**Certainly! Here’s the full, expanded 3-page report incorporating the details from the `Standard8QueenGA` and `OptimisedExtendedGA` implementations, as well as the explanations for the Horses Puzzle, Crowded Queens Puzzle, and the theoretical analysis.**

**---**

**# Genetic Algorithm Implementation for the 8-Queens Problem and Extensions**

**## Introduction**

**This report outlines the design, implementation, and optimization of a genetic algorithm (GA) to solve the classic 8-queens puzzle. The GA framework is based on evolutionary principles, such as selection, crossover, and mutation, which iteratively evolve solutions to the problem. In addition to the standard 8-queens problem, I extended the algorithm to solve a Horses puzzle and a more complex 20x20 Crowded Queens puzzle. Optimizations, including hyperparameter tuning, smart mutation, and custom crossover strategies, further enhanced the performance of these implementations. Finally, the report provides a theoretical comparison with other evolutionary strategies.**

**## 1. Basic Implementation and Hyperparameter Tuning**

**The initial implementation of the GA, encapsulated in the `Standard8QueenGA` class, aimed to solve the 8-queens problem by reducing conflicts among queens on an 8x8 board. Each potential solution (chromosome) in the population represents a board configuration, and the algorithm iteratively selects and evolves these configurations to achieve a solution.**

**### Population Initialization and Fitness Function**

**Each chromosome is represented by an array of integers, where each element denotes the row position of a queen in a specific column. The fitness function evaluates a configuration by counting non-attacking pairs of queens, giving higher fitness scores to configurations with fewer conflicts.**

**### Hyperparameter Tuning**

**To improve efficiency, I conducted a systematic hyperparameter tuning process using the `optimise\_parameters` function. The primary parameters considered were population size and mutation probability. The function evaluated each combination of these parameters by running multiple trials, measuring the average generations needed to reach a solution, and assessing runtime. The tested parameter ranges included:**

**- \*\*Population Sizes\*\*: [20, 50, 100, 250, 500]**

**- \*\*Mutation Probabilities\*\*: [0.01, 0.03, 0.08, 0.15, 0.3]**

**Results indicated that a population size of 250 and a mutation probability of 0.15 yielded the best performance, achieving a balance between diversity and convergence speed. Larger populations and moderate mutation rates helped the algorithm avoid local optima, enhancing solution efficiency. This optimized configuration provided a solid foundation for tackling more complex puzzles, as detailed in subsequent sections.**

**## 2. Extensions and Optimizations**

**### Advanced Crossover and Mutation Techniques**

**In the `OptimisedExtendedGA` class, I implemented advanced crossover and mutation techniques to handle more complex configurations. The main enhancements include a fixed 50% crossover and a strategic mutation approach, `smart\_mutate`.**

**- \*\*50% Crossover\*\*:**

**Instead of a single-point crossover, the algorithm creates offspring by combining half of each parent’s genes. This approach promotes diversity by ensuring each child inherits a balanced mix of traits, increasing the chance of evolving promising configurations.**

**- \*\*Smart Mutation\*\*:**

**The `smart\_mutate` function targets queens with the highest number of conflicts, moving them to positions with fewer attacks based on trial placements. This selective mutation focuses on problematic areas of each chromosome, significantly accelerating convergence. For instance, in a large board configuration, random mutation may struggle with local optima, whereas smart mutation directly addresses high-conflict areas.**

**### Horses Puzzle Implementation**

**As an extension of the GA, I implemented the Horses puzzle, which involves placing multiple horses on a chessboard without threatening each other based on knight-like “L” moves. This variation required unique handling of positions and movement constraints, encapsulated in the `HorsesState` class.**

**- \*\*Population and Fitness\*\*:**

**Each configuration in this puzzle is represented by an array of coordinates, one for each horse. The fitness function calculates the number of non-attacking pairs by checking each pair of horses and determining if any can attack each other based on knight moves. This constraint-based fitness evaluation introduces computational complexity, as the algorithm needs to verify both row and column uniqueness.**

**- \*\*Mutation Strategy for Horses\*\*:**

**The `simple\_horse\_mutate` function allows each horse to change positions on the board without overlap. This mutation strategy ensures that no two horses occupy the same position, respecting the unique positioning requirements of knight-like movement. This approach was essential for handling the discrete board structure of the Horses puzzle, where overlapping positions would invalidate the solution.**

