Morpion Solitaire: Algorithm Evaluation and Performance Analysis

Doha El Hitary
delhitary@gmail.com
Université Paris-Dauphine

July 31, 2024

Abstract

This report presents the implementation and results of an automated version of the Morpion Solitaire game, utilizing advanced search algorithms such as Nested Monte Carlo Search (NMCS), Nested Rollout Policy Adaptation (NRPA) and an Enhanced NRPA algorithm. The report details the fundamental concepts of Morpion Solitaire, the algorithms used, and the results obtained from these methods.

1 Introduction to Morpion Solitaire

Morpion Solitaire is a combinatorial game that challenges players to extend a series of moves on a grid. The objective is to maximize the number of moves by forming lines of five connected dots. The game exists in two variants: 5T (Classic) and 5D (Diagonal). Given its complexity, it serves as an ideal testbed for evaluating search algorithms.

Game Rules:

- The starting grid contains tokens arranged in a predefined manner.
- The player must place a new token on the grid each turn.
- Each newly placed token must complete a line of five tokens in a straight line (horizontal, vertical, or diagonal).
- The game ends when no more tokens can be placed according to the rules.

1.1 State of the Art in NMCS and NRPA for Morpion Solitaire

Several studies have explored the application of NMCS (Nested Monte Carlo Search) and NRPA (Nested Rollout Policy Adaptation) to Morpion Solitaire. Cazenave's work on Nested Monte Carlo Search has shown its effectiveness in handling the game's complexity [1]. Rosin introduced Nested Rollout Policy Adaptation, which has also been applied successfully to Morpion Solitaire, demonstrating promising results [2]. Additionally, Wang et al. investigated the use of MCTS principles in Morpion Solitaire, further advancing the field [3].

Various aspects of Monte Carlo Tree Search (MCTS) and its applications have been explored in different studies. Coulom's work on efficient selectivity and backup operators in MCTS highlights improvements in tree search algorithms [4]. Gelly and Silver's research demonstrates the effectiveness of MCTS and rapid action value estimation in the game of Go, providing a strong foundation for applying these techniques to other complex games [5]. Browne et al. present a comprehensive survey of MCTS methods, detailing their applications and effectiveness across various domains [6]. Chaslot et al. introduce a new framework for game AI using MCTS, illustrating its versatility and robustness [7]. Finally, Winands, Björnsson, and Saito examine the application of MCTS in the game Lines of Action, further showcasing the adaptability and power of MCTS in diverse gaming scenarios [8].

2 NMCS (Nested Monte Carlo Search)

NMCS is chosen for its robust exploration and exploitation balance in combinatorial optimization problems like Morpion Solitaire. It systematically evaluates potential moves through nested simulations, making it adept at discovering effective strategies in large search spaces. NMCS's ability to perform sequential evaluations allows it to progressively refine move selection, making it a powerful tool for optimizing game strategies.

2.1 How NMCS is Implemented

NMCS implementation consists of several key components:

- Sequential Search: NMCS performs a nested search where each level sequentially evaluates moves. This involves simulating possible moves, assessing their outcomes, and selecting the best moves based on these evaluations.

- Simulations and Evaluations: At each level, NMCS conducts a series of simulations to evaluate potential moves. The algorithm then selects moves that yield the highest scores from these simulations.

- Progressive Refinement: NMCS iteratively refines its move selection by conducting deeper levels of search, progressively improving the strategy based on the outcomes of previous levels.

The implementation starts with initializing the game grid. The algorithm then performs multiple iterations of nested searches, with each level conducting simulations and selecting the best moves based on the evaluations. This process is repeated, refining the strategy through progressive search depth.

2.2 Methodology

The NMCS algorithm employs a nested search structure to evaluate possible moves. Each level of the search performs independent simulations and selects moves based on the results. The NMCS algorithm uses a nested structure to explore possible grid configurations by performing sequential evaluations. Below is an outline of the NMCS methodology:

Initialization:

- Start with an initial game grid.

