

4. BÚSQUEDA LOCAL

IA 3.2 - Programación III

1° C - 2023

Lic. Mauro Lucci

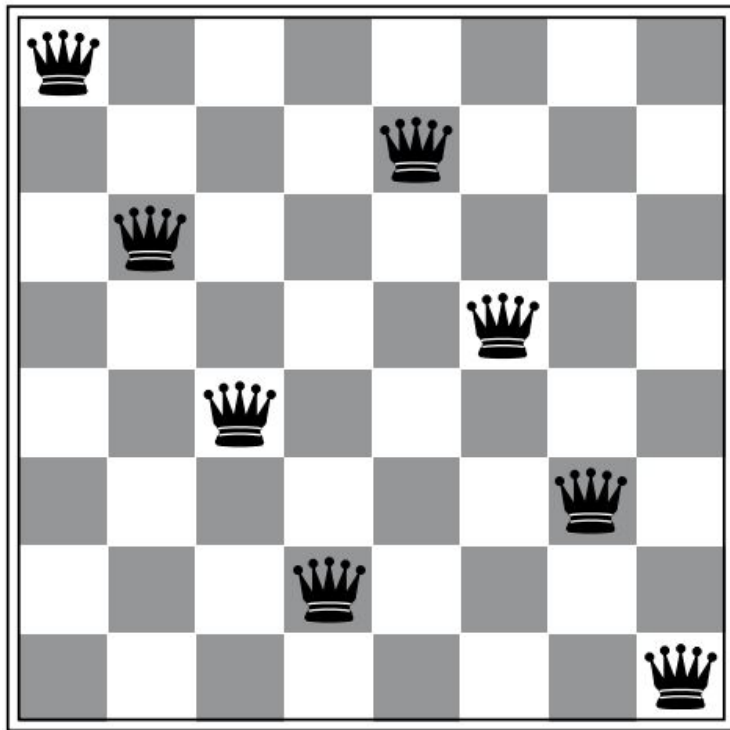


Repaso

- Los algoritmos de búsqueda vistos hasta ahora están diseñados para explorar el espacio de estados de forma sistemática desde el estado inicial.
- Mantienen uno o más caminos en memoria y registran cuáles alternativas ya han sido exploradas en cada punto del camino.
- Cuando un objetivo es encontrado, el camino al objetivo constituye una solución del problema.
- Pero en muchos problemas, el camino al objetivo es irrelevante.

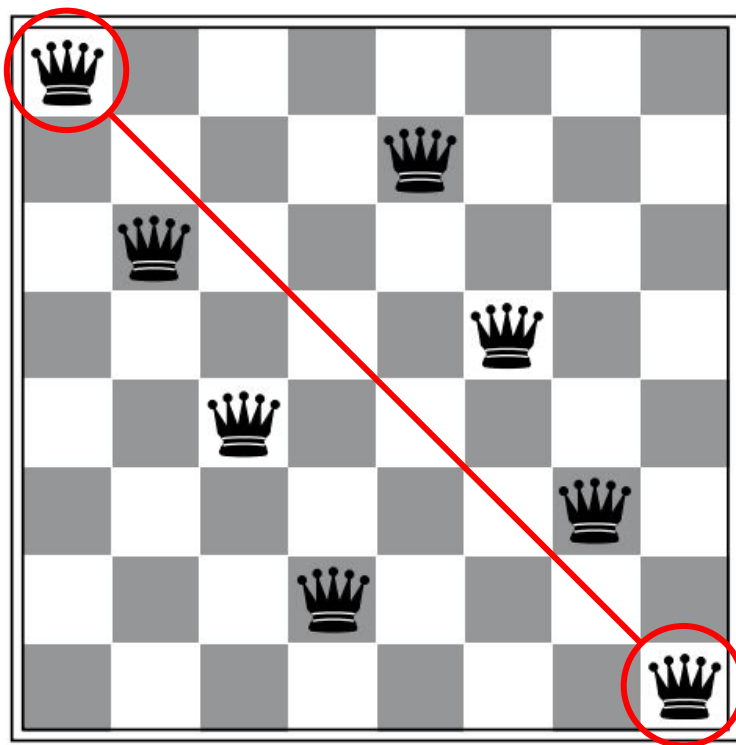
Problema de las 8 Reinas

— — —



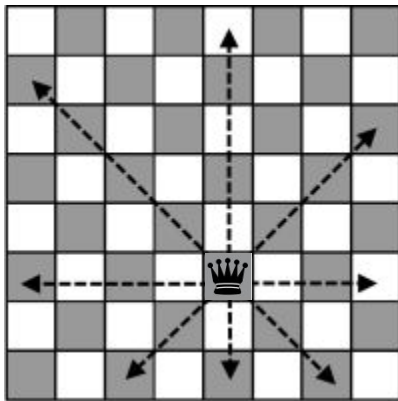
Problema de las 8 Reinas

— — —



Descripción

Objetivo. Ubicar 8 reinas en un tablero de ajedrez de forma tal que ninguna reina ataque a otra.



Reglas. Cada casilla puede contener a lo sumo una reina.



Desafío

— — —

¿Es posible resolver el problema?

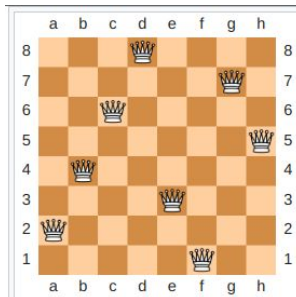


Respuesta

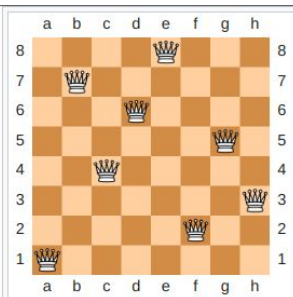
— — —

En total, hay 92 soluciones.

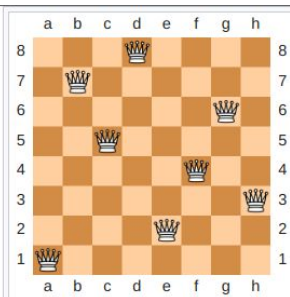
Sin contar soluciones simétricas, hay 12 soluciones.



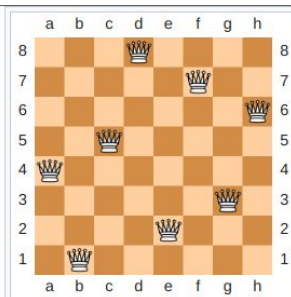
Solution 1



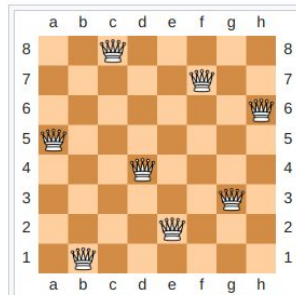
Solution 2



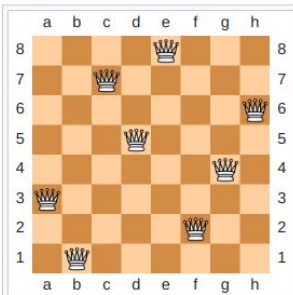
Solution 3



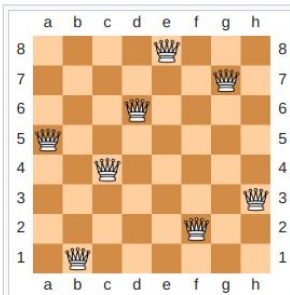
Solution 4



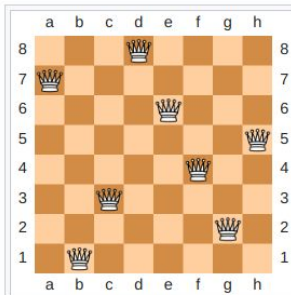
Solution 5



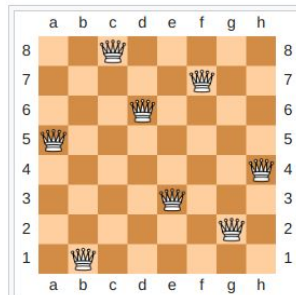
Solution 6



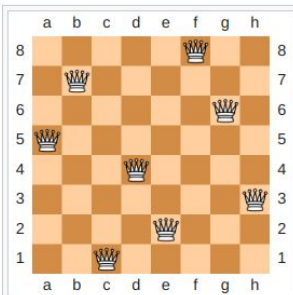
Solution 7



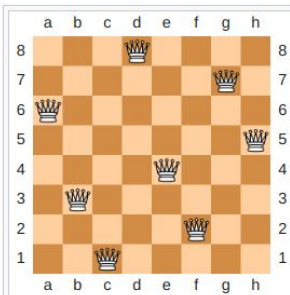
Solution 8



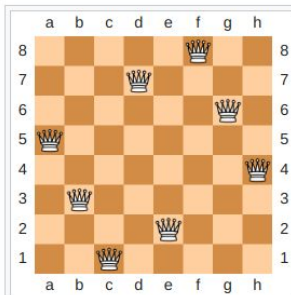
Solution 9



Solution 10



Solution 11



Solution 12

Formulación incremental (I)

— — —

- **Estados.** Matrices M de 8×8 tales que:

- $M_{ij} = 1$ si hay reina en la fila i y columna j .
- $M_{ij} = 0$ si no hay reina en la fila i y columna j .
- $(\sum_{ij} M_{ij}) \leq 8$ (a lo sumo 8 reinas en el tablero).

Hay $1 + 64 + 64 \times 63 + \dots + 64 \times 63 \times \dots \times 57 \approx \mathbf{1,8 \times 10^{14}}$ estados.

- **Estado inicial.** $M_{ij} = 0$ para toda fila i y columna j .
- **Acciones.** R_{ij} : poner una reina en la fila i y columna j .
- **Modelo transicional.** Si $M_{ij} = 0$, entonces $(R_{ij}(M))_{ij} = 1$.
- **Test objetivo.** $\sum_{ij} M_{ij} = 8$ y ninguna reina está siendo atacada.
- **Costo de camino.** Costo unitario individual (es irrelevante).

