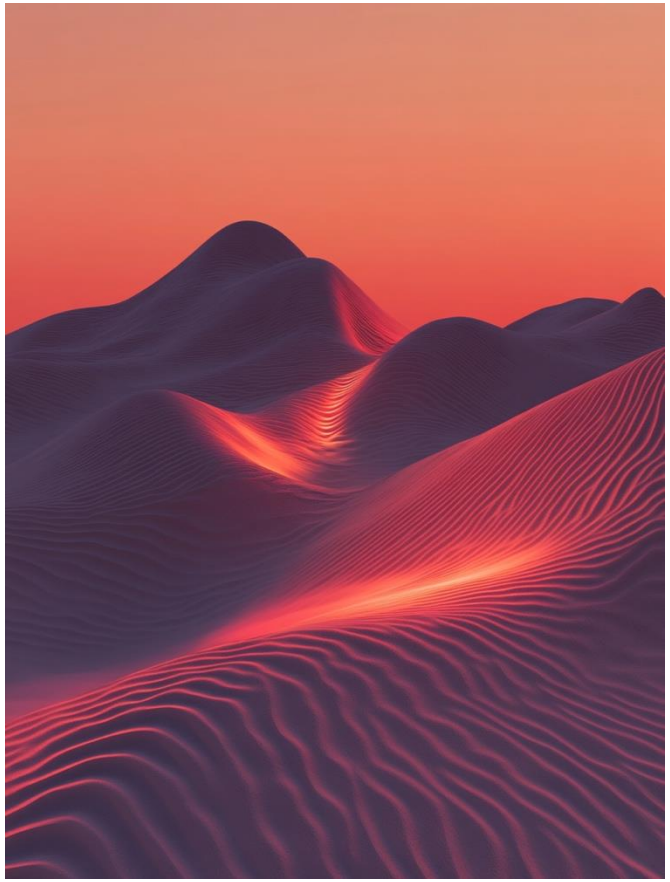
ALGORITMOS DE BÚSQUEDA LOCAL

ALGORITMOS DE BÚSQUEDA LOCAL

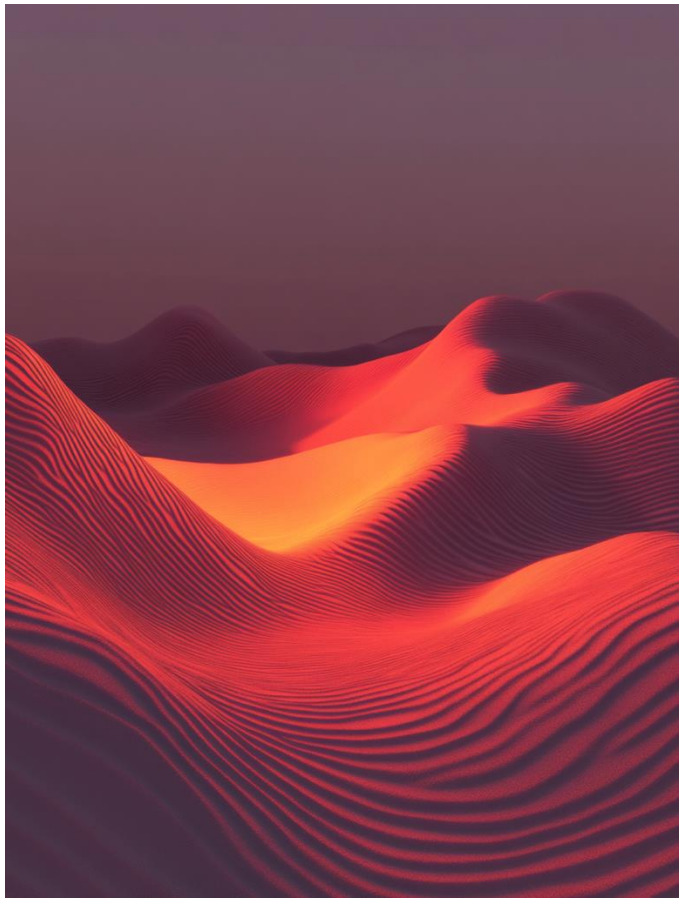
Los algoritmos de búsqueda que vimos la clase anterior se diseñan para explorar sistemáticamente espacios de búsqueda. Esta forma sistemática se alcanza manteniendo uno o más caminos en memoria y registrando qué alternativas se han explorado en cada punto a lo largo del camino y cuáles no.

Cuando se encuentra un objetivo, **el camino** a ese objetivo también constituye una **solución** al problema.

Pero hay problemas en donde **no** nos importa el camino, sino que importa la configuración final. Por ejemplo, un algoritmo que resuelva Sudokus.



ALGORITMOS DE BÚSQUEDA LOCAL



Si no importa el camino al objetivo, podemos considerar una clase diferente de algoritmos que no se preocupen en absoluto de los caminos.

Los algoritmos de búsqueda local funcionan con un solo estado actual y generalmente se mueve sólo a los *vecinos* del estado. Estos algoritmos tienen dos ventajas:

1. Usan poca memoria
2. Pueden encontrar soluciones razonables en espacios de estado grandes o infinitos (continuos).

ALGORITMOS DE BÚSQUEDA LOCAL

Problemas de optimización

Los algoritmos de búsqueda local son útiles para resolver **problemas de optimización** puros, en los cuales el objetivo es encontrar el mejor estado según una función objetivo.

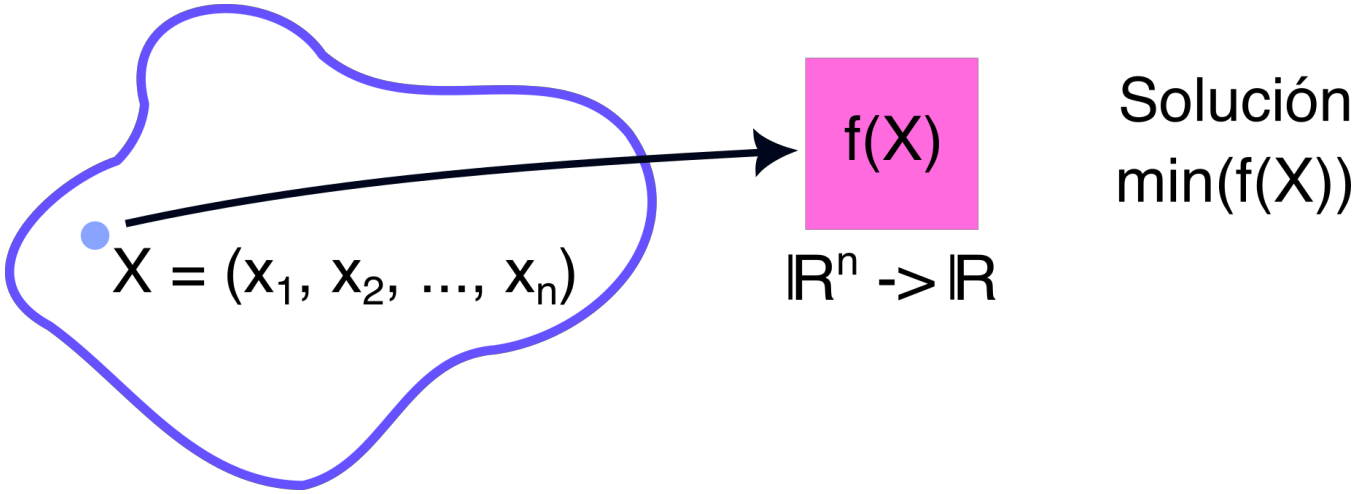
Un problema de optimización consiste en minimizar o maximizar el valor de una variable. En otras palabras, se trata de calcular o determinar el valor mínimo o el valor máximo de una función.

