3. Búsqueda avanzada

Heurísticos de Búsqueda

Iñigo Lopez-Gazpio · inigo.lopez@ehu.eus

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Búsqueda local

¿ Cuál es el problema de los algoritmos de búsqueda local ?

Búsqueda local

Problemas – El criterio para terminar el algoritmo

- ¿ Espacio de estados completamente analizado ?
- ¿ Tiempo de ejecución ?
- ¿ Número de iteraciones ?
- ¿ Estricta mejora del fitness del estado actual ?

Búsqueda local

Normalmente **la no mejora** en el vecindario provoca la terminación

Ejemplo: algoritmo Basic Hill-Climbing

```
Make the initialNode with initial problem state
currentNode = initialNode

WHILE NOT local_best_FOUND DO
     EXPAND currentNode AND KEEP bestSuccessor
     IF f(currentNode) "better or equal than"
     f(bestSuccessor) THEN
          local_best_FOUND
     ELSE
     currentNode = bestSuccessor


RETURN currentNode's state
```

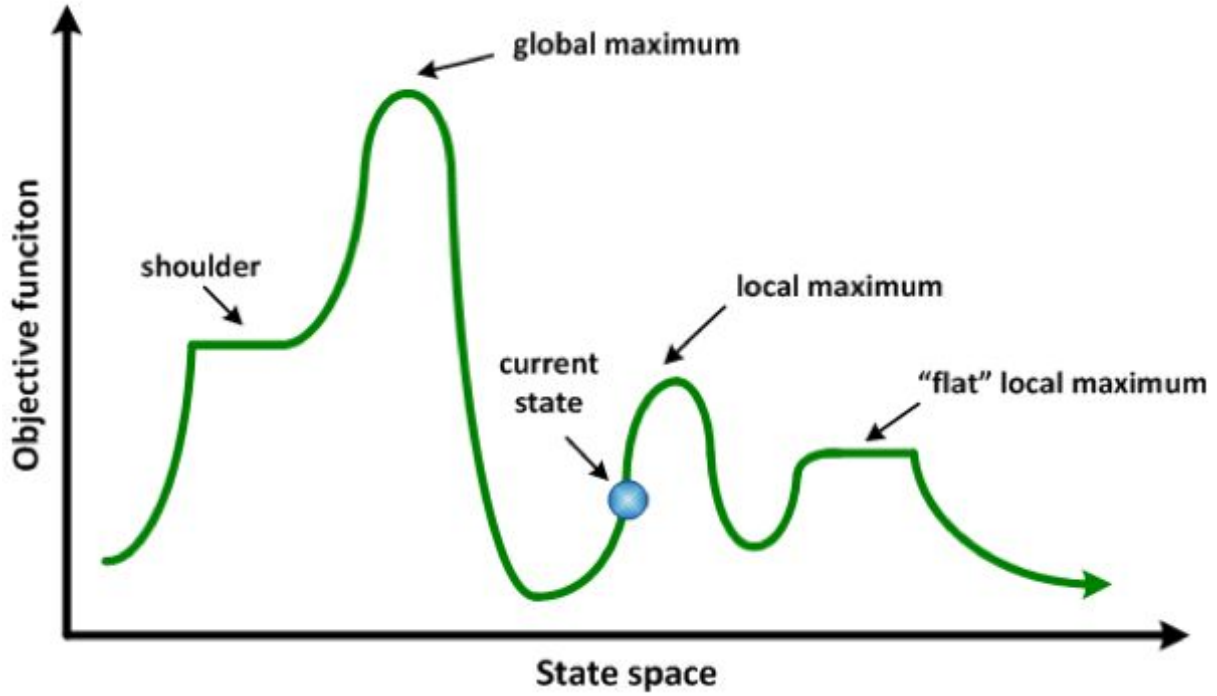
```
Función EXPAND


bestSuccessor = currentNode's firstSuccessor

FOR EACH of the rest of currentNode's Successors DO
     Compute f(successor)
     IF f(successor) "better than" f(bestSuccessor) THEN
          bestSuccessors.append( successor )

RETURN bestSuccessors
```

Búsqueda local

¿ A qué problema estamos haciendo referencia ?

Búsqueda local

El problema de los óptimos locales



Búsqueda local - Ejemplo

¿ A dónde transicionamos si el fitness del actual candidato es 11 ?

Búsqueda local

¿ Soluciones posibles contra el problema de los óptimos locales ?

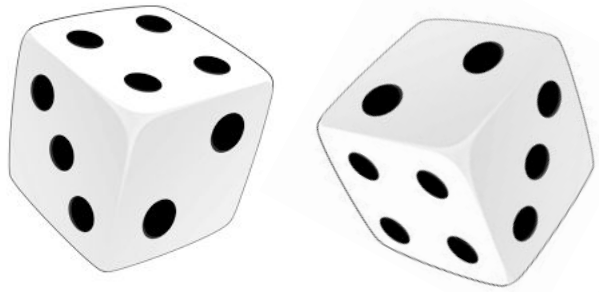
Búsqueda local

Soluciones posibles contra el problema de los óptimos locales

1. Cambiar la inicialización
2. Cambiar el landscape (el entorno)
3. Aceptar soluciones peores (a corto plazo)

Búsqueda local - 1. Cambiar la inicialización

Búsqueda local - 1. Cambiar la inicialización

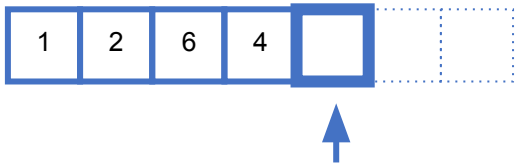


Variantes del algoritmo Basic Hill-Climbing

- **Stochastic Hill-Climbing**
  - Selección entre el vecindario de manera aleatoria (esta probabilidad es dinámica y varía)
  - Mayor complicación para encontrar un óptimo global
- **Random-Restart Hill-Climbing**
  - Lanzamos N veces el algoritmo Hill-Climbing (de manera independiente)
  - Muy potente si se paraleliza y $N \rightarrow \infty$
- **Local Beam Search**
  - En lugar de 1 candidato (current) mantenemos K candidatos en todo momento (en la misma ejecución)
  - El vecindario se calcula en base a los K candidatos actuales
- **(Stochastic) Multi-start Local Search**
  - Una combinación de lo anterior. Local Beam Search + aleatoriedad en la inicialización + aleatoriedad selección
- **Greedy Randomized Adaptive Search (GRASP)**
  - Formulación incremental estocástica + Búsqueda local
- **Iterated Local Search**
  - Añadir perturbación o mutación en el candidato

GRASP - Greedy Randomized Adaptive Search Procedure

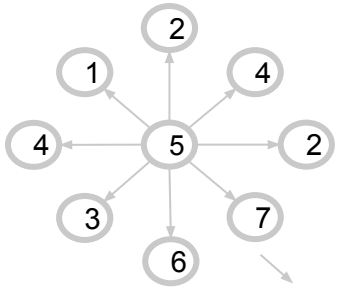
Algoritmo incremental

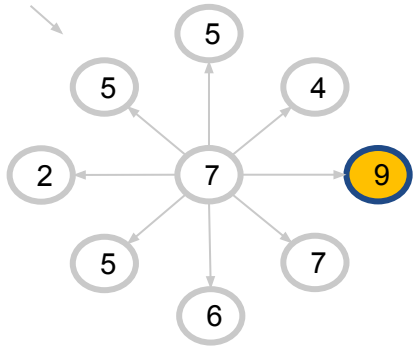


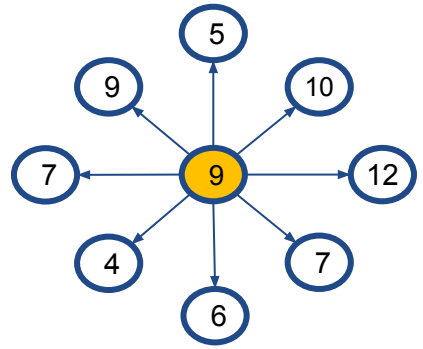
Opciones restantes…..

+

Búsqueda local







¿ Problemas ?

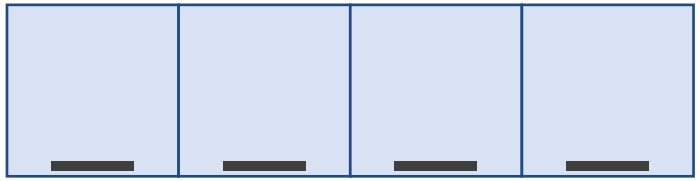
Los algoritmos incrementales son deterministas y tienden a producir los mismos resultados

Solución

Añadir aleatoriedad

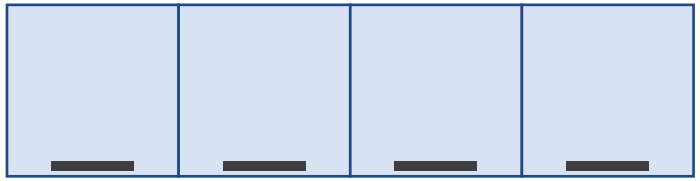
GRASP - Ejemplo

Opción	Fitness
A	25
B	10
C	11
D	15



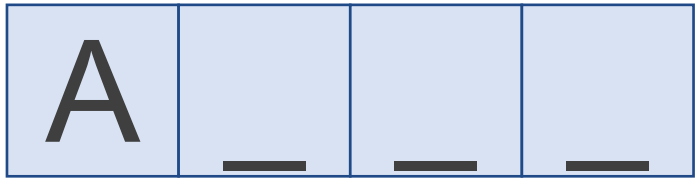
GRASP - Ejemplo

Opción	Fitness
A	25
B	10
C	11
D	15



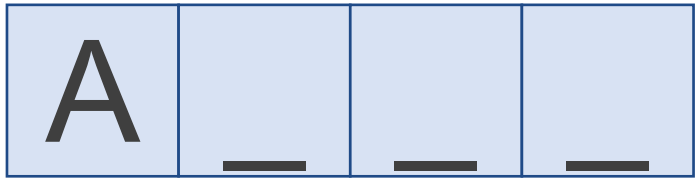
GRASP - Ejemplo

Opción	Fitness
~~A~~	~~25~~
B	10
C	11
D	15



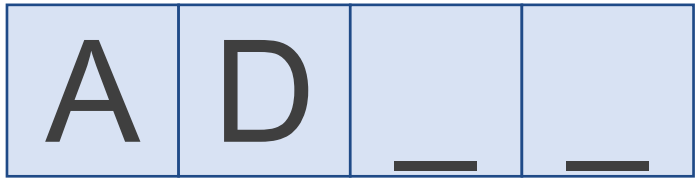
GRASP - Ejemplo

Opción	Fitness
~~A~~	~~25~~
B	10
C	11
D	15



GRASP - Ejemplo

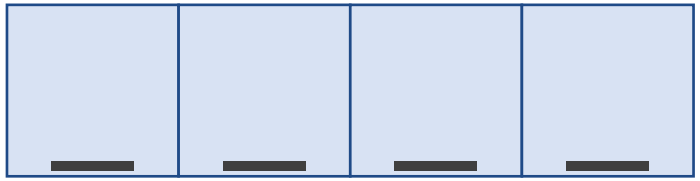
Opción	Fitness
~~A~~	~~25~~
B	10
C	11
~~D~~	~~15~~



