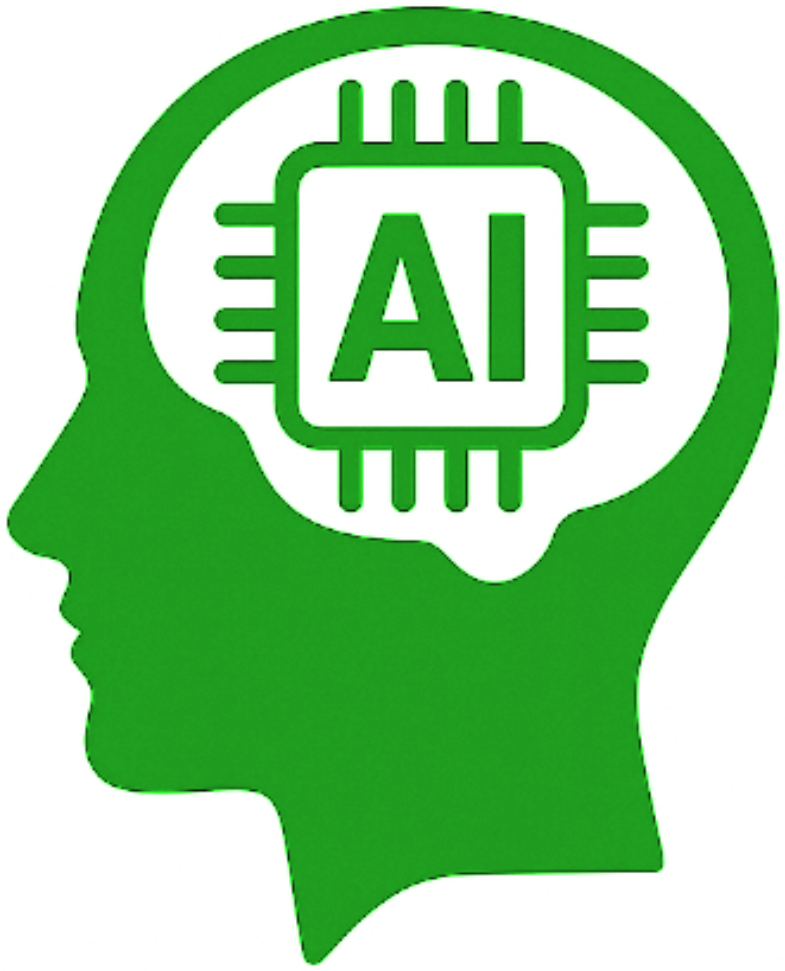
Practice 3 - Artificial Intelligence

Local search. Games.



Author: Fernando Pastor Peralta (897113)

Delivery date: 04/11/2025

Group: Tuesday B, 4:00 PM - 7:00 PM

Index

Introduction:

We will work on local search, specifically applying and analyzing three algorithms—Hill Climbing, Simulated Annealing, and Genetic Algorithm—on the 8-queens problem, starting from the Java AIMA code provided to us. We will collect metrics on success or failure, average number of steps, and other relevant measurements. This entire first part is implemented in the file *NQueensLocal.java*.

We will also address adversarial games. After modeling a simple game tree, we will implement MINIMAX and α–β pruning to compute the value of the tree.

Methodology and analysis of results:

HillClimbingSearch:

nQueensHillClimbingSearch_Statistics:

We perform numExperiments (in this case, 10,000 experiments called from the main method) using HillClimbingSearch.

To create completely different experiments, we randomly generate boards using an auxiliary function randomNQueensBoard() and ensure that they do not repeat previously generated initial boards by using a hash-based function. Each time a new initial state is generated, it is inserted into the hash; if it is already present, it is rejected and a new board is generated for the experiment. To identify an initial state in the hash, we create a signature of the board using the auxiliary function sacarFirmaBoard(board).

After performing the search with AttackingPairsHeuristic() and running the agent, we record the number of steps by counting how many actions the agent performed. We also determine whether the search was successful by checking the search status. Depending on success or failure, we accumulate statistics on the number of successes and failures, as well as the total steps for each experiment. Finally, we print the computed data to standard output, as shown in section 2.1 of the appendix image.

From the obtained data, we observe that the algorithm using the attacking pairs heuristic is very fast but unreliable, achieving only about a 15% success rate, although with great speed since the average number of steps is 3. We also observe that the average cost of a successful run is slightly higher than that of failures (≈ 4 steps per success versus ≈ 3 per failure). Therefore, the initial state is highly determinant, and restarts are necessary to find acceptable successful paths.

nQueensRandomRestartHillClimbing:

In this function, we repeatedly execute HillClimbingSearch with the attacking pairs heuristic and random initial states until the first solution is found. The implementation is very similar to the previous case, collecting data in the same way.

From the results obtained (section 2.2 in the appendix images), we can see that the average number of steps in the successful attempt is 3, as in the previous case. However, the number of attempts required to achieve success is highly dependent on probability, with considerable dispersion: sometimes it succeeds sooner, sometimes later, but it rarely succeeds in the first few attempts. This is consistent with the previously observed 85% failure rate per attempt.