Los problemas de optimización se pueden dividir en dos categorías:

- Un problema de optimización con variables discretas se conoce como *optimización discreta*, en la que un objeto como un número entero, una permutación o un grafo se debe encontrar en un conjunto contable.
- Un problema con variables continuas se conoce como *optimización continua*, en la que se debe encontrar un valor óptimo de una función continua.

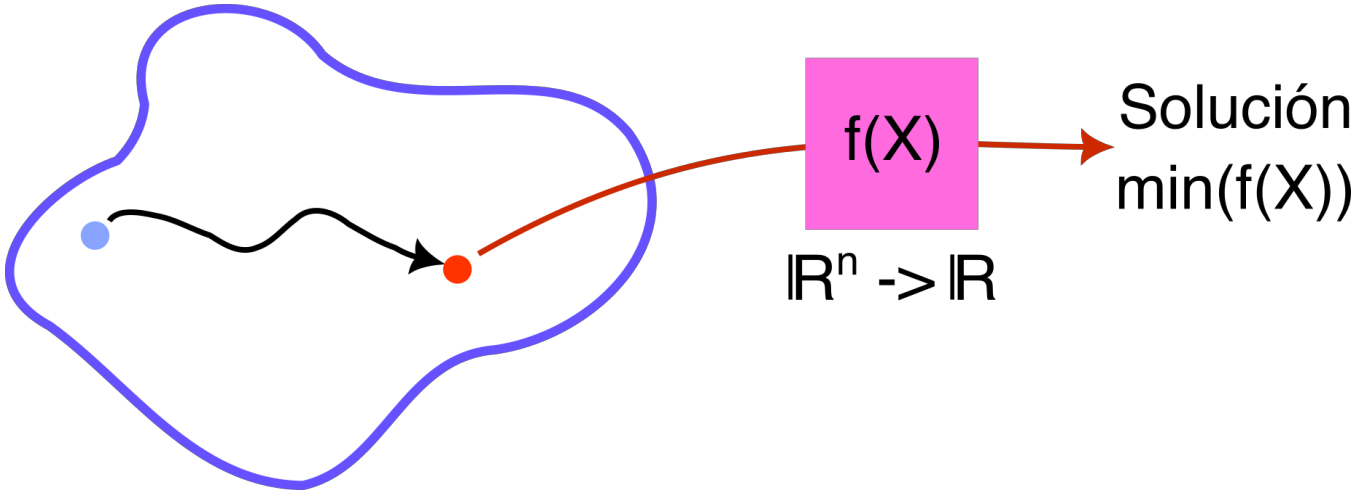
ALGORITMOS DE BÚSQUEDA LOCAL

Problemas de optimización



ALGORITMOS DE BÚSQUEDA LOCAL

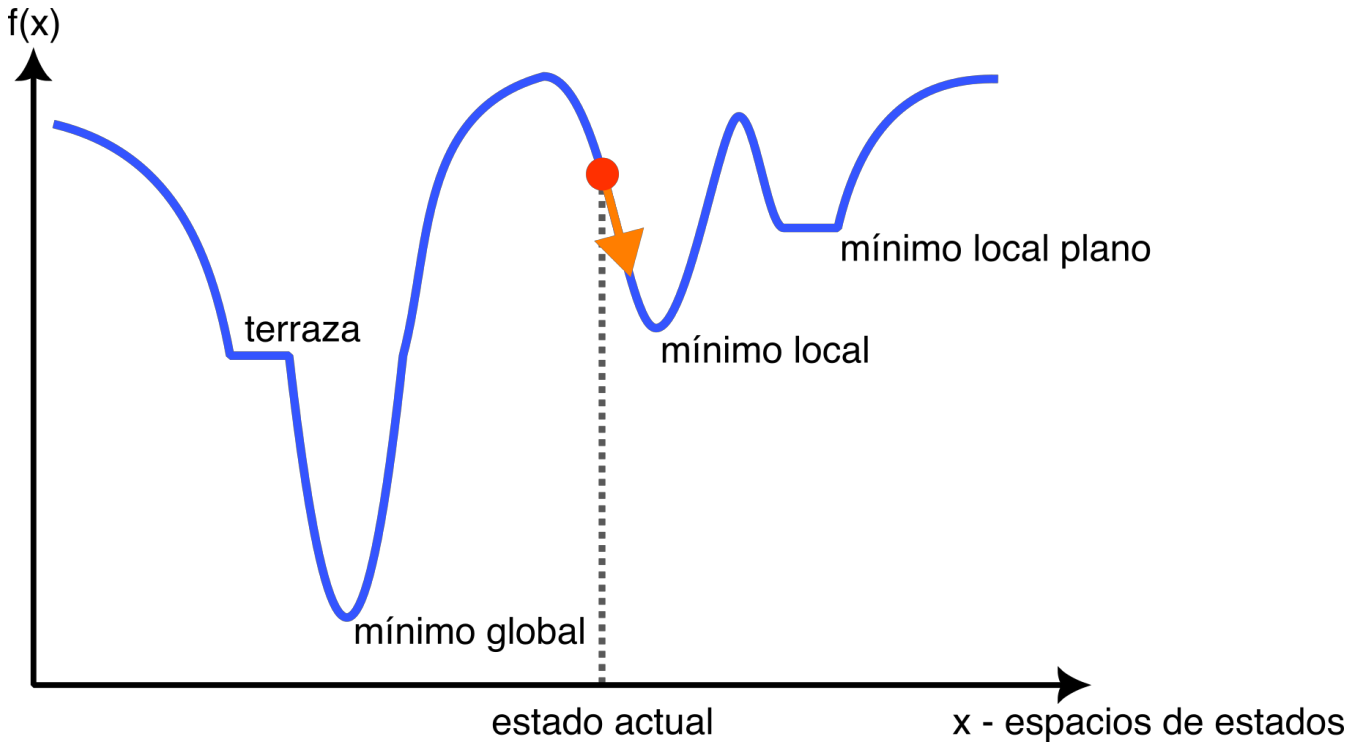
Problemas de optimización



ALGORITMOS DE BÚSQUEDA LOCAL

Problemas de optimización

Pasemos a un caso de una sola variable así podemos observar:



ALGORITMOS DE BÚSQUEDA LOCAL

Problemas de optimización

Hay múltiples algoritmos de búsqueda local, los que veremos hoy son:

Espacio discreto:

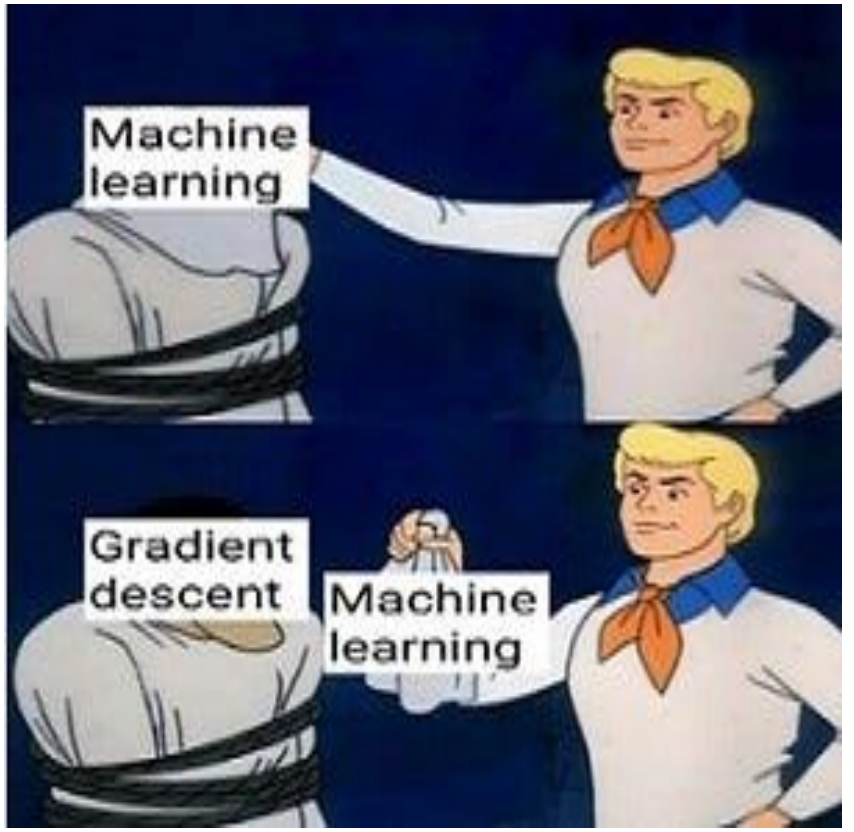
- **Gradiente descendiente o ascendente**
- **Simulated annealing**
- **Búsqueda Local Beam**
- **Algoritmos genéticos**

Espacio continuo:

- **Gradiente descendiente o ascendente en espacio continuo**

GRADIENTE DESCENDIENTE O ASCENDENTE

GRADIENTE DESCENDENTE O ASCENDENTE DISCRETO

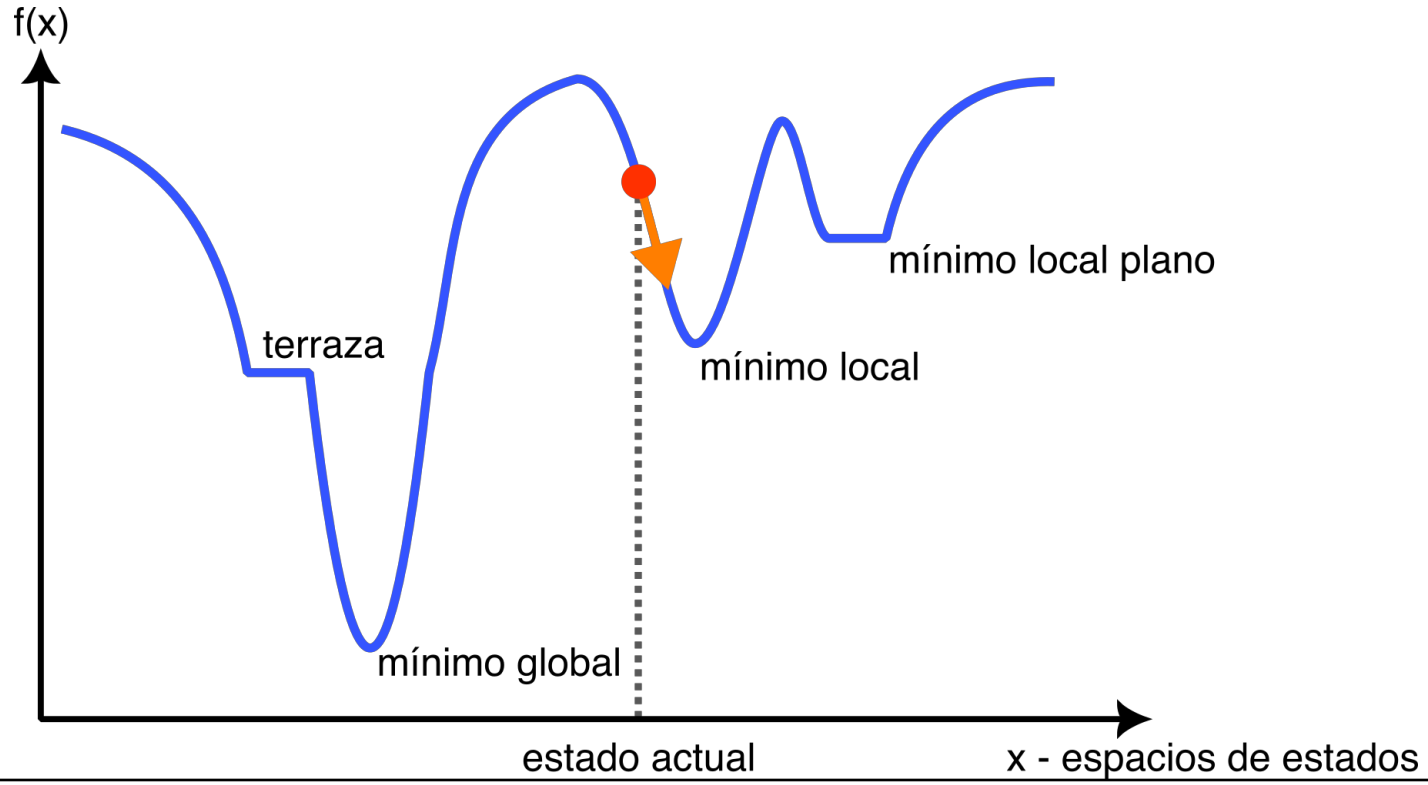


GRADIENTE DESCENDENTE O ASCENDENTE DISCRETO

Este algoritmo que continuamente se mueve en dirección de mayor valor decreciente o creciente. Termina la búsqueda en donde ningún vecino está más bajo. Este algoritmo no miras más allá de lo que tiene de los vecinos.