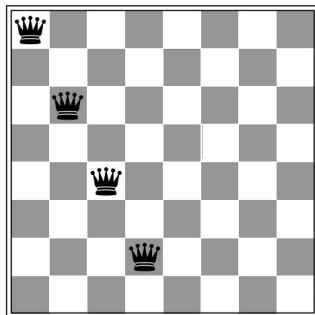
Formulación incremental (II)

— — —

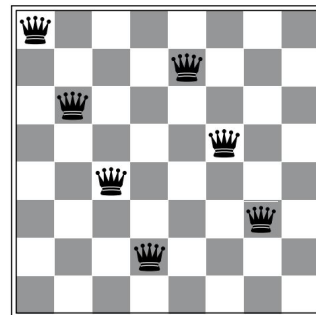
- Una mejor formulación debería prohibir poner reinas en casillas atacadas.
- Hay a lo sumo una reina por fila y por columna.
- **Estados.** Tuplas (f_1, f_2, \dots, f_n) con $0 \leq n \leq 8$, donde $f_i \in \{1, 2, \dots, 8\}$ es la fila donde se ubica la reina de la columna i , y tal que ninguna reina es atacada. Hay **2057** estados.

📖 Ejemplos.

$(1, 3, 5, 7) =$



$(1, 3, 5, 7, 2, 4, 6) =$



Formulación incremental (II)

- **Estado inicial.** $()$ tupla vacía.
- **Acciones.** R_i : poner una reina en la fila i de la columna vacía más a la izquierda.
- **Modelo transicional.**

$$\text{RESULTADO}((f_1, \dots, f_j), R_i) = (f_1, \dots, f_j, i).$$

- **Test objetivo.**

$$f_1, f_2, \dots, f_8 > 0.$$

- **Costo de camino.** Costo unitario individual (es irrelevante).

Formulación incremental (II)

- Para $n = 8$,
 - Formulación incremental (I): $1,8 \times 10^{14}$ estados.
 - Formulación incremental (II): 2057 estados.
- Para $n = 100$,
 - Formulación incremental (I): 10^{400} estados.
 - Formulación incremental (II): 10^{52} estados.

¿Cómo tratamos este tipo de problemas?

Búsqueda local

Los **algoritmos de búsqueda local** operan sobre un único estado actual y generalmente se mueven a un sucesor de dicho estado. Normalmente, **los caminos seguidos por la búsqueda no se guardan.**

Son útiles en problemas donde el camino al objetivo es irrelevante y lo importante es el objetivo en sí mismo.

A pesar de no ser sistemáticos, tienen dos ventajas.

1. Consumen muy poca memoria.
2. Encuentran soluciones *razonables* en espacios de búsqueda grandes o infinitos.

— — —



Problemas de optimización

— — —

- Un **problema de optimización** tiene una **función objetivo** h , donde, dado un estado s ,

$h(s)$: **valor objetivo** de s ,

y el objetivo es encontrar un estado s que **maximice** $h(s)$.

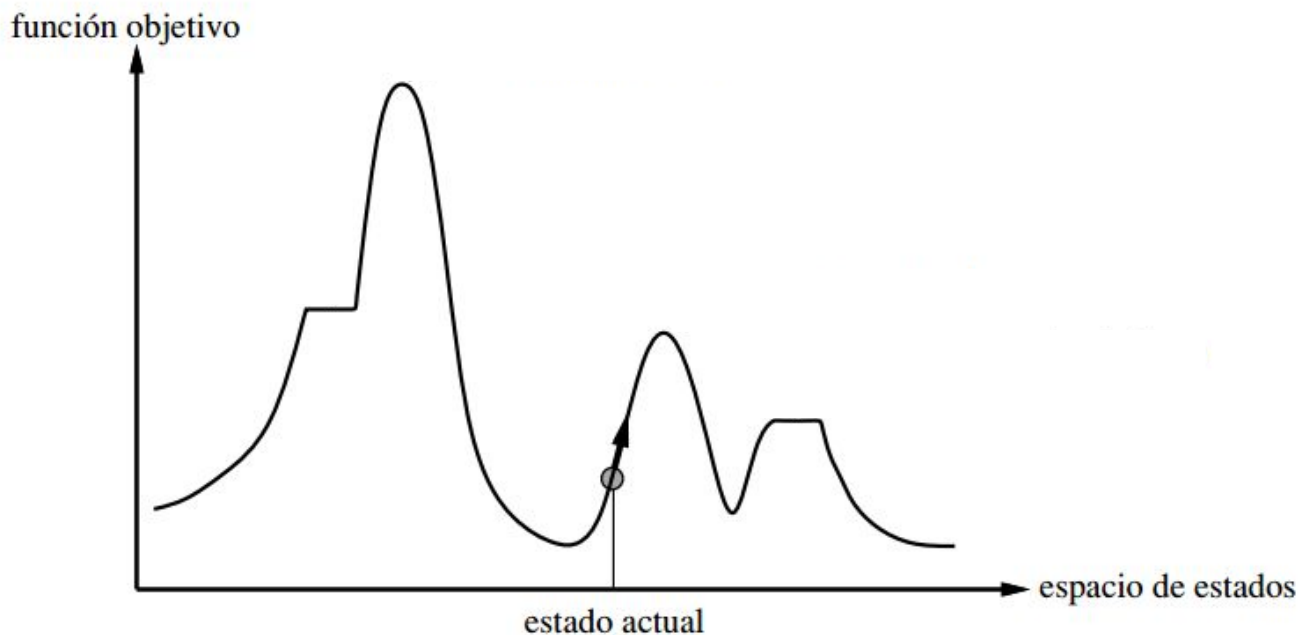
-  **Nota.** Minimizar h es equivalente a maximizar $-h$.
-  **Ejemplo.** El problema del viajante es un problema de optimización que minimiza la función objetivo:

h : distancia recorrida por el tour.

- Los algoritmos de búsqueda local también son muy útiles para resolver este tipo de problemas.

Paisaje del espacio de estados

Los algoritmos de búsqueda local exploran el **paisaje**:



Formulación de estado completo – 8 Reinas

— — —

Los algoritmos de búsqueda local usan típicamente una **formulación de estado completo**.

- **Estados.** Tuplas (f_1, f_2, \dots, f_8) donde $f_i \in \{1, 2, \dots, 8\}$ es la fila donde se ubica la reina de la columna i . **Todas las reinas están ubicadas.**

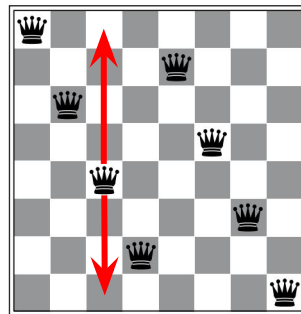
Hay $8^8 = 16.777.216$ estados.

- **Estado inicial.** Cualquier estado.

- **Acciones.**

R_{ij} : mover la reina de la columna j a la fila i de la misma columna.

Cada estado tiene $7 \times 8 = 56$ sucesores.



Formulación de estado completo – 8 Reinas

— — —

- **Modelo transicional.**

$\text{RESULTADO}((f_1, \dots, f_{j-1}, f_j, f_{j+1}, \dots, f_8), R_{ij}) = (f_1, \dots, f_{j-1}, \mathbf{i}, f_{j+1}, \dots, f_8).$

- **Función objetivo.**

h : número de pares de reinas que se atacan mutuamente.

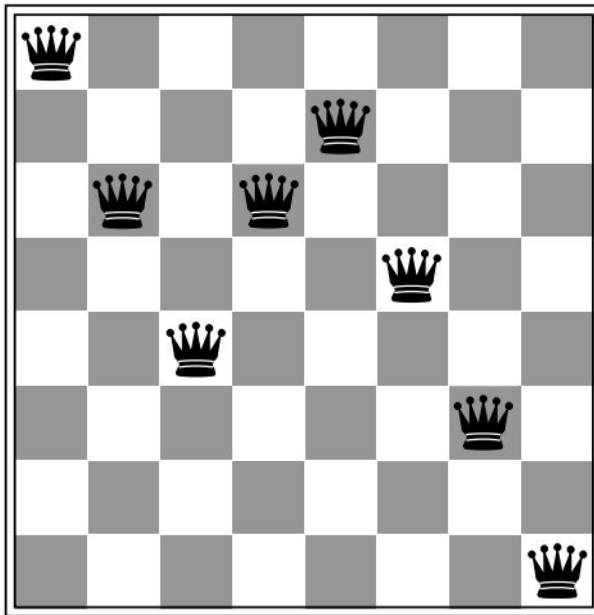


Ejercicio

— — —

Calcular el número de pares de reinas que se atacan mutuamente en el siguiente estado.

$S =$

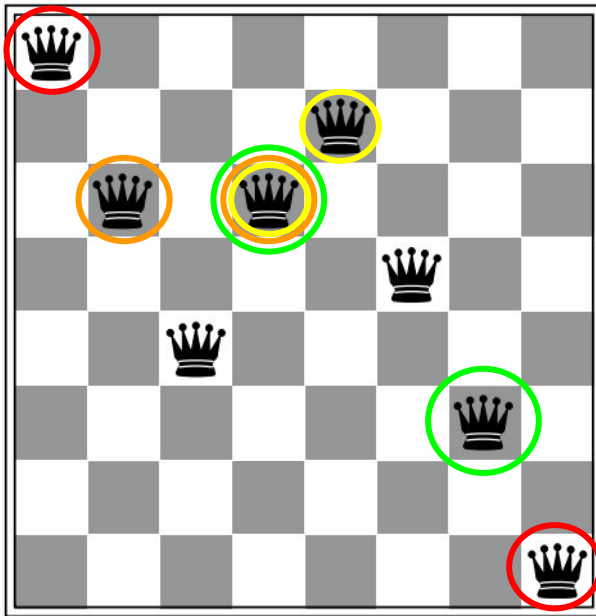




Respuesta

Calcular el número de pares de reinas que se atacan mutuamente en el siguiente estado.

