Problemas de optimización

Inteligencia Artificial

OPTIMIZACIÓN

En algunos casos, solo nos interesa el **estado final**, no el camino para llegar allí.

Los algoritmos de **búsqueda local** operan explorando desde un estado inicial hacia estados vecinos, sin registrar los caminos recorridos ni los estados alcanzados. Esto significa que no son sistemáticos, podrían nunca explorar una parte del espacio de búsqueda donde realmente está la solución.

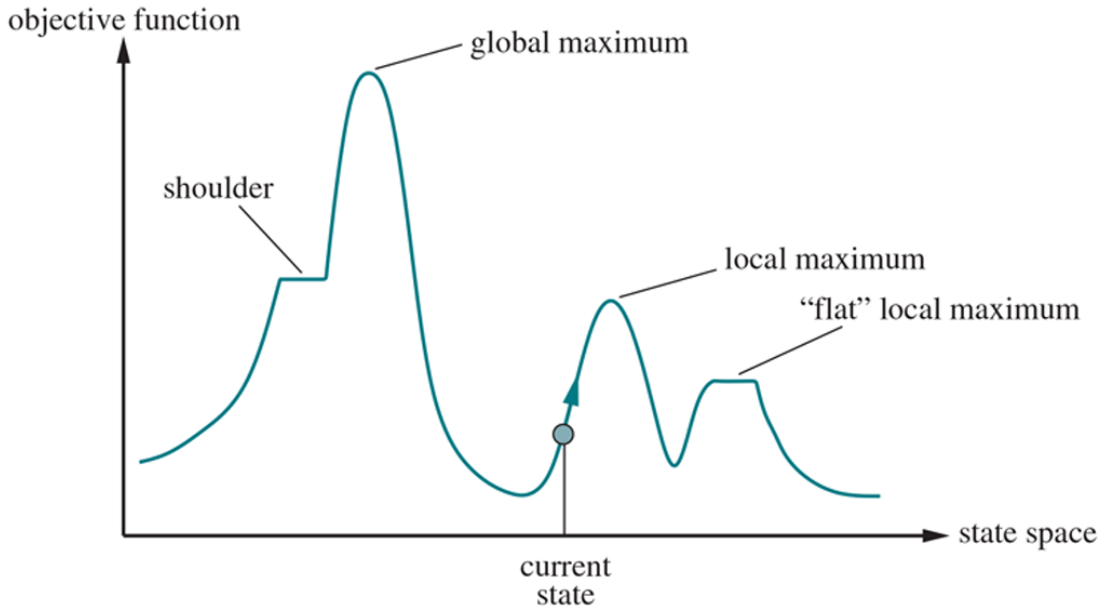
OPTIMIZACIÓN

Sin embargo, tienen dos ventajas clave:

- Requieren muy poca memoria.
- Pueden encontrar soluciones razonables en espacios de estado muy grandes o infinitos, donde los algoritmos sistemáticos son inadecuados.

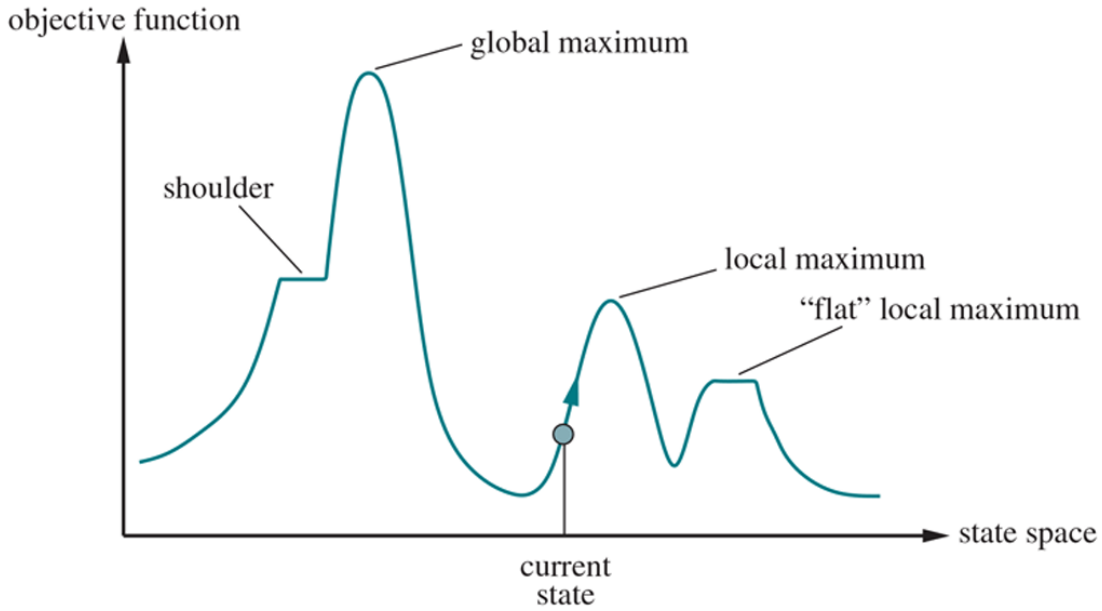
Los **algoritmos de búsqueda local** también pueden resolver **problemas de optimización**, en los cuales el objetivo es encontrar el **mejor estado** según una **función objetivo**.

