Entrega Práctica 3 - Inteligencia Artificial

Búsqueda local. Juegos.

Autor: Fernando Pastor Peralta (897113)

Fecha de entrega: 04/11/2025

Grupo: Martes B, 16:00-19:00

Índice

Introducción:

Trabajaremos en búsqueda local, concretamente aplicaremos y analizaremos tres algoritmos Hill-Climbing, Simulated Annealing y Genetic Algorithm en el problema de las 8 reinas, partiendo de la base que nos da el código AIMA en Java que nos proporcionan. Recopilaremos métricas de éxito o fracaso, pasos medios y otras métricas relevantes de nuestras mediciones. Toda esta primera parte se plasmará en el fichero NQueensLocal.java.

También, tocaremos los juegos adversarios. Tras modelar un árbol de juego sencillo, implementaremos MINIMAX y poda α-β para calcular el valor del árbol.

Metodología y análisis de los resultados:

HillClimbingSearch / Escalada:

nQueensHillClimbingSearch_Statistics:

Hacemos numExperiments (en este caso, llamamos con 10000 experimentos desde el método main) sobre HillClimbingSearch.

Para crear experimentos totalmente diferentes, creamos aleatoriamente tableros mediante una función auxiliar randomNQueensBoard() y nos aseguramos que no se repitan con otros tableros iniciales que hemos generado antes mediante un función hash, en la que insertamos cada estado inicial cada vez que tenemos uno nuevo, si está repetido no nos dejará meterlo y volveremos a generar un nuevo tablero para el nuevo experimento. Para identificar un estado inicial en la función hash creamos una firma del estado inicial para poder instalarlo cómodamente, esto lo hacemos mediante la función auxiliar sacarFirmaBoard(board).

Tras hacer búsqueda con AttackingPairsHeuristic() y poner al agente en acción, sacamos los pasos viendo cuantas acciones hizo el agente y vemos si la búsqueda fue exitosa consultando el estado de la búsqueda, dependiendo de si se tuvo éxito o no, se toman medidas sobre cuántos éxitos/fracasos llevamos y sobre los pasos acumulados por cada experimento en cada uno. Después, mostramos por la salida estándar los datos calculados correctamente, tal y cómo se puede ver en el apartado 2.1 de la foto del anexo.

De los datos obtenidos, podemos ver que el algoritmo con la heurística de parejas en ataque, es muy rápido pero poco fiable, ya que logra aproximadamente solo un 15% de éxito, pero con mucha rapidez ya que su número de pasos medio es de 3. Además, también vemos que el coste medio de un éxito supera un poco al de fracasos (por éxito ≈ 4, por fracaso ≈ 3). Por lo tanto, el estado inicial parece ser muy determinante, y los reinicios resultan necesarios para encontrar caminos exitosos aceptables.

nQueensRandomRestartHillClimbing:

En esta función ejecutamos múltiples veces HillClimbingSearch con la heurística de parejas en ataque con estados iniciales aleatorios, hasta que aparece la primera solución. La implementación es muy similar a la del punto anterior, tomando los datos de la misma forma.

De los resultados que obtenemos (2.2 las fotos del anexo), podemos ver que el número medio de pasos en el intento exitoso es de 3, como en el punto anterior, sin embargo, la cantidad de intentos que hay que hacer para tener un éxito es algo muy dependiente de la probabilidad (datos bastante dispersos), que unas veces tarda más y otras menos, pero difícilmente sale entre los primeros intentos (también coherente con el 85% de fracaso por intento declarado en el punto anterior).

SimulatedAnnealing / Enfriamiento Simulado:

Usamos prácticamente misma implementación que nQueensHillClimbingSearch_Statistics, tomando mismos datos y mediante el algoritmo de SimulatedAnnealing con la heurística de parejas en ataque. Hacemos 2 ejecuciones con 1000 experimentos cada uno, una schedule(T(t)=k*e^(−λt)) con temperatura T alta (k grande (20) y lambda pequeña (0.045)) y otra con temperatura T baja (k pequeña (5) y lambda mayor (0.1)).

