Further AI

Eight Queens Assignment

Jake West-Gomila

November 19, 2025

1 Introduction

This report describes the implementation of the eight queens problem, and several extensions of it, using genetic algorithms. The eight queens problem involves placing eight queens on a chessboard such that none attack one another. The problem is computationally challenging to solve by brute force because of the vast number of possible configurations, $\frac{64!}{56!}$, motivating the use of more efficient search strategies.

Genetic algorithms, inspired by natural selection, favour individuals whose "genes" confer advantageous characteristics or behaviours, increasing the likelihood that these traits propagate through successive generations (Ruse, 1975). In this context, selection, reproduction and mutation serve as computational analogues of evolutionary processes, enabling a population of candidate solutions to improve over time and converge towards a desirable state (Ponce-Cruz and Ramírez-Figueroa, 2010).

The report first details the base implementation and evaluates its performance on the eight queens problem. It then examines several optimisations and extensions, including hyperparameter tuning, generalising the approach to the n-queens problem and exploring an evolutionary mechanism based on viral infection.

2 Basic Implementation

The eight queens problem has only 92 distinct solutions when ignoring symmetry, despite the large number of possible placements. Because the problem has a low solution density, a poorly designed genetic algorithm effectively performs a near random, or even less efficient, search over the space. The basic implementation refines the provided EightQueensState model by removing unused methods and simplifying the cost function. It applies the core components of a genetic algorithm: selection, crossover, mutation and iterative evolution of the population.

The fitness function is defined as the difference between the maximum possible number of attacking queen pairs and the number actually present, with a squared fitness term used to reinforce the advantage of highly fit individuals. Roulette selection enforces this proportional preference during parent choice, while elitism ensures that the top fraction of individuals carry over unchanged, preserving strong genetic material that might otherwise be degraded by random variation. The population then gradually evolves toward a valid solution by selecting fitter candidates more frequently for reproduction. Mutation is also applied to maintain diversity and stop the algorithm getting stuck in local minima. Throughout the implementation, unnecessary computations are avoided where possible, but the primary emphasis remains on reducing time complexity by pruning the search space (Figure 1).

3 Advanced Implementation

3.0.1 Permutation State Representation

The first enhancement is to represent the chessboard as a permutation, recognising that each queen must occupy a unique row. This reduces the search space to 8! possible configurations and removes all row and column conflicts, leaving only diagonal conflicts to address. In practice, this substantially accelerates convergence and improves scalability to larger n-queens instances: for example, the runtime for the eight queens problem decreased from $2.39s$ to $0.01s$ under otherwise identical algorithmic conditions, a reduction of 99.6%.

Maintaining the permutation invariant introduces some complexity, particularly for crossover and mutation operators. Mutation is adapted to swap two row values, ensuring the representation remains a valid permutation. Crossover is more challenging: several operators exist in the literature, but order crossover was adopted here as it preserves partial ordering across generations and maintains valid permutations (Moraglio and Poli, 2011).

3.0.2 Empirically Derived Time Complexity

Using a maximum of $20,000$ generations, the time complexity was estimated by measuring the runtime of the permutation-based genetic algorithm for increasing values of n and fitting a line of best fit to the resulting data (Figure 2). The fitted curve indicates exponential growth in runtime as n increases.

3.0.3 Hyperparameter Tuning

Hyperparameter tuning is performed by varying the algorithm's parameters and evaluating the resulting performance. Because genetic algorithms rely on several stochastic processes, certain initial populations may, by chance, favour particular hyperparameter settings. To minimise this sensitivity and obtain results that are robust to random initialisation, each configuration is evaluated ten times using ten different random seeds, and the mean time required to reach a solution is used as the performance metric.

In general, larger population sizes tend to increase runtime and runtime variance, though notable counterexamples appear in the results. As shown in Figure 3, identifying optimal hyperparameters without systematic experimentation is challenging, since, aside from population size, there is no clear relationship between parameter settings and performance. Higher runtimes also show substantially greater variability, suggesting that hyperparameters introducing more randomness may result in less consistent performance. This observation aligns with the motivation for the next section, which explores adding greater directionality through viral infection.

