Hill Climbing

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

12-2022

Contenido

1. Búsqueda de escalada (Hill Climbing)
   - Aplicaciones

Búsqueda de escalada

- Las búsquedas anteriormente vistas tiene el inconveniente de requerir mucha memoria, ya que necesitan ir guardando todo el árbol (BFS) o una parte de éste (DFS), así como los nodos que ya han sido visitados.
- Existe un grupo de algoritmos, llamados de **búsqueda local**, que intentan resolver este problema.
  - Lo logran no guardando todos los nodos (o el camino).
  - Aunque..., no siempre llegan a la mejor solución posible.
- Sin embargo, han logrado buenos resultado en problemas donde no se conocer el nodo que se está buscando, sólo se conocen las características que debe cumplir.

- El algoritmo de búsqueda local básico es el llamado "Ascensión de colinas" (Hill Climbing, en inglés).
- Este algoritmo intenta encontrar, lo más pronto posible, el punto máximo de una función (de ahí el nombre de ascensión de colinas).
- También se les conoce como **algoritmos de optimización**, ya que tratan de encontrar el máximo de una función.

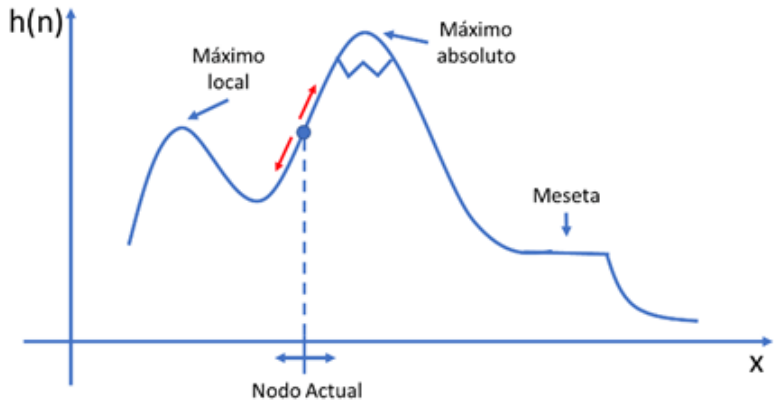
- Ascensión de colinas también utiliza una función heurística, $h(n)$, para evaluar un nodo n. El valor resultante indica que tan "bueno" es el nodo (función objetivo).
- De la misma manera que en el caso anterior (A*), el diseño de $h(n)$ no es trivial y depende completamente del problema.
- Eso si, la condición que debe cumplir la función heurística es que su punto máximo se logre en el nodo que tiene la solución buscada.

Procedure 1 HILL_CLIMBING

Input: $v : Vertex, g : Graph$

```
current ← v
found ← FALSE
while not found do
    Expand the current node to generate all its neighbors
    best ← NULL
    bestValue ← 0
    for each v in neighbors do
        if h(v) > bestValue then
            bestValue ← h(v)
            best ← v
        end if
    end for
    if h(current) > bestValue then
        found ← TRUE
    else
        current ← best
    end if
end while
```

- Como se pude observar, el algoritmo de escalada es un algoritmo avaro y tiene los mismo problemas que cualquier algoritmo avaro tiene: cada vez que se ejecuta, lo único que puede garantizar es llegar a un óptimo local.



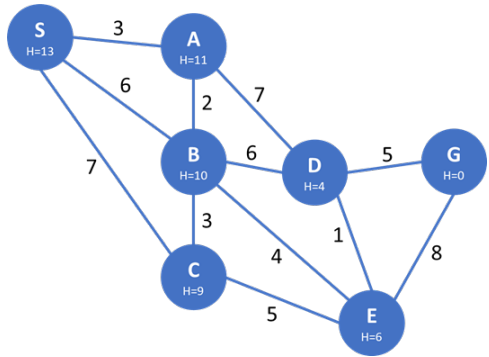
Para tratar de escapar de los óptimo locales, se han diseñado varios mecanismos que han dado lugar a nuevos algoritmos:

- **Ascensión de Colinas con Reinicio Aleatorio:** Se ejecuta el algoritmo original varias veces, partiendo siempre de un nodo inicial aleatoriamente seleccionado, quedándose con la mejor solución.
- **Ascensión de Colinas Aleatorio:** Se selecciona un vecino en forma aleatoria, de entre todos los que son mejores que el nodo actual.
- **Ascensión de Colinas con Primera Opción:** Se selecciona un vecino en forma aleatoria. Si el vecino es mejor que el actual, se hace le considera el nuevo actual y el proceso continúa. Si no es mejor, se selecciona otra de forma aleatoria. Esto es muy útil cuando el número de vecinos es muy grande.

Aplicaciones

- Las aplicaciones para el algoritmo de búsqueda de escalada tienen un espacio de búsqueda discreto (como un grafo). Por lo general, no se conoce el nodo final, sólo se conocen las características que debe tener y, normalmente, se puede iniciar en cualquier nodo del espacio, por lo que se puede seleccionar en forma aleatoria, aunque en algunos problemas se tiene que iniciar en un estado muy específico.
- Los problemas con espacios de búsqueda continuos (como las funciones) se pueden resolver con este método "discretizándolos" de alguna forma y, por lo general, se logra llegar muy cerca del resultado óptimo

- Apliquemos el algoritmo de búsqueda de escalada, al problema de encontrar la distancia mínima (no el camino, porque no se guarda, aunque si modificamos un poco el algoritmo lo podríamos obtener, pero perderíamos la ventaja de la poca memoria que requiere) que hay para llegar del nodo S al G



Si usamos como función la h(n) el proceso sería el siguiente:

1. Hacemos actual $= S$, cuya evaluación es $H = 13$. El costo es 0.
2. Como S no es G, generamos sus vecinos que son: A con 11, B con 10 y C con 9.
3. El mejor de los vecinos es el que tenga la menor evaluación. En este caso es C con 9. Como 9 es mejor que 13 del nodo actual nos movemos a él haciendo nodo actual $= C$. El nuevo costo es 7.
4. Como C no es G, generamos sus vecinos que son: S con 13, B con 10 y E con 6. 5. El mejor de los vecinos es E con 6. Como 6 que 9, nos movemos a E. El nuevo costo se obtiene sumando el costo actual 7 más 5 lo que da un costo de 12.
5. Como E no es G, generamos sus vecinos que son: C con 9, B con 10, D con 4 y G con 0.
6. El mejor de los vecinos es G con 0. Como 0 es mejor que 6, nos movemos a él haciendo nodo actual $= G$. El nuevo costo se obtiene sumando el costo actual 12 más 8, lo que da un costo de 20.
7. Como G es el nodo final, el proceso termina y regresamos un costo de 20.