De los datos que se obtienen, visibles en las fotos con 2.3 del anexo, podemos concluir, que por lo menos con la k y lambda que hemos definido, el algoritmo de SimulatedAnneling es muy inferior frente al HillClimbingSearch. Veamos con más detalle los resultados:

- k grande y lambda pequeña: aproximadamente un 98.5% de fracaso, con un número de pasos medio de 67 y coste medio de éxito y fallo muy similar, alrededor de 66.
- k pequeña y lambda mayor: aproximadamente un 90% de fracaso, con un número de pasos medio de 28 y coste medio de éxito y fallo de 28 y 24 respectivamente.

De estos resultados podemos concluir que con este límite, cuanto más fría es la temperatura, mejores resultados tenemos en todos los ámbitos. Si queremos obtener buenos resultados en el problema de las 8-reinas con esta configuración determinada, deberemos bajar la temperatura y aumentar los experimentos para aumentar la probabilidad de obtener resultados exitosos.

GeneticAlgorithm / Algoritmos Genéticos:

En nuestra implementación diseñamos cada individuo como una lista de enteros de longitud N, con alfabeto {0, 1,…N-1} que señaliza cada fila marcada en cada columna. La población se representa como una función hash de individuos. La FitnessFunction (FF) devuelve mayores valores cuantos menos ataques hay entre reinas, lo utilizaremos también para definir el estado objetivo. Tras ejecutar el geneticAlgorithm con los parámetros que se pueden modificar, tomamos las distintas medidas y las presentamos por salida standard.

En el anexo se ven 3 ejecuciones con distintos juegos de parámetros, veamos las diferencias en los resultados:

- Pop=50, pmut=0,15 → objetivo alcanzado, fitness=28, iteraciones=361, tiempo=1760 ms.

- Pop=30, pmut=0,25 → objetivo alcanzado, fitness=28, iteraciones=933, tiempo=1510 ms.

- Pop=150, pmut=0,05 → no alcanza objetivo, fitness=25, iteraciones=14, tiempo=707 ms.

Aquí vemos que más población implica mayor coste temporal por iteración, mayor aceleración de convergencia por generaciones. Además, la mutación más alta mantiene más diversidad y tiene mayor tasa de exploración. El fallo en el último experimento es coherente con el estancamiento por no tener suficiente mutación.

Juegos:

Para este apartado creamos 3 clases:

- EjemplosArbolJuego.java: Construimos el árbol de juego con hojas tipo double con el material proporcionado por el profesorado. Ejecutamos una vez con cada algoritmo (una con MinimaxArbolJuego y otra con AlfaBetaArbolJuego), imprimimos valor, árbol y nodos visitados.

- MinimaxArbolJuego.java: el método ejecutar delega la ejecución con los parámetros correctamente inicializados a minimax(). Pasamos un array de enteros contador que suma 1 por cada llamada recursiva que no sea la raíz. Si el nodo es terminal, marcamos como visitado y devolvemos su valor; si es MAX, recorremos hijos y nos quedamos con el máximo; si es MIN, con el mínimo. Guardamos el resultado en el nodo para luego mostrarlo por pantalla.

- AlfaBetaArbolJuego.java: misma estructura que minimax, pero con los parámetros alpha y beta. En MAX actualizamos alpha = max(alpha, value) y podamos si alpha ≥ beta, en MIN actualizamos beta = min(beta, value) y podamos si alpha ≥ beta. Los nodos podados serán [X].

Según el resultado mostrado en la última imagen del anexo, con ambos algoritmos, la poda no altera el resultado, solo evita explorar partes inútiles (menos nodos visitados). Tras obtener α=3 en la primera rama, en la segunda rama MIN se cumple β≤α al ver la primera hoja (=2) y se podan los dos hijos restantes ([X]).

Conclusión:

En la búsqueda local en 8-reinas (con heurística de parejas en ataque), tras ejecutar los tres algoritmos podemos concluir que: el HillClimbingSearch es muy rápido pero poco fiable, que el geneticAlgorithm es el más robusto encontrando soluciones y SimulatedAnnealing es ampliamente menos eficiente en todos los ámbitos (para que compita hay que aumentar iteraciones y/o enfriar más lento).

En juegos, Minimax y α-β devuelven mismo valor, pero α-β reduce los nodos explorados sin alterar el resultado.

Anexo:

```
===== 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
------------------------------
```