GRASP + Stochastic. Ejemplo

Elaboramos una selección de las mejores opciones.
(longitud ***k***)

Opción	Fitness
A	25
B	10
C	11
D	15



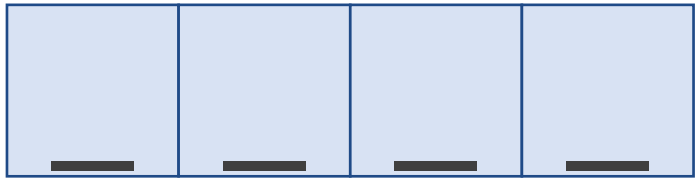
GRASP + Stochastic. Ejemplo

Opción	Fitness
A	25
B	10
C	11
D	15

Elaboramos una selección de las mejores opciones. (longitud ***k***)

Opción	Fitness
A	25
D	15

Elegir opción final de manera aleatoria



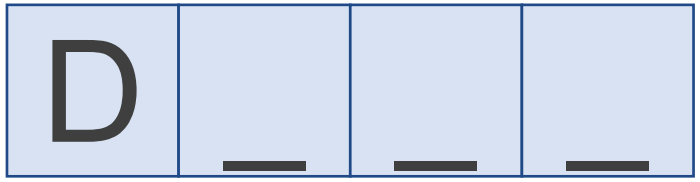
GRASP + Stochastic. Ejemplo

Opción	Fitness
A	25
B	10
C	11
D	15

Elaboramos una selección de las mejores opciones. (longitud ***k***)

Opción	Fitness
A	25
D	15

Elegir opción final de manera aleatoria



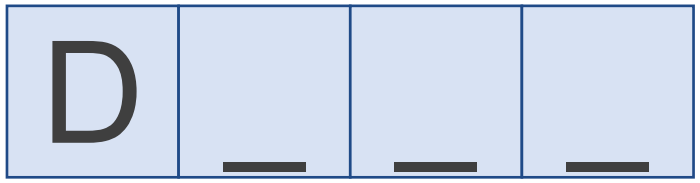
GRASP + Stochastic. Ejemplo

Opción	Fitness
A	25
B	10
C	11
~~D~~	~~15~~

Elaboramos una selección de las mejores opciones. (longitud ***k***)

Opción	Fitness
A	25
D	15

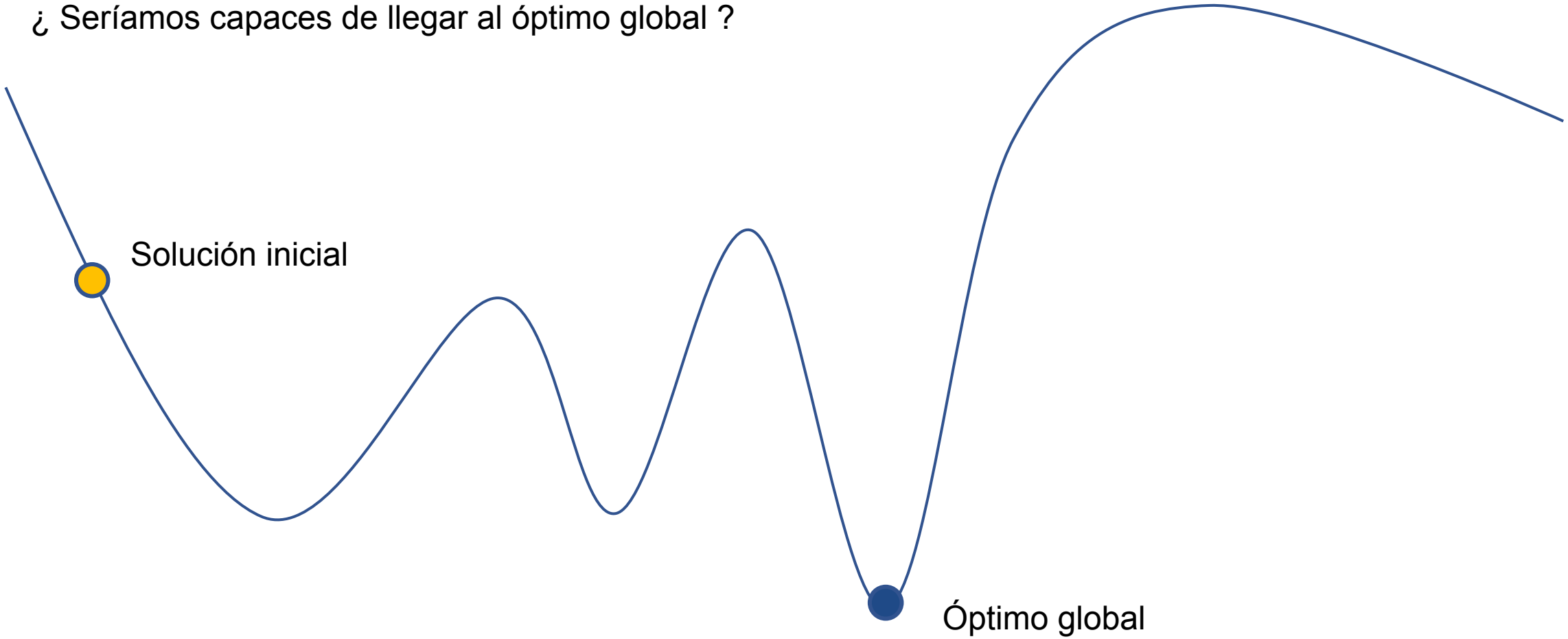
Elegir opción final de manera aleatoria



¿ Puedes describir el comportamiento del algoritmo en referencia al hiperparámetro *k* ?

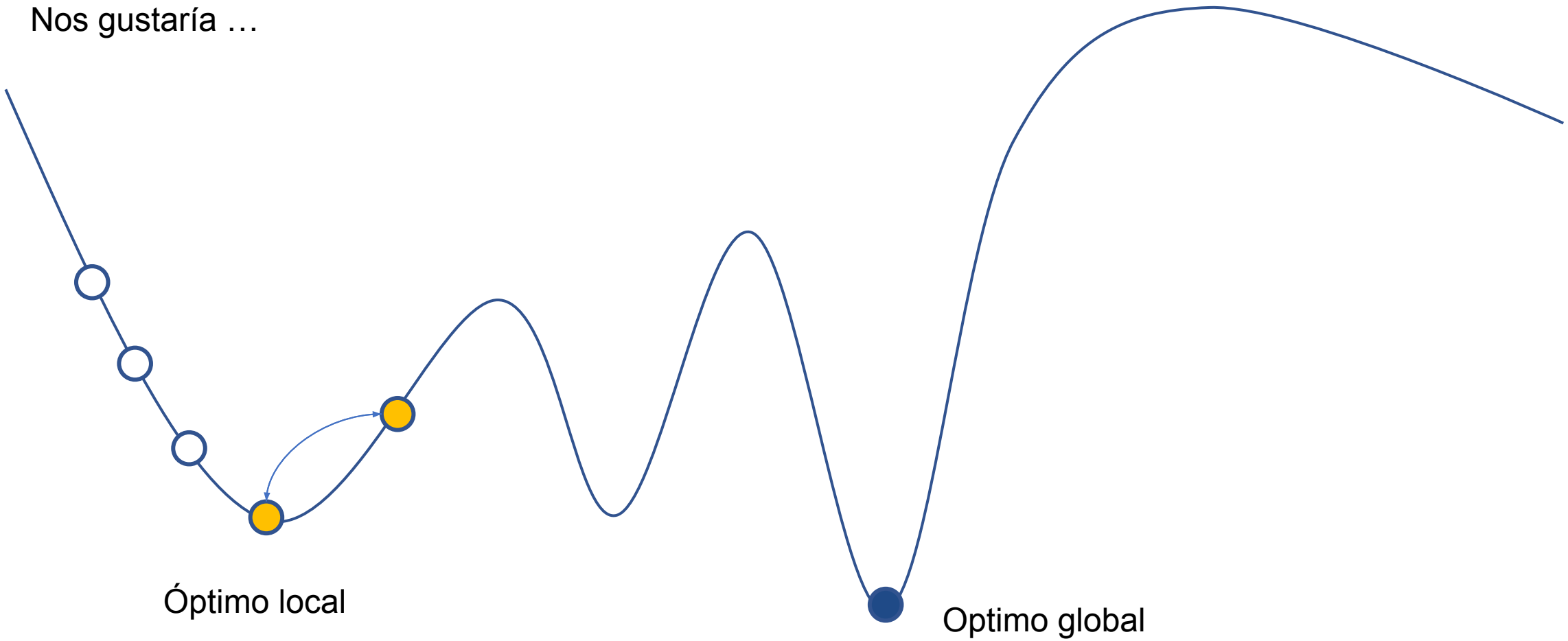
ILS - Iterated Local Search - ejemplo minimización

¿ Seríamos capaces de llegar al óptimo global ?



