Algoritmos de búsqueda local

Los algoritmos vistos hasta ahora se diseñan para explorar sistemáticamente espacios de búsqueda. Esta forma sistemática se alcanza manteniendo uno o más caminos en memoria y registrando qué alternativas se han explorado en cada punto a lo largo del camino y cuales no. Cuando se encuentra un objetivo, el camino a ese objetivo también constituye una solución al problema.

Sin embargo, en muchos problemas el camino es irrelevante. Por ejemplo, en el problema de las 8-reinas, lo que importa es la configuración final de las reinas, no el orden en el que fueron colocadas. Esta clase de problemas incluyen muchas aplicaciones importantes como diseño de circuitos integrados, disposición del suelo de una fabrica, job-shop scheduling, optimización de redes de telecomunicaciones, o en la ruta de un vehículo.

Si no importa el camino al objetivo, podemos considerar una clase diferente de algoritmos que no se preocupen en absoluto de los caminos. Los algoritmos de búsqueda local funcionan con un solo estado actual y generalmente se mueve sólo a los vecinos del estado. Típicamente, los caminos seguidos por la búsqueda no se retienen. Aunque los algoritmos de búsqueda local no son sistemáticos, tienen dos ventajas claves:

1. Usan muy poca memoria
2. Pueden encontrar a menudo soluciones razonables en espacios de estados grandes o infinitos para los cuales son inadecuados los algoritmos sistemáticos.

Además de encontrar los objetivos, los algoritmos de búsqueda local son útiles para resolver problemas de optimización, en los cuales el objetivo es encontrar el mejor estado según una función objetivo.

Para entender la búsqueda local, consideremos la forma o el paisaje del espacio de estados de la figura 4.10. El paisaje tiene posición (definido por el estado) y elevación (definido por la función heurística).Si la elevación corresponde a un costo, entonces el objetivo es encontrar el valle más bajo (un mínimo global). Si la elevación corresponde a una función objetivo, entonces el objetivo es encontrar el pico más alto (un máximo global). Los algoritmos de búsqueda local exploran este paisaje. Un algoritmo de búsqueda local completo siempre encuentra un objetivo si existe. Un algoritmo de búsqueda local óptimo siempre encuentra un mínimo/máximo global.

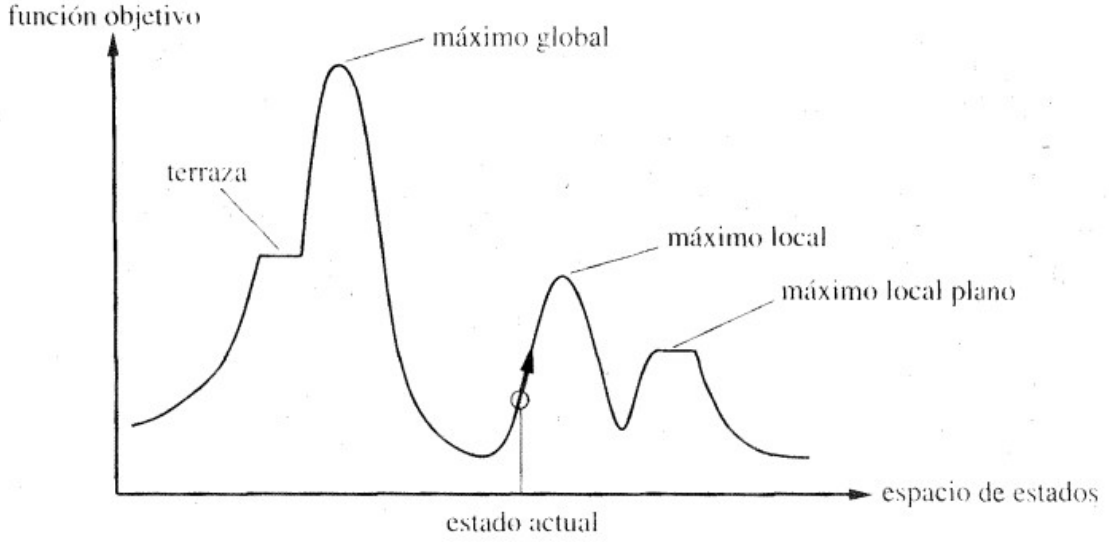

Figura 4.10 Un paisaje del espacio de estados unidimensional en el cual la elevación corresponde a la función objetivo. El objetivo es encontrar el máximo global. La búsqueda de ascensión de colinas modifica el estado actual para tratar de mejorarlo, como se muestra con la flecha. Se definen en el texto varios rasgos topográficos.