OPTIMIZACIÓN



Para entender la búsqueda local, imaginemos los estados de un problema distribuidos en un paisaje de espacio de estados, como se muestra en la Figura, Cada **punto (estado)** en este paisaje tiene una **"elevación"**, definida por el valor de la **función objetivo**.

OPTIMIZACIÓN



- Si la elevación corresponde a una función objetivo que debe maximizarse, entonces el objetivo es encontrar el pico más alto—un máximo global—y este proceso se llama **ascenso por colinas (hill climbing).**
- Si la elevación representa un costo que debe minimizarse, entonces el objetivo es encontrar el valle más profundo—un mínimo global—y este proceso se llama **descenso de gradiente (gradient descent).**

HILL CLIMBING

OPTIMIZACIÓN-HILL CLIMBING

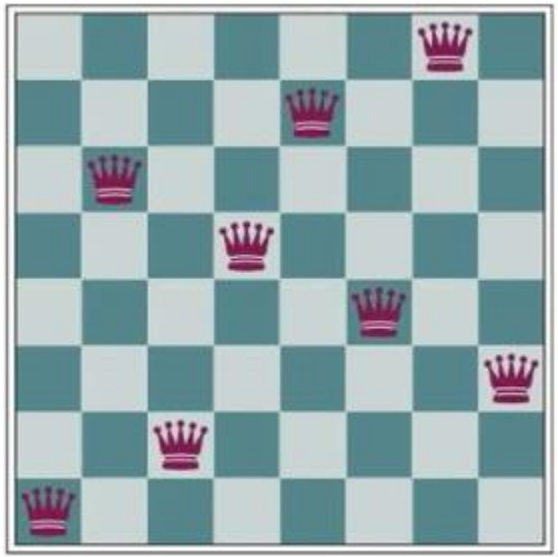
El **algoritmo de búsqueda por ascenso de colinas** (*hill-climbing search algorithm*) mantiene un único **estado actual** y, en cada iteración, **se mueve al estado vecino con el valor más alto**—es decir, avanza en la dirección que proporciona el **ascenso más pronunciado**.

El algoritmo **termina cuando alcanza un "pico"**, donde ningún vecino tiene un valor más alto. **El ascenso por colinas no mira más allá de los vecinos inmediatos del estado actual**.