SimulatedAnnealing:

We use virtually the same implementation as nQueensHillClimbingSearch_Statistics, taking the same data and using the SimulatedAnnealing algorithm with the attacking pairs heuristic. We perform 2 runs with 1000 experiments each, one schedule($T(t)=k*e^{(-\lambda t)}$) with high temperature T (large k (20) and small lambda (0.045)) and another with low temperature T (small k (5) and larger lambda (0.1)).

From the data obtained, visible in photos 2.3 of the appendix, we can conclude that, at least with the k and lambda values we have defined, the SimulatedAnneling algorithm is significantly inferior to HillClimbingSearch. Let's examine the results in more detail:

- Large k and small lambda: approximately a 98.5% failure rate, with an average number of steps of 67 and a very similar average success and failure cost, around 66.
- small k and large lambda: approximately 90% failure rate, with an average number of steps of 28 and average success and failure costs of 28 and 24 respectively.

From these results, we conclude that, under this configuration, lower temperatures yield better results across all metrics. To obtain good results for the 8-queens problem with this setup, we must reduce the temperature and increase the number of experiments to raise the probability of successful outcomes.

Genetic Algorithm:

In our implementation, we designed each individual as a list of integers of length N, with the alphabet {0, 1,…N-1}, indicating the selected row in each column. The population is represented as a hash-based structure of individuals. The Fitness Function (FF) returns higher values as the number of attacking queen pairs decreases, and it is also used to define the goal state. After executing the genetic algorithm with configurable parameters, we collect the measurements and print them to standard output.

The appendix shows three executions with different parameter sets. The differences in results are as follows:

- Pop=50, pmut=0.15 →goal reached, fitness=28, iterations=361, time=1760 ms.
- Pop=30, pmut=0.25 → goal reached, fitness=28, iterations=933, time=1510 ms.
- Pop=150, pmut=0.05 → goal not reached, fitness=25, iterations=14, time=707 ms.

We observe that a larger population implies a higher time cost per iteration but faster convergence across generations. Higher mutation rates maintain greater diversity and promote exploration. The failure in the last experiment is consistent with stagnation caused by insufficient mutation.

Games:

For this section, we created three classes:

- EjemplosArbolJuego.java: We construct the game tree with leaf nodes of type double, using the material provided by the instructors. We execute the tree once with each algorithm (MinimaxArbolJuego and AlfaBetaArbolJuego) and print the value, the tree, and the number of visited nodes.
- MinimaxArbolJuego.java: The ejecutar() method delegates execution to minimax() with correctly initialized parameters. We pass an integer counter array that increments by one for each recursive call except the root. If the node is terminal, it is marked as visited and its value is returned; if it is a MAX node, we traverse the children and keep the maximum value; if it is a MIN node, we keep the minimum. The result is stored in the node and later displayed.
- AlfaBetaArbolJuego.java: This follows the same structure as minimax, but includes the parameters α and β. In MAX nodes, we update α = max(α, $value$) and prune if $\alpha \geq \beta$; in MIN nodes, we update β= min(β, $value$) and prune if $\alpha \geq \beta$. Pruned nodes are marked as [X].

According to the result shown in the last appendix image, both algorithms return the same value; pruning does not change the result, it only avoids exploring unnecessary parts of the tree (fewer visited nodes). After obtaining α = 3 in the first branch, in the second branch the MIN node satisfies $\beta \leq \alpha$ upon seeing the first leaf (= 2), and the remaining two children are pruned ([X]).

Conclusion:

In local search for the 8-queens problem (using the attacking pairs heuristic), after executing the three algorithms we can conclude that HillClimbingSearch is very fast but unreliable, the Genetic Algorithm is the most robust in finding solutions, and Simulated Annealing is significantly less efficient across all metrics (to be competitive, it requires more iterations and/or slower cooling).

In adversarial games, Minimax and α–β pruning return the same value, but α–β pruning reduces the number of explored nodes without altering the result.