GRADIENTE DESCENDENTE O ASCENDENTE DISCRETO

Este algoritmo que continuamente se mueve en dirección de mayor valor decreciente o creciente. Termina la búsqueda en donde ningún vecino está más bajo. Este algoritmo no miras más allá de lo que tiene de los vecinos.



GRADIENTE DESCENDENTE O ASCENDENTE DISCRETO

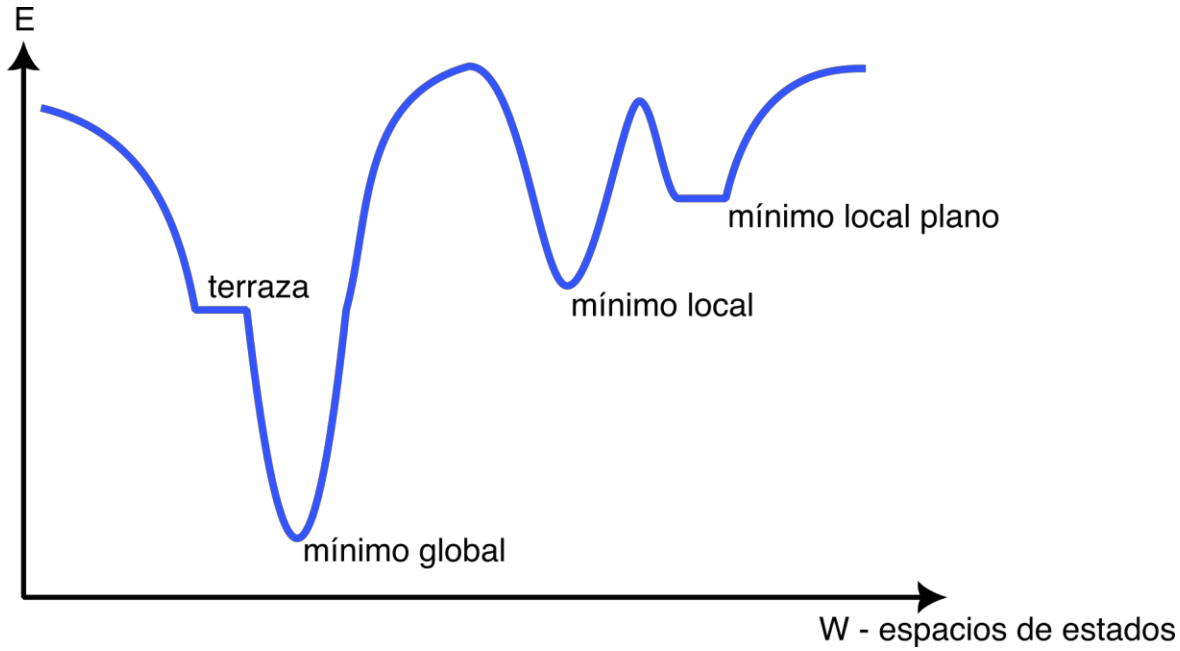
Esta búsqueda se le llama búsqueda local voraz (greedy), porque avanza a un estado vecino sin pensar en donde ir después.

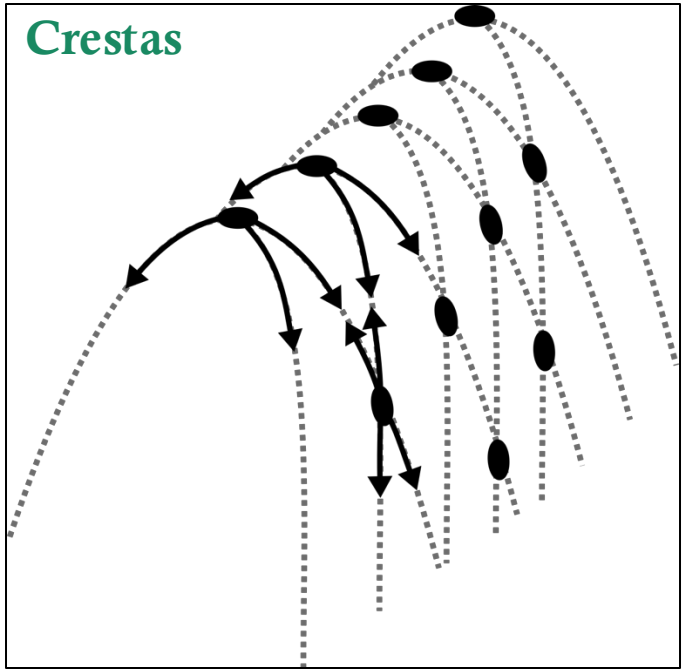
Avanza muy rápido hacia una solución, pero es fácil llegar a un estado no ideal. Se atasca por los siguientes motivos:

- **Mínimo (máximo) local:** Un mínimo local es un valle que es más bajo que sus estados vecinos, pero estás más arriba que el mínimo global.
- **Meseta:** Una meseta es un área donde la función de evaluación es plana.
- **Crestas:** Las crestas causan una secuencia de mínimos locales que hace muy difícil la navegación.

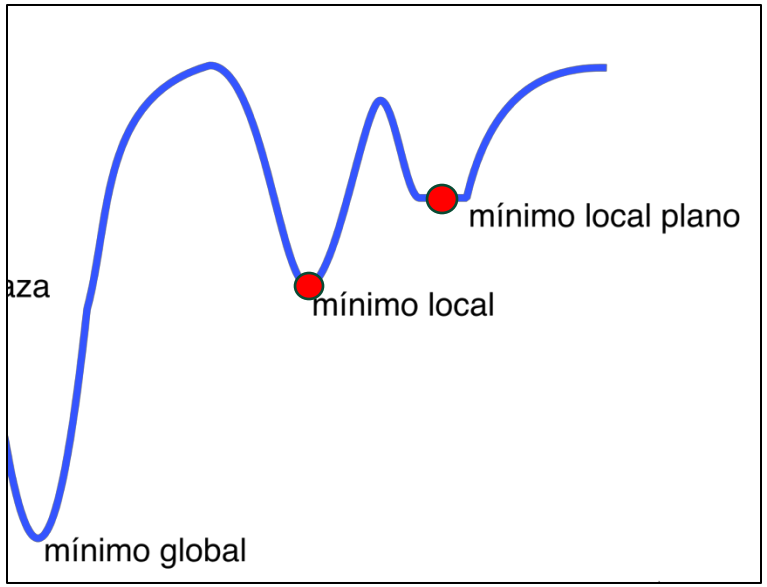
GRADIENTE DESCENDENTE O ASCENDENTE DISCRETO

Esta búsqueda se le llama búsqueda local voraz (greedy), porque avanza a un estado vecino sin pensar en donde ir después.

