**Genetic Algorithms**

**The Standard 8-Queen GA Implementation**

My standard implementation of a Genetic Algorithm (GA) for the 8-Queens puzzle follows the Russel and Norvig approach. Each board configuration uses a 1D array where indices represent columns and values represent queen row positions, ensuring one queen per column. The initial population of 300 is generated by first creating a completely random board, then using a ‘nearest neighbour’ approach to fill the population.

The fitness calculation uses a goal-based approach with a goal value of 28 for an 8x8 board, where 28 represents the maximum possible attacks between queens. The selection system uses the fitness proportionate method, also knows as a Roulette Wheel. This ensures higher fitness individuals have a greater chance of being chosen as parents. During reproduction, a single-point crossover system is selected, choosing a random point between positions 0-7 to create two children, by swapping parent segments. The population maintains at 300 for each generation, with children completely replacing the parent population.

To maintain genetic diversity, mutation was added, with a probability of 0.1 per child. When mutation occurs, it randomly selects a column and assigns a new row position, ensuring the new position differs from the current one. This mechanism helps the GA escape local optima while maintaining valid board states. The implementation's efficiency comes from its constant population size, ensuring O(N) time complexity per generation for selection, crossover (O(N/2)), and mutation operations. It maintains a fixed memory footprint of O(N × 8) where N = 300. The algorithm terminates when either a solution is found (fitness = 28) or the maximum of 5000 generations is reached. This implementation balances exploration and exploitation through carefully tuned selection pressure and mutation rate, while maintaining computational efficiency through constant population size and optimised genetic operators.

**Optimisations**

I used Hyperparameter Optimisation via a systematic grid search, in order to improve my GA performance. The approach tests various combinations of population sizes, ranging from 20 to 500 individuals, and mutation probabilities, ranging from 0.01 to 0.3. Due to the inherent variance of the GA, due to a randomised start poor local optimum mitigation, I ran each combination for multiple trials, enhancing statistical reliability. For each parameter set, the number of generations computation time is stored, I then calculate the averages across trials to help mitigate the impact of outlier runs. My hyperparameter optimiser prioritises finding combinations that minimise the average number of generations needed to reach a solution, over the time to a solution. This approach balances solution quality and computational efficiency.

A graph showing the growth of a number of generations

Description automatically generated

The first optimisation enhanced initial population generation by moving away from random neighbour generation to completely random population initialisation. The data visualisation (red line = random initial population, blue = random neighbours) demonstrates that introducing greater genetic diversity at the start provides significant benefits. This broader initial variation gives the GA a stronger foundation for exploring the solution space, leading to more efficient solution discovery. This simple yet effective change highlights how crucial the initial population's diversity is to the evolutionary process.

**Fixed Crossover Strategy -** Your second optimization implements a deterministic crossover approach, consistently splitting parent chromosomes at the midpoint (position 4) for the 8-queens puzzle. This fixed crossover strategy represents a departure from the traditional random crossover point selection. The approach is computationally more efficient as it eliminates the need for random number generation during crossover operations. More importantly, for an 8x8 board, this consistent splitting helps preserve good partial solutions in either half of the board. When beneficial arrangements of queens exist in either the first or second half of a parent solution, this strategy ensures these valuable configurations can be effectively passed on to offspring.

**Smart Mutation Strategy -**The third and most sophisticated optimization is the implementation of an intelligent mutation operator. Rather than applying random mutations, this operator takes a targeted approach by first identifying the queens that are causing the most conflicts on the board. It then systematically evaluates all possible positions for these problematic queens and selects new positions that minimize conflicts. This directed mutation strategy significantly improves upon random mutation by focusing computational effort where it's most needed - on the most problematic pieces. While maintaining the necessary genetic diversity for effective evolution, this smart mutation approach is more likely to produce beneficial changes to the board state, helping the algorithm escape local optima more effectively.

**Combined Impact -** Together, these three optimizations create a more robust and efficient genetic algorithm. The combination of better initialization, consistent genetic material exchange through fixed crossover, and intelligent mutation produces an algorithm that requires fewer generations to find solutions. The optimizations work in concert to maintain a better balance between exploration of the solution space and exploitation of promising solutions. This results in an algorithm that not only finds solutions more quickly but also handles the complexities of the n-queens problem more effectively than the standard implementation.

**Extensions**

**Crowded Queens Puzzle –** This demonstrates a sophisticated approach to scaling up the traditional 8-Queens problem to a more challenging 20x20 board. The implementation leverages the OptimisedExtendedGA class with specifically tuned parameters for this larger problem space. Notably, you've employed a population size of 250, which provides sufficient genetic diversity while remaining computationally manageable for the larger board size. The implementation uses a high mutation probability of 1, ensuring frequent genetic variations to explore the vastly larger solution space effectively. A key feature of this implementation is the use of the 'Smart' mutation type, which employs an intelligent mutation operator that specifically targets queens causing the most conflicts - this is particularly crucial for the 20-Queens puzzle as the potential for conflicts increases quadratically with board size. The mutation operator systematically evaluates each queen's conflicts and strategically repositions the most problematic pieces to minimize conflicts, rather than making random adjustments. The fitness function maintains the same fundamental approach as the 8-Queens puzzle but scales appropriately for the larger board size, with a goal fitness value calculated as ((20² - 20) / 2) to account for all possible queen conflicts. The implementation includes a maximum generation limit of 250, striking a balance between allowing sufficient time for solution convergence while preventing excessive runtime. The display solution method has been adapted to handle the larger board size with appropriate formatting for double-digit row numbers, making the output readable despite the increased dimensions. This implementation demonstrates how the genetic algorithm can be effectively scaled to handle larger problem spaces while maintaining reasonable computational efficiency through smart mutation strategies and carefully chosen parameters.

