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 achieves computational efficiency through its fixed population size N = 300. Each generation has a time complexity of O(N), which breaks down into O(N) for selection, O(N) for crossover operations (processing N/2 pairs), and O(N) for mutation checks. The space complexity remains constant at O(N × 8), storing the population of N chromosomes each representing 8 queens. 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 parameters.

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, 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 generatedMy first optimisation changes how the initial population is created, by moving away from random neighbour generation to a completely random population initialisation. The data visualisation runs each approach over 30 trials (red line = random initial population, blue = random neighbours). This show that by 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.

A graph showing the growth of a number of generations

Description automatically generatedMy second optimisation implements a fixed crossover approach, where half of each parents' 'DNA' is shared during reproduction (red line = fixed half way crossover, blue = random crossover). This approach is computationally efficient by eliminating random number generation during crossover. For an 8x8 board, this consistent splitting aligns with the Building Block Hypothesis - where complex problems are solved by preserving and combining well-fitted solution components. In N-Queens, queens in different board halves have reduced diagonal interactions, making these halves semi-independent. When beneficial arrangements exist in either half of a parent solution, this fixed strategy ensures these valuable partial solutions are preserved and effectively passed to the offspring, while providing structured exploration of the search space.

A graph showing the growth of the generation

Description automatically generatedAn optimisation I attempted to implement was Elitism. This is where a subset of the fittest in the population are carried over to the next generation. I tested multiple Elitism rates to see which could lead to an optimisation. However, none were successful. The chart shows the sub optimal GA (red line =1 percent elitism in the population, blue = no elitism). This led me to look further and plot the fitness histories of these trials. An interesting observation emerged, where I saw little ‘oscillation’ in the fitness over the generations, for trials with any form of elitism. The reduced performance with elitism appeared to be due to premature convergence, where preserving the best solutions actually limited the algorithm's ability to explore the full solution space. The observed lack of fitness oscillation suggests that elitism was overly constraining the evolutionary process, preventing the algorithm from escaping local optima through natural fitness fluctuations.

Multiple other optimisations were tried, but they all seemed to slow down solution convergence. I tried various cross over techniques and adaptive mutations, where I altered the mutation rate based off fitness stagnation. I also tried to scale mutation as the diversity of the population decreased. I concluded that the space complexity was larger for these attempted optimisations and they added computation slowed down convergence.

Extensions

The Crowded Queens Puzzle

This extension scales up the 8-Queens puzzle to a 20x20 board. Through some hyperparameter optimisation, a population size of 250 was selected, along with the mutation probability of 1. This allows for frequent variation to explore the significantly larger solution space. Due to the fact that queen attacks grow quadratically with the board size, a ‘Smart’ mutation function was built. By targeting queens with the highest attack counts and strategically repositioning them, the ‘Smart’ mutator enabled the GA to converge to a solution in a reasonable time. The GA has a 250-generation limit, providing sufficient time for convergence while managing runtime. This implementation demonstrates effective scalability for larger problem spaces.

The Cavalry Puzzle

This puzzle uses horses instead of queens, with the same game logic in mind, placing all the horses where none can attack each other. The HorsesState class handles the state and fitness of each individual using [x, y] coordinate pairs, over the 1D arrays used in the 8-Queens puzzle. I selected 35 knights to be placed without any attacks on a 20x20 board. Although seeming arbitrary, I looked to demonstrate complexity without hampering time computation time. The fitness calculation is examining each pair of horses, calculating absolute differences in their x and y coordinates to identify potential attacks (checking for the 2-1 or 1-2 coordinate differences that define the horse moves). The GA parameters were tuned for this specific puzzle, 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 uses a simple\_horse\_mutate function that creates a random mutation of a horse on the board.

The Chained Queens Puzzle

This extension adds an additional constraint to the queens. At least three queens must form a horse-move chain. The ChainQueensState class implements logic for both the standard queen placement and a horse-chain validation. The fitness function first checks for traditional queen conflicts, then applies a board-size penalty if there isn't a chain of three queens connected by horse moves. To handle this more complex problem space, the GA uses a population size of 500 for genetic diversity and a mutation probability of 0.5 to balance exploration and exploitation. The implementation uses the 'Smart' mutation type, operates on a 12x12 board, and runs for up to 1000 generations. A chained queen is depicted with a ‘C’ on the display board and unchained with a ‘Q’.