Nested Search:

- For each level of depth:
  - **Simulation**: Perform a series of simulations to evaluate potential moves.
  - **Move Selection**: Select the moves that yield the highest scores based on the simulations.

Progressive Search:

- Conduct deeper levels of search to progressively refine the move selection strategy.

Iterative Improvement:

- Repeat the nested search and move selection for a specified number of iterations to continuously improve the strategy.

The NMCS algorithm systematically evaluates and refines move selections through nested simulations and progressive search, enabling it to effectively optimize game strategies.

3 NRPA (Nested Rollout Policy Adaptation)

NRPA is selected for its ability to dynamically adapt strategies through nested rollouts, making it particularly effective for complex combinatorial games like Morpion Solitaire. Its hierarchical learning and adaptation capabilities enable efficient optimization of move sequences by learning and adjusting policies at various levels of nesting. This makes NRPA an ideal choice for exploring vast search spaces and refining strategies based on observed outcomes.

3.1 How NRPA is Implemented

NRPA implementation involves several core components:

- Policy Representation: A policy in NRPA is represented by a vector that assigns weights to potential moves. These weights influence the probability of selecting each move during playouts.

- Adaptation Mechanism: The algorithm iteratively adapts the policy based on the outcomes of simulated playouts, adjusting weights to favor more successful moves and reducing the likelihood of less effective ones.

- Playouts and Nesting: At each level of nesting, NRPA performs a series of playouts, selecting moves based on the current policy. After each playout, the policy is updated through adaptation, reinforcing successful strategies and penalizing unsuccessful ones.

The implementation begins with initializing the game grid and policy. For each level of nesting, the algorithm executes multiple iterations, each involving a sequence of playouts. Following each playout, the policy is adapted to reinforce successful moves and penalize less effective ones, gradually refining the strategy.

3.2 Methodology

The NRPA algorithm is implemented with multiple levels of depth to explore possible grid configurations. Each level conducts simulations and adapts the move selection policy based on the results obtained.

```
class NRPA:
    def __init__(self, level, iterations, version, alpha=1.0):
        self.level = level
        self.iterations = iterations
```

```
        self.alpha = alpha
        self.version = version

    # other methods...
```

Listing 1: NRPA Class

Below is an outline of the NRPA methodology:

Initialization:

- Start with an initial game grid and a randomly initialized policy.

Nested Search:

- For each level of depth:
  - **Simulation**: Perform a series of playouts, simulating the game by selecting moves based on the current policy.
  - **Policy Adaptation**: After each playout, adapt the policy by increasing the weights of successful moves and decreasing the weights of less successful ones.
  - **Policy Update**: Update the policy at each level based on the outcomes of the playouts to favor successful strategies.

Iterative Improvement:

- Repeat the nested search and policy adaptation for a specified number of iterations to progressively refine the strategy.

The NRPA algorithm systematically refines the move selection policy by leveraging nested simulations and adaptive learning, enabling it to effectively explore and optimize game strategies.

4 Enhanced NRPA

To conduct some exploration, we chose to develop a version of the NRPA algorithm that differs from the standard NRPA. Our goal is to investigate new approaches

and techniques that could potentially improve the algorithm's performance and efficiency in solving complex combinatorial search problems.
By introducing new modifications such as an exploration constant and enhanced playout strategies, we aim to determine whether these changes can lead to better results and a more efficient search process.

Here is a detailed explanation of these enhancements:

4.1 Exploration Constant

- **Standard Version:** The standard version of NRPA uses a simple policy based on exponential weights of the moves to select moves at each step.
- **Enhanced Version:** The enhanced version introduces an exploration constant (`exploration_constant`), which is a parameter used to balance exploration and exploitation of the moves. This allows the algorithm to better explore the search space, rather than focusing solely on moves with the highest weights.

4.2 Enhanced NRPA Playouts

- **Standard Version:** Playouts in the standard version consist of making successive moves following the current policy until no more moves are possible.
- **Enhanced Version:** Playouts in the enhanced version (`enhanced_nrpa_playout`) follow a similar strategy but can be influenced by the exploration constant, allowing for more varied exploration of the moves.