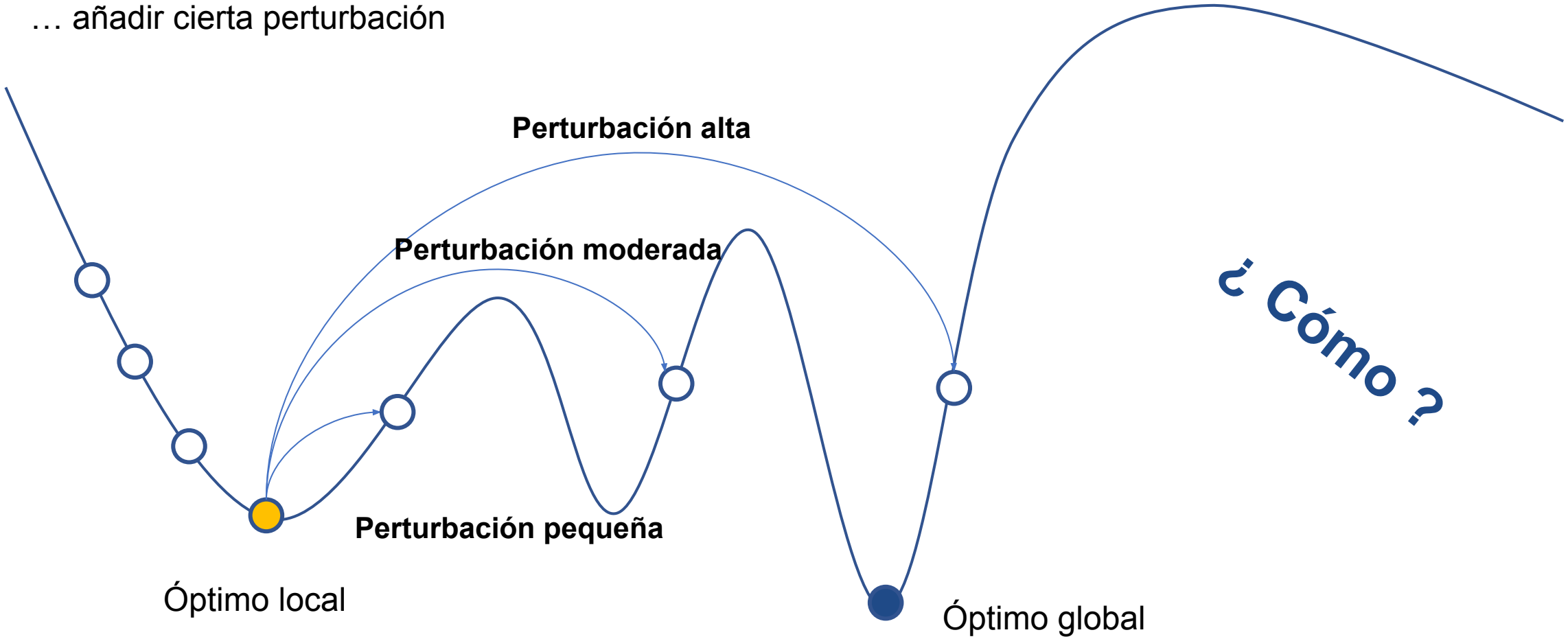
ILS - Iterated Local Search - ejemplo minimización

ILS - Iterated Local Search - ejemplo minimización



ILS - Iterated Local Search - ejemplo minimización

… añadir cierta perturbación



ILS - Iterated Local Search - ejemplo minimización

¿ Cuánta perturbación ?

Intensificar — Diversificar

Pequeña — **Alta**

Se consiguen distintos comportamientos para los algoritmos.

Un grado demasiado elevado hará nuestros algoritmos altamente ineficientes

Búsqueda local - 2. Cambiar el landscape (el entorno)

Búsqueda local - 2. Cambiar el landscape (el entorno)

Métodos para relajar el entorno

- **Variable Neighborhood Descent (VND)**
  - Método que permite explorar vecindades más lejanas (Hiperparámetro ***k***)
- **Variable Neighborhood Search (VNS)**
  - VND + **estocástico**
- **Métodos Smoothing**
  - Relajamos la función objetivo (Similar a **admisibilidad / dominancia** para el algoritmo A*)

VND - Variable Neighborhood Descent

Procedure VND: Given N_k, (k=1,.., k_{max}) and some x∈X

01. **Begin**
02. **Repeat**
03. k=1;
04. **Repeat**
05. x' ← local-search(x , $N_k(x)$);
06. **if** (f(x') < f(x)) **then** x ←x'
07. **else** k←k+1;
08. **until** $k = k_{max}$;
09. **until** (some stop condition is satisfied)
10. **end.**

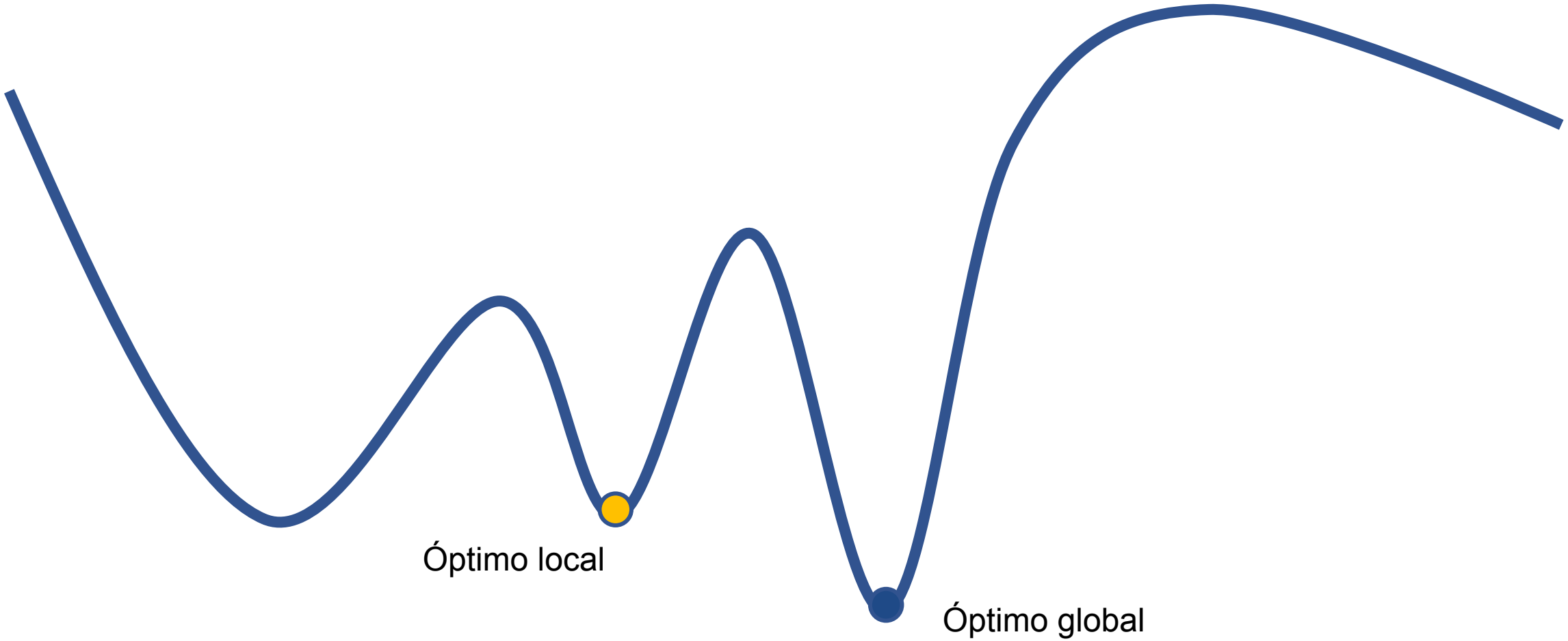
VNS - Variable Neighborhood Search

Procedure VND: Given N_k, (k=1,.., k_{max}) and some x∈X

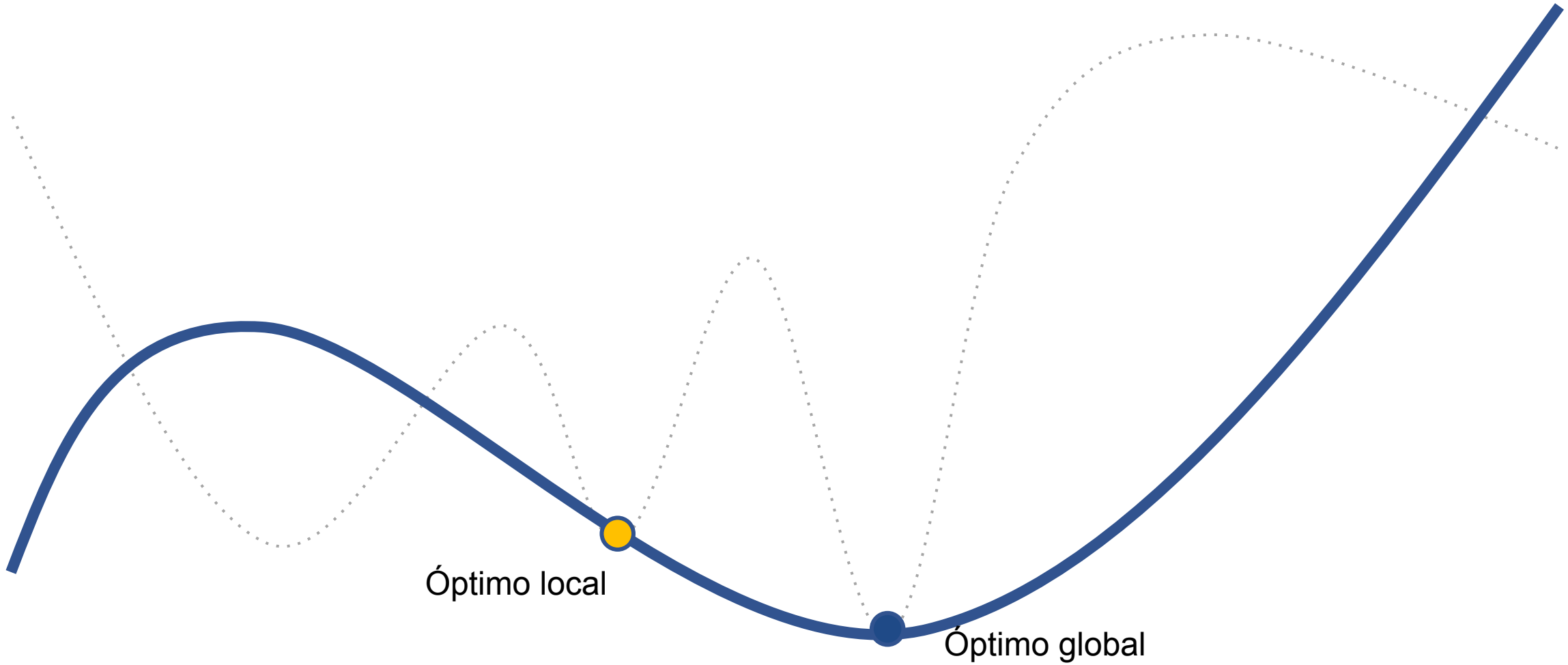
01. **Begin**
02. **Repeat**
03. k=1;
04. **Repeat**
05. ~~x' ← local search(x , N_k (x));~~
06. **if** (f(x') < f(x)) **then** x ←x'
07. **else** k←k+1;
08. **until** $k = k_{max}$;
09. **until** (some stop condition is satisfied)
10. **end.**