Búsqueda de ascensión de colinas

Consiste en un bucle que continuamente se mueve en dirección del valor creciente, es decir, cuesta arriba. Termina cuando alcanza un pico en donde ningún vecino tiene un valor más alto. El algoritmo no mantiene un árbol de búsqueda, sino una estructura de datos del nodo actual que necesita sólo el registro del estado y su valor de función objetivo. La ascención de colinas no mira delante más allá de los vecinos inmediatos del estado actual. Se parece a lo que haríamos cuando tratamos de encontrar la cumbre del Everest con neblina mientras sufrimos de amnesia.

```
AscencionDeColinas
   NodoActual = NodoInicial;
   Ciclo
      L = VECINOS(NodoActual);
      SigEval = -INF;
      SigNodo = NULL;
      Para Todo x En L
      Si (EVAL(x) > SigEval)
         SigNodo = x;
         SigEval = EVAL(x);
      Si SigEval <= EVAL(NodoActual)
         //Regresamos el NodoActual, ya que no hay un mejor vecino
         Regresa NodoActual;
      NodoActual = SigNodo;
```

Veamos el ejemplo de las 4-Reinas con la siguiente configuración inicial:

	1	2	3	4
A				
B				
C		**R**	**R**	
D	**R**			**R**

El objetivo es colocar las reinas en el tablero sin que se ataquen entre si, por lo tanto es lógico que nuestra función de evaluación o heurística este relacionada con esto. Tomemos como heurística h = número de pares de reinas que se atacan. En nuestro estado inicial h = 4. Representemos nuestro nodo inicial indicando la posición de cada reina, es decir:

NodoInicial = [D1,C2,C3,D4]

En el algoritmo el NodoActual toma el valor del NodoInicial, luego comienza el ciclo y lo primero que hace es obtener la lista de todos los vecinos del NodoActual. Los vecinos son todos aquellos movimientos que podemos hacer a partir de nuestro nodo actual. Para este ejemplo consideraremos mover una sola reina sobre su misma columna unicamente. Si cada reina se puede mover sobre su propia columna únicamente entonces cada reina se puede colocar de 3 maneras distintas, como tenemos 4 reinas esto quiere decir que tenemos un total de 4X3 = 12 posibles vecinos.

Del estado inicial entonces tenemos los vecinos mostrados en el siguiente tablero. Escribimos su h en su correspondiente posible posición, es decir si movemos la reina D1 a C1 entonces tendríamos un h = 4, entonces colocamos el 4 en C1.

	1	2	3	4
A	**4**	**2**	**2**	**4**
B	**3**	**4**	**4**	**3**
C	**4**	**R**	**R**	**4**
D	**R**	**5**	**5**	**R**

El algoritmo revisa que vecino ofrece un mejor desempeño (el algoritmo de arriba máximiza, solo debemos invertirlo para minimizar porque buscamos h = 0). En este caso tenemos dos vecinos que ofrecen una mejora en la h de 2, supongamos que elegimos el primero entonces nuestro tablero queda de la siguiente manera

	1	2	3	4
A		**R**		
B				
C			**R**	
D	**R**			**R**

Por lo tanto nuestro nuevo NodoActual es [D1,A2,C3,D4] h = 2

Volvemos a calcular los vecinos:

	1	2	3	4
A	**4**	**R**	**2**	**2**
B	**2**	**4**	**3**	**1**
C	**2**	**4**	**R**	**2**
D	**R**	**5**	**3**	**R**

Tenemos un vecino que nos dan un h = 1, hacemos ese movimiento y obtenemos el siguiente tablero

	1	2	3	4
A		**R**		
B				**R**
C			**R**	
D	**R**			

Por lo tanto nuestro nuevo NodoActual es [D1,A2,C3,B4] h = 1

Volvemos a calcular los vecinos

	1	2	3	4
A	**3**	**R**	**2**	**2**
B	**3**	**3**	**3**	**R**
C	**2**	**3**	**R**	**2**
D	**R**	**4**	**1**	**2**

Tenemos un vecino que nos da h = 1, el algoritmo dice que nos detengamos pues no obtuvimos una h más chica. El algoritmo hace esto para evitar las terrazas, sin embargo, en este ejemplo particular de las N-Reinas, sabemos que podemos salir de la terraza y que hay solamente máximos globales. Por lo tanto vamos a continuar y solo nos detendremos cuando todas las h de los vecinos generados sean mayores estrictamente a la del NodoActual.

	1	2	3	4
A		**R**		
B				**R**
C				
D	**R**		**R**	

Por lo tanto nuestro nuevo NodoActual es [D1,A2,D3,B4] h = 1

Calculamos los nuevos vecinos

	1	2	3	4
A	**1**	**R**	**2**	**3**
B	**2**	**2**	**3**	**R**
C	**0**	**3**	**1**	**3**
D	**R**	**3**	**R**	**3**

Tomamos el movimiento con h = 0

	1	2	3	4
A		**R**		
B				**R**
C	**R**			
D			**R**	

Por lo tanto nuestro nuevo NodoActual es [C1,A2,D3,B4] h = 0

Aunque ya logramos la solución el algoritmo termina hasta que busquemos a todos los vecinos y no haya un vecino con una h menor entonces sigamos

Calculamos los nuevos vecinos

	1	2	3	4
A	**1**	**R**	**2**	**1**
B	**2**	**2**	**2**	**R**
C	**R**	**2**	**2**	**3**
D	**1**	**2**	**R**	**1**

No hay un vecino con una h menor o igual a 0 por lo tanto terminamos y regresamos la solución que es el NodoActual = [C1,A2,D3,B4]