4.3 Adaptation Strategy

- **Standard Version:** The adaptation strategy in the standard version updates the policy weights based on the results of the playouts.
- **Enhanced Version:** The adaptation strategy in the enhanced version also uses the exploration constant to adjust the move weights more sophisticatedly.

4.4 Overall Algorithm Structure

- **Standard Version:** The basic NRPA algorithm performs recursive calls for different search levels and adapts the policy based on the results at each level.

- **Enhanced Version:** The enhanced algorithm follows the same structure but with improved policy and playout management due to the addition of the exploration constant.

4.5 Technical Details of the Enhancements

- **Exploration Constant (`exploration_constant=1.414`):** This addition allows regulating the degree of exploration by influencing the move weights. A higher value will favor more exploration, while a lower value will favor more exploitation of known good moves.

- **Enhanced Playout:** The `enhanced_nrpa_playout` method follows the same logic as `nrpa_playout` but with adjustments to account for the exploration constant.

In summary, the enhanced version of NRPA is hypothesized to potentially surpass the standard version mainly by introducing the exploration constant, which is expected to allow for a better balance between exploration and exploitation of moves, and enhanced playouts that leverage this new approach. These improvements could **potentially** lead to more efficient searches and better results for complex combinatorial search problems like the Morpion Solitaire game, though this remains to be confirmed through further experimentation and analysis.

5 Performance and Results

This section analyzes the performance of the algorithms tested: NMCS (Nested Monte Carlo Search), NRPA (Nested Rollout Policy Adaptation), and Enhanced NRPA. The results, as shown in the image provided, are summarized and discussed below.

5.1 NMCS

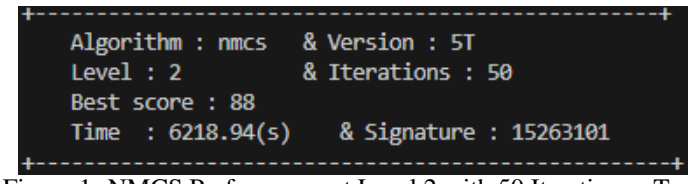

Figure 1: NMCS Performance at Level 2 with 50 Iterations - Test1

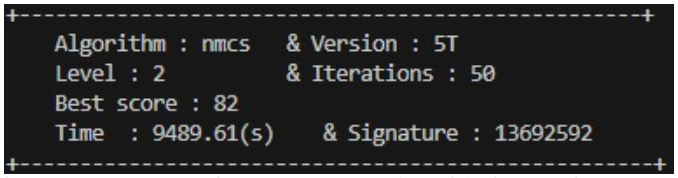

Figure 2: NMCS Performance at Level 2 with 50 Iterations - Test2

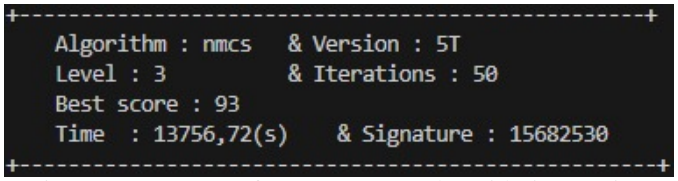

Figure 3: NMCS Performance at Level 3 with 50 Iterations

Note: The number of iterations is small and the tests were conducted with limited resources.

Performance: NMCS demonstrated expected solid performance in achieving high scores, with 90 at Level 2 and 93 at Level 3 on a very small number of iterations. The ability to reach these scores suggests that NMCS is effective in exploring the search space and finding optimal or near-optimal solutions, as anticipated for one of the most robust search methods. However, this comes at a significant computational cost.

Computation Time: The time taken for NMCS to complete the computations is considerably high. For Level 2, it took 6218.94 seconds, and for Level 3, it took 13756.72 seconds. This indicates that while NMCS is powerful, it is also computationally intensive, which can be a major drawback in scenarios with limited computational resources, as expected for such a comprehensive search technique. Given that the environment in which these tests were conducted had very limited resources, this high computational time is not surprising.