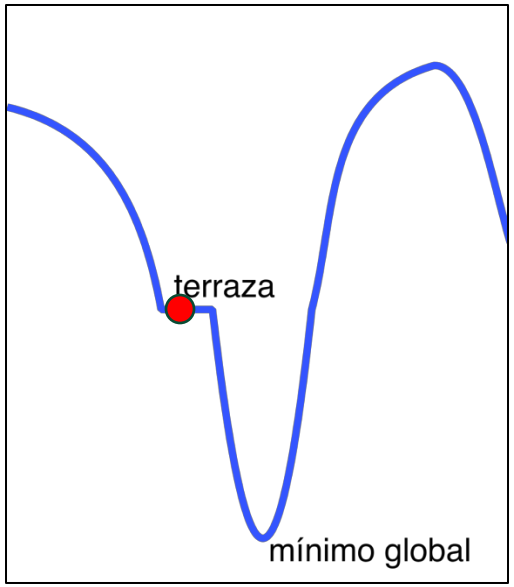




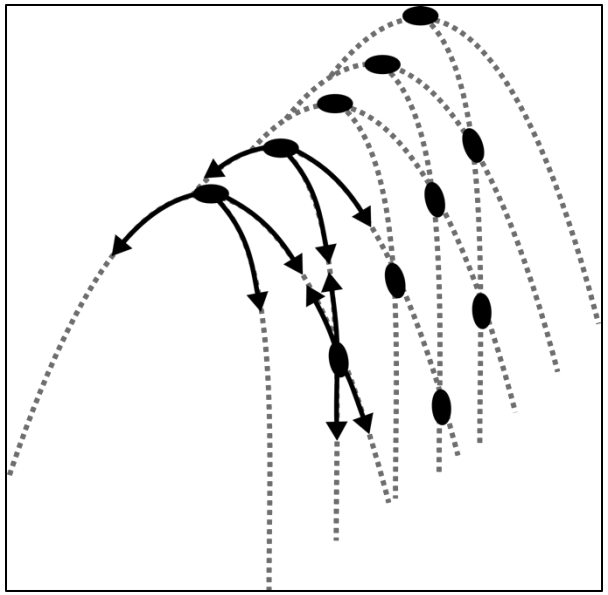
GRADIENTE DESCENDENTE O ASCENDENTE DISCRETO



Mínimo (máximo) local



Mesetas



Crestas

GRADIENTE DESCENDENTE O ASCENDENTE DISCRETO

Variantes de este algoritmo:

- **Ascenso o descenso estocástico:** Se escoge aleatoriamente la dirección de entre los movimientos ascendentes posibles. Converge más lento, pero en ciertos casos encuentra mejor solución.
- **Ascenso o descenso estocástico de primera opción:** Posee una planificación el cual genera sucesores de forma estocástica al azar hasta que se genera una que es mejor que el estado actual
- **Ascenso o descenso de reinicio aleatorio:** Esto conduce a una serie de búsquedas en ascensión de colinas desde estados iniciales generados aleatoriamente, parándose cuando se encuentra un objetivo.