x' ← Random (N_k (x))
X'' ←local_search (N_k (x'))

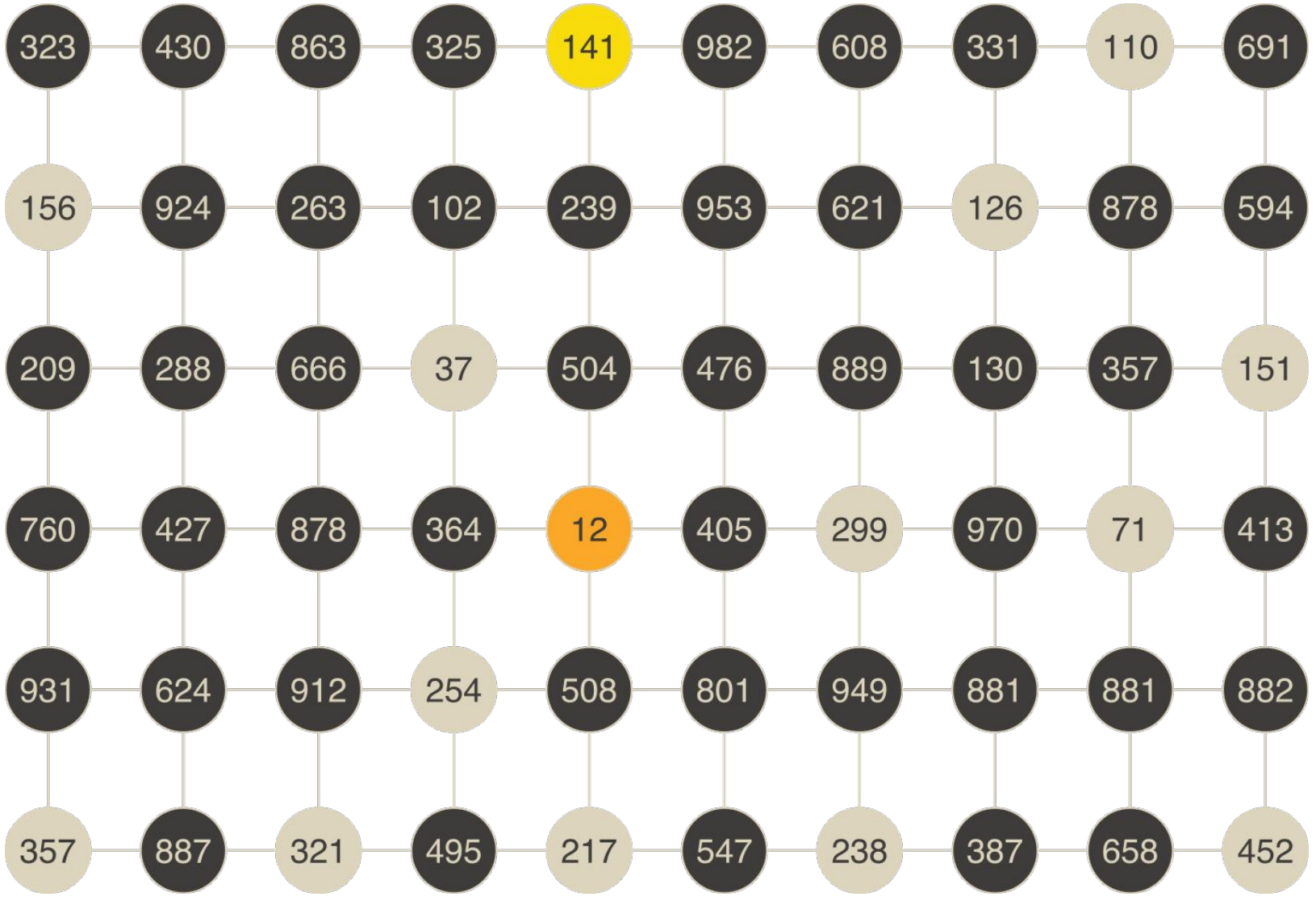
Métodos smoothing



Métodos smoothing

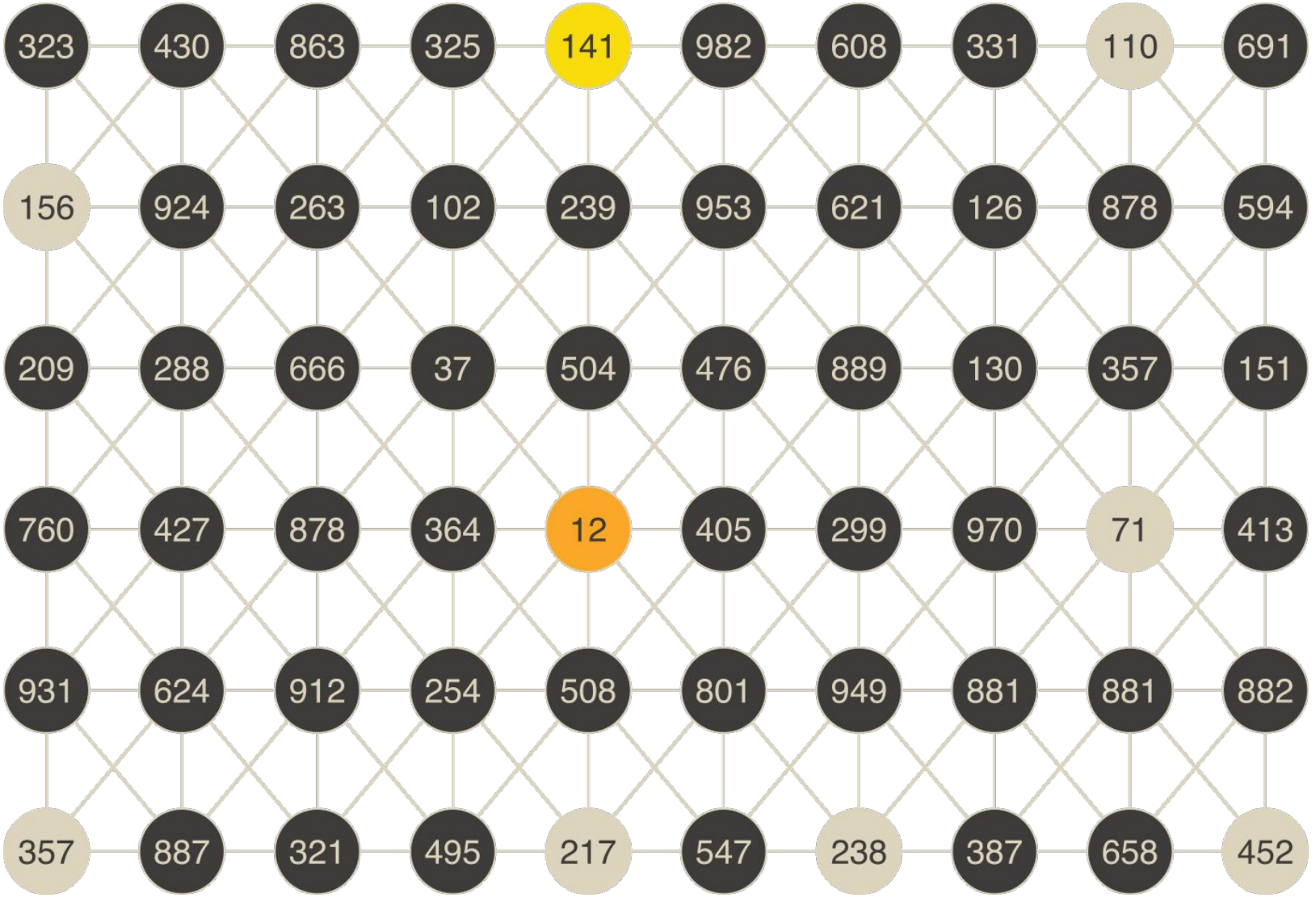


Métodos smoothing



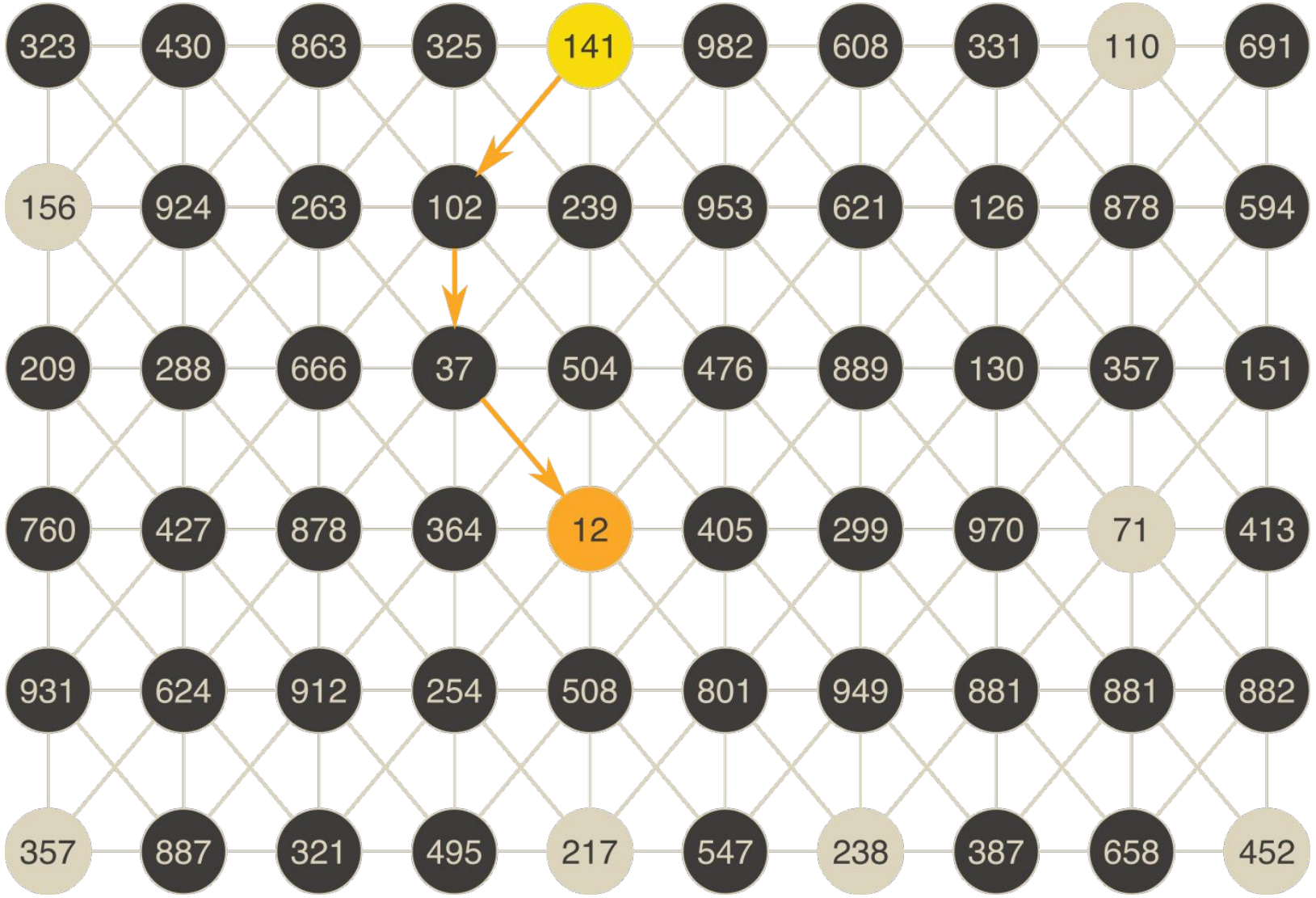
15 óptimos locales

Métodos smoothing

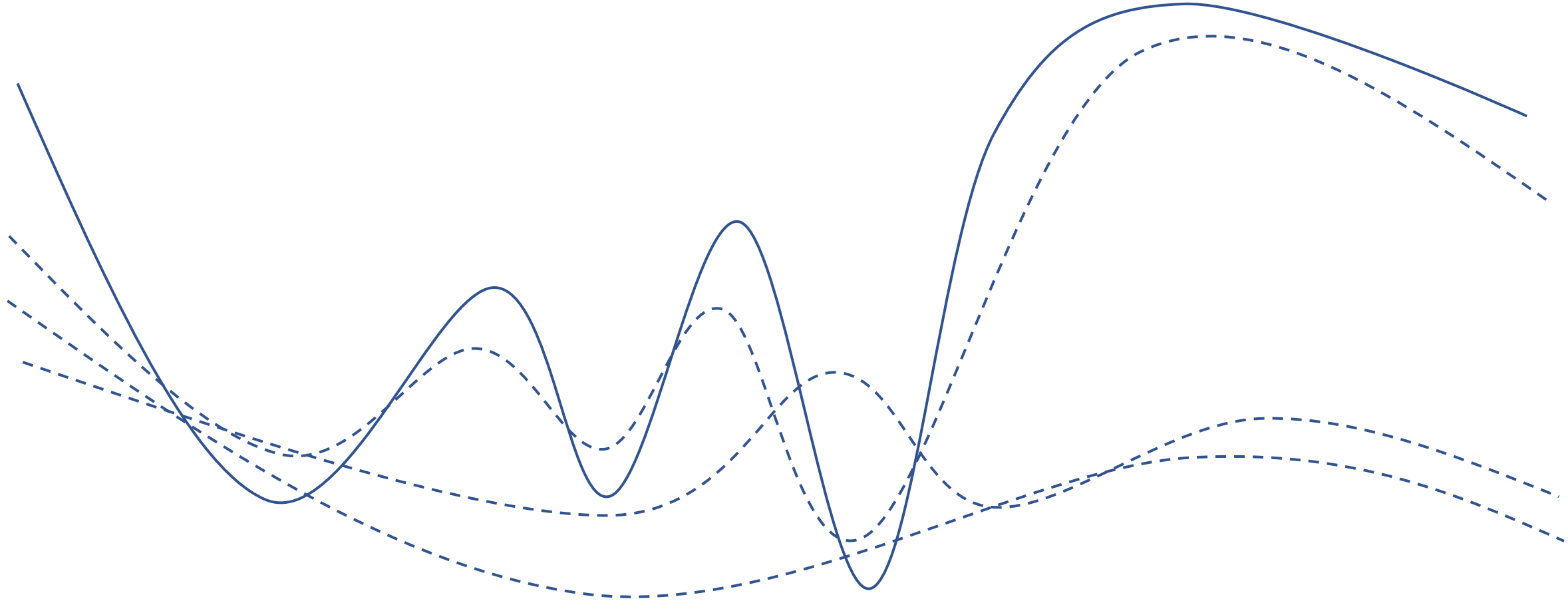


9 óptimos locales

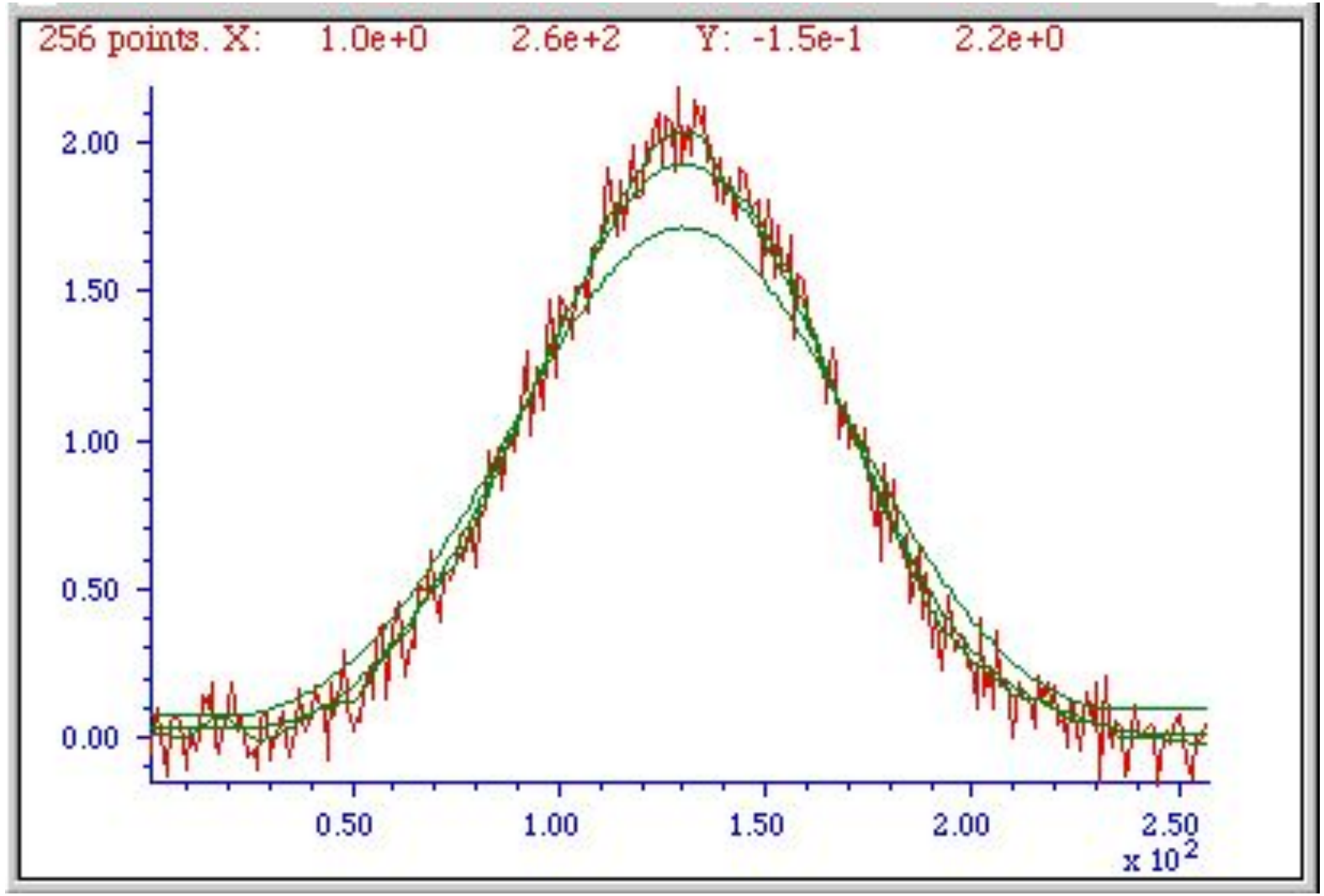
Métodos smoothing



Métodos smoothing



Métodos smoothing



Métodos smoothing

```
Input:

    SMOOTHING (f,a)    – función que modifica la función objetivo
    LOCAL_SEARCH (s,f) – algoritmo heurístico búsqueda local
    UPDATE (a)         – función de actualización de factor



s' = s0
a = a0

WHILE  A > 1 DO
    f'  = SMOOTHING (f,a)
    s'' = LOCAL_SEARCH (s', f')
    a   = UPDATE (a)
END
```