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← problem.INITIAL
    while true do
        neighbor ← a highest-valued successor state of current
        if VALUE(neighbor) ≤ VALUE(current) then return current
        current ← neighbor
```

OPTIMIZACIÓN-HILL CLIMBING

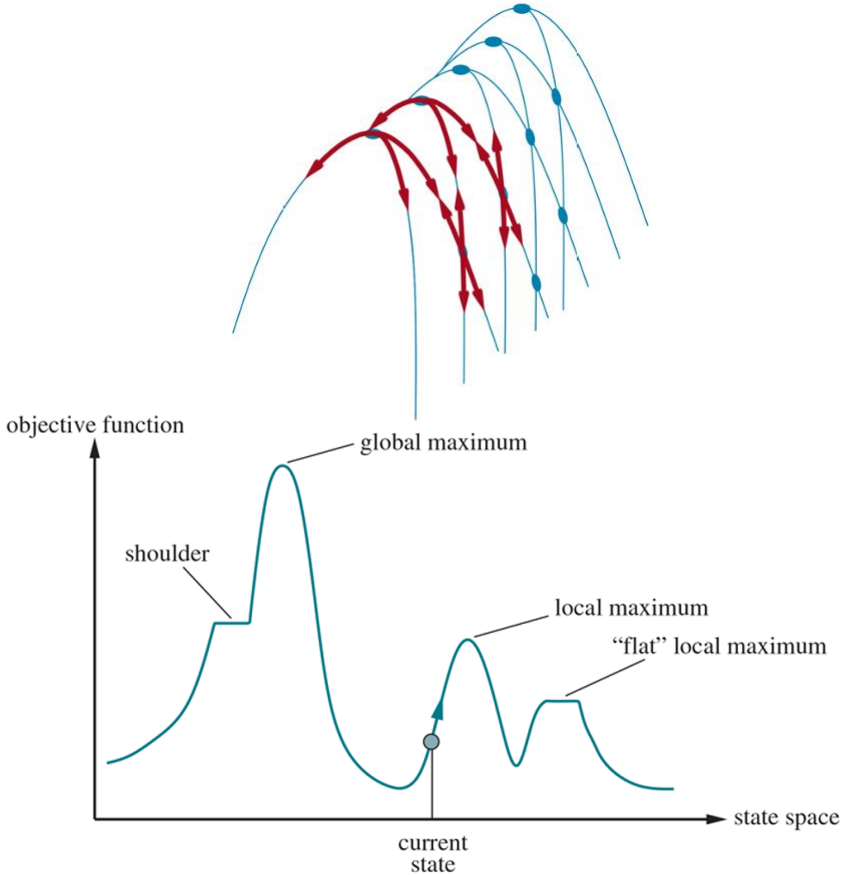
Para adaptar **Hill Climbing**, que usualmente **maximiza**, simplemente lo aplicamos a **la versión negativa de la función heurística**, es decir, usamos **−h como función objetivo**.



OPTIMIZACIÓN-HILL CLIMBING

Hill Climbing puede quedarse atascado por cualquiera de las siguientes razones:

- **MÁXIMOS LOCALES:** Un máximo local es un pico que es más alto que cada uno de sus estados vecinos, pero más bajo que el máximo global. Los algoritmos de ascensión de colinas que alcanzan la vecindad de un máximo local serán arrastrados hacia la cima, pero luego quedarán atrapados sin otra opción para continuar.
- **CORDILLERAS (RIDGES):** Las cordilleras resultan en una secuencia de máximos locales que son muy difíciles de atravesar para los algoritmos codiciosos.
- **MESETAS**: Una meseta es un área plana del paisaje del espacio de estados. Puede ser un máximo local plano, del cual no existe una salida ascendente, o un hombro, desde el cual es posible progresar.



OPTIMIZACIÓN-HILL CLIMBING

Se han inventado muchas variantes de **Hill Climbing**.

•**Ascensión de colinas estocástica** (**stochastic hill climbing**) elige aleatoriamente entre los movimientos ascendentes; la probabilidad de selección puede variar con la inclinación del movimiento ascendente. Generalmente, este método converge más lentamente que la ascensión más pronunciada, pero en algunos paisajes de estados, encuentra mejores soluciones.

•**Ascensión de colinas con primera elección** (**first-choice hill climbing**) implementa la ascensión estocástica generando sucesores aleatoriamente hasta que se encuentra uno que sea mejor que el estado actual. Esta es una buena estrategia cuando un estado tiene **muchos** (por ejemplo, miles) de sucesores.

OPTIMIZACIÓN-HILL CLIMBING

Se han inventado muchas variantes de **Hill Climbing**.

- **Búsqueda en ascenso con reinicio aleatorio** (*random-restart hill climbing*), que adopta el dicho: "Si al principio no tienes éxito, inténtalo de nuevo." Este método realiza una serie de búsquedas de ascenso de colinas desde estados iniciales generados aleatoriamente hasta encontrar una solución. Es **completo con probabilidad 1**, porque eventualmente generará un estado meta como el estado inicial.

NOTA: *Si cada búsqueda de ascenso de colinas tiene una probabilidad **p** de éxito, entonces el número esperado de reinicios necesarios es **1/p**. Para instancias del problema de las 8 reinas sin movimientos laterales permitidos, **p ≈ 0.14**, por lo que se necesitan aproximadamente **7 iteraciones** para encontrar una solución (6 fracasos y 1 éxito).*

SIMULATED ANNEALING

OPTIMIZACIÓN-SIMULATED ANNEALING

- Un algoritmo de **búsqueda en ascenso de colinas** que **nunca realiza movimientos "hacia abajo"** (hacia estados con menor valor o mayor costo) siempre es vulnerable a quedar atrapado en un **máximo local**.