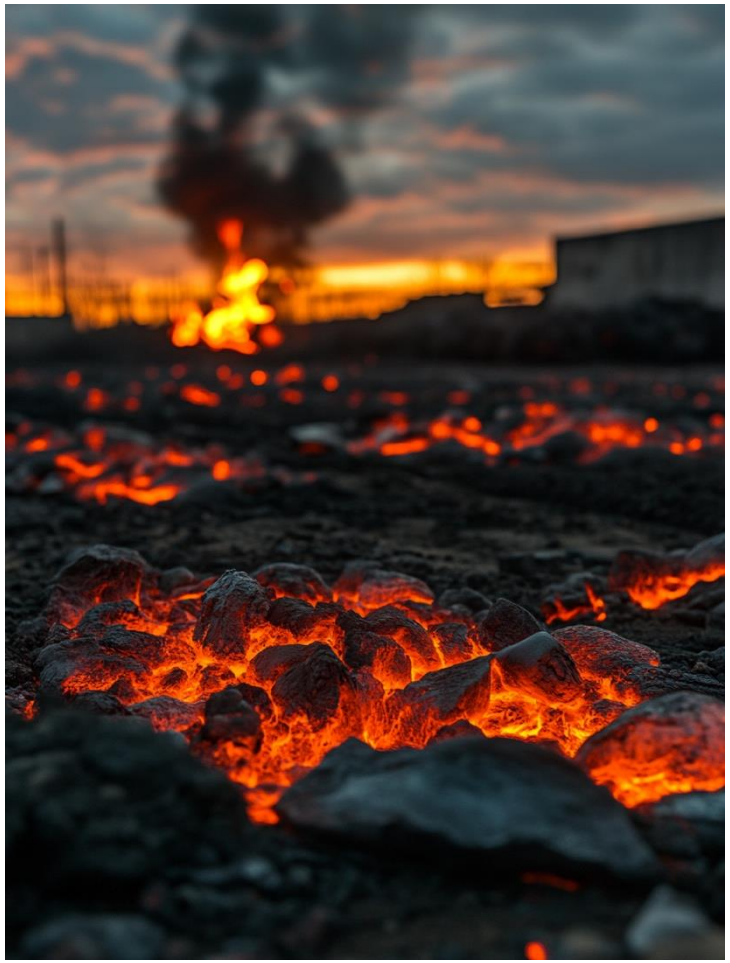
SIMULATED ANNEALING

SIMULATED ANNEALING

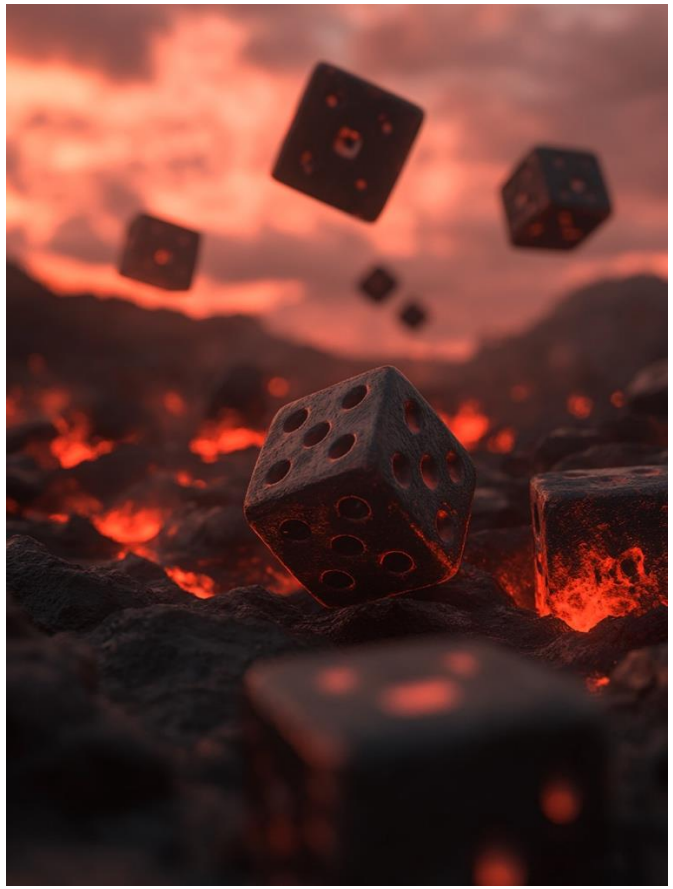
La idea para lograr esta sacudida al azar es usar un nuevo parámetro que llamamos temperatura. Cuando más temperatura haya más probable de que podamos movernos en dirección contraria, cuando baja la temperatura, baja esta probabilidad. Si la temperatura es cero estamos en un algoritmo de descenso de colina puro.

Este algoritmo arranca con mucha temperatura y a medida que avanza, se va enfriando, de igual forma que el metal.

Se puede demostrar que, si se disminuye la temperatura bastante despacio, el algoritmo encontrará un óptimo global con probabilidad cerca de uno.



SIMULATED ANNEALING



Para determinar si se hace el cambio, se hace uso de la distribución de Boltzmann, el cual es una distribución de probabilidad de partículas en un sistema a través de varios estados posibles:

$$F \propto e^{-\frac{\Delta E}{kT}}$$

Esta distribución establece que cuando más grande es la diferencia de energía (o en nuestro caso la diferencia de la función de costo) menos probable es que se elija, pero si la **temperatura es alta, esta probabilidad es mayor y es más fácil que de todas formas se elija.**

BÚSQUEDA LOCAL BEAM

BÚSQUEDA LOCAL BEAM

Buscar desde un estado inicial únicamente es una medida un poco extrema de ahorro de memoria.

En cambio, **búsqueda local beam** guarda la información de k estados y sobre ellos realiza la búsqueda independientemente. Estos se inician al azar.

En cada paso se generan sucesores de los k estados. Si alguno cumple el objetivo, se termina, en cambio si no, se seleccionan los k mejores sucesores de la lista.

A simple vista pareciera lo mismo de correr k veces a gradiente descendiente o ascendente, pero a diferencia de esto es que entre los procesos de búsquedas hay pase de información:

Si un estado genera varios sucesores buenos y los otros k-1 estados generan sucesores malos, entonces el efecto es que el primer estado abandona la búsqueda de los otros y se queda con los sucesores del primer estado.

El algoritmo rápidamente abandona las búsquedas infructuosas y mueve sus recursos a donde se hace la mayor parte del progreso.

BÚSQUEDA LOCAL BEAM

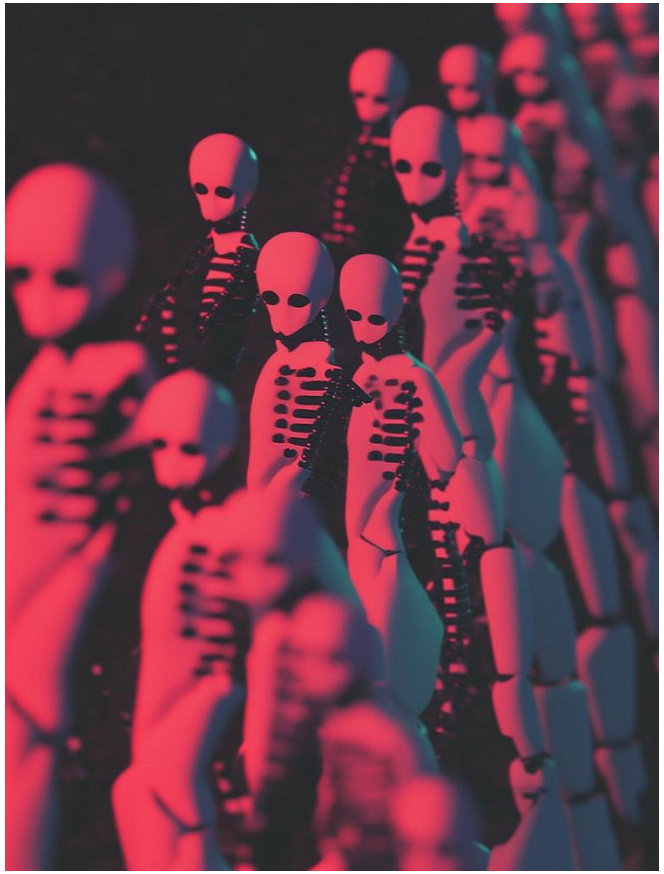
El principal **inconveniente** de este algoritmo es que puede sufrir carencia de diversidad de estados, enfocándose en una sola parte muy limitada.

Una variante es la **búsqueda de haz estocástica**, en el cual en vez de elegir a los k mejores sucesores, se eligen aleatoriamente, con una función de temperatura similar a la de *simulated annealing*.

Esta búsqueda muestra un parecido con el proceso de selección natural, por lo cual los *sucesores* (descendientes) de un *estado* (organismo) pueblan la siguiente generación según su *valor* (idoneidad o salud).