Búsqueda local - 3. Aceptar soluciones peores (a corto plazo)

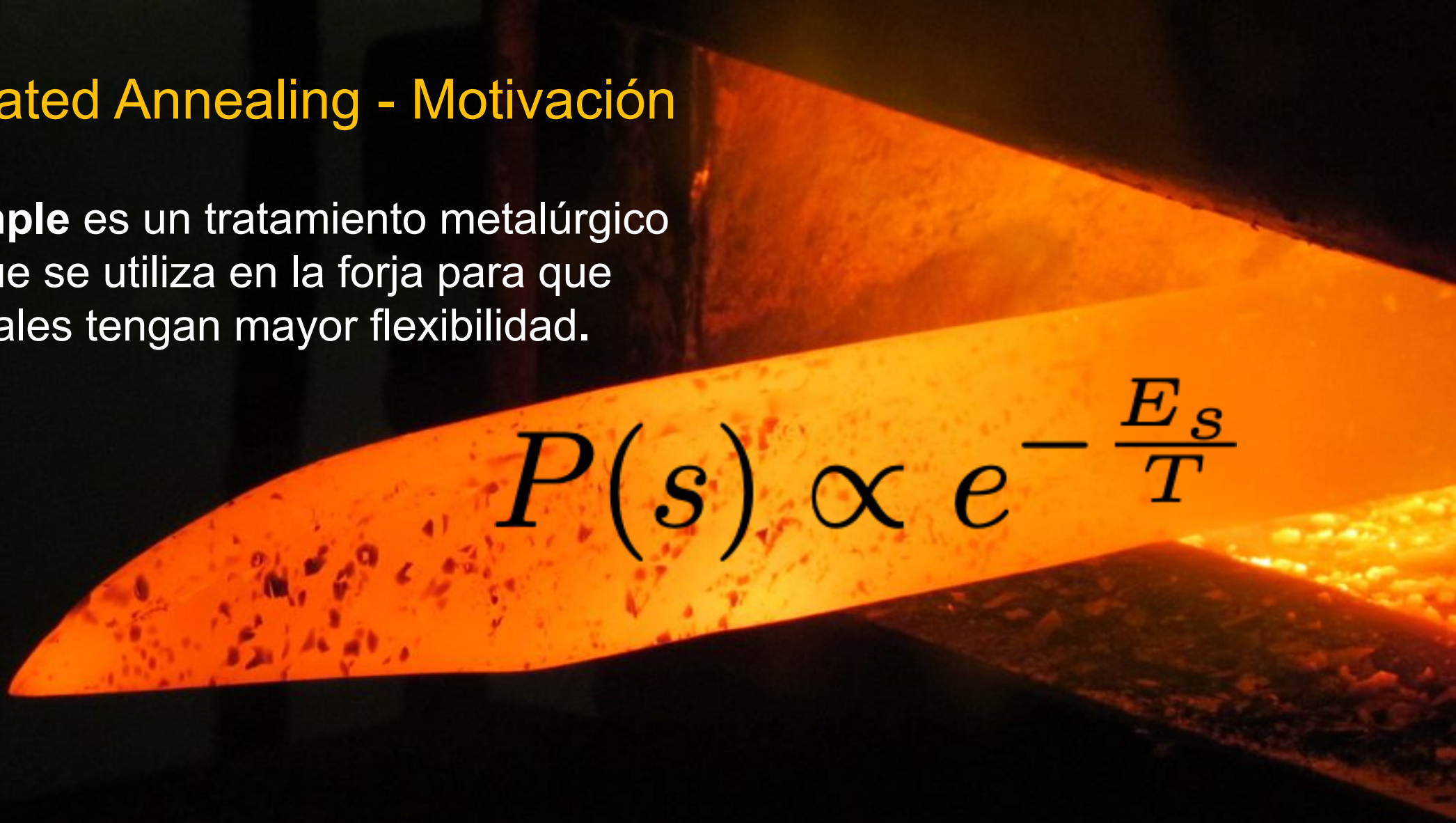
Búsqueda local - 3. Aceptar soluciones peores (a corto plazo)

Métodos más conocidos

- **Simulated annealing (SA)**
  - Algoritmo basado en la temperatura para el dilema de la exploración vs explotación
- **Tabu search (TS)**
  - Buffer o memoria para candidatos prohibidos
- **Algoritmos con energía de exploración (e.g. Algoritmo del demonio)**
  - Explorar soluciones “malas” consume energía de exploración

SA - Simulated Annealing - Motivación

El **destemple** es un tratamiento metalúrgico térmico que se utiliza en la forja para que los materiales tengan mayor flexibilidad.



En este proceso, si la **temperatura** es la adecuada, las moléculas se mueven a su posición **óptima de mínima energía**.

SA - Simulated annealing

Moléculas → Candidatos

Energía → Fitness

$$P(s_i \rightarrow s_j) = \frac{P(s_i)}{P(s_j)} = e^{-\frac{\Delta E}{T}}$$

$$P(s_i \rightarrow s_j) = e^{-\frac{f(s_j) - f(s_i)}{T}}$$

SA - Simulated annealing

Aceptamos transicionar a una solución peor en base a dos factores:

1. Energía / Diferencia entre el fitness
2. Temperatura / Iteraciones del algoritmo

¿ Por qué ? Principalmente dos factores

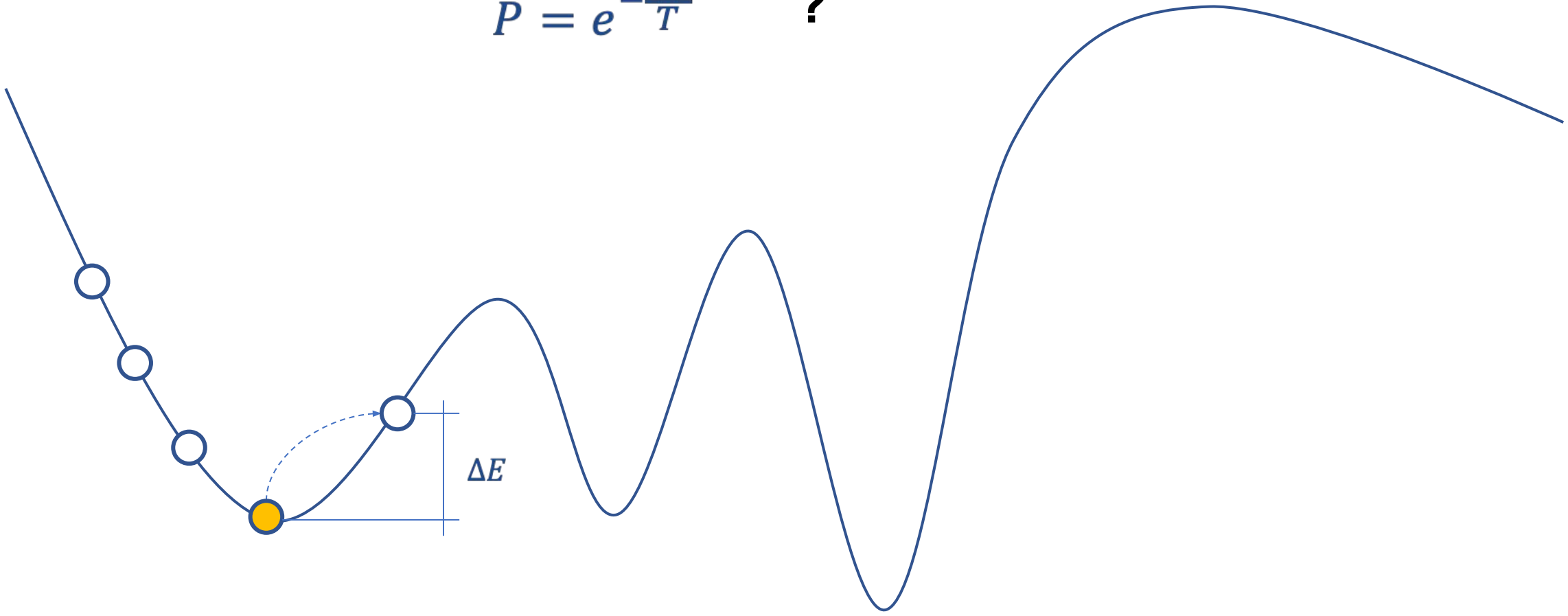
1. Escapar de óptimos locales
2. Regular la exploración y la explotación

Comportamiento

- Si el sucesor tiene un **fitness mayor**, **aceptamos** la solución
- Si el sucesor tiene un **fitness menor**, **aceptamos o no** la solución según un criterio

SA - Simulated annealing

$$P = e^{-\frac{\Delta E}{T}}$$?



SA - Simulated annealing

```
Make the initialNode with the initial problem state
currentNode = initialNode
i = 1
T = temperatureFunction(i)

WHILE T != 0 AND i < MAX_VALUE DO
    successors = EXPAND currentNode's state
    nextNode = randomly selected successor among Successors
    ΔE = f(currentNode) - f(nextNode) (minimization problem)
    IF ΔE > 0 THEN
        CurrentNode = NextNode
    ELSE
        currentNode = NextNode only with probability e^(ΔE/T)
    T = temperatureFunction(++i)

RETURN currentNode's state
```

SA - Simulated annealing

Valores altos para $|\Delta E / T|$

- La tasa de aceptación es baja
- Exploration vs **exploitation**

Valores bajos para $|\Delta E / T|$

- La tasa de aceptación es alta
- **Exploration** vs exploitation

SA - Simulated annealing

La tasa de aceptación basada en $|\Delta E / T|$

- Es pequeña si **ΔE** es grande
- Es pequeña si la **Temperatura** es baja

Nota: Recuerda que si **ΔE** > 0 entonces la solución se acepta directamente

ΔE = fitness(current) - fitness(new)

Ejercicio en clase - completa esta tabla !