**### Crowded Queens Puzzle on a 20x20 Board**

**To test the scalability of the GA, I extended the algorithm to solve a 20x20 Crowded Queens puzzle, involving 20 queens on a 20x20 board. This larger configuration significantly increases the solution space, necessitating strategic enhancements to the GA’s core functions.**

**- \*\*Adaptations for Larger Board Size\*\*:**

**I recalculated the goal fitness to fit the 20x20 board size, setting it based on non-attacking pairs for 20 queens. Additionally, the algorithm required a larger population (250) and a mutation probability of 1.0 to maintain genetic diversity on a larger board. These values were carefully tuned to balance solution quality with runtime efficiency, ensuring that the GA could explore the vast solution space without premature convergence.**

**- \*\*Smart Mutation with High Mutation Rates\*\*:**

**For the Crowded Queens puzzle, `smart\_mutate` proved especially useful. The algorithm identified queens in high-conflict positions and strategically repositioned them, a necessity given the large search space. High mutation rates further contributed to genetic diversity, reducing the risk of the algorithm becoming stuck in local optima. This approach allowed the GA to converge on solutions within a feasible number of generations, even for this complex configuration.**

**## 3. Theoretical Analysis and Comparison with Evolutionary Strategies**

**### Genetic Algorithms (GA) and Complexity Analysis**

**Genetic algorithms are well-suited to combinatorial problems like the 8-queens puzzle, where discrete configurations and constraints increase the complexity of the solution space. The optimizations used in `OptimisedExtendedGA`—such as `smart\_mutate` and power-aware mutation in Evolving Queens—introduce computational overhead but reduce the total number of generations needed for convergence. This added complexity is offset by the algorithm’s accelerated convergence rate, making it more efficient for large-scale configurations.**

**For instance, the strategic mutation techniques used in `smart\_mutate` reduce overall runtime by prioritizing high-conflict queens, while 50% crossover enhances exploration by mixing traits effectively. Although these advanced techniques require more computation per generation, they yield a lower total generation count, achieving efficient solutions with minimal resource usage.**

**### Comparison with Evolutionary Strategies (ES)**

**Evolutionary Strategies (ES) is another optimization algorithm similar to GA but operates differently in continuous spaces, focusing primarily on mutation. While GAs use selection, crossover, and mutation to evolve solutions, ES relies on mutation with self-adaptive rates that evolve alongside solutions. This self-adaptation is particularly advantageous for continuous problems, such as parameter tuning for neural networks, where mutation-based exploration allows ES to fine-tune solutions effectively without crossover.**

**In contrast, the 8-queens and Crowded Queens puzzles require discrete, constraint-based solutions, making GAs a more suitable choice. For example, in large-scale routing applications, GAs dynamically adapt paths to minimize latency, a flexibility beneficial for complex logistical optimization. In these cases, ES would lack the required structure to maintain valid configurations, whereas GA excels by evolving discrete configurations while adhering to specific constraints.**

**### Real-World Application of Genetic Algorithms**

**Genetic algorithms have seen widespread use in industries like logistics and network optimization. A notable application is in optimizing vehicle routing, where GAs minimize delivery times and operational costs by dynamically evolving routing paths. This adaptability allows GAs to outperform traditional algorithms, especially in cases where diverse constraints (e.g., traffic, distance, delivery windows) need to be optimized simultaneously. This flexibility highlights the GA’s utility in dynamic, constraint-driven environments, showcasing its broader applicability beyond theoretical puzzles.**

**## Conclusion**

**The genetic algorithm developed in this project successfully solved the 8-queens puzzle and extended to tackle the Horses and Crowded Queens puzzles. Through strategic parameter tuning and advanced techniques, such as smart mutation, power-aware mutation, and optimized crossover, the GA demonstrated its capability to handle complex, constraint-based puzzles. The theoretical comparison underscores the GA’s suitability for discrete configurations, where its evolutionary approach can efficiently navigate high-dimensional spaces. The results from both the Horses and Crowded Queens extensions emphasize the GA’s adaptability, validating it as a versatile tool for solving real-world optimization problems across diverse domains.**

**---**

**This report integrates the technical enhancements and theoretical insights from the `Standard8QueenGA` and `OptimisedExtendedGA` implementations, showcasing the algorithm’s adaptability and performance in both standard and extended configurations. Let me know if you'd like further customization or additional details in specific sections!**