ALGORITMOS GENÉTICOS

ALGORITMOS GENÉTICOS

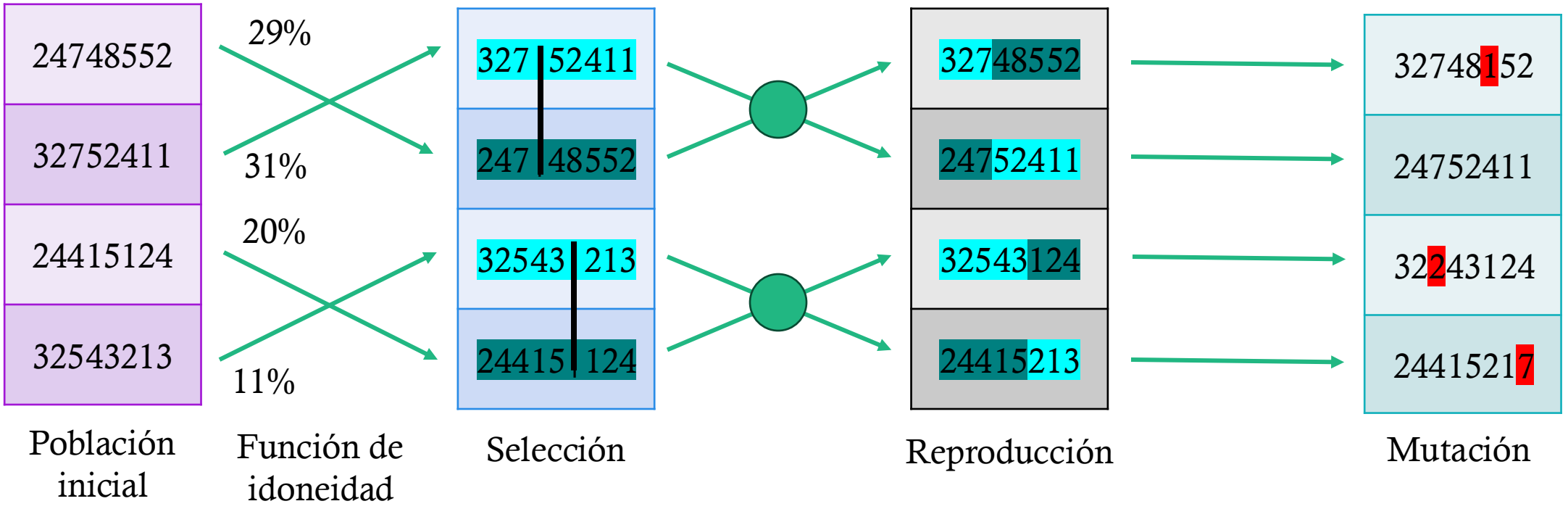


Un **algoritmo genético** es una variante de la búsqueda local beam estocátisca en la que los estados sucesores se generan combinando dos estados padres (reproducción), más que modificar un solo estado (reproducción asexual).

La analogía con la selección natural es la misma que anteriormente.

Los algoritmos genéticos comienzan con un conjunto k de estados aleatorios, llamados **población**. Cada estado, o **individuo**, está representado como una cadena de caracteres sobre un **alfabeto finito** que representa el código genético del estado.

ALGORITMOS GENÉTICOS



ALGORITMOS GENÉTICOS

Como en la búsqueda beam local estocástica, los algoritmos genéticos combinan una tendencia ascendente con exploración aleatoria y cambian la información entre los hilos paralelos de búsqueda.

La ventaja del algoritmo genético viene de la operación de cruce para combinar bloques grandes de letras que han evolucionado independientemente para así realizar funciones útiles, de modo que se aumente el nivel de granularidad en el que funciona la búsqueda.

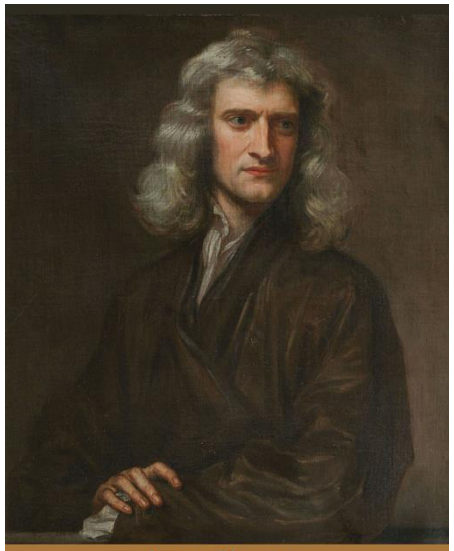
```
def gen_alg(poblation, fun_ido):
  new_poblation = []

  while 1:
    for i in range(len(poblation)):
      x = sel_aleatory(poblation, fun_ido)
      y = sel_aleatory(poblation, fun_ido)
      child = reproduction(x, y)
      dice = random()
      if dice < 0.01:
        child = mutate(child)
      new_poblation.append(child)
    poblation = new_poblation

    for z in poblation:
      if z is solution:
        return z

def reproduction(x, y):
  n = len(x)
  c = random_int(0, n)
  return x[:c] + y[c:]
```

BÚSQUEDA EN ESPACIOS CONTINUOS

BÚSQUEDA EN ESPACIOS CONTINUOS

Los algoritmos que vimos hasta ahora no se pueden aplicar en casos de espacio continuos ya que los sucesores que nos presentan son infinitos.

La literatura de espacio de búsqueda continuas es enorme y es tan vieja como la época de Newton y Liebniz (siglo XVII).



Isaac Newton
(1643 - 1727)

Gottfried Wilhelm Leibniz
(1646 - 1716)

BÚSQUEDA EN ESPACIOS CONTINUOS

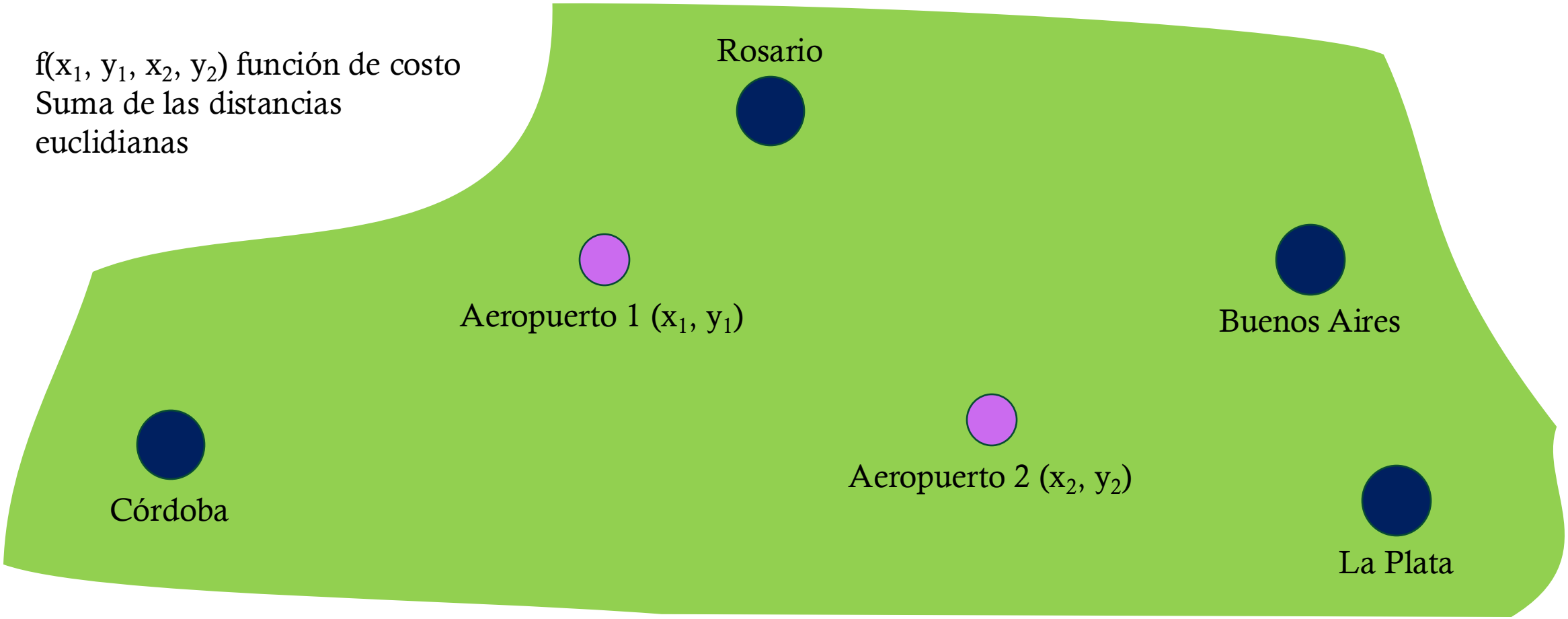
Pensemos un ejemplo, se quiere colocar dos aeropuertos nuevos en Argentina, de tal forma que la distancia al cuadrado de cada gran ciudad de Argentina (Buenos Aires, La Plata, Córdoba y Rosario) con respecto a los aeropuertos sea mínima.