3.0.4 Viral Infection

Viral infection is introduced to address a key limitation of standard genetic algorithms: beyond selection, they lack directionality. Crossover and mutation introduce variation but do so blindly, causing promising partial solutions to be repeatedly found and lost. Constraint satisfaction techniques can help mitigate this, and Kanoh et al. (1997) draw an analogy between constraint propagation and viral infection. In their formulation, known partial solutions to a constraint satisfaction problem are encoded as "viruses", which then infect individuals in the population at a rate governed by two hyperparameters: the infection rate and each virus's infectivity. This mechanism replaces mutation and introduces guided variation, ensuring that modifications consistently push individuals toward constraint satisfaction rather than introducing random noise.

The initial virus population is generated by sampling valid pairs of queen placements, with pairs weighted by their proximity to the centre of the board. This design is motivated by the minimum remaining values heuristic from constraint satisfaction (Russell and Norvig, 2022), which encourages exploring the most highly constrained regions first, as they yield the most informative conflicts. Fitness is maintained as defined in the previous variants as it is proportional to that defined by Kanoh et al. (1997).

To preserve the permutation invariant during infection, each modification is implemented as a swap between the infected value and the value at the target position. Infectivity is updated after each infection attempt by doubling it on success or halving it on failure, with bounds of -100 and -0.001 applied to reduce the risk of stagnation in local minima.

For $n > 15$, all variants of the implemented genetic algorithms frequently become trapped in local minima, with the viral infection implementation taking in excess of 60 minutes to solve $n = 20$: the population converges to a set of high fitness individuals, but lacks the diversity or directional mechanisms needed to make the final transitions to a valid solution (Figure 4). A hybrid approach, in which viral infection is combined with a targeted local search procedure or a more exploratory mutation strategy, appears promising and represents a compelling direction for further investigation.

The performance of the viral infection variant was compared with that of the permutation variant to assess whether the hypothesised benefits of increased directionality translate into practical gains. Both experiments used the same random seed and comparable hyperparameters: a population of 200 individuals, a 5% elitism rate, and a 25% mutation or infection rate. Overall, the viral infection variant did not outperform the simpler permutation algorithm (Figure 5; Figure 6). In some cases, such as $n = 19$, the viral mechanism substantially degraded performance, increasing runtime by approximately 60%; this suggests that the removal of stochastic search can be detrimental when the algorithm must escape difficult regions of the search space. Kanoh et al. (1997)'s viral infection genetic algorithm was shown to be most effective in search spaces with low constraint density, a characteristic that the n-queens problem does not exhibit. Consequently, n-queens may be unsuitable for this mechanism.

Nevertheless, the viral variant exhibits a smoother exponential runtime profile than the permutation algo-

rithm (Figure 5), indicating greater predictability arising from its increased directionality and the absence of random mutation. Such predictability may be advantageous in settings where modelling algorithm performance is required. Further hyperparameter tuning may reveal the expected performance improvements, and there remains substantial scope for experimenting with virus construction, infectivity scoring and the evolution of the virus population throughout the search.

4 Comparison to Genetic Programming

Genetic programming is a generalisation of genetic algorithms, and the two share core mechanisms such as evolving populations of individuals through crossover and reproduction. The key distinction is that genetic algorithms operate on candidate solutions within the problem's search space, whereas genetic programming evolves executable computer programs, typically encoded as syntax trees to allow structural manipulation (Langdon and Poli, 2013). Fitness in genetic algorithms is often defined by proximity to a goal, such as counting attacking queen pairs in the eight queens problem, while in genetic programming it is usually determined by executing programs on test inputs and evaluating their outputs against a benchmark (Langdon and Poli, 2013). Genetic programming is well suited to domains where direct manipulation of solutions is difficult, and has been applied in areas such as intelligent control and flexible job-shop problem (FJSP) scheduling (Ponce-Cruz and Ramírez-Figueroa, 2010; Xu et al., 2023). However, its individuals are more complex and may be unnecessary or inappropriate for problems that allow simpler representations.