Genetic Algorithms and Particle Swarm Optimisations

The Particle Swarm Optimisation (PSO) algorithm is inspired by the collective behaviour of flocking birds. The algorithm operates by simulating a ‘swarm’ of particles (potential solutions) that move through the search space to find an optimal solution, with each particle adjusting its trajectory based on its own experience and the experience of its neighbours.

The PSO starts with a set of particles that are randomly distributed throughout the search space, each assigned both a position and velocity. These particles are then evaluated using a fitness function that measures how well each particle solves the optimisation problem. During optimisation, each particle maintains memory of its personal best position based on its individual experience, while the entire swarm collectively tracks the global best position discovered by any particle. The algorithm continually updates each particle's velocity based on its distance from both its personal best and the global best positions, carefully balancing exploration of new areas with exploitation of known good solutions.

PSO's movement mechanics are controlled by parameters that influence how particles explore the search space and converge on solutions. The algorithm iteratively updates the particles based on inertia, cognitive attraction to personal best positions, and social attraction to the global best position. PSO is known to be particularly valuable for complex, high-dimensional problems.

Genetic Algorithms (GAs) and PSOs are both inspired by nature, however they both approach problem solving in distinctly different ways. GAs draw from biological evolution, using selection, crossover, and mutation to evolve solutions across generations. Whereas PSOs simulates social behaviour, with particles moving through the solution space guided by personal and global best positions, similar to how birds flock or fish school.

The key differences are ultimately how they get to their solutions. GAs create new generations and don't maintain a memory between each generation. This is due to them relying on both fitness-based selection and genetic diversity to explore the solution space. On the other hand, PSOs continuously update particle positions and maintain a memory of both personal and global best positions. Also, PSOs generally require less parameter tuning compared to GAs, which may need tuning during initialisation, mutation, crossover, and selection.

Genetic Algorithms in Medicine

A very notable achievement by GAs has been in breast cancer detection, through mammography screening. GAs have proven to be especially valuable to help radiologists analyse large amounts of imaging data, both more efficiently and accurately. GAs have demonstrated their ability to detect edge cases in various imaging modalities (MRI, CT, and ultrasound). These detections of microcalcifications in mammograms, and in image segmentations, have proven to improve breast cancer detection.

GAs have also proven to be effective in heart disease diagnosis. Some systems have been developed that combine GAs with multilayered neural networks to predict cardiovascular disease risk. These systems were particularly effective because they could optimise neural network weights in fewer iterations, making the diagnostic process more efficient and accurate.

GAs have also had success during brain tumour detection. The algorithms have been implemented in a comprehensive process that includes image capture, pre-processing, filtration, and image segmentation before applying GA techniques for final analysis. In the specific area of image segmentation, a breakthrough GA was built that treated segmentation as a global optimisation problem. This algorithm utilised a fitness function that considered multiple factors including image similarity, pixel intensity, and spatial positioning.

It is undeniable that GAs have a unique property in searching a complex space. These medical advancements have been driven by the pioneering research of algorithms that operate on the basis of nature. With a subtle irony of using nature to fight nature.

References

* Adıgüzel Mercangöz, B. (ed.) (2021) *Applying Particle Swarm Optimization: New Solutions and Cases for Optimized Portfolios*. International Series in Operations Research & Management Science, vol. 306. 1st edn. Cham: Springer. doi:10.1007/978-3-030-70281-6.
* Brownlee, J., 2023. *A Gentle Introduction to Particle Swarm Optimization*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/>
* AnalytixLabs, 2023. *A Complete Guide to Genetic Algorithm: Advantages, Limitations & More*. [online] Medium. Available at: <https://medium.com/@byanalytixlabs/a-complete-guide-to-genetic-algorithm-advantages-limitations-more-738e87427dbb>
* Ghaheri, A., Shoar, S., Naderan, M., & Hoseini, S.S., 2015. *The applications of genetic algorithms in medicine*. *Oman Medical Journal*, 30(6), pp.406–416. Available at: <https://pmc.ncbi.nlm.nih.gov/articles/PMC4678452/>
* Mourya, A.K., Tyagi, P., & Bhatnagar, A., 2016. *Genetic Algorithm and their applicability in Medical Diagnostic: A Survey*. *International Journal of Scientific & Engineering Research*, 7(12), pp.1143–1145. Available at: <http://www.ijser.org>
* Wikipedia, 2024. *Genetic algorithm*. [online] Available at: <https://en.wikipedia.org/wiki/Genetic_algorithm>