- En contraste, una **búsqueda aleatoria pura**, que se mueve a un estado sucesor sin considerar su valor, eventualmente encontrará el máximo global, pero será extremadamente ineficiente.

- Por lo tanto, parece razonable combinar Hill Climbing con un movimiento aleatorio para lograr tanto eficiencia como completitud.

El *simulated annealing* es uno de estos algoritmos.

OPTIMIZACIÓN-SIMULATED ANNEALING

En **metalurgia,** *annealing* es el proceso utilizado para **templar o endurecer metales y vidrio**, calentándolos a una temperatura alta y luego enfriándolos gradualmente, permitiendo que el material alcance un estado cristalino de **baja energía**.

Para explicar el **simmulated annealing**, cambiamos nuestra perspectiva de **búsqueda en ascenso de colinas** a **descenso por gradiente**. (Es decir, minimizando el costo)

OPTIMIZACIÓN-SIMULATED ANNEALING

Es similar a la búsqueda en ascenso de colinas (*hill climbing*). Sin embargo, en lugar de elegir el **mejor** movimiento, selecciona un **movimiento aleatorio**. Si el movimiento mejora la situación, siempre se acepta. De lo contrario, el algoritmo acepta el movimiento con una probabilidad menor que 1.

Esta probabilidad **disminuye exponencialmente** con la medida de que tan malo es el movimiento, es decir:

$$\Delta E = \text{Valor}(\text{nueva solución}) - \text{Valor}(\text{solución actual})$$

Además, la probabilidad **también disminuye a medida que la "temperatura" T baja**: los "malos" movimientos son más probables al inicio (cuando T es alta) y se vuelven menos probables a medida que T disminuye.

$$e^{\Delta E/T}$$

OPTIMIZACIÓN-SIMULATED ANNEALING

Si la función de enfriamiento (*schedule*) reduce T a 0 lo suficientemente lento, entonces, según una propiedad de la distribución de Boltzmann:

$$e^{\Delta E/T}$$

La probabilidad se concentrará en el óptimo global, lo que hará que el algoritmo lo encuentre con una probabilidad cercana a 1.

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

OPTIMIZACIÓN-SIMULATED ANNEALING

Ten en cuenta que la condicion de parada del algoritmos esta asociada a la Temperatura T:

Cuándo ocurre que T=0?

1. **Enfriamiento geométrico:**

$$T(t) = T_0 \cdot \alpha^t$$

Donde α (por ejemplo, 0.99) es un factor de reducción en cada iteración.

2. **Enfriamiento logarítmico:**

$$T(t) = \frac{T_0}{1 + \beta \log(1 + t)}$$

Donde β controla la rapidez de enfriamiento.

3. **Enfriamiento lineal:**

$$T(t) = T_0 - \lambda t$$

Donde λ es la tasa de decremento.

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

EVOLUTIONARY ALGORITHMS

OPTIMIZACIÓN-EVOLUTIONARY ALGORITHMS

Los **algoritmos evolutivos** pueden verse como variantes de la búsqueda estocástica en haz que están explícitamente motivadas por la metáfora de la selección natural en biología: existe una población de individuos (estados), en la que los individuos más aptos (con mayor valor) producen descendencia (estados sucesores) que poblarán la siguiente generación, un proceso llamado **recombinación**.

Los algoritmos evolutivos varían en los siguientes aspectos:

•El tamaño de la población.