5 Real World Applications of Genetic Algorithms

Planning and managing manufacturing processes is inherently complex, and product assembly represents a substantial proportion of overall manufacturing and labour costs (Wang et al., 2009). To remain competitive, manufacturers must reduce these costs, making the optimisation of assembly operations a key priority (Wang et al., 2009). Assembly scheduling is typically formulated as a FJSP, which seeks to minimise an objective, most often the total completion time, by determining an effective sequence of operations across multiple machines (Dauzère-Pérès et al., 2024). Genetic algorithms have been successfully applied to FJSP in semiconductor manufacturing, achieving a 65% reduction in scheduling violations over a five-day planning horizon (Cavalieri, Crisafulli, and Mirabella, 1999).

6 Future Work

A natural extension of this work would be to search for all fundamental solutions, which would require explicitly addressing the symmetries of the n-queens problem and could enable further pruning of the search space. Another direction would be to evolve viruses over time or encode larger partial solutions, potentially increasing the usefulness of viral infection as a guiding mechanism. Finally, an ensemble approach may be advantageous: when fitness is already high, the iterative genetic algorithm cycle of selection, crossover and mutation, or infection, can be inefficient, and incorporating a more directional local search method, such as backtracking, may yield faster convergence in these scenarios.

7 Conclusion

Although evolutionary algorithms are effective on a wide range of optimisation problems, including the eight queens puzzle, their success often depends on the mechanisms chosen and how well they exploit problem structure (Russell and Norvig, 2022). This report examined a permutation-based genetic algorithm for the eight queens problem, evaluated its performance and investigated several enhancements, including hyperparameter tuning, generalising the approach to larger n-queens instances and incorporating a viral infection mechanism inspired by constraint based reasoning. The report also considered real world applications of evolutionary algorithms and briefly compared genetic algorithms with genetic programming to highlight the differences in representation and fitness evaluation.

References

Cavalieri, S., Crisafulli, F., and Mirabella, O., 1999. A genetic algorithm for job-shop scheduling in a semiconductor manufacturing system. 2, pp.957–961.

Dauzère-Pérès, S., Ding, J., Shen, L., and Tamssaouet, K., 2024. The flexible job shop scheduling problem: A review. *European journal of operational research*, 314(2), pp.409–432.

Kanoh, H., Matsumoto, M., Hasegawa, K., Kato, N., and Nishihara, S., 1997. Solving constraint-satisfaction problems by a genetic algorithm adopting viral infection. *Engineering applications of artificial intelligence*, 10(6), pp.531–537.

Langdon, W. and Poli, R., 2013. *Foundations of genetic programming*. Springer Berlin Heidelberg.

Moraglio, A. and Poli, R., 2011. Geometric crossover for the permutation representation. *Intelligenza artificiale*, 5(1), pp.49–63.

Ponce-Cruz, P. and Ramírez-Figueroa, F.D., 2010. *Intelligent control systems with LabVIEW™*. Springer.

Ruse, M., 1975. Charles darwin's theory of evolution: An analysis. *Journal of the history of biology* [Online], 8(2), pp.219–241. JSTOR: 4330635. [Accessed November 16, 2025].

Russell, S. and Norvig, P., 2022. *Artificial Intelligence: A modern approach*. Pearson Education Limited.

Wang, L., Keshavarzmanesh, S., Feng, H.-Y., and Buchal, R.O., 2009. Assembly process planning and its future in collaborative manufacturing: a review. *The international journal of advanced manufacturing technology*, 41(1), pp.132–144.