$s =$



$$h(s) = 4$$

Búsqueda de ascensión de colinas

Hill-climbing search

Es el algoritmo de búsqueda local más simple. Es un bucle que en cada iteración se mueve a un sucesor con **mejor** valor objetivo.

En el paisaje del espacio de estados, siempre se mueve cuesta arriba por una colina.

Termina cuando no puede seguir avanzando, es decir, cuando ningún sucesor mejora el valor objetivo actual.

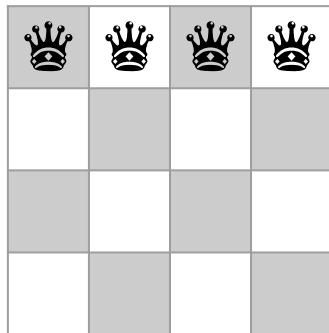
También es llamada **búsqueda local avara**, ya que no mira más allá de los sucesores inmediatos del estado actual.

— — —



Ejemplo – 4 Reinas

— — —

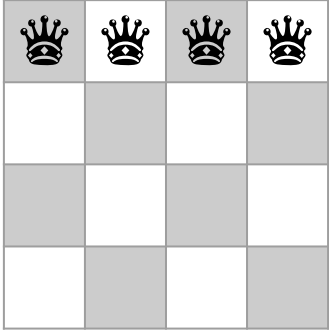


$h = 6$

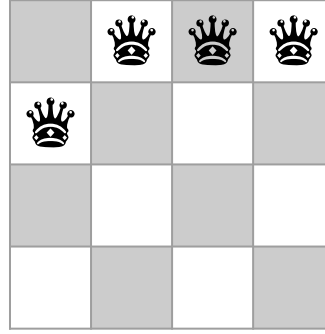
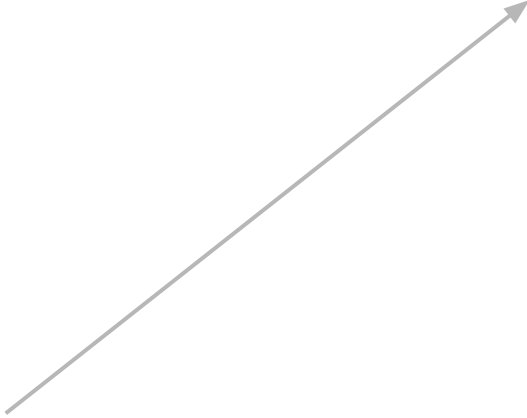


Ejemplo – 4 Reinas

— — —



$h = 6$

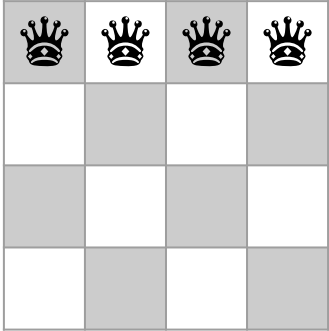


$h = 4$

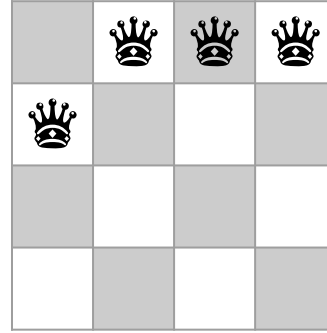
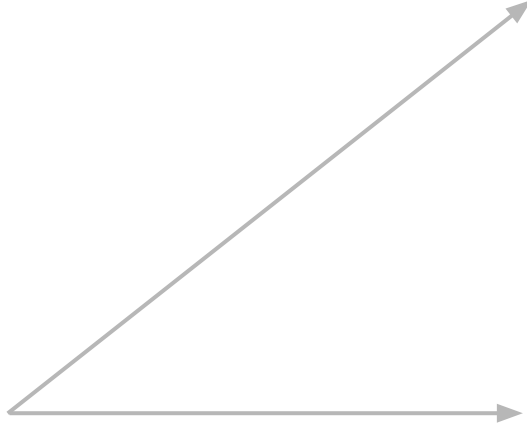


Ejemplo – 4 Reinas

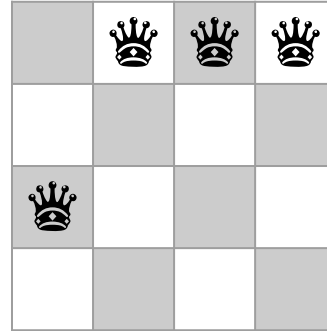
— — —



$h = 6$



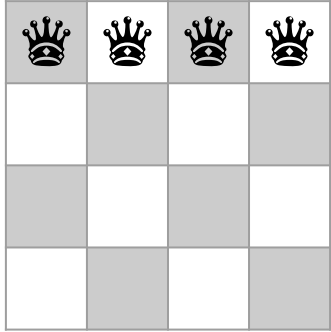
$h = 4$



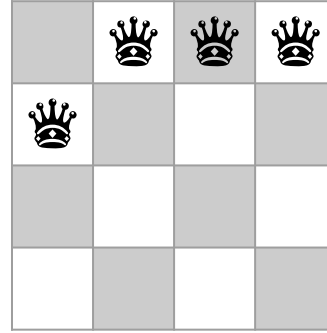
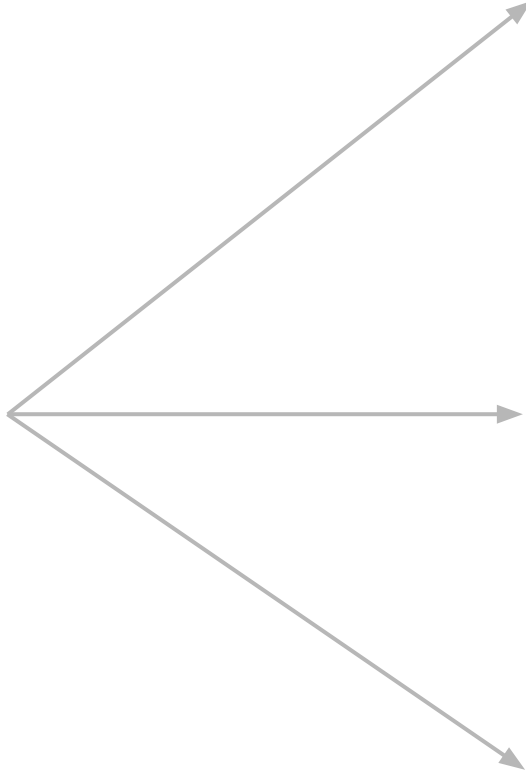
$h = 4$



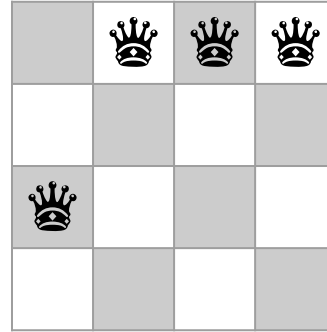
Ejemplo – 4 Reinas



$h = 6$



$h = 4$







$h = 4$

Continúa...



Ejemplo – 4 Reinas

— — —

			
4			
4			
...			

$h = 6$

El número de la fila i y columna j representa el valor objetivo del estado que se obtiene de aplicar la acción R_{ij} al estado actual.



Ejemplo – 4 Reinas

— — —

♔	♔	♔	♔
4	5	5	4
4	4	4	4
4	3	3	4


$$h = 6$$

El número de la fila i y columna j representa el valor objetivo del estado que se obtiene de aplicar la acción R_{ij} al estado actual.







Ejemplo – 4 Reinas

— — —

			
4	5	5	4
4	4	4	4
4	3	3	4

$h = 6$



$h = 3$



Ejemplo – 4 Reinas

— — —

♔	♔	♔	♔
4	5	5	4
4	4	4	4
4	3	3	4

$h = 6$



♔	6	♔	♔
1	5	2	3
3	4	2	1
3	♔	2	3

$h = 3$



Ejemplo – 4 Reinas

— — —

♔	♔	♔	♔
4	5	5	4
4	4	4	4
4	3	3	4

$h = 6$



♔	6	♔	♔
1	5	2	3
3	4	2	1
3	♔	2	3

$h = 3$



		♔	♔
♔			
	♔		

$h = 1$



Ejemplo – 4 Reinas

— — —

♔	♔	♔	♔
4	5	5	4
4	4	4	4
4	3	3	4

$h = 6$



♔	6	♔	♔
1	5	2	3
3	4	2	1
3	♔	2	3

$h = 3$



3	4	♔	♔
♔	3	2	3
3	3	1	0
3	♔	2	1

$h = 1$



Ejemplo – 4 Reinas

— — —

♔	♔	♔	♔
4	5	5	4
4	4	4	4
4	3	3	4

$h = 6$



♔	6	♔	♔
1	5	2	3
3	4	2	1
3	♔	2	3

$h = 3$



3	4	♔	♔
♔	3	2	3
3	3	1	0
3	♔	2	1

$h = 1$



		♔	
♔			
			♔
	♔		

$h = 0$



Ejemplo – 4 Reinas

— — —

4	5	5	4
4	4	4	4
4	3	3	4

$h = 6$



	6		
1	5	2	3
3	4	2	1
3		2	3

$h = 3$



3	4		
	3	2	3
3	3	1	0
3		2	1

$h = 1$



$h = 0$

Encontramos una solución en 3 pasos.