ΔE

T	1	10	50	100	150	250	500
1							
10							
50							
100							
150							
250							
500							

$$P = e^{-\frac{\Delta E}{T}}$$

```
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})
```

Funciones para la temperatura

Existen múltiples funciones de temperatura…

¿ Se te ocurre alguna opción ?

TS - Tabu Search

Se permiten soluciones malas…

… pero las **registramos en un buffer**

TS - Tabu Search

Greedy Hill Climbing

Elegimos la mejor solución existente en el vecindario para transicionar

Búsqueda tabú

Elegimos ~~la mejor~~ solución ~~existente en el vecindario~~ para transicionar

Es posible repetir candidatos !

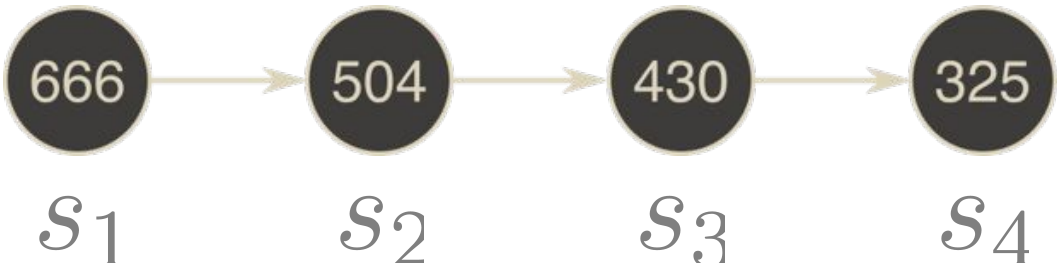
TS - Tabu Search

Buffer Tabú

Registramos candidatos visitados

Memoria

s_4
s_3
s_2



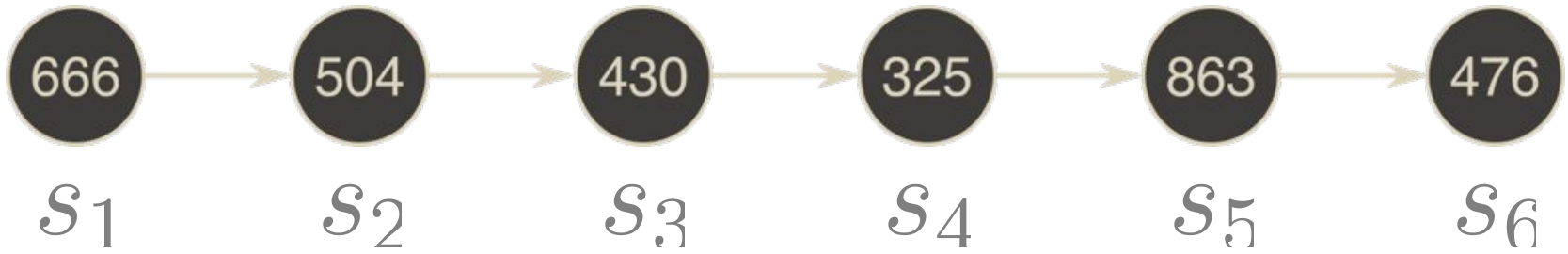
TS - Tabu Search

Buffer Tabú

Registramos candidatos visitados

Memoria

s_6
s_5
s_4



TS - Tabu Search

Buffer Tabú

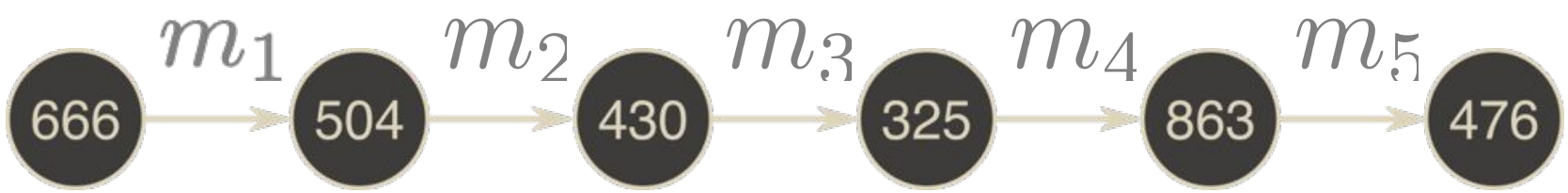
Registramos candidatos visitados

Registramos las acciones

Memoria

m_5
m_4
m_3

Se utilizan memorias de capacidades diferentes
Intensificar vs. diversificar



TS - Tabu Search

Aspiration criteria

Permitir soluciones o movimientos tabú
Con el único objetivo de mejorar el fitness

Intensificar (memoria medio plazo)

Reutilizar los factores **más frecuentes** de las soluciones buenas

Diversificar (memoria largo plazo)

Reutilizar los factores **menos frecuentes** de todas las soluciones

TS - Tabu Search

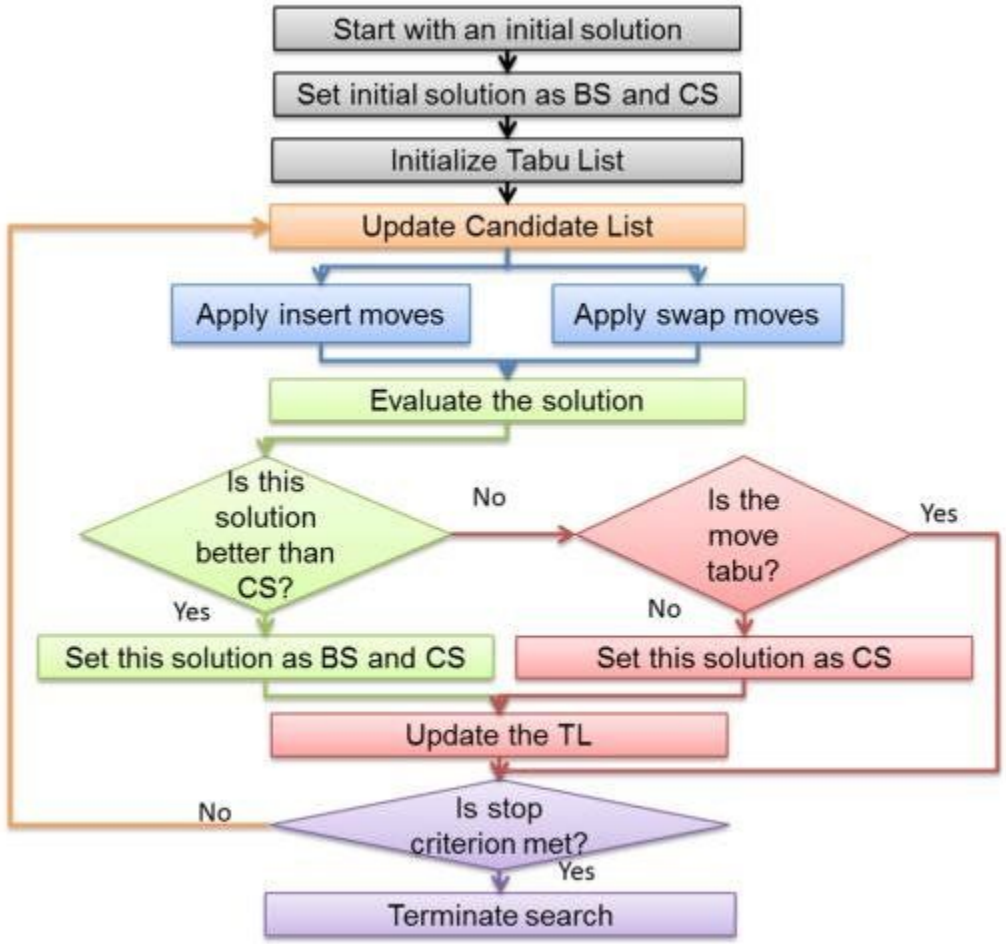
```
sBest ← s0
bestCandidate ← s0
tabuList ← []
tabuList.push(s0)
while (not stoppingCondition())
    sNeighborhood ← getNeighbors(bestCandidate)
    bestCandidate ← sNeighborhood[0]
    for (sCandidate in sNeighborhood)
        if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
            bestCandidate ← sCandidate
        end
    end
    if (fitness(bestCandidate) > fitness(sBest))
        sBest ← bestCandidate
    end
    tabuList.push(bestCandidate)
    if (tabuList.size > maxTabuSize)
        tabuList.removeFirst()
    end
end
return sBest
```