Xu, M., Mei, Y., Zhang, F., and Zhang, M., 2023. Genetic programming for dynamic flexible job shop scheduling: Evolution with single individuals and ensembles. *Ieee transactions on evolutionary computation*, 28(6), pp.1761–1775.

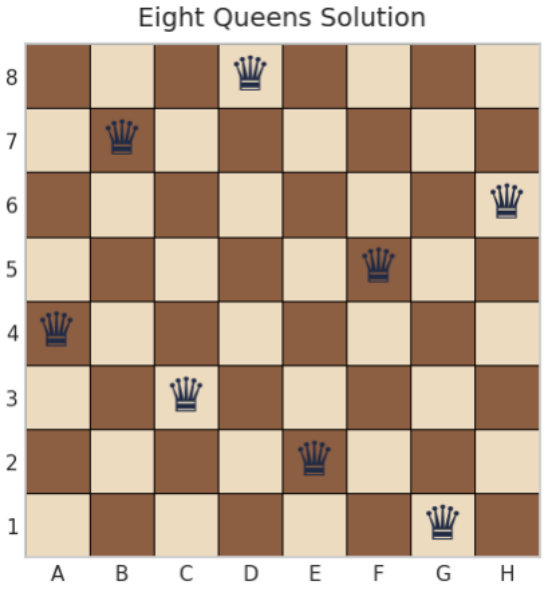

Figure 1: Shows a sample eight queens solution, generated by the basic algorithm, taking $1.726s$, 548 iterations and $10,216$ mutations to arrive at a solution.

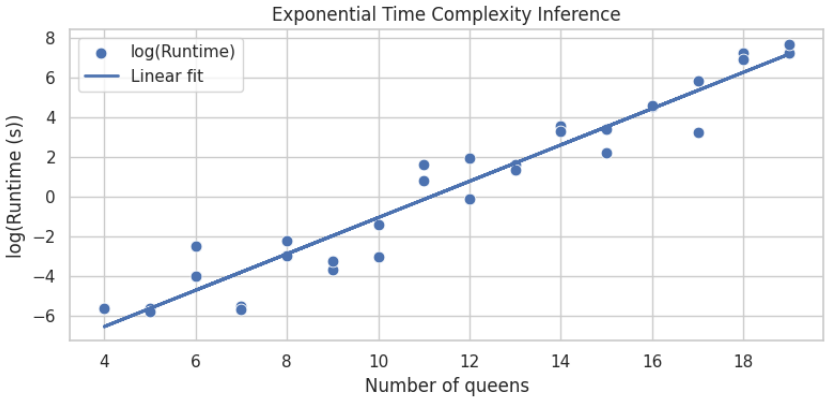

Figure 2: Time complexity inferred from a runtime experiment using a permutation-based state representation with fixed hyperparameters. The $n = 20$ instance did not complete, so its plotted runtime is an underestimate.

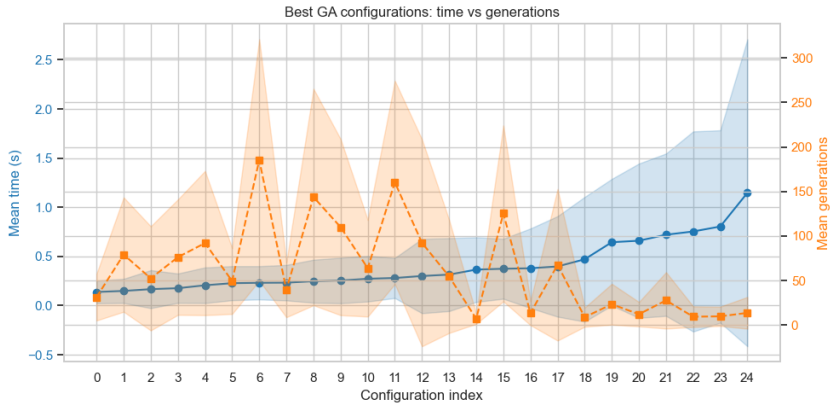

Figure 3: Mean runtime and mean number of generations for each hyperparameter configuration, with shaded regions showing linearly interpolated standard deviation error bars. The weak correspondence between mean runtime and mean generations indicates that runtime cannot be reliably predicted from these hyperparameters alone. Configurations used can be found in the accompanying Python notebook, but they include varying the population size and the elitism and mutation rates.