Algoritmo general de búsqueda de ascensión de colinas

— — —

```
1 function ASCENSIÓN-COLINAS(problema) return estado que es un máximo local
2   actual ← nodo con el estado inicial del problema
3   do
4     vecino ← nodo con el estado sucesor de actual
5               con mejor valor objetivo
6     if (vecino.valor ≤ actual.valor) then return actual.estado
7     actual ← vecino
```


Algoritmo general de búsqueda de ascensión de colinas

— — —
No se mantiene ninguna frontera, solo el nodo actual que es un objeto con un estado y un valor objetivo.

```
1 function ASCENSIÓN-COLINAS(problema) return estado que es un máximo local
2   actual ← nodo con el estado inicial del problema
3   do
4     vecino ← nodo con el estado sucesor de actual
5               con mejor valor objetivo
6     if (vecino.valor ≤ actual.valor) then return actual.estado
7     actual ← vecino
```

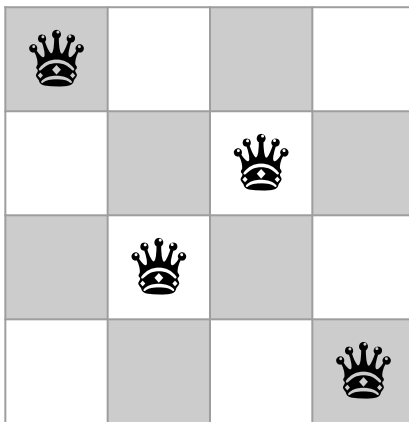
En caso de empate, se elige de forma aleatoria.



Ejercicio

— — —

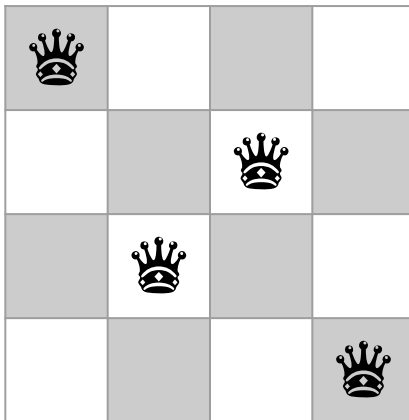
Resolver el problema de las **4 Reinas** usando la búsqueda de ascensión de colinas desde el siguiente estado inicial:





Respuesta

— — —



$$h = 2$$



Respuesta

— — —

👑	3	2	4
3	4	👑	2
2	👑	4	3
4	2	3	👑

$$h = 2$$



Respuesta

👑	3	2	4
3	4	👑	2
2	👑	4	3
4	2	3	👑

$$h = 2$$

La búsqueda de ascensión de colinas quedó **atascada**.

No hay ningún estado sucesor con valor objetivo mayor que el del estado actual.

Atascos

— — —

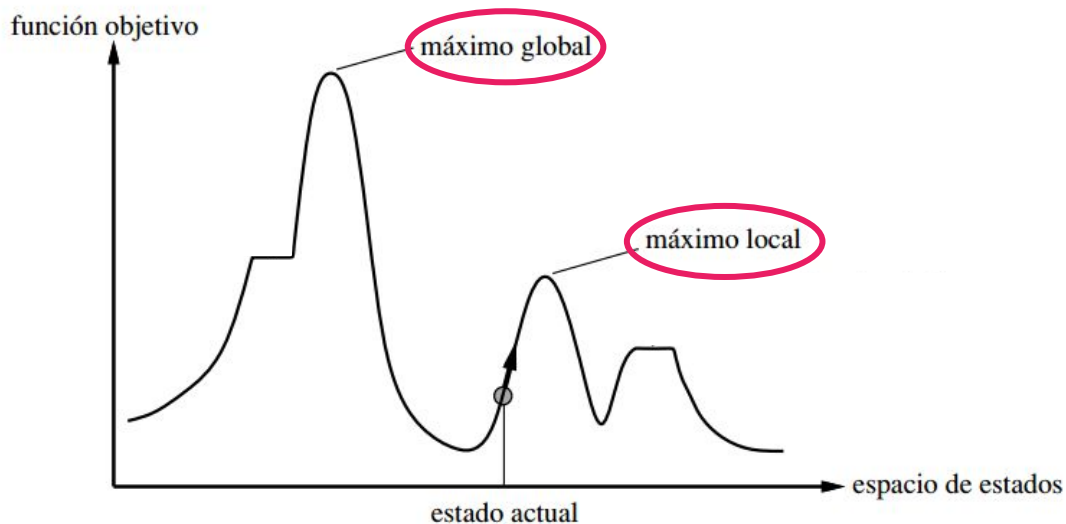
La ascensión de colinas a menudo hace un rápido progreso hacia buenas soluciones...

Pero se puede **atascar** en un estado que no es óptimo por los siguientes motivos.

1. **Máximos locales.**
2. **Crestas.**
3. **Mesetas.**

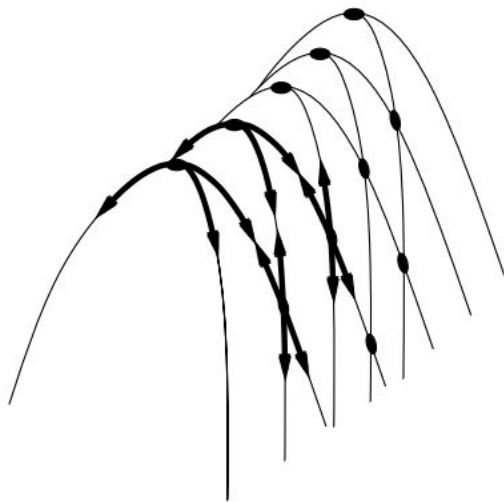
1. Máximos locales

Un **máximo local** es un pico que es más alto que cada uno de sus estados sucesores pero más bajo que el máximo global.



2. Crestas

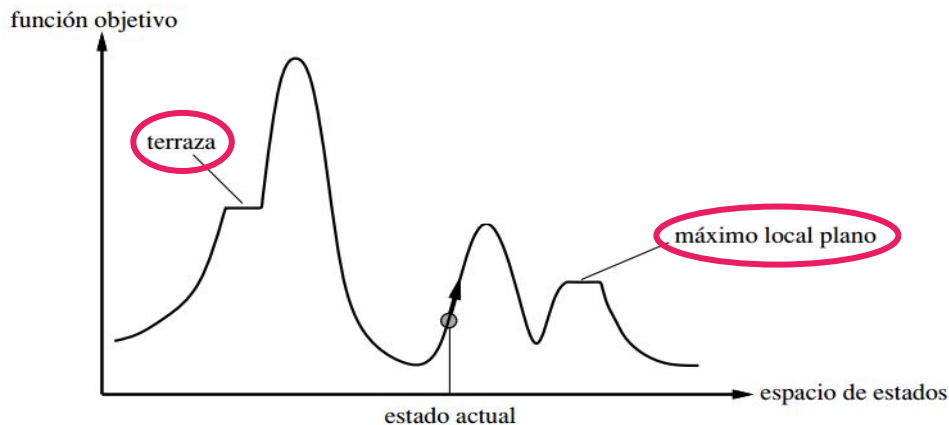
Una **cresta** es una secuencia de máximos locales que no están directamente conectados el uno con el otro. De cada máximo local, todas las acciones disponibles apuntan hacia abajo.



3. Mesetas

Una **meseta** es un área del paisaje del espacio de estados donde la elevación es plana. Es decir, donde el mejor estado sucesor tiene el mismo valor objetivo que el estado actual.

Si no existe una salida ascendente, es un **máximo local plano**, de lo contrario es una **terrazza**.





Ejemplo – 8 Reinas

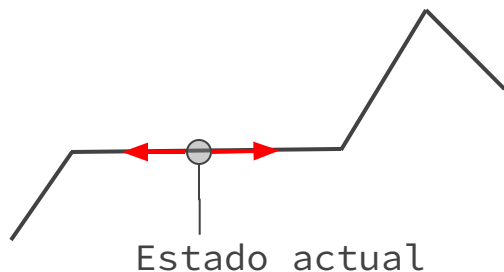
— — —

- La formulación de estado completo tiene casi **17 millones** de estados.
- Empezando en un estado aleatorio, la búsqueda de ascensión de colinas:
 - se atasca en un máximo local **86%** de las veces, haciendo en promedio **3** pasos.
 - alcanza un máximo global **14%** de las veces, haciendo en promedio **4** pasos.
- **¡No está nada mal!** Pero podemos hacerlo mejor.

Movimiento lateral

— — —

- La ascensión de colinas termina al encontrar una meseta.
- Tiene sentido si estamos en un máximo local plano. Pero, ¿si estamos en una terraza?
- Un **movimiento lateral** consiste en moverse a un estado sucesor con el mismo valor objetivo que el estado actual.

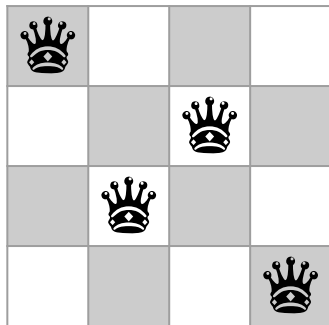


- Para evitar caminos cíclicos, es usual **limitar el número de movimientos laterales consecutivos**.



