Sistemas Inteligentes

Proyecto 3

Búsqueda Local: Hill Climbing

Definición

Hacer una función en Python (el archivo que la contenga deberá llamarse **`Proyecto3.py`**) llamada:

`busquedaHC(N, lateral, M)`

que reciba como entrada un número entero **N**, el número de reinas en el tablero de **N X N**, y la variable booleana lateral que puede ser **`true`**, cuando se permiten movimientos laterales y **`false`**, cuando no se permiten, y un número entero **M**, el número máximo de inicios a realizar. La función deberá resolver el problema de las **N-reinas** utilizando el algoritmo *Hill Climbing* con reinicio aleatorio (para evitar óptimos locales). La solución consiste en encontrar la posición de las **N** reinas, en un tablero de **N X N**, de tal forma que no se atacan.

Antecedentes

Hay algunos problemas en los que los caminos desde la raíz hasta el estado meta no importan, sólo importa el estado meta, el cual es la solución.

Para este tipo de problemas los algoritmos de búsqueda local son excelentes ya que son fáciles de implementar, ocupan poca memoria (casi constante) y, utilizando una buena heurística, logran llegar a la solución óptima o al menos muy cerca de ella.

El algoritmo de búsqueda local básico es el llamado *Hill Climbing* (Ascensión de Colinas). Este algoritmo evalúa todos los vecinos del estado actual (estados a donde se puede llegar a partir del estado actual, aplicando alguna acción válida), utilizando una función de evaluación (función heurística), diseñada específicamente para el problema y, una vez evaluados, selecciona el vecino que tenga la mejor evaluación mayor que la evaluación del nodo actual. Este proceso continúa hasta que no se existe ningún vecino con una evaluación mejor que el actual (óptimo local) dando como solución el nodo actual.

Desde luego que, en algunas ocasiones, el proceso no termina exactamente en la solución óptima (óptimo global) sino en un óptimo local, por lo que se tienen que utilizar algunas variantes del método para tratar de salir de él y llegar al óptimo global. Una de las variantes más comúnmente usadas es *Hill Climbing* con reinicio aleatorio, en el cual se corre algoritmo un número n de veces, partiendo en cada una de ellas de un punto inicial seleccionado en forma aleatoria, y se regresa la mejor solución encontrada, esperando que el número n sea lo suficientemente grande como para llegar al óptimo global.

Si en el algoritmo de *Hill Climbing* permitimos que se mueva no sólo a estados mejores que el actual sino también a los que sean iguales, es decir, permitiendo movimientos laterales (verificando que no se regrese al estado anterior), el número de veces para el caso de las **8-reinas**

es de alrededor de **n = 2**. En promedio, cada 2 veces que se corre, llega al óptimo global. Si no permitimos movimientos laterales el número sube a **n = 7**. Para el caso de las **8-reinas**, como sabemos exactamente cuáles son las condiciones de la solución, sabremos exactamente cuándo llega al óptimo global.

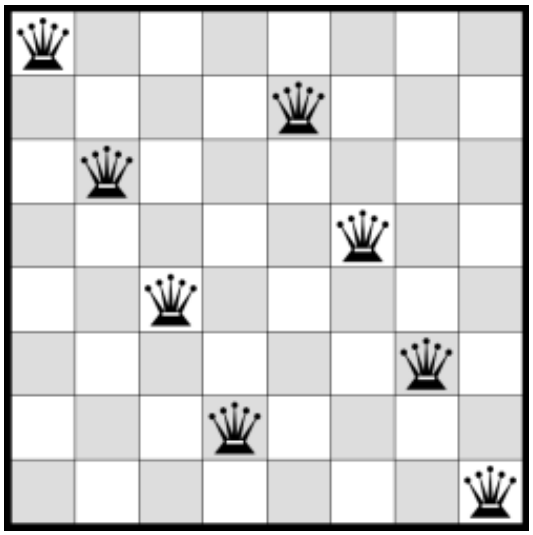
Consideraciones

1. Usar *Hill Climbing* con reinicio aleatorio. Dejar que el usuario selecciones si se permiten movimientos laterales o no.
2. **Entrada**: Recibirá como parámetros:
    a. **N**: el número de reinas y tamaño del tablero (el tablero es de **NXN**).
    b. **lateral**: true si permite movimientos laterales o false si no.
    c. **M**: El número máximo de reinicios a realizar, donde se puede dar la opción de que lo haga hasta que encuentre una solución, es decir, **M = infinito**.
3. **Salida**: La función no regresará nada, sólo **imprimirá**:
    a. El estado solución (tablero y la posición de las reinas), o la mejor solución en caso de que no se haya encontrado.
    b. El número **m** de veces que se reinició hasta encontrar la solución (**m** $\leq$ **M**).
    c. Un letrero que diga si se encontró o no se encontró la solución.

    Por ejemplo, si el usuario da un número máximo de iteraciones igual a 5 y después de 5 reinicios no logró encontrar la solución, responderá “No se encontró la solución”, con una n = 5. Si la logra encontrar en 5 o menos, responderá “Solución Encontrada”, y n igual al número de reinicios que requirió.

4. En el caso de las 8-reinas, en promedio, debe haberse corrido 7 veces para encontrar una solución, si no se permiten movimientos laterales, y 2 veces, si se permiten.
5. El estado de inicio se va a generar con **una reina por columna**, donde el cuadrito que ocupa en el tablero se genera **en forma aleatoria**.
6. Las acciones permitidas son:
    a. Mover una reina un cuadrito hacia arriba.
    b. Mover una reina un cuadrito hacia abajo.

    Eso significa que cada estado actual tiene a lo más 16 vecinos.
7. La función de evaluación (heurística) será el número de reinas que se están atacando (más fácil de calcular) o el número de pares de reinas que se están atacando. Por ejemplo, para el siguiente tablero (estado) del problema de las 8-reinas:



La evaluación es 1 (un par de reinas atacándose) o 2 (dos reinas atacándose).

Probando el programa

- El programa será probado colocando todos los archivos necesarios para que corra en el mismo folder.
- El script utilizado (colocado en el archivo **`main.py`**) será el siguiente:

```
import busquedaHC from Proyecto3.py

N = 8            # o el número de reinas que tenga el problema
lateral = true   # o false
M = 5            # o el número máximo que se desee reiniciar
busquedaHC(N, lateral, M)
```

- La función contestará imprimiendo la salida solicitada.