•La representación de cada individuo. En los **algoritmos genéticos**, cada individuo es una cadena sobre un alfabeto finito (a menudo una cadena booleana), al igual que el ADN es una cadena sobre el alfabeto **ACGT**. En las **estrategias evolutivas**, un individuo es una secuencia de números reales, y en la **programación genética**, un individuo es un programa informático.

OPTIMIZACIÓN-EVOLUTIONARY ALGORITHMS

El número de mezcla, **(ρ)** representa la cantidad de padres que se combinan para formar descendencia. El caso más común es

ρ = 2: dos padres combinan sus "genes" (partes de su representación) para formar descendencia.

Cuando ρ = 1, se tiene una búsqueda estocástica en haz (que puede verse como una reproducción asexual). Es posible tener ρ > 2, lo cual es poco común en la naturaleza pero fácil de simular en computadoras.

OPTIMIZACIÓN-EVOLUTIONARY ALGORITHMS

Proceso de selección para elegir a los individuos que se convertirán en los padres de la siguiente generación:

- una posibilidad es seleccionar entre todos los individuos con una probabilidad proporcional a su puntaje de aptitud(valor).

- Otra posibilidad es seleccionar aleatoriamente n individuos ($n > \rho$) y luego elegir los más aptos como padres.

OPTIMIZACIÓN-EVOLUTIONARY ALGORITHMS

Proceso de selección para elegir a los individuos que se convertirán en los padres de la siguiente generación:

- una posibilidad es seleccionar entre todos los individuos con una probabilidad proporcional a su puntaje de aptitud(valor).

- Otra posibilidad es seleccionar aleatoriamente n individuos ($n > \rho$) y luego elegir los más aptos como padres.

OPTIMIZACIÓN-EVOLUTIONARY ALGORITHMS

RECOMBINACIÓN:
Un enfoque común (asumiendo **ρ = 2**) es seleccionar aleatoriamente un **punto de cruce** (*crossover point*) para dividir las cadenas de cada padre y recombinar las partes para formar dos hijos:

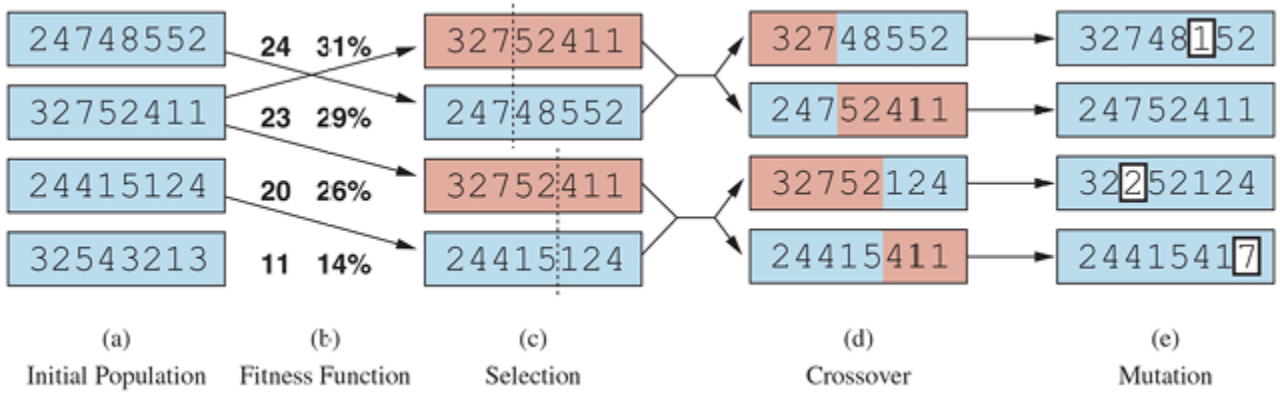
- Uno con la primera parte del padre 1 y la segunda parte del padre 2.
- Otro con la segunda parte del padre 1 y la primera parte del padre 2.

Tasa de Mutación:

- Determina con qué frecuencia la descendencia tendrá mutaciones aleatorias en su representación.

OPTIMIZACIÓN-EVOLUTIONARY ALGORITHMS

Para un estado del problema de las 8 reinas: el dígito en la posición c representa el número de fila de la reina en la columna c. En cada estado es evaluado por la función de aptitud. Los valores de aptitud más altos son mejores, por lo que en el problema de las 8 reinas utilizamos el número de pares de reinas que no se atacan. Para una solución óptima, este valor es **8 × 7 / 2 = 28.**



OPTIMIZACIÓN-EVOLUTIONARY ALGORITHMS

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Inteligencia Artificial

https://stability.ai/blog/stable-diffusion-public-release

https://openai.com/blog/chatgpt/

https://ai.googleblog.com/2022/06/minerva-solving-quantitative-reasoning.html

https://www.youtube.com/watch?v=jMvLCZBXbtc

https://www.deepmind.com/blog/tackling-multiple-tasks-with-a-single-visual-language-model

https://www.deepmind.com/blog/building-safer-dialogue-agents

https://www.deepmind.com/publications/a-generalist-agent

https://www.deepmind.com/research/highlighted-research/alphago

https://ai.facebook.com/research/cicero/

https://openai.com/blog/whisper/

https://valle-demo.github.io/