Entonces el espacio de estado está definido por 4 coordenadas: (x_1, y_1), (x_2, y_2) correspondiente a la ubicación de los aeropuertos.

Es un espacio cuatro-dimensional o que los estados están definidos por 4 variables. Moverse sobre este espacio se corresponde a moverse a movimientos de los aeropuertos.

La función objetivo $f(x_1, y_1, x_2, y_2)$ es relativamente fácil de calcularla usando la suma de la distancia euclidiana con cada ciudad.

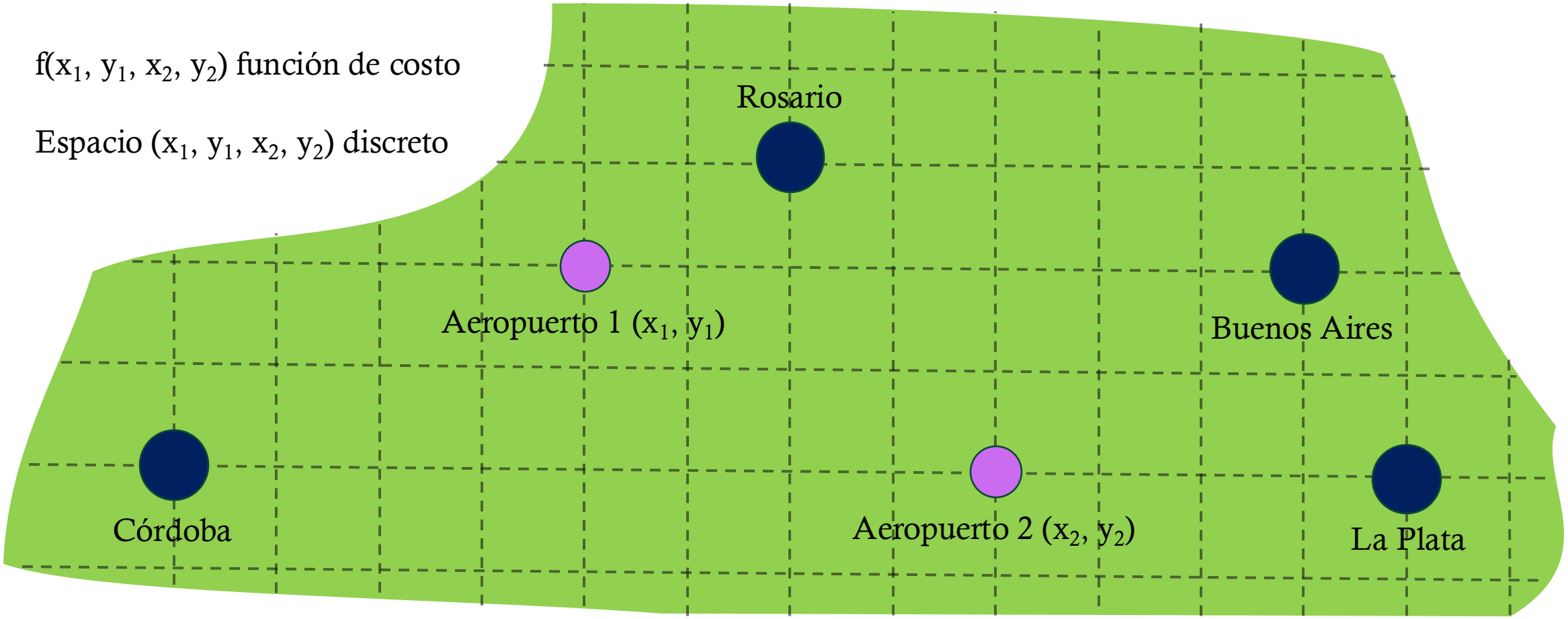
BÚSQUEDA EN ESPACIOS CONTINUOS



BÚSQUEDA EN ESPACIOS CONTINUOS

$f(x_1, y_1, x_2, y_2)$ función de costo

Espacio (x_1, y_1, x_2, y_2) discreto



BÚSQUEDA EN ESPACIOS CONTINUOS

Una forma de resolver esto es **discretizar el espacio**.

Por ejemplo, podemos mover de a pequeños pasos a los aeropuertos en dirección x o y, y en una cantidad fija.

Con 4 variables, nos da 8 sucesores para cada estado. De ahí aplicamos cualquier algoritmo de búsqueda local.

BÚSQUEDA EN ESPACIOS CONTINUOS

Hay métodos de usar el gradiente para encontrar un máximo o mínimo:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}\right)$$

En algunos casos podemos resolver la ecuación diferencial:

$$\nabla f = 0$$

Para el ejemplo si tuviéramos un solo aeropuerto, sería la media aritmética de la ubicación de las ciudades.

BÚSQUEDA EN ESPACIOS CONTINUOS

En muchos casos esto no puede resolverse directamente. Pero podemos hacer uso del algoritmo de gradiente, haciendo un cambio local

$$X_{new} = X - \alpha \nabla f(X)$$

En donde alfa es una constante pequeña, que en redes neuronales se llama **constante de aprendizaje.**

Hay casos que no tenemos la función objetivo de forma diferenciable. En estos casos se usa el gradiente empírico evaluando pequeño incrementos y decrementos de cada coordenada.

BÚSQUEDA EN ESPACIOS CONTINUOS

La elección de alfa es sumamente importante, porque si alfa es muy pequeña, necesitamos demasiados pasos, en cambio sí es muy grande nunca puede converger.