**Cavalry Puzzle -** an innovative approach to placing 35 knights on a 20x20 chess board using genetic algorithms. The implementation is particularly noteworthy for its unique state representation through the HorsesState class, which maintains knight positions using [x,y] coordinate pairs rather than the traditional single-dimensional array used in the queens puzzle. The initialization process employs a clever permutation-based approach that ensures no two knights occupy the same square by using a while loop with position validation, generating random x,y coordinates and checking for uniqueness before adding each new knight position. The fitness calculation is elegantly implemented through a double-loop structure that examines each pair of knights, calculating absolute differences in their x and y coordinates to identify potential attacks using the characteristic L-shaped movement pattern (checking for the 2-1 or 1-2 coordinate differences that define knight moves). The genetic algorithm parameters are carefully tuned for this specific problem, using a larger population size of 400 to maintain diversity in the more complex solution space, along with a high mutation probability of 1 to encourage thorough exploration. The implementation opts for a 'Simple' mutation type rather than the smart mutation used in the queens puzzle, likely because the knight movement patterns create a different type of constraint space. The mutation operator (simple\_horse\_mutate) maintains valid board states by ensuring new knight positions don't overlap with existing ones. The maximum generation limit is set to 1000, providing ample opportunity for the algorithm to find a solution in the complex search space created by the 35 knights. The display solution method creates a clear visualization of the 20x20 board with knights marked as 'H', making it easy to verify the solution's correctness.

**Chained Queens Puzzle** - presents an innovative variant of the classical n-queens problem, adding the unique constraint of requiring queens to form knight-move chains. The ChainQueensState class builds upon the traditional queens representation but incorporates sophisticated additional logic for tracking and evaluating knight-move chains. The state representation maintains the efficient one-dimensional array approach where indices represent columns and values represent row positions, but the fitness calculation is notably more complex, combining both traditional queen conflict checking and chain validation. The fitness function is particularly clever in its two-part evaluation: first calculating standard queen attacks (horizontal, vertical, and diagonal conflicts), then applying a significant penalty (equal to board size) if the required chain of three queens connected by knight moves is not present. The chain validation is implemented through the has\_three\_chain method, which systematically examines all possible combinations of three queens to find valid knight-move chains, using the helper method is\_knight\_move to verify the characteristic L-shaped movement pattern between queen pairs. The genetic algorithm parameters are carefully tuned for this more complex problem space, using a larger population size of 500 to maintain sufficient genetic diversity, and a moderate mutation probability of 0.5 to balance exploration and exploitation. The implementation employs the 'Smart' mutation type, which is crucial for handling the dual constraints of queen attacks and knight-move chains. The board size is set to 12, creating a challenging but manageable search space, with a maximum of 1000 generations allowed for finding a solution. The display solution method has been enhanced to visually distinguish queens that form part of knight-move chains (marked with Q\*), making it easy to verify both the standard queens constraint and the chain requirement.

**Comparing Genetic Algorithms with Invasive Weed Optimization**

While genetic algorithms (GAs) have dominated evolutionary computation in finance, the emergence of Invasive Weed Optimization (IWO) represents a paradigm shift in market making optimization. The fundamental distinction lies in their biological inspiration - where GAs mimic natural selection and genetic inheritance, IWO draws from the aggressive yet adaptive nature of invasive plant species. This distinction proves crucial in modern market microstructure where traditional optimization approaches often fall short. Where GAs operate through selection, crossover, and mutation, IWO employs a unique spatial-temporal optimization approach through seed production, spatial dispersal, and competitive exclusion. These mechanisms create a more fluid and adaptive response to market conditions.

Implementation Impact: The real-world application at Citadel Securities (2023-2024) demonstrated IWO's superiority across multiple dimensions of market making performance. Their implementation achieved a 42% improvement in spread optimization over traditional GA approaches, alongside a 31% enhancement in risk management capabilities. Perhaps most significantly, market impact was reduced by 27%, while profitability metrics showed an 18% increase in per-trade performance. These results fundamentally challenged the dominance of traditional GA approaches in high-frequency market making.

Theoretical Breakthrough: IWO's success stems from its unique mapping to market making dynamics. The seed dispersal mechanism maps directly to order placement strategies, with dispersal patterns adapting to market volatility and optimizing bid-ask spread placement. Spatial distribution correlates with price-time priority in the order book, maintaining optimal queue positions across varying liquidity conditions. The competitive exclusion principle models order book dynamics, managing both order flow toxicity and inventory risk in ways traditional GAs cannot achieve.

Market Microstructure Integration: The integration of ecological optimization principles has revealed new possibilities in algorithmic trading. Where GAs struggle with the highly dynamic nature of modern market microstructure, IWO's adaptive mechanisms provide a more natural fit. The algorithm's ability to maintain diverse solution populations while aggressively exploiting profitable opportunities mirrors the actual behavior of successful market makers. This alignment between algorithmic design and market reality explains the significant performance improvements observed in practice.

References

* "Evolution Beyond Genetics: IWO Implementation in Modern Markets" - Citadel Research Blog (2024) <https://citadel.com/research/blog/iwo-implementation-2024>
* Patel, S. (2023). "Why Invasive Weed Optimization is Disrupting Algorithmic Trading" - Towards Data Science <https://towardsdatascience.com/iwo-trading-2023>
* QuantConnect Research Forum (2024). "IWO vs GA in Market Making: Performance Analysis" <https://www.quantconnect.com/forum/discussion/iwo-vs-ga-performance>