Búsqueda Local y Optimización

Sistemas Inteligentes

Dr. Víctor de la Cueva
vcueva@itesm.mx

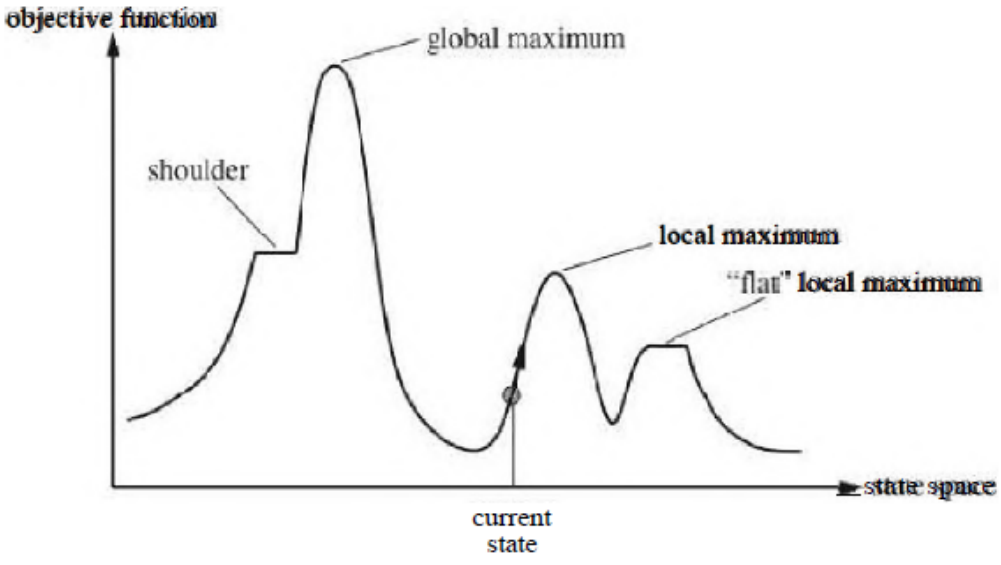
Búsqueda Global vs Local

- Los algoritmos que hemos visto hasta ahora buscan por TODO (búsqueda global) el espacio de búsqueda recorriéndolo sistemáticamente.
- Cuando se encuentra un estado meta, la solución es el camino completo de la raíz a la meta.
- En muchos problemas el camino es irrelevante, lo que importa es llegar a un estado meta, el cual es la solución (e.g. 8-reinas).
- Los algoritmos de búsqueda local operan usando un solo nodo actual (más que múltiples caminos) y generalmente se mueven sólo a los vecinos de dicho nodo.
- Típicamente, los caminos no son guardados.

Búsqueda Local

- Aunque los algoritmos de búsqueda local no son sistemáticos, tienen un par de ventajas clave :
  - Usan muy poca memoria, generalmente una cantidad constante.
  - Generalmente, pueden encontrar soluciones razonables en espacios de estados muy grandes o infinitos (continuos) para los cuales los sería imposible aplicar los algoritmos sistemáticos.
- Además de encontrar metas, estos algoritmos son útiles para resolver problemas de optimización en los cuales se trata de encontrar el mejor estado de acuerdo a una función de optimización.

Ejemplo de espacio de estados continuo



Hill Climbing (Ascensión de Colinas)

- Es la búsqueda local básica.
- En cada paso el nodo actual es reemplazado por el "mejor" vecino.
- Los vecinos reciben una evaluación por medio de una función, llamada función de evaluación.
- El "mejor" vecino es el que tiene el mejor valor (mayor o menor dependiendo del problema).
- No observa más allá de la vecindad (los nodos a los que se llega con una sola acción) del estado actual (e.g. tratar de encontrar la punta del Everest, con neblina y con amnesia).
- Es a veces llamada "Búsqueda Voraz Local" (Greedy Local Search).

Ventajas y desventajas de HC

- Ventajas:
  - Ocupa muy poca memoria. Sólo el nodo actual, sus vecinos y la función objetivo de cada vecino.
  - Es sumamente simple de implementar.
- Desventaja:
  - Sólo garantiza llegar a un óptimo local (desde luego que a veces puede encontrar a un óptimo global dependiendo de dónde inicie).

Algoritmo Hill Climbing

```
función ASCENSIÓN-COLINAS(problema) devuelve un estado que es un máximo local
  entradas: problema, un problema
  variables locales: actual, un nodo
                     vecino, un nodo

  actual ← HACER-NODO(ESTADO-INICIAL[problema])
  bucle hacer
    vecino ← sucesor de valor más alto de actual
    si VALOR[vecino] ≤ VALOR[actual] entonces devolver ESTADO[actual]
    actual ← vecino
```

HC para el 8-queens

- Si se aplica HC directamente al problema de las 8-reinas, con una reina por columna y como acción válida mover una reina un cuadrito hacia arriba o hacia abajo:
  - El 86% de las veces llega a un óptimo local que no es el óptimo absoluto. Sólo el 14% de las veces encuentra la solución.
- Si se le permite moverse a un vecino aún y cuando sea igual (sideways movements) se aumenta el porcentaje de 14% a 94%.

Quitando voracidad a HC

- El problema con el algoritmo HC es que siempre selecciona el mejor (Voraz), esto sólo garantiza llegar a un óptimo local, el cual algunas veces es el absoluto.
- Una forma de mejorarlo es quitarle algo de voracidad, permitiendo que seleccione algunas veces un vecino aún y cuando no sea el mejor.
- Se han inventado algunos enfoques para hacer esto en los cuales hay las dos opciones:
  - Permitir la selección uno mejor aunque no sea tan bueno.
  - Permitir la selección de uno peor.