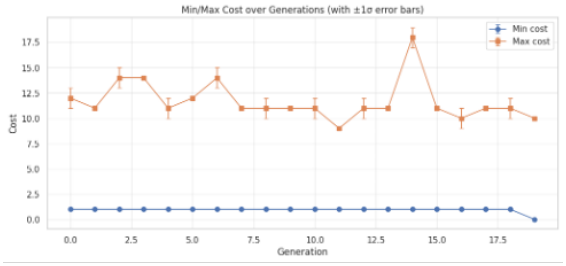

Figure 4: Illustration of the algorithm becoming trapped on a plateau of high fitness local minima, where a substantial portion of the total runtime is spent attempting to escape to a complete solution that is only one move away. The cost shown represents the number of attacking queen pairs.

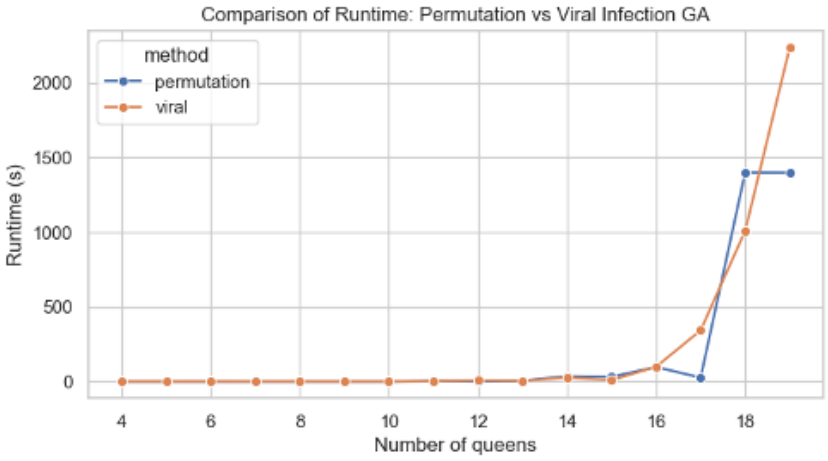

Figure 5: Showing runtimes for 4 to 19 queens using both the permutation and viral infection variants of the genetic algorithm.

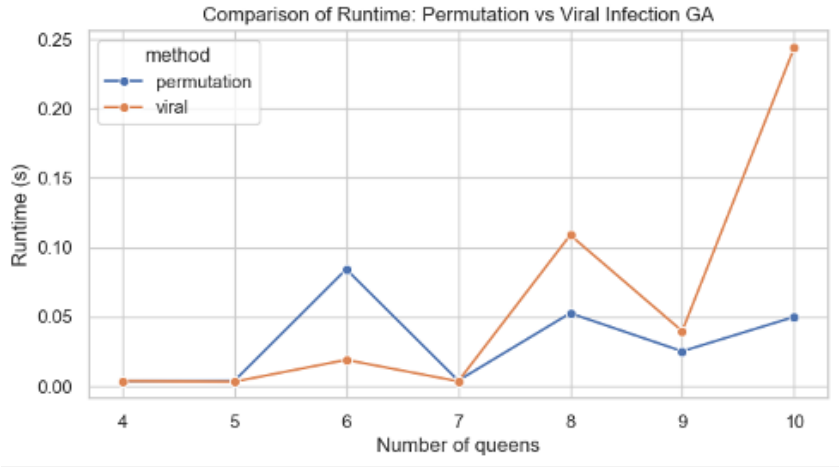

Figure 6: Showing runtimes for 4 to 10 queens using both the permutation and viral infection variants of the genetic algorithm.