Ejemplo – 4 Reinas

— — —







$h = 2$



Ejemplo – 4 Reinas

— — —

	3	2	4
3	4		2
2		4	3
4	2	3	

$h = 2$



Ejemplo – 4 Reinas

— — —

	3	2	4
3	4		2
2		4	3
4	2	3	

$h = 2$

Paso
lateral



$h = 2$



Ejemplo – 4 Reinas

— — —

👑	3	2	4
3	4	👑	2
2	👑	4	3
4	2	3	👑

$h = 2$

Paso
lateral



2	1	2	4
3	2	👑	3
👑	👑	4	4
4	2	3	👑

$h = 2$



Ejemplo – 4 Reinas

♔	3	2	4
3	4	♔	2
2	♔	4	3
4	2	3	♔

$h = 2$

Paso
lateral

2	1	2	4
3	2	♔	3
♔	♔	4	4
4	2	3	♔

$h = 2$

	♔		
		♔	
♔			
			♔

$h = 1$



Ejemplo – 4 Reinas

♔	3	2	4
3	4	♔	2
2	♔	4	3
4	2	3	♔

$h = 2$

Paso
lateral

2	1	2	4
3	2	♔	3
♔	♔	4	4
4	2	3	♔

$h = 2$

3	♔	2	3
3	2	♔	2
♔	2	2	4
3	2	1	♔

$h = 1$



Ejemplo – 4 Reinas

♔	3	2	4
3	4	♔	2
2	♔	4	3
4	2	3	♔

$h = 2$

Paso
lateral

2	1	2	4
3	2	♔	3
♔	♔	4	4
4	2	3	♔

$h = 2$

3	♔	2	3
3	2	♔	2
♔	2	2	4
3	2	1	♔

$h = 1$

	♔		
♔			
		♔	♔

$h = 1$

Paso
lateral



Ejemplo – 4 Reinas

♔	3	2	4
3	4	♔	2
2	♔	4	3
4	2	3	♔

$h = 2$

Paso
lateral

2	1	2	4
3	2	♔	3
♔	♔	4	4
4	2	3	♔

$h = 2$

3	♔	2	3
3	2	♔	2
♔	2	2	4
3	2	1	♔

$h = 1$

3	♔	2	1
3	3	1	0
♔	3	2	1
3	4	♔	♔

$h = 1$

Paso
lateral



Ejemplo – 4 Reinas

♔	3	2	4
3	4	♔	2
2	♔	4	3
4	2	3	♔

$h = 2$

Paso
lateral

2	1	2	4
3	2	♔	3
♔	♔	4	4
4	2	3	♔

$h = 2$

3	♔	2	3
3	2	♔	2
♔	2	2	4
3	2	1	♔

$h = 1$

	♔		
			♔
♔			
		♔	

$h = 0$

3	♔	2	1
3	3	1	0
♔	3	2	1
3	4	♔	♔

$h = 1$

Paso
lateral



Ejemplo – 8 Reinas

— — —

Sin movimientos laterales:

- Se atasca en un máximo local **86%** de las veces, haciendo en promedio **3** pasos.
- Se alcanza un máximo global **14%** de las veces, haciendo en promedio **4** pasos.

Limitando a 100 el número de movimientos laterales consecutivos:

- Se atasca en un máximo local **6%** de las veces, haciendo en promedio **64** pasos.
- Se alcanza un máximo global **94%** de las veces, haciendo en promedio **21** pasos.

Otras estrategias de ascensión de colinas

— — —

La ascensión de colinas clásica siempre elige el estado sucesor con el mejor valor objetivo. Otras estrategias:

1. **Ascensión de colinas estocástica.** Elige aleatoriamente entre todos los estados sucesores con un valor objetivo más alto que el del estado actual. También puede asignar probabilidades diferenciadas según la diferencia en el valor objetivo.

Su convergencia suele ser más lenta, pero a veces encuentra mejores soluciones.

2. **Ascensión de colinas de primera opción.** Los estados sucesores se generan de a uno y se elige el primer estado sucesor cuyo valor objetivo sea más alto que el del estado actual.


Es útil cuando los estados tienen muchos sucesores.

Complejidad

- Los algoritmos de ascensión de colinas descritos hasta ahora son **incompletos** 👎.
- Pueden no encontrar un objetivo cuando se atascan en un máximo local.

Reinicio aleatorio

— — —

- Si la ascensión de colinas no tiene éxito, conviene intentar de vuelta desde otro estado inicial.
- **Ascensión de colinas de reinicio aleatorio.** Realiza una serie de búsquedas de ascensión de colinas desde un estado inicial generado aleatoriamente, hasta encontrar un estado objetivo.
- **Complejidad.**  (con una cierta probabilidad).

Si cada búsqueda de ascensión de colinas tiene probabilidad p de éxito, entonces se esperan $1/p$ reinicios para encontrar un estado objetivo.



Ejemplo – 8 Reinas

— — —

- Sin movimientos laterales, $p \approx 0.14$, se esperan $1/0.14 \approx \mathbf{7.14}$ reinicios.
- Con movimientos laterales, $p \approx 0.94$, se esperan $1/0.94 \approx \mathbf{1.06}$ reinicios.

El reinicio aleatorio permite resolver problemas con hasta 3M de reinas en menos de un minuto.



Resumen

- ❑ Una búsqueda local actúa evaluando y modificando un estado actual (o más de uno), en lugar de explorar sistemáticamente caminos desde un estado inicial.
- ❑ Se caracterizan por su bajo consumo de memoria.
- ❑ Suelen aplicarse en problemas donde lo único que importa es el estado de una solución, no el camino para alcanzarlo, y en problemas de optimización.
- ❑ Normalmente, requieren de una formulación de estado completo del problema y de una función objetivo que alcance su máximo valor en estados objetivos.



Resumen

- ❑ La búsqueda local más simple es el algoritmo de ascensión de colinas, el cual siempre se mueve a un estado sucesor que mejore el valor objetivo actual.
- ❑ El éxito de la ascensión de colinas depende fuertemente del paisaje de espacio de estados. En general, puede quedar atascado en un máximo local que no es global.
- ❑ Los movimientos laterales pueden ayudar a desatascar la búsqueda.
- ❑ El reinicio aleatorio agrega completitud con una cierta probabilidad.



Próximamente

— — —

La ascensión de colinas no puede escapar de un máximo local si:

1. Es un pico.
2. Es un máximo local plano.
3. Es una terraza y los movimientos laterales generan un camino cíclico o el número de movimientos laterales no es suficiente para encontrar un camino ascendente.

¿Cómo escapar en estas condiciones?

Búsqueda Tabú

Tabu Search

Incorpora al algoritmo de ascensión de colinas dos elementos.

1. Un mecanismo para escapar de un máximo local.

Cuando la búsqueda se atasca, permite moverse a un estado sucesor con **peor** valor objetivo.

2. Memoria de corto plazo para evitar caminos cíclicos.

Normalmente, se mantiene una **lista tabú** que prohíbe ciertas acciones que podrían revertir acciones realizadas recientemente.

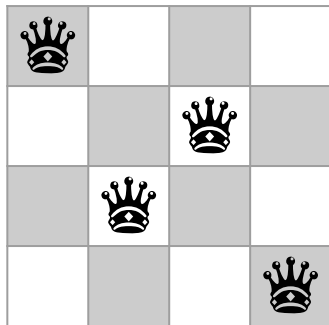
— — —



Ejemplo – 4 Reinas

Lista tabú = { }

— — —



$h = 2$



Ejemplo – 4 Reinas

Lista tabú = { }

— — —

	3	2	4
3	4		2
2		4	3
4	2	3	

$h = 2$



Ejemplo – 4 Reinas

Lista tabú = { }

	3	2	4
3	4		2
2		4	3
4	2	3	

$h = 2$

Empeora
el valor
objetivo

$h = 3$







Ejemplo – 4 Reinas

Lista tabú = $\{R_{3\ 2}\}$

	3	2	4
3	4		2
2		4	3
4	2	3	

$h = 2$

Empeora
el valor
objetivo

$h = 3$

Revertir esta acción, es decir, mover de vuelta a la reina de la columna 2 a la fila 3, podría generar un camino cíclico.

Agregamos la acción $R_{3\ 2}$ a la lista tabú.



Ejemplo – 4 Reinas

Lista tabú = $\{R_{32}\}$

	3	2	4
3	4		2
2		4	3
4	2	3	

$h = 2$

Empeora
el valor
objetivo

		4	5
3	4		3
1	-	4	4
3	2	3	

$h = 3$



Ejemplo – 4 Reinas

Lista tabú = $\{R_{3,2}, R_{1,1}\}$

	3	2	4
3	4		2
2		4	3
4	2	3	

$h = 2$

Empeora
el valor
objetivo

		4	5
3	4		3
1	-	4	4
3	2	3	

$h = 3$

$h = 1$



Ejemplo – 4 Reinas

Lista tabú = $\{R_{32}, R_{11}\}$

♔	3	2	4
3	4	♔	2
2	♔	4	3
4	2	3	♔

$h = 2$

Empeora
el valor
objetivo

♔	♔	4	5
3	4	♔	3
1	-	4	4
3	2	3	♔

$h = 3$

-	♔	2	3
3	3	♔	2
♔	-	2	3
3	2	1	♔

$h = 1$



Ejemplo – 4 Reinas

Lista tabú = $\{R_{3,2}, R_{1,1}, \mathbf{R_{4,4}}\}$

	3	2	4
3	4		2
2		4	3
4	2	3	

$h = 2$

Empeora
el valor
objetivo

		4	5
3	4		3
1	-	4	4
3	2	3	

$h = 3$

-		2	3
3	3		2
	-	2	3
3	2	1	

$h = 1$

$h = 2$

Empeora
el valor
objetivo



Ejemplo – 4 Reinas

Lista tabú = $\{R_{32}, R_{11}, R_{44}\}$

	3	2	4
3	4		2
2		4	3
4	2	3	

$h = 2$

Empeora
el valor
objetivo

		4	5
3	4		3
1	-	4	4
3	2	3	

$h = 3$

-		2	3
3	3		2
	-	2	3
3	2	1	

$h = 1$

-		3	3
5	4		
	-	1	3
3	3	0	-

$h = 2$

Empeora
el valor
objetivo



Ejemplo – 4 Reinas

Lista tabú = $\{R_{32}, R_{11}, R_{44}, \mathbf{R_{23}}\}$

	3	2	4
3	4		2
2		4	3
4	2	3	

$h = 2$

Empeora
el valor
objetivo

		4	5
3	4		3
1	-	4	4
3	2	3	

$h = 3$

-		2	3
3	3		2
	-	2	3
3	2	1	

$h = 1$

Empeora
el valor
objetivo

-		3	3
5	4		
	-	1	3
3	3	0	-

$h = 2$

$h = 0$

Algoritmo general de búsqueda tabú

```
1 function TABÚ(problema) return estado
2   actual ← nodo con el estado inicial del problema
3   mejor ← actual
4   tabu ← inicialmente vacía
5   while no se cumpla el criterio de parada
6       vecino ← nodo con el estado sucesor de actual con mejor valor
           objetivo que no sea un estado tabú
7       if (mejor.valor < vecino.valor) then mejor ← vecino
8       actualizar la lista tabú
9       actual ← vecino
10  return mejor.estado
```


Algoritmo general de búsqueda tabú

```
1 function TABÚ(problema) return estado
2   actual ← nodo con el estado inicial del problema
3   mejor ← actual
4   tabu ← inicialmente vacía
5   while no se cumpla el criterio de parada
6     vecino ← nodo con el estado sucesor de actual con mejor valor
        objetivo que no sea un estado tabú
7     if (mejor.valor < vecino.valor) then mejor ← vecino
8     actualizar la lista tabú
9     actual ← vecino
10  return mejor.estado
```

Se mantiene un nodo con el estado con mejor valor objetivo encontrado hasta el momento y se lo retorna tras cumplirse el criterio de parada.