Appendix:

```
===== 2.1 HillClimbingSearch / Escalada =====
NQueens HillClimbing con 10000 estados iniciales diferentes -->
Fallos: 85,31%
Coste medio fallos: 3,08
Éxitos: 14,69%
Coste medio éxitos: 4,05
   Número de pasos medio: 3

===== 2.2 Random-Restart Hill Climbing =====
Numero de experimentos: 26
Número medio de pasos en conseguir el éxito: 3

===== 2.3 SimulatedAnnealing / Enfriamiento Simulado (dos Schedulers) =====
NQueensDemo Simulated Annealing con 1000 estados iniciales diferentes -->
Parámetros Scheduler: Scheduler (20,0.045,100)
Fallos: 98,50%
Coste medio fallos: 66,72
Éxitos: 1,50%
Coste medio éxitos: 64,00
Número de pasos medio: 67

NQueensDemo Simulated Annealing con 1000 estados iniciales diferentes -->
Parámetros Scheduler: Scheduler (5,0.10,100)
Fallos: 90,10%
Coste medio fallos: 28,63
Éxitos: 9,90%
Coste medio éxitos: 24,80
Número de pasos medio: 28
```

```
===== 2.4 GeneticAlgorithm / Algoritmos Genéticos =====
GeneticAlgorithm
Parámetros iniciales:    Población: 50, Probabilidad mutación: 0.15
Mejor individuo=
--Q-----
-----Q--
---Q----
-Q------
-------Q
----Q---
------Q-
Q-------


Tamaño tablero  = 8
Fitness         = 28.0
Es objetivo     = true
Tamaño población= 50
Iteraciones     = 272
Tiempo          = 1161ms.

--- FIN ---
```

```
===== 2.1 HillClimbingSearch / Escalada =====
NQueens HillClimbing con 10000 estados iniciales diferentes -->
Fallos: 85,20%
Coste medio fallos: 3,06
Éxitos: 14,80%
Coste medio éxitos: 4,05
Número de pasos medio: 3
```

```
===== 2.1 HillClimbingSearch / Escalada =====
NQueens HillClimbing con 10000 estados iniciales diferentes -->
Fallos: 85,50%
Coste medio fallos: 3,06
Éxitos: 14,50%
Coste medio éxitos: 4,08
Número de pasos medio: 3
```

```
===== 2.2 Random-Restart Hill Climbing =====
Numero de experimentos: 16
Número medio de pasos en conseguir el éxito: 3
```

```
===== 2.2 Random-Restart Hill Climbing =====
Numero de experimentos: 9
Número medio de pasos en conseguir el éxito: 3
```

```
===== 2.2 Random-Restart Hill Climbing =====
Numero de experimentos: 12
Número medio de pasos en conseguir el éxito: 3
```

```
===== 2.3 SimulatedAnnealing / Enfriamiento Simulado (dos Schedulers) =====
NQueensDemo Simulated Annealing con 1000 estados iniciales diferentes -->
Parámetros Scheduler: Scheduler (20,0.045,100)
Fallos: 98,50%
Coste medio fallos: 66,66
Éxitos: 1,50%
Coste medio éxitos: 64,47
Número de pasos medio: 67

NQueensDemo Simulated Annealing con 1000 estados iniciales diferentes -->
Parámetros Scheduler: Scheduler (5,0.10,100)
Fallos: 90,40%
Coste medio fallos: 28,72
Éxitos: 9,60%
Coste medio éxitos: 24,31
Número de pasos medio: 28
```

```
===== 2.3 SimulatedAnnealing / Enfriamiento Simulado (dos Schedulers) =====
NQueensDemo Simulated Annealing con 1000 estados iniciales diferentes -->
Parámetros Scheduler: Scheduler (20,0.045,100)
Fallos: 98,90%
Coste medio fallos: 66,61
Éxitos: 1,10%
Coste medio éxitos: 65,18
Número de pasos medio: 67

NQueensDemo Simulated Annealing con 1000 estados iniciales diferentes -->
Parámetros Scheduler: Scheduler (5,0.10,100)
Fallos: 89,70%
Coste medio fallos: 28,65
Éxitos: 10,30%
Coste medio éxitos: 25,35
Número de pasos medio: 28
```

```
===== 2.4 GeneticAlgorithm / Algoritmos Genéticos =====
GeneticAlgorithm
Parámetros iniciales:    Población: 50, Probabilidad mutación: 0.15
Mejor individuo=
----Q---
------Q-
-Q------
-----Q--
--Q-----
Q-------
---Q----
-------Q


Tamaño tablero  = 8
Fitness         = 28.0
Es objetivo     = true
Tamaño población= 50
Iteraciones     = 361
Tiempo          = 1760ms.
```

```
===== 2.4 GeneticAlgorithm / Algoritmos Genéticos =====
GeneticAlgorithm
Parámetros iniciales:    Población: 30, Probabilidad mutación: 0.25
Mejor individuo=
-----Q--
--Q-----
------Q-
-Q------
-------Q
----Q---
Q-------
---Q----


Tamaño tablero  = 8
Fitness         = 28.0
Es objetivo     = true
Tamaño población= 30
Iteraciones     = 933
Tiempo          = 1510ms.
```

```
===== 2.4 GeneticAlgorithm / Algoritmos Genéticos =====
GeneticAlgorithm
Parámetros iniciales:    Población: 150, Probabilidad mutación: 0.05
Mejor individuo=
---Q----
----Q---
-Q------
-----Q--
--------
--Q---Q-
Q-------
-------Q


Tamaño tablero  = 8
Fitness         = 25.0
Es objetivo     = false
Tamaño población= 150
Iteraciones     = 14
Tiempo          = 707ms.
```

```
Valor con MINIMAX: 3.0
3.0-MAX
    3.0-MIN
            [3.0]
          [12.0]
            [8.0]
    2.0-MIN
            [2.0]
            [4.0]
            [6.0]
    2.0-MIN
          [14.0]
            [5.0]
            [2.0]

Nodos visitados: 12
------------------------------
Valor con poda Alfa Beta: 3.0
3.0-MAX
    3.0-MIN
            [3.0]
          [12.0]
            [8.0]
    2.0-MIN
            [2.0]
              [X]
              [X]
    2.0-MIN
          [14.0]
            [5.0]
            [2.0]

Nodos visitados: 10
------------------------------
```