5.2 NRPA

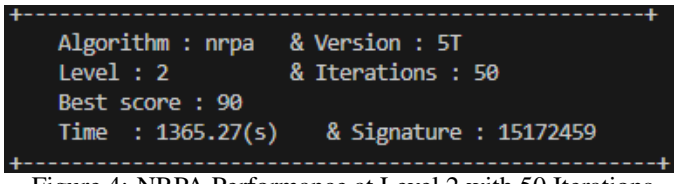

Figure 4: NRPA Performance at Level 2 with 50 Iterations

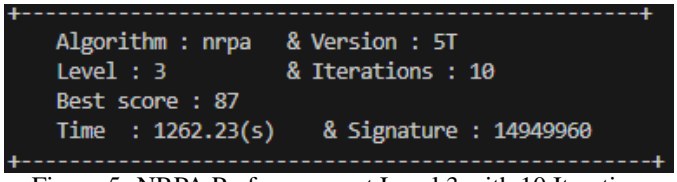

Figure 5: NRPA Performance at Level 3 with 10 Iterations

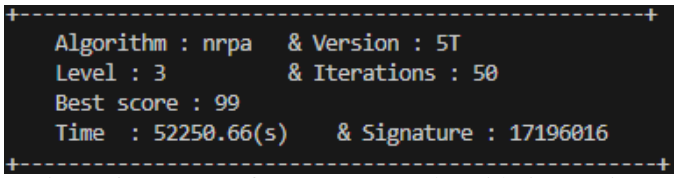

Figure 6: NRPA Performance at Level 3 with 50 Iterations

Note: The number of iterations is small and the tests were conducted with limited resources.

Performance: NRPA showed expected efficient performance with relatively shorter computation times while maintaining competitive scores. At Level 2, NRPA achieved a best score of 90, matching the NMCS result but in significantly less time. The Level 3 test with fewer iterations (10 iterations) yielded a best score of 87, indicating potential for higher scores with more iterations. Additionally, with 50 iterations at Level 3, NRPA achieved an impressive best score of 99. This demonstrates NRPA's ability to quickly reach good solutions, which is anticipated given its design. It suggests that with more iterations, NRPA could potentially achieve even better results.

Computation Time: The time taken for NRPA is much shorter compared to NMCS. For Level 2, it took 1365.27 seconds, and for Level 3, it took 1262.23 seconds with only 10 iterations. However, for 50 iterations at Level 3, the computation time increased to 52250.66 seconds. This indicates that while NRPA remains efficient, its computation time can increase significantly with more iterations. Nonetheless, it still proves to be more efficient than NMCS, especially in scenarios where computational resources are limited, aligning with its reputation for efficiency.

Scalability: NRPA's performance and efficiency suggest that it can handle more iterations and higher levels more effectively within limited computational resources. Despite the longer computation time at higher iterations, NRPA remains a versatile and scalable option for optimizing game strategies and solving complex combinatorial problems. This is expected for one of the most resource-

efficient methods. Studies have indicated that NRPA scales well in comparison to other search methods, providing a good balance between performance and computational demands.

5.3 Enhanced NRPA

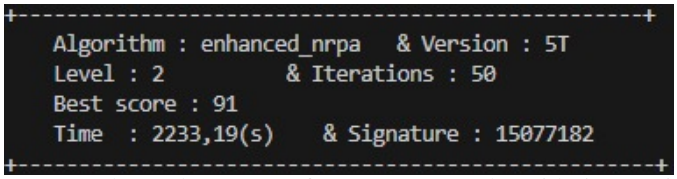

Figure 7: Enhanced NRPA Performance at Level 2 with 50 Iterations

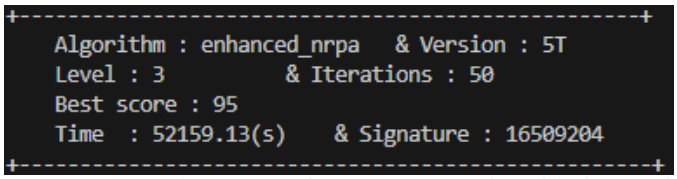

Figure 8: Enhanced NRPA Performance at Level 3 with 50 Iterations

Note: The number of iterations is small and the tests were conducted with limited resources.

