¿Qué es Hill Climbing?

Hill Climbing (o **ascenso de colina**) es un algoritmo de **búsqueda local** usado para encontrar una solución que optimice (maximice o minimice) una función.
La idea es sencilla:

- **Partes de una solución inicial** (un valor cualquiera).
- **Exploras vecinos** (soluciones cercanas).
- **Te mueves al vecino mejor** si mejora la función.
- Repites hasta que no haya mejoras.

Analogía simple

Imagina que estás en una montaña con niebla:

- Solo puedes ver unos pasos a tu alrededor.
- Das pasos siempre hacia donde sube más la pendiente.
- Eventualmente llegas a la cima **local** (aunque puede no ser la cima más alta globalmente).

Pasos del algoritmo

1. **Elegir un punto inicial** (ejemplo: una temperatura inicial al azar).
2. **Evaluar la función objetivo** en ese punto (ejemplo: qué tan buena es esa temperatura).
3. **Generar vecinos** (valores cercanos).
4. **Comparar**: si un vecino es mejor, moverte a él.
5. **Repetir** hasta que no haya mejora o se alcance un límite de pasos.

Uso típico

1. **Optimización de funciones matemáticas**
    - Ejemplo: encontrar la temperatura ideal para máxima eficiencia energética.
2. **Problemas de búsqueda**
    - Ejemplo: asignar recursos, rutas o configuraciones.
3. **IA y aprendizaje automático**
    - Ajustar parámetros de un modelo de forma simple.
4. **Problemas con restricciones físicas**

- Ejemplo: mantener motores o reactores en una temperatura estable.

Ventajas

✅ Sencillo de implementar.
✅ Rápido para problemas pequeños.
✅ Funciona bien si la función es unimodal (tiene un único máximo).

Ejemplo aplicado (temperaturas)

Supongamos que queremos encontrar la **temperatura ideal de un motor** que funciona mejor cerca de 90 °C.

- Definimos `f(t) = -(t-90)**2 + 300`.
- Hill Climbing empezará en un valor aleatorio, por ejemplo, 70 °C.
- Irá probando valores cercanos (71, 69, 72, etc.).
- Si uno es mejor, se mueve ahí.
- Repite hasta que encuentre el punto donde no hay mejora → aprox. **90 °C**.

Ejemplo numérico claro usando la función del motor f(t)=−(t−90)2+300.

1) Paso a paso conceptual

1. **Elegir estado inicial**: seleccionas una temperatura inicial t0.
2. **Evaluar** la función objetivo f(t0).
3. **Generar vecinos** cercanos (por ejemplo t0+Δ y t0−Δ).
4. **Evaluar los vecinos** y escoger el mejor vecino (el que tiene mayor f si buscas maximizar).
5. **Moverte** al vecino si mejora: t1 ← vecino mejor. Si ninguno mejora, **detener** (óptimo local o meseta).
6. **Repetir** hasta criterio de paro (sin mejoras, alcanzar iteraciones máximas, o mejora menor que ε).

2) Pseudocódigo (versión simple — vecinos ±Δ)

```
t = t0
valor = f(t)
while not stopping_condition:
    up = t + delta
    down = t - delta
    if f(up) > valor and f(up) >= f(down):
        t = up
        valor = f(up)
    elif f(down) > valor:
        t = down
        valor = f(down)
    else:
```

```
        break   # no hay mejora: detener
return t, valor
```

3)Ejecución numérica (ejemplo detallado)

- **Función**: f(t)=−(t−90)2+300 (máximo en t=90).
- **Parámetros**: temperatura inicial t0=70.0, paso Δ=1.5.
- Evaluamos vecinos `t+Δ` y `t-Δ` y nos movemos al que mejore.

Voy mostrando las iteraciones (cálculos explícitos para las primeras):

Iteración 0 (estado inicial)

- t0=70.0
- f(70.0)=−(70−90)2+300=−202+300=−400+300=−100.00

Iteración 1 (vecinos 71.5 y 68.5)

- tup=71.5 → f(71.5)=−(71.5−90)2+300
  =−18.52+300=−342.25+300=−42.25
- tdown=68.5 → f(68.5)=−(21.52)+300=−462.25+300=−162.25
- Mejor vecino: **71.5** (porque −42.25>−162.25 y −42.25>−100).
- Movimiento: t1=71.5.