https://en.wikipedia.org/wiki/Tabu_search

TS - Tabu Search

1. Initialisation
2. Intensification
3. Selection
4. Diversification
5. Escape
6. Conclusion

BS: Best solution
CS: Curent solution
TL: Tabu list



https://www.sciencedirect.com/science/article/pii/S2405896317317354?via%3Dihub

TS - Tabu Search - Ejercicio

Tabu List

Short term memory

Aspiration criteria

Actions stored

https://en.wikipedia.org/wiki/Tabu_search

Intensification

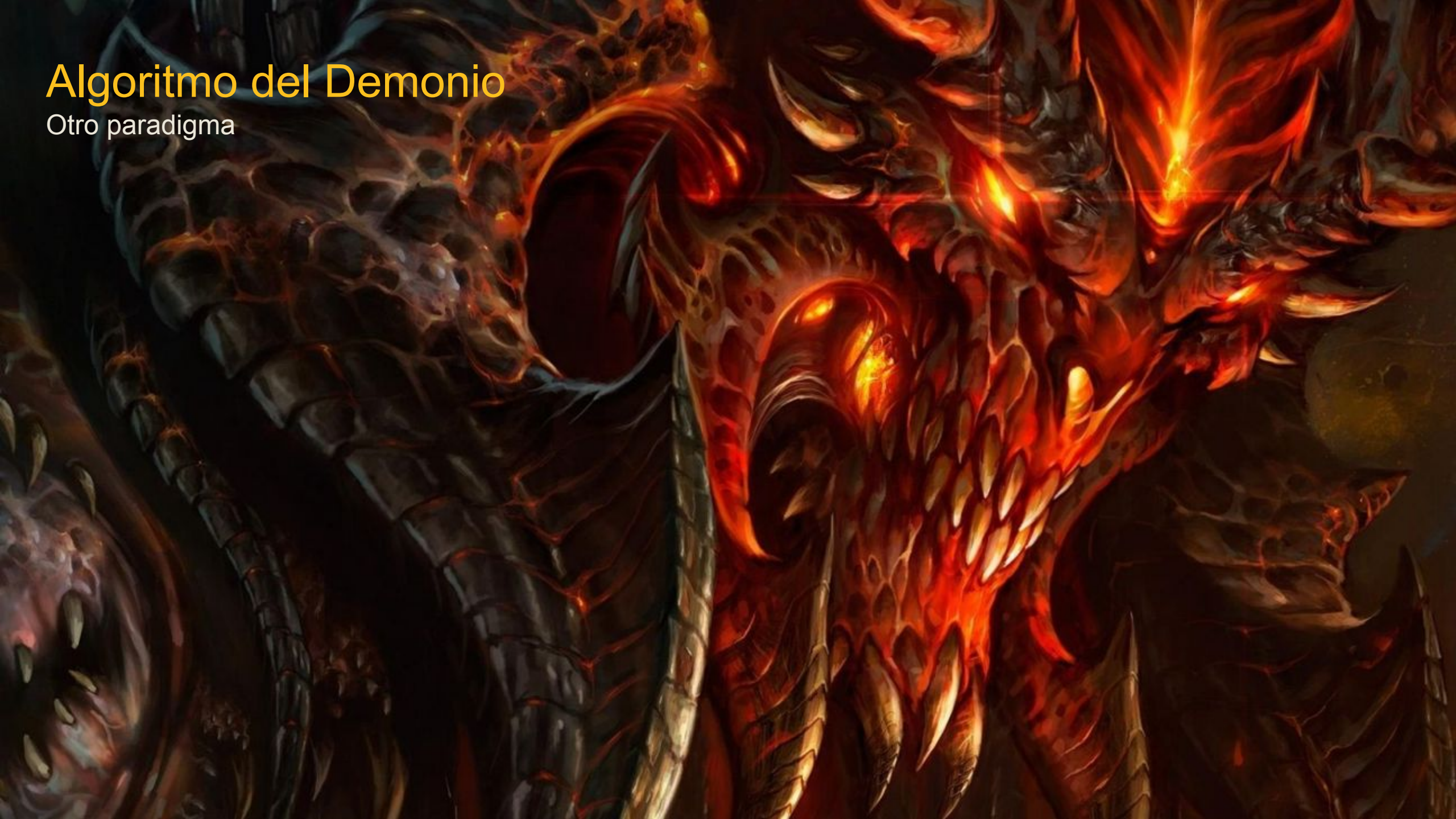
Medium term memory

Diversification

Long term memory

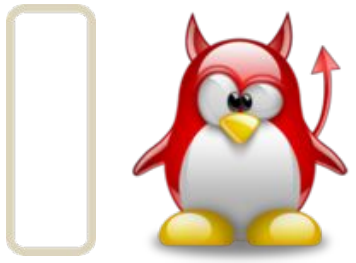
Algoritmo del Demonio

Otro paradigma



Algoritmo del demonio

Energía



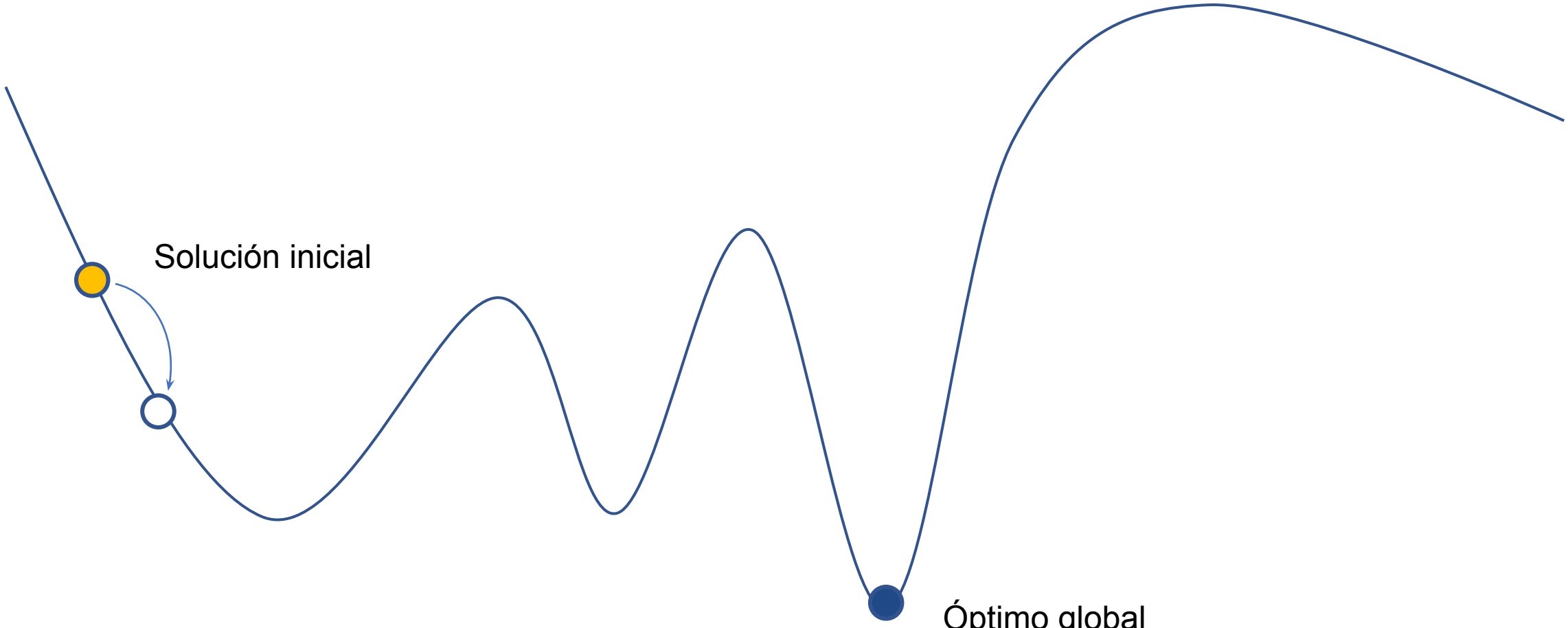
Se inicializa en 0

Solución inicial

Óptimo global

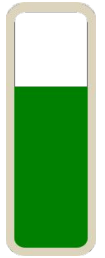
Algoritmo del demonio

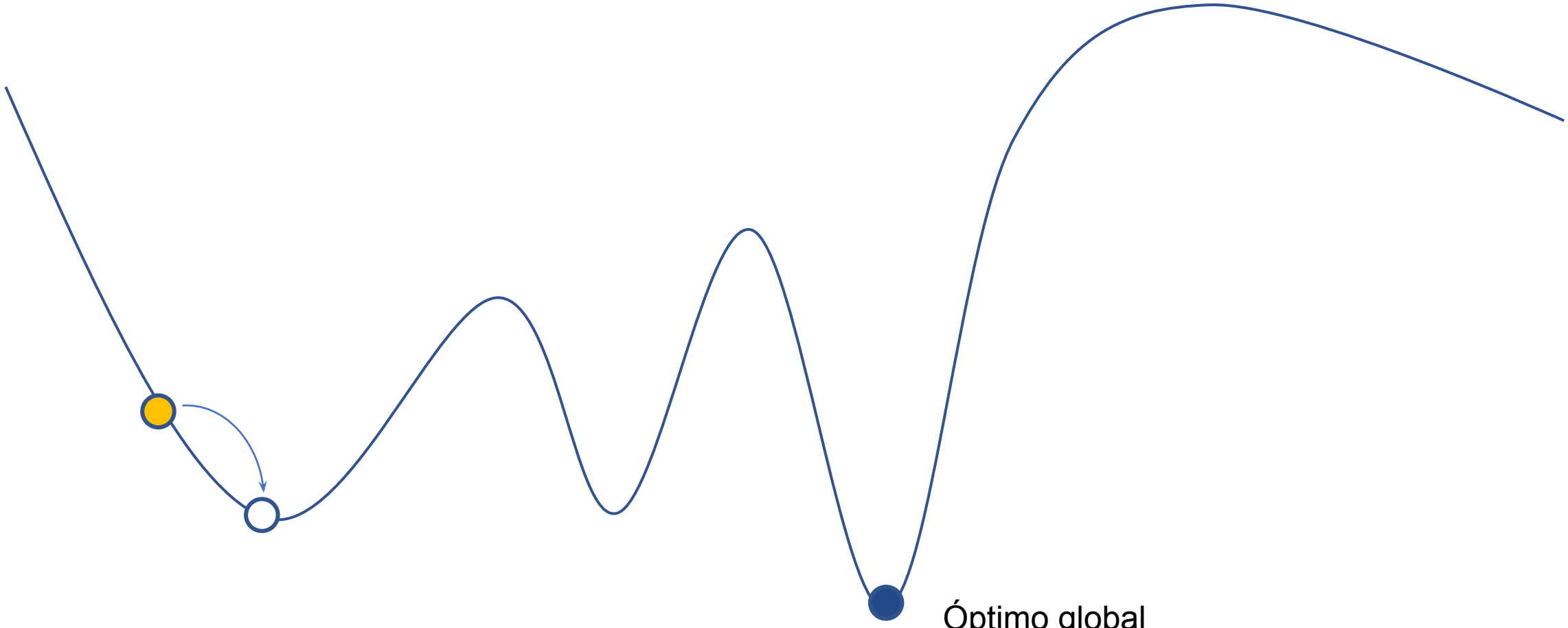
Energía



Algoritmo del demonio

Energía





Algoritmo del demonio

Deabruaren Energia