levels in the game Nim and enhance the player experience through balanced and evolving challenges. The traditional victory formula for Nim was avoided, making defining an independent fitness function complex. Instead, we employed a competitive coevolutionary fitness function, where individuals are evaluated against each other, allowing the fitness criteria to evolve with the population.

Our experiments were designed to compare three competitive fitness evaluation methods: K-random, round-robin, and Single-Elimination Tournament (SET). These methods were tested in different search space dimensions, varying the maximum number of matches per stack (N). K-Random showed superior performance for smaller search spaces (N=5), evolving better individuals faster than Round Robin and SET. However, as the search space increased (N=10), Round Robin outperformed the other methods, indicating that the efficiency of the fitness evaluation method depends on the complexity of the problem space.

The results showed that GAs with competitive coevolutionary fitness functions can effectively generate human-like game strategies in the Nim game. This approach can lead to more engaging and balanced difficulty levels, improving player satisfaction. However, human users' satisfaction and engagement with the game are needed to be evaluated.

6. Final Remarks

As a fundamental model in combinatorial game theory, the Nim game offers profound insights into impartial games and is an excellent educational tool. Its connection to binary numbers and the concept of Nim-sum provides a clear framework for understanding winning strategies. However, designing varying difficulty levels for computer-based Nim play presents unique challenges.

Our exploration of using genetic algorithms (GAs) to address these challenges has demonstrated the potential for creating more engaging and balanced gameplay

experiences. By avoiding reliance on the traditional winning formula and employing a competitive coevolutionary fitness function, our approach allows the difficulty levels to evolve, simulating human-like play and maintaining player interest.

This study highlights the benefits of integrating advanced AI techniques in game design to enhance player experience and further educational objectives in mathematical reasoning and problem-solving. Future work may explore refining these algorithms and applying similar methods to other games, ultimately contributing to the broader field of game theory and AI-driven game design.

Acknowledgements

The authors would like to thank the following Brazilian Agencies: CAPES and the National Council for Scientific and Technological Development - CNPq. DB thanks Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ) for the financial support through the grants E-26/210.798/2024.

Referências

Angeline, P. e Pollack, J. (1993). Competitive environments evolve better solutions for complex tasks. In *Proceedings of the 5th International Conference on Genetic Algorithms*, page 264–270. Morgan Kaufmann.

Berseth, G., Haworth, M. B., Kapadia, M., e Faloutsos, P. (2014). Characterizing and optimizing game level difficulty. In *Proceedings of the 7th International Conference on Motion in Games*, pages 153–160.

Bouton, C. L. (1901). Nim, a game with a complete mathematical theory. *The Annals of Mathematics*, 3(1/4):35–39.

Fernández-Ares, A., Mora, A., García-Sánchez, P., Castillo, P. A., e Merelo, J. (2017). Analysing the influence of the fitness function on genetically programmed bots for a real-time strategy game. *Entertainment Computing*, 18:15–29.

Funes, P. e Pujals, E. (2005). Intransitivity revisited coevolutionary dynamics of numbers games. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 515–521.

Hauptman, A. e Sipper, M. (2005). Gp-endchess: Using genetic programming to evolve chess endgame players. In *European Conference on Genetic Programming*, pages 120–131. Springer.

Hynek, J. (2004). Evolving strategy for game playing. In *4th international ICSC symposium on engineering intelligent systems*, pages 1–6.

Jaśkowski, W., Krawiec, K., e Wieloch, B. (2008). Winning ant wars: Evolving a human-competitive game strategy using fitnessless selection. In *Genetic Programming: 11th European Conference, EuroGP 2008, Naples, Italy, March 26-28, 2008. Proceedings 11*, pages 13–24. Springer.

Ladjici, A. A., Tiguercha, A., e Boudour, M. (2014). Nash equilibrium in a two-settlement electricity market using competitive coevolutionary algorithms. *International Journal of Electrical Power & Energy Systems*, 57:148–155.

Luke, S., Sullivan, K., e Abidi, F. (2011). Large scale empirical analysis of cooperative coevolution. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 151–152.

Luke, S. e Wiegand, R. P. (2002). When coevolutionary algorithms exhibit evolutionary dynamics. In *2002 Genetic and Evolutionary Computation Conference Workshop Program*, pages 236–241.

Mukhar, K. (2013). *Using evolutionary computing to develop game strategy for a non-deterministic game*. PhD thesis, University of Colorado Colorado Springs.

Oliehoek, F. A., De Jong, E. D., e Vlassis, N. (2006). The parallel nash memory for asymmetric games. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 337–344.

Panait, L. e Luke, S. (2002). A comparison of two competitive fitness functions. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 503–511.

Popovici, E., Bucci, A., Wiegand, R. P., e De Jong, E. D. (2012). Coevolutionary principles.

Potter, M. A. (1997). *The design and analysis of a computational model of cooperative coevolution*. George Mason University.

Rosin, C. D. e Belew, R. K. (1995). Methods for competitive co-evolution: finding opponents worth beating. In *ICGA*, pages 373–381. Citeseer.

Rosin, C. D. e Belew, R. K. (1996). A competitive approach to game learning. In *Proceedings of the ninth annual conference on Computational learning theory*, pages 292–302.

Rosin, C. D. e Belew, R. K. (1997). New methods for competitive coevolution. *Evolutionary computation*, 5(1):1–29.

Rougetet, L. (2018). Machines designed to play nim games (1940–1970): A possible (re) use in the modern french mathematics curriculum? *Teaching and Learning Discrete Mathematics Worldwide: Curriculum and Research*, pages 229–250.

Samothrakis, S., Lucas, S., Runarsson, T. P., e Robles, D. (2012). Coevolving game-playing agents: Measuring performance and intransitivities. *IEEE Transactions on Evolutionary Computation*, 17(2):213–226.

Sipper, M., Azaria, Y., Hauptman, A., e Shichel, Y. (2007). Designing an evolutionary strategizing machine for game playing and beyond. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(4):583–593.

Stoffová, V. et al. (2019). Computer games as a tool for the development of algorithmic thinking. *European Proceedings of Social and Behavioural Sciences*, 53.

Szubert, M., Jaśkowski, W., Liskowski, P., e Krawiec, K. (2013). Shaping fitness function for evolutionary learning of game strategies. In *Proceedings of the 15th annual conference on genetic and evolutionary computation*, pages 1149–1156.

Takagi, H. e Pallez, D. (2009). Paired comparison-based interactive differential evolution. In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pages 475–480. IEEE.

Vrajitoru, D. (2007). Competitive coevolution versus objective fitness for an autonomous motorcycle pilot. In *2007 IEEE International Conference on Electro/Information Technology*, pages 557–562. IEEE.

Watson, R. A. e Pollack, J. B. (2001). Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 702–709.

Wilisowski, Ł. e Dreżewski, R. (2015). The application of co-evolutionary genetic programming and td (1) reinforcement learning in large-scale strategy game vcmi. In *Agent and Multi-Agent Systems: Technologies and Applications: 9th KES International Conference, KES-AMSTA 2015 Sorrento, Italy, June 2015, Proceedings*, pages 81–93. Springer.