Variantes de HC

- HC Estocástico
  - De entre los mejores, se selecciona uno en forma aleatoria.
- HC de Primera Opción
  - Se selecciona un vecino en forma aleatoria. Si es mejor se deja, si es peor se deshecha y se selecciona otro.
  - Muy bueno cuando hay muchos vecinos.
- HC con Reinicio Aleatorio
  - Si no se llega a la solución, selecciona otro inicio en forma aleatoria e inicia de nuevo.
  - Depende de la probabilidad p (distribución geométrica) de llegar a la solución en un HC normal (se requieren 1/p iteraciones en promedio).

Recocido Simulado

- Un algoritmo que nunca baja (HC) es incompleto (garantiza llegar a un óptimo local).
- Un algoritmo completamente aleatorio (camino aleatorio) es probabilísticamente completo pero muy ineficiente (lento).
- Si se combina resulta el Recocido Simulado (*Simulated Annealing*).
- La idea se tomó del enfriamiento de los metales. Se encontró que si un metal se calienta mucho y se enfría muy (pero muy) lentamente, sus cristales se acomodan en forma óptima y el metal obtenido es muy duro (si se enfría rápido se hace quebradizo).

Ciclo del SA

- Es similar al del HC, sólo que en lugar de seleccionar al mejor, selecciona uno al azar:
  - Si mejora, se acepta.
  - Si empeora se acepta con una probabilidad (menor a 1).
  - Dicha probabilidad disminuye exponencialmente con "que tan mala es la evaluación", es decir, la cantidad ΔE por la que se empeora la evaluación.
  - La probabilidad también disminuye conforme disminuye la temperatura (al inicio es más grande y se va enfriando).
- Se puede demostrar que si T disminuye muy (pero muy) despacio, se encuentra un óptimo global con probabilidad cercana a 1.

Recocido Simulado

función TEMPLE-SIMULADO(*problema, esquema*) **devuelve** un estado solución
  entradas: *problema*, un problema
    esquema, una aplicación desde el tiempo a «temperatura»
  variables locales: *actual*, un nodo
    siguiente, un nodo
    T, una «temperatura» controla la probabilidad de un paso hacia abajo

  actual ← HACER-NODO(ESTADO-INICIAL[*problema*])
  para $t \leftarrow 1$ **a** ∞ **hacer**
    $T \leftarrow esquema[t]$
    si $T = 0$ **entonces devolver** *actual*
    siguiente ← un sucesor seleccionado aleatoriamente de *actual*
    $\Delta E \leftarrow$ VALOR[*siguiente*] − VALOR[*actual*]
    si $\Delta E > 0$ **entonces** *actual* ← *siguiente*
    en caso contrario *actual* ← siguiente sólo con probabilidad $e^{\Delta E/T}$

Una versión de la Ascensión de Colinas estocástico donde se permite descender a algunos movimientos.
La entrada del esquema determina el valor de T como una función del tiempo.

Variantes de Búsqueda Local

- Búsqueda de Haz Local
  - Guardar sólo un nodo en la memoria es muy extremo.
  - El algoritmo de Haz Local guarda la pista de k estados.
  - Comienza con k estados iniciales generados en forma aleatoria.
  - En cada paso se generan TODOS los vecinos de los k estados
    - Si alguno es un objetivo, paramos.
    - Si no, se seleccionan los k mejores sucesores de la lista completa y repetimos.
- Búsqueda de Haz Estocástica
  - Análoga a HC Estocástico.
  - Escoge k sucesores aleatoriamente con una probabilidad de elegir a un sucesor como una función creciente de su valor.
  - Se parece a la sección natural (los mejores tienen mayor probabilidad de ser seleccionados).

Algoritmos Genéticos

- Utilizan la heurística de la selección natural, donde el mejor adaptado tiene mayores probabilidades de sobrevivir, tener mayor descendencia y pasarles su información genética.
- En la naturaleza, cada individuo se puede considerar una respuesta que se logró adaptar para resolver un problema (falta de alimento, depredadores, etc.).
- Los operadores fundamentales de búsqueda son la cruza de la información genética entre padres y la mutación.
- La función que evalúa a los individuos se conoce como función de aptitud (*fitness function*).

Algoritmo Genético

```
función ALGORITMO-GENÉTICO(población, IDONEIDAD) devuelve un individuo
  entradas: población, un conjunto de individuos
            IDONEIDAD, una función que mide la capacidad de un individuo

  repetir
    nueva_población ← conjunto vacío
    bucle para i desde 1 hasta TAMAÑO(población) hacer
      x ← SELECCIÓN-ALEATORIA(población, IDONEIDAD)
      y ← SELECCIÓN-ALEATORIA(población, IDONEIDAD)
      hijo ← REPRODUCIR(x,y)
      si (probabilidad aleatoria pequeña) entonces hijo ← MUTAR(hijo)
      añadir hijo a nueva_problación
    población ← nueva_población
  hasta que algún individuo es bastante adecuado, o ha pasado bastante tiempo
  devolver el mejor individuo en la población, de acuerdo con la IDONEIDAD

función REPRODUCIR(x,y) devuelve un individuo
  entradas: x,y, padres individuales

  n ← LONGITUD(x)
  c ← número aleatorio de 1 a n
  devolver AÑADIR(SUBCADENA(x, 1, c),SUBCADENA(y, c + 1, n))
```

Proyecto 4

Referencia

- S. Russel and P. Norvig. Inteligencia Artificial un enfoque moderno. 2ª edición, Pearson, España (2004).