Algoritmo general de búsqueda tabú

Se incorpora una lista tabú y se evitan estados generados con acciones que sean tabú.

```
1 function TABÚ(problema) return estado
2   actual ← nodo con el estado inicial del problema
3   mejor ← actual
4   tabu ← inicialmente vacía
5   while no se cumpla el criterio de parada
6     vecino ← nodo con el estado sucesor de actual con mejor valor
7     objetivo que no sea un estado tabú
8     if (mejor.valor < vecino.valor) then mejor ← vecino
9     actualizar la lista tabú
10    actual ← vecino
11  return mejor.estado
```

Se actualiza el estado haya o no mejorado el valor objetivo

Algoritmo general de búsqueda tabú

¿Qué criterio de parada elegir?

```
1 function TABÚ(problema) return estado
2   actual ← nodo con el estado inicial del problema
3   mejor ← actual
4   tabu ← inicialmente vacía
5   while no se cumpla el criterio de parada
6       vecino ← nodo con el estado sucesor de actual con mejor valor
           objetivo que no sea un estado tabú
7       if (mejor.valor < vecino.valor) then mejor ← vecino
8       actualizar la lista tabú
9       actual ← vecino
10  return mejor.estado
```

Tabúes

— — —

¿Qué elementos se pueden almacenar en la lista tabú?

1. **Acciones.**
2. **Estados.**
3. **Proposiciones.**

A continuación veremos con más detalle cada una de estas alternativas.

1. Acciones

Acciones que podrían revertir acciones realizadas recientemente.



Ejemplo - 8 Reinas.

Suponer que inicialmente la reina de la columna j está en la fila i_1 y se la mueve a la fila i_2 (acción R_{i_2j}).

1. Como ya vimos, una posibilidad es agregar a la lista tabú la acción R_{i_1j} .
2. Otra posibilidad más restrictiva es prohibir mover esa reina por completo. Es decir, agregar a la lista tabú las acciones R_{1j}, \dots, R_{8j} .

2. Estados

Estados que hayan sido visitados recientemente.

En general, consume más memoria. La representación de un estado suele ocupar más memoria que la de una acción.

Además, para representaciones complejas puede no ser sencillo determinar si dos estados son iguales.



Ejemplo - 8 Reinas.

Un estado es una tupla de 8 elementos, mientras que una acción es un par de elementos.

3. Proposiciones

Proposiciones sobre atributos de los estados o de las acciones.

Usualmente las proposiciones de la lista tabú se dicen que son **tabú activas**.

Los estados sucesores que verifiquen proposiciones tabú activas son ignorados.



Ejemplo - 8 Reinas.

Las reinas de las columnas j_1 y j_2 se atacan mutuamente.

Memoria de corto plazo

La lista tabú funciona como una **memoria de corto plazo**, que almacena únicamente la información reciente y no toda la historia del algoritmo.

Para esto, existen dos enfoques.

1. Limitar el tamaño de la lista tabú.
2. Quitar los elementos de la lista que hayan estado por más de un cierto número de iteraciones.

A este valor se lo conoce como **tenor de tabú**.

Criterio de parada

— — —

La búsqueda tabú puede terminar si se alcanza un estado objetivo.

De lo contrario, algunos criterios comunes son:

1. Tras cierto número de **iteraciones totales** (o tiempo de CPU).
2. Tras cierto número de **iteraciones sin mejoras** en el mejor valor objetivo.
3. Cuando la función objetivo sobrepasa cierto **valor umbral**.

El segundo criterio es el más habitual.

Seteo de parámetros

— — —

¿Qué valor elegir para los parámetros?

- Número de iteraciones totales.
- Tamaño de la lista tabú.
- Tenor de tabú.
- ...

No es obvio. **Se determinan experimentalmente.**

Los algoritmos de búsqueda tabú suelen ser extremadamente sensibles al seteo de los parámetros.

No es raro que la performance mejore/empeore drásticamente al cambiar el valor de algunos parámetros.

Componentes adicionales

En problemas difíciles se suelen incorporar a la búsqueda tabú algunos componentes adicionales para mejorar su efectividad.

Entre los más comunes se encuentran:

1. **Criterio de aspiración.**
2. **Diversificación.**

1. Criterio de aspiración

Los tabúes pueden volverse demasiado restrictivos:

- Pueden prohibir acciones atractivas, incluso cuando no hay peligro de ciclar, o
- Pueden conducir a un estancamiento en la búsqueda.

Un **criterio de aspiración** permite ignorar tabúes.

Lo más sencillo es permitir un movimiento, incluso si es tabú, siempre que el valor objetivo del estado sucesor sea mayor al del mejor estado encontrado hasta el momento (es claro que el nuevo estado no había sido visitado anteriormente).

2. Diversificación

Los algoritmos de búsqueda local, en particular la búsqueda tabú, tienden a pasar la mayor parte del tiempo en una porción restringida del espacio de estados, ignorando quizás regiones más interesantes.

La **diversificación** es un mecanismo que fuerza la búsqueda en regiones inexploradas. Típicamente usa **memoria de largo plazo**.

Lo más sencillo es la **memoria de frecuencia**, que registra por cuántas iteraciones se preservaron algunos componentes de la solución.



Ejemplo - 8 Reinas.

Memoria de frecuencia. Para toda fila i y columna j , número de iteraciones en donde la reina de la columna j estuvo en la fila i .

2. Diversificación

— — —

A partir de esta información, existen dos estrategias de diversificación.

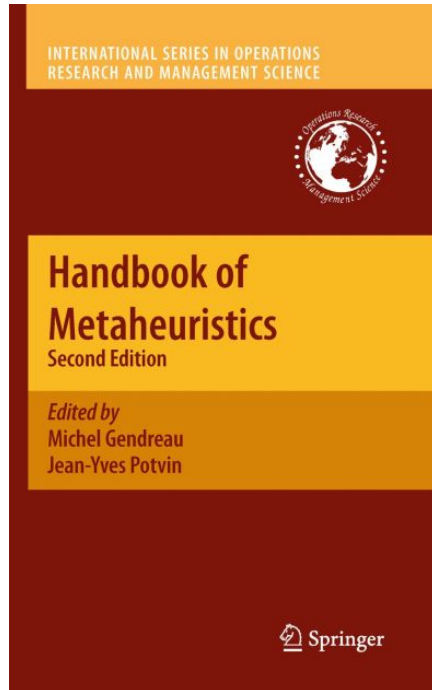
1. **De reinicio**. Reiniciar la búsqueda desde un estado con componentes poco frecuentes.
2. **Continua**. A la hora de elegir un estado sucesor, priorizar aquellos con componentes poco frecuentes (agregar un término de frecuencia a la función objetivo).

Ejemplo - 8 Reinas.

Diversificación de reinicio. Tras cierto número de iteraciones, reiniciar la búsqueda tabú desde el estado que tiene a cada reina en su fila menos frecuente.

Bibliografía

— — —



Capítulo 2. Tabu Search

Michel Gendreau and Jean-Yves Potvin

Búsqueda local en espacios continuos

En el mundo real, las magnitudes (longitud, masa, tiempo, etc.) suelen asumir valores continuos.

Ninguno de los algoritmos de búsqueda local vistos hasta ahora son aplicables a problemas con espacios de estados continuos.

En estos casos, los estados tienen **infinitos sucesores**.