Performance: Enhanced NRPA outperformed the standard NRPA in terms of the best score achieved at Level 2, with a score of 91. This indicates that the enhancements, such as incorporating MCTS principles, contributed to a more refined and effective search process. At Level 3, Enhanced NRPA achieved a best score of 95 with 50 iterations, further demonstrating its potential to achieve better results with similar computational efforts, even in environments with limited resources. The improvements suggest that Enhanced NRPA can achieve competitive scores while refining the search process effectively.

Computation Time: The time taken for Enhanced NRPA at Level 2 was 2233.19 seconds, which is longer than standard NRPA but still significantly shorter than NMCS. At Level 3, the computation time for Enhanced NRPA was 52159.13 seconds with 50 iterations. This indicates that while Enhanced NRPA is more computationally intensive than standard NRPA, it remains more efficient than NMCS. Given the limited computational resources in the testing environment, this efficiency is promising. However, further experimentation is needed to confirm its scalability and efficiency in more resource-constrained settings.

Scalability: Enhanced NRPA balances performance and computation time well, suggesting it can handle more complex scenarios efficiently. The integration of MCTS principles helps in achieving a better exploration-exploitation balance, making it suitable for higher levels and more iterations. This suggests potential for scalability even with limited resources, although further testing is required to verify this in larger-scale applications. Studies have indicated that integrating MCTS principles can improve search efficiency, supporting the hypothesis that Enhanced NRPA could perform well in more demanding scenarios.

6 Conclusion

This study evaluated the performance of three algorithms: NMCS (Nested Monte Carlo Search), NRPA (Nested Rollout Policy Adaptation), and Enhanced NRPA, under conditions with limited computational resources and a small number of iterations.

Performance: All three algorithms demonstrated their known strengths in exploring the search space effectively. NMCS showed robust performance with high scores but at a significant computational cost. NRPA achieved competitive scores more efficiently, highlighting its design for quick convergence to good solutions. Enhanced NRPA, incorporating MCTS principles, showed potential for even better results, suggesting its enhancements contributed to a more refined search process. However, these results were obtained under limited resources, and further testing is necessary to confirm their generalizability.

Computation Time: NMCS required substantial computation time, consistent with its comprehensive search approach. NRPA was significantly faster, making it a more efficient option, particularly in resource-constrained environments. Enhanced NRPA, while more computationally intensive than standard NRPA, remained more efficient than NMCS. These findings reflect each method's computational demands and efficiency.

In conclusion, while these preliminary results are promising, they are based on tests with limited iterations and computational resources. Therefore, it is essential to interpret these findings cautiously and acknowledge the need for further investigation to validate the scalability and efficiency of these algorithms in broader applications. Each algorithm's performance aligns with its known characteristics, providing a foundation for future research and optimization.

References

[1] Tristan Cazenave. Nested Monte Carlo Search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[2] Christopher D. Rosin. Nested Rollout Policy Adaptation for Monte Carlo Tree Search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[3] Hui Wang. Tackling Morpion Solitaire with AlphaZero-like Ranked Reward Reinforcement Learning. Available at arXiv: `https://arxiv.org/abs/2103.04931`, 2021.

[4] Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Computers and Games*, pp. 72-83, 2006.

[5] Sylvain Gelly and David Silver. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artificial Intelligence*, 175(11):1856-1875, 2011.

[6] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Phil Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1-43, 2012.

[7] Guillaume Chaslot, Mark Winands, H. Jaap van den Herik, Jos Uiterwijk, and Bruno Bouzy. Monte-Carlo Tree Search: A New Framework for Game AI. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pp. 216-217, 2008.

[8] Mark H. M. Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte Carlo Tree Search in Lines of Action. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):239-250, 2010.