Iteración 2 (vecinos 73.0 y 70.0)

- tup=73.0 → f(73)=−(17)2+300=−289+300=11.00
- tdown=70.0 → ya sabemos f(70)=−100
- Mejor: **73.0** → nos movemos a t2=73.0.

A continuación se resumen más iteraciones (mismos cálculos):

Iter	Temperatura actual (°C)	f(t)	Movimiento
0	70.0	-100.00	inicio
1	71.5	-42.25	up
2	73.0	11.00	up
3	74.5	59.75	up
4	76.0	104.00	up
5	77.5	143.75	up
6	79.0	179.00	up
7	80.5	209.75	up
8	82.0	236.00	up
9	83.5	257.75	up

Iter	Temperatura actual (°C)	f(t)	Movimiento
10	85.0	275.00	Up

Observación: con Δ=1.5 el algoritmo sube paso a paso hacia el óptimo (90°C). Si continuáramos, seguiríamos acercándonos; cuando lleguemos a un punto donde ni `t+Δ` ni `t-Δ` mejoren más, el algoritmo se detendrá (óptimo local respecto a Δ).

4) Parámetros y su efecto

- **Paso (Δ)**:
  - Muy pequeño → convergencia lenta, mejor precisión.
  - Muy grande → puede "saltar" el óptimo o oscilar; ayuda a escapar de pequeños óptimos locales, pero puede perder resolución.
- **Vecinos**: generar más vecinos por iteración (p. ej. muestrear k vecinos aleatorios) ayuda a explorar mejor.
- **Criterio de paro**: no mejora, mejora < ε, o tope de iteraciones.
- **Rangos/boundaries**: fijar límites (p. ej. temperatura entre 0 y 200 °C) y truncar vecinos fuera del rango.

5) Problemas comunes y soluciones

- **Óptimos locales**: usar *random restarts* (ejecutar muchas veces con diferentes t0 y quedarte con el mejor resultado).
- **Mesetas** (muchos vecinos con mismo valor): usar selección aleatoria entre iguales o disminuir Δ adaptativamente.
- **Cordilleras / ridge**: cambiar la estrategia de vecinos (direcciones no alineadas al eje).
- **Necesidad de explorar**: usar variantes estocásticas o algoritmos que acepten peores soluciones (Simulated Annealing) para evitar quedar atrapados.

6) Variantes útiles para enseñar

- **Random-Restart Hill Climbing**: ejecutar HC varias veces desde distintos t0.
- **Stochastic Hill Climbing**: muestrear k vecinos aleatorios y moverse al mejor entre ellos (o elegir aleatoriamente un vecino mejor).
- **Simulated Annealing**: permite aceptar peores soluciones con cierta probabilidad (buen contraste frente a Hill Climbing).

7) Consejos prácticos para ejercicios en clase

- Pide que los alumnos prueben distintos Δ y comparen convergencia.
- Que midan cuántas ejecuciones con random restart son necesarias para encontrar el óptimo global en funciones multimodales.
- Que visualicen la trayectoria sobre la curva (como el gráfico que ya generamos) para ver cómo sube y dónde se queda atrapado.
- Comparar HC con Simulated Annealing en la misma función multimodal (p. ej. `sin` + `cos`).

Los **5 ejemplos cotidianos,** pensados desde la perspectiva de **maximización** (buscar lo mejor posible):

Ejemplo	Se maximiza o minimiza?	Qué significa en la vida real?
1. Subir una montaña a ciegas	**Maximizar** altura f(x)	Encontrar la cima más alta.
2. Buscar la mejor señal de Wi-Fi	**Maximizar** intensidad de señal	Quedarte donde el Wi-Fi es más fuerte.
3. Ajustar la temperatura del aire acondicionado	**Minimizar** la incomodidad (o maximizar comodidad)	Llegar a la temperatura más confortable (ej. 23 °C).
4. Mejorar una receta de cocina	**Maximizar** el “sabor” (una función subjetiva de ingredientes)	Encontrar la combinación que sabe mejor.
5. Estudiar para un examen	**Maximizar** rendimiento académico (notas, retención de info)	Elegir la estrategia más efectiva de estudio.

En resumen:

- **Hill Climbing puede usarse para ambos casos**:
  - **Maximización** → buscar el valor más grande posible de la función objetivo.
  - **Minimización** → buscar el valor más pequeño posible (ej. minimizar el error en un modelo).
- En los ejemplos que vimos: 4 son de **maximización** (buscar lo mejor) y 1 de **minimización** (ajustar el aire acondicionado para reducir incomodidad).