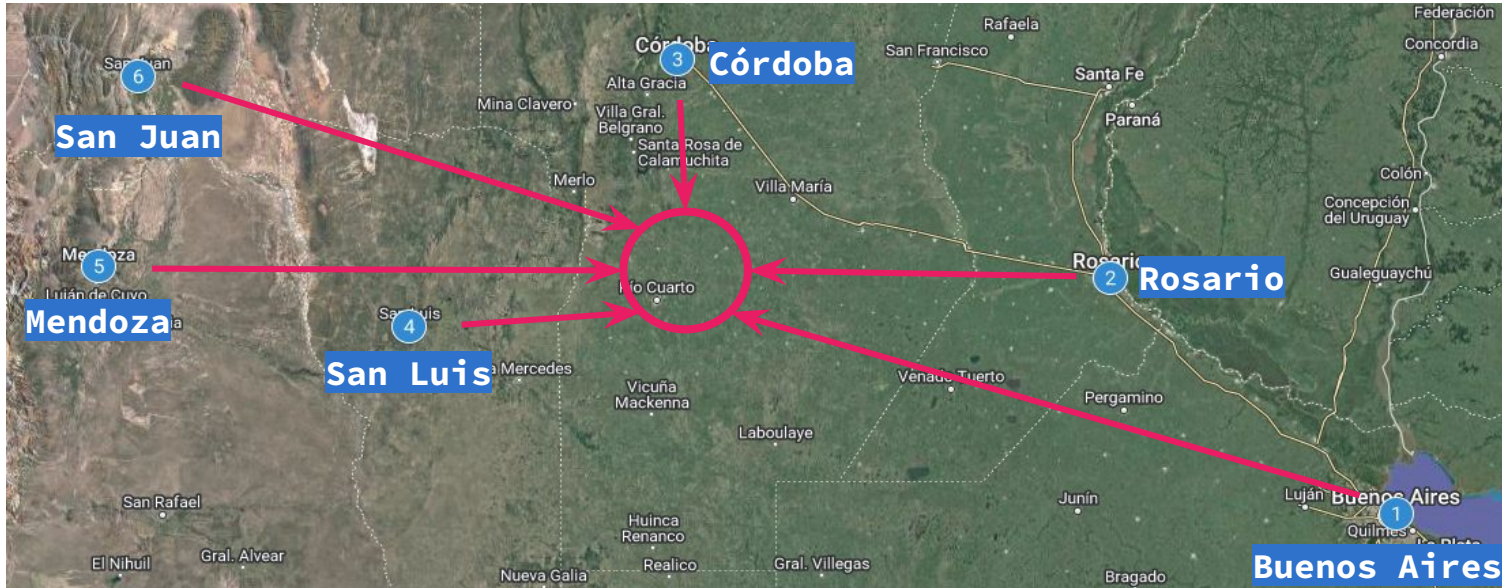
¿Cómo buscamos soluciones en espacios continuos?

— — —



Problema de colocación de aeropuerto

Objetivo. Ubicar un aeropuerto, de forma tal que la suma de las distancias al cuadrado de cada ciudad del mapa al aeropuerto sea mínima.



Formulación

— — —

- **Estados.** Coordenadas (x,y) con la ubicación del aeropuerto, donde x e y son números reales.
- **Estado inicial.** Cualquier estado.
- **Acciones.** Vectores (i,j) en \mathbb{R}^2 que mueven el aeropuerto.
- **Modelo transicional.** $\text{RESULTADO}((x,y),(i,j)) = (x+i,y+j)$
- **Función objetivo.**

$$f(x,y) = - \sum_{c \in C} ((x - x_c)^2 + (y - y_c)^2)$$

Donde C es el conjunto de ciudades y las constantes (x_c, y_c) son las coordenadas de la ciudad c .



Ejemplo – Aeropuerto

Suponiendo que las ciudades tienen las siguientes coordenadas.



La función objetivo se simplifica a:

$$f(x, y) = -6x^2 + 12x - 6y^2 + 6y - 13$$

Discretizar vs. no discretizar

— — —

Veremos dos técnicas para resolver problemas de este tipo.

1. **Discretizar** el espacio de estados.
2. **No discretizar** el espacio de estados.

1. Discretizar

Discretizar consiste en restringir el problema de forma tal que, para todo estado S , el conjunto

$ACCIONES(S)$

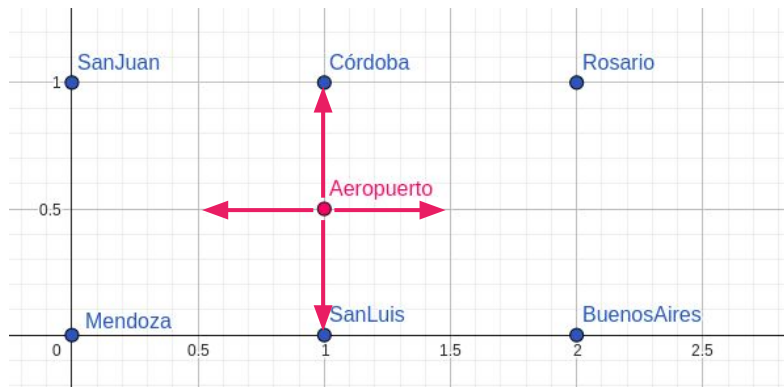
sea **finito**.

Podemos aplicar entonces cualquiera de los algoritmos de búsqueda local descritos anteriormente.



Ejemplo – Aeropuerto

El aeropuerto se puede mover únicamente vertical u horizontalmente en una cantidad fija $\pm\delta$.



$\delta=0.5$

Es decir,

$$\text{ACCIONES}(x,y) = \{(x+\delta,y), (x-\delta,y), (x,y+\delta), (x,y-\delta)\}.$$

2. No discretizar

Buscar máximos locales en espacios de estados continuos requiere repasar algunos conceptos de **cálculo**.

La literatura sobre el tema es enorme, veremos una muy breve introducción.

Gradiente

— — —

- Sea un campo escalar

$$f : \mathbb{R}^n \rightarrow \mathbb{R}.$$

- El **gradiente** de f , notado $\nabla(f)$, es el campo vectorial

$$\nabla(f)(x_1, \dots, x_n) = \left(\frac{\partial f}{\partial x_1}(x_1, \dots, x_n), \dots, \frac{\partial f}{\partial x_n}(x_1, \dots, x_n) \right).$$

donde $\frac{\partial f}{\partial x_i}$ es la **derivada parcial** de f respecto a x_i .



Ejemplo – Aeropuerto

— — —

El gradiente de la función objetivo:

$$f(x, y) = -6x^2 + 12x - 6y^2 + 6y - 13$$

es el campo vectorial:

$$\nabla f(x, y) = (-12x + 12, -12y + 6)$$

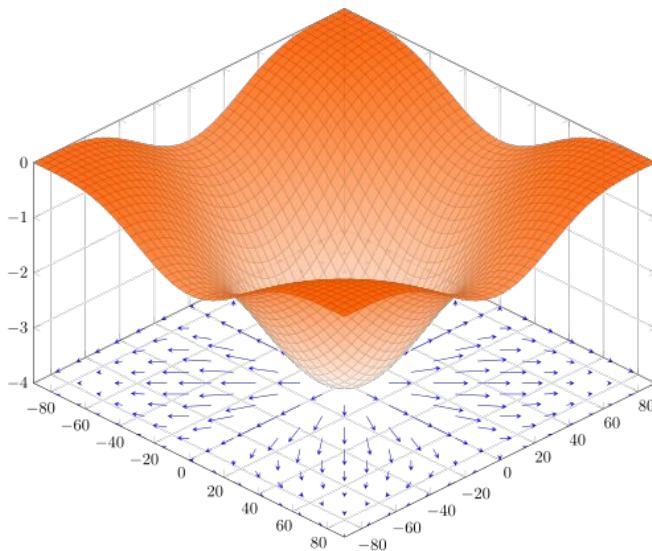
Interpretación geométrica

— — —

Dado un punto $(a_1, \dots, a_n) \in \mathbb{R}^n$, el vector $\nabla(f)(a_1, \dots, a_n)$ muestra la dirección en la que hay que moverse estando en el punto (a_1, \dots, a_n) para aumentar el valor de f lo más rápido posible y su módulo indica cuán rápido aumenta.

Ejemplo.

$$f(x) = -(\cos^2(x) + \cos^2(y))^2$$



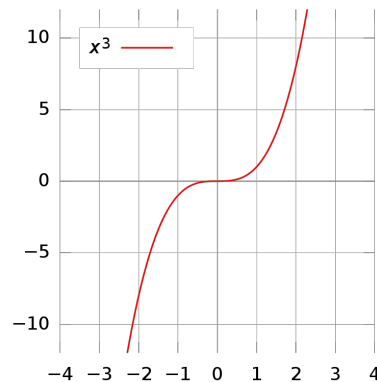
Puntos estacionarios

— — —

- Dada una función diferenciable (derivable en cualquier dirección), un **punto estacionario** es un punto del dominio donde todas las derivadas parciales son 0. Es decir,

$$\nabla f(x_1, \dots, x_n) = (0, \dots, 0)$$

- Un punto estacionario puede ser:
 - un **máximo local**,
 - un **mínimo local**,
 - o un **punto de silla**.
- Algunas veces, es posible calcularlos analíticamente.





Ejemplo – Aeropuerto

Los puntos estacionarios de la función objetivo

$$f(x, y) = -6x^2 + 12x - 6y^2 + 6y - 13$$

se obtienen resolviendo el sistema de ecuaciones

$$\nabla f(x, y) = (0, 0)$$

es decir,

$$\begin{cases} -12x + 12 = 0 \\ -12y + 6 = 0 \end{cases}$$

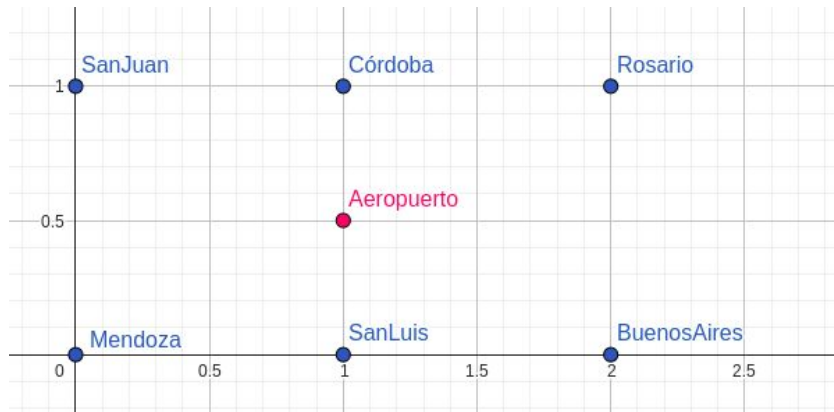


Ejemplo – Aeropuerto

La solución del sistema

$$\begin{cases} -12x + 12 = 0 \\ -12y + 6 = 0 \end{cases}$$

es el punto $p = (1, 0.5)$.



Claramente, p es un máximo global de f porque es el centro geométrico del rectángulo cuyos vértices están en Mendoza, San Juan, Rosario y Buenos Aires.

Existen criterios para clasificar un punto estacionario. Por ejemplo, el criterio de la **derivada segunda**.

Aproximación numérica

- Desafortunadamente, no siempre es posible hallar los puntos estacionarios de forma analítica.
 - Por ejemplo, la ecuación $\nabla f(x_1, \dots, x_n) = (0, \dots, 0)$ podría ser difícil de despejar, tener muchas variables, infinitas soluciones, etc.
- Los máximos y mínimos locales se pueden aproximar **numéricamente**.

Ascensión de gradiente

El algoritmo de **ascensión de gradiente**:

1. Parte de un punto x del dominio.
2. En cada iteración se mueve al punto

$$x \leftarrow x + \alpha \cdot \nabla(f)(x),$$

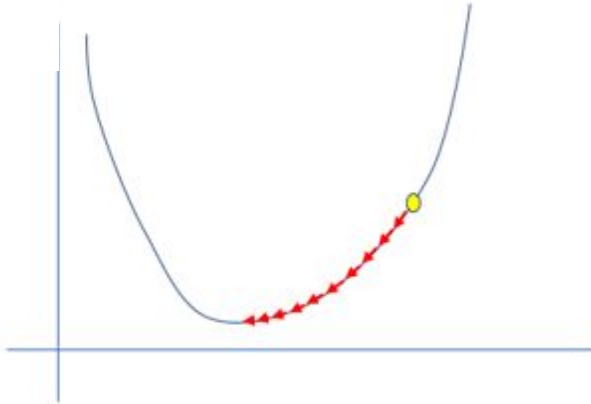
donde α es una constante pequeña, llamada **tamaño de paso**.

Si se busca un mínimo local, se mueve al punto

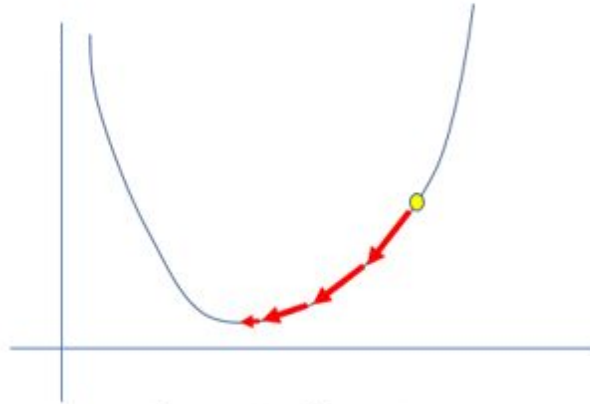
$$x \leftarrow x - \alpha \cdot \nabla(f)(x).$$



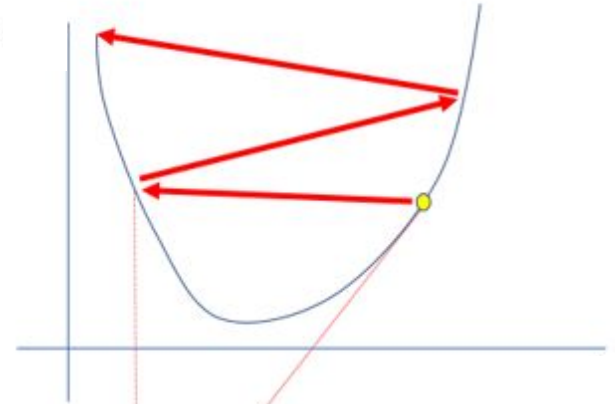
Ejemplos



α muy chico,
requiere muchas
iteraciones



α adecuado,
requiere menos
iteraciones.



α muy grande,
puede escapar del
mínimo local.

Funciones no diferenciables

— — —

- Las funciones no diferenciables son más difíciles de tratar.
- No es posible obtener una fórmula cerrada para $\nabla(f)$.
- Si el gradiente se puede calcular **localmente**, entonces todavía es posible aplicar la ascensión de gradiente.



Ejemplo – Colocar dos aeropuertos

El objetivo es ubicar 2 aeropuertos de forma tal que la suma de las distancias al cuadrado de cada ciudad del mapa al aeropuerto más cercano sea mínima.





Ejemplo – Colocar dos aeropuertos

— — —

La función objetivo ahora es:

$$f(x_1, y_1, x_2, y_2) = - \sum_{c \in C} \min((x_1 - x_c)^2 + (y_1 - y_c)^2, (x_2 - x_c)^2 + (y_2 - y_c)^2)$$

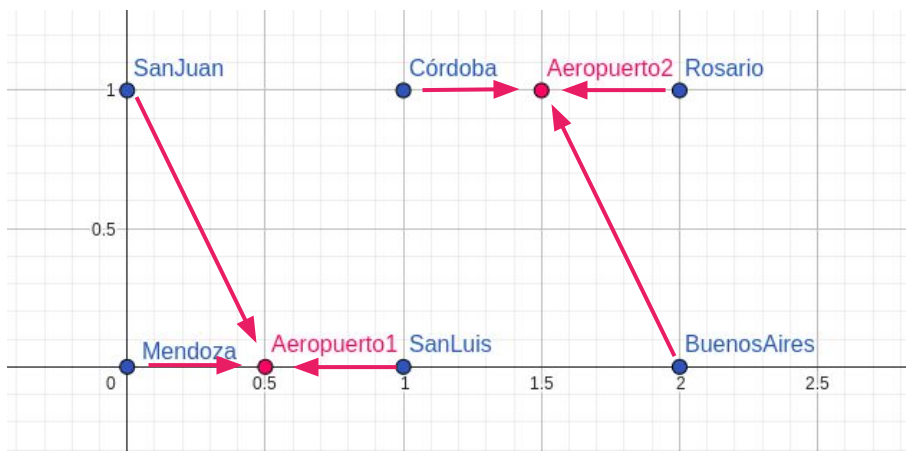
donde (x_1, y_1) es la coordenada del primer aeropuerto y (x_2, y_2) es la coordenada del segundo aeropuerto.

Ya no es posible calcular las derivadas parciales de f , pero sí es posible calcular el valor del gradiente en un punto.



Ejemplo – Colocar dos aeropuertos

Suponer que las ciudades se ubican en las siguientes coordenadas.



Si la coordenada del primer aeropuerto es $(0.5, 0)$ y la del segundo aeropuerto es $(1.5, 1)$, entonces es fácil determinar cuál es el aeropuerto más cercano para cada ciudad. Las flechas que salen de cada ciudad apuntan al aeropuerto más cercano.



Ejemplo – Colocar dos aeropuertos

— — —

- La función objetivo en un punto responde a la ley:

$$f(x_1, y_1, x_2, y_2) = - \sum_{i=1}^2 \sum_{c \in C_i} ((x_i - x_c)^2 + (y_i - y_c)^2),$$

donde C_i es el conjunto de ciudades que tienen al aeropuerto de la coordenada (x_i, y_i) como más cercano.

- El gradiente en un punto responde a la ley:

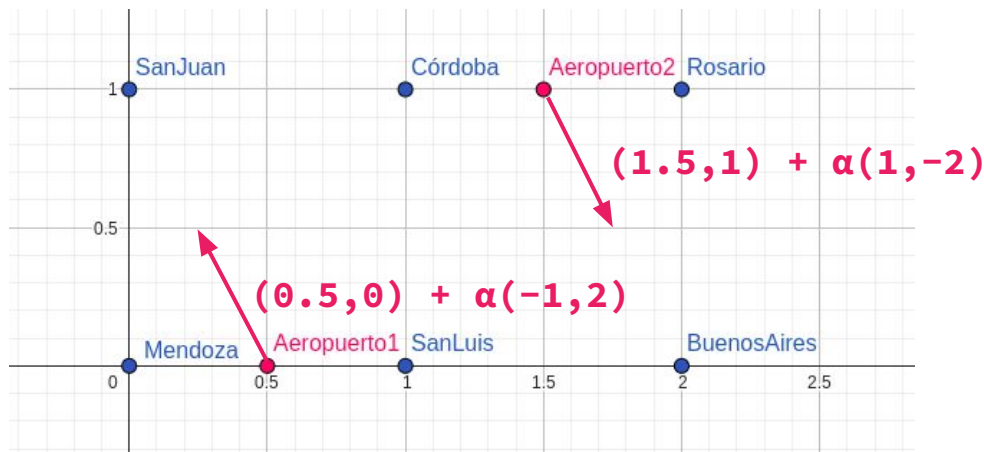
$$\nabla(f)(x_1, y_1, x_2, y_2) = \left(-2 \sum_{c \in C_1} (x_1 - x_c), -2 \sum_{c \in C_1} (y_1 - y_c), -2 \sum_{c \in C_2} (x_2 - x_c), -2 \sum_{c \in C_2} (y_2 - y_c) \right).$$



Ejemplo – Ubicar dos aeropuertos

— — —

En el punto $(0.5, 0, 1.5, 1)$, el gradiente es $(-1, 2, 1, -2)$.





Resumen

- ❑ La búsqueda tabú recorre el paisaje del espacio de estados de forma más sofisticada que la ascensión de colinas.
- ❑ Incorpora un mecanismo para escapar de máximos locales y una memoria de corto plazo para evitar estados visitados recientemente.
- ❑ Para implementarla, es necesario experimentar para encontrar los valores más adecuados para los parámetros.
- ❑ La ascensión de gradiente se utiliza para buscar un máximo local en problemas con espacios de estados continuos.
- ❑ En cada iteración, se mueve en la dirección de la ascensión más empinada, que está dada por el vector gradiente en ese punto.
- ❑ Es fundamental elegir un tamaño de paso adecuado.



Próximamente

— — —

En algunos problemas, nuestras decisiones están afectadas por las acciones de **otros agentes** que **compiten** en nuestra contra.

Veremos problemas de **búsqueda entre adversarios**.