


# Búsqueda local iterada

- 
- No es un reinicio desde ~~cero~~, sino que partimos de la solución a la que hayamos llegado (un mín local)
  - Hacemos perturbaciones (modificaciones) a esas soluciones para cambiarlas un poco.

**Recocido simulado** nos permite tomar soluciones que peores  
(bajo ciertas restricciones)

**En búsqueda tabú** acepta peores soluciones pero llevando una memoria de las últimas soluciones que se hayan intercambiado

Una vez que llegamos a un óptimo local  
-> Una solución se logro mejorar -> “buena”

- A partir de esta sol podríamos buscar otra mejor.
- Alterarla, lleva a una sol peor pero con el fin de encontrar una mejor para que nos saque de esa zona de búsqueda





- Al llegar al óptimo local vamos a modificar esa sol de alguna forma, esta forma es utilizando un método de perturbación.
- Esta sol ya esta "optimizada", aunque empeore al modificarla (perturbarla) no empeora tanto como volver a elegir una sol aleatoria (reiniciar la búsqueda).
- Además, no se pierde la optimización ya realizada, ni los recursos empleados en ella.

- Iterada: cada vez que caemos en un óptimo local buscamos perturbar la sol actual para poder escapar de ese óptimo.
- La perturbación tiene que ser "adecuada". → Esq. rep. sol.  
→ Esq. cod.
- La perturbación depende del esquema de representación:
  - Vectores binarios: tomar un porcentaje de las variables. Si tenemos un vector de tamaño 50 tomamos 10 y las intercambiamos
  - En permutaciones: si intercambiaba 2, ahora tomo 5 y los intercambio entre ellos

Necesitamos que:

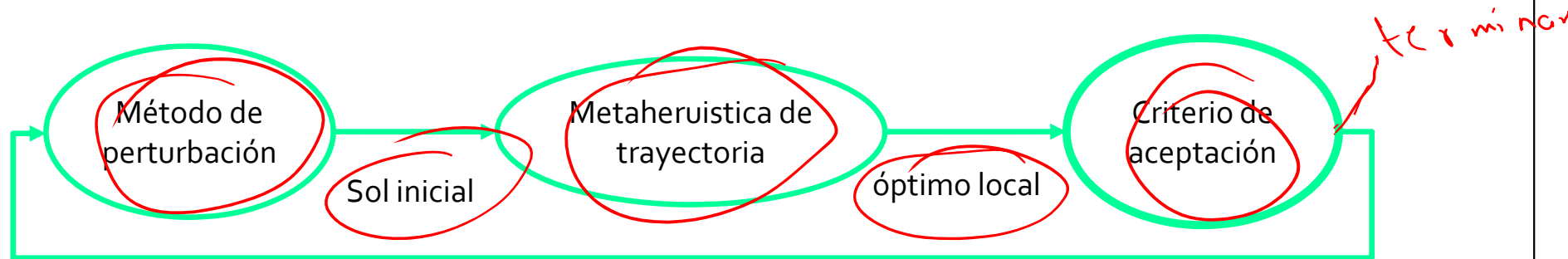
- La perturbación sea lo suficientemente fuerte para salir del espacio de búsqueda...
- Pero si es muy fuerte... nos vamos a un reinicio total, (búsqueda aleatoria)
- La misma perturbación puede llevar a caer en el mismo óptimo local

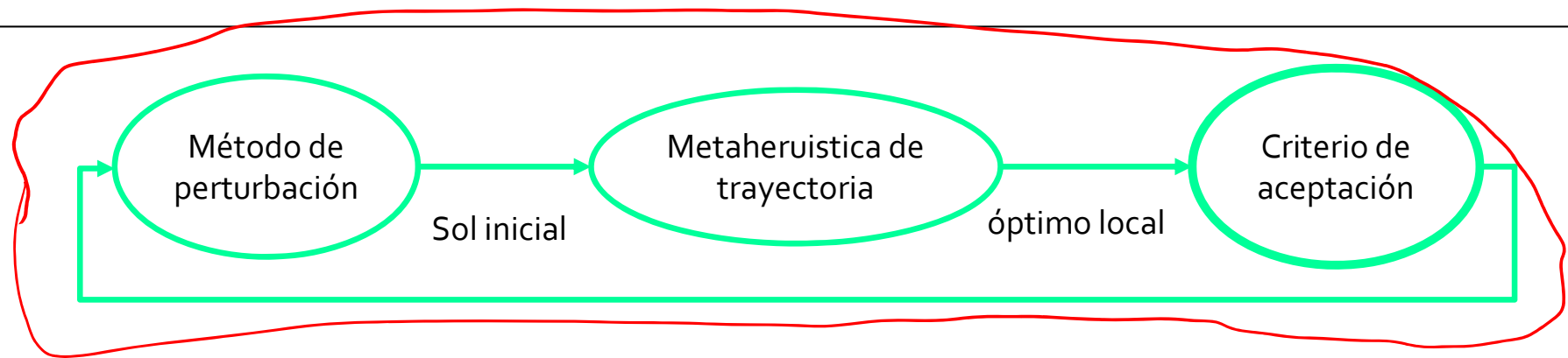
peor e s

En algún punto será necesario evaluar si o si a todos los vecinos de la solución  
(para poder garantizar que se trata del óptimo local)

## Idea general:

- Recibir una sol aleatoria (o generar una aleatoria) *Generador de sol aleatorios*
- Usar búsqueda local (hill climbing)
- Se puede combinar la estrategia de búsqueda iterada con cualquier búsqueda local
- ~~Su método de recocido simulado recibe como parámetro una solución inicial~~
- ~~(así podemos pasar la sol que encontramos con otro algoritmo como parámetro a SA)~~
- Mientras no se cumpla algún criterio de paro (no iteraciones, no evaluaciones, tiempo de ejecución) aplicar método de perturbación





- El método de perturbación tomará en cuenta el proceso de búsqueda de una forma similar a una memoria (si en la pasada se modifico ciertas variables ahora otras)
- Aplicar búsqueda local
- Actualizar a la mejor sol encontrada *Me < actual?*
- Criterio de aceptación:
  - ¿Quien es la nueva sol?
  - Es posible llegar a una sol peor: en vez de iterar a partir de esta sol mejor, retomar a partir de la que estaba trabajando y no de la nueva que acabo de construir (una de las que están “almacenadas” en mi historial)
- Perturbación y criterio de aceptación
  - > Eso posiblemente me lleve a un local diferente

## Componentes

TSP

**Solución Inicial:** Se genera una solución aleatoria para empezar la búsqueda local iterada.  
Utilizando...

**Estrategia de búsqueda local:** Se aplica la estrategia de búsqueda local para refinar la solución actual y así mejorar la distancia total del recorrido (mínimo local).  
*hill climbing*

**Perturbación:** Se realiza intercambiando  $n$  ciudades aleatorias en la solución actual.  
*¿cuál? ¿xq?*

**Evaluación:** La func de evaluación toma una solución (una permutación de ciudades) y la matriz de distancias entre ciudades. Luego, itera sobre las ciudades en la solución, sumando las distancias entre cada par de ciudades consecutivas. La suma total de las distancias de la solución representa la evaluación (longitud total del recorrido).  
*valor fijo → 30° del tamaño*

**Criterio de termino:** Se repite un número fijo de veces, determinado por el parámetro iteraciones.  
Cuando se alcanza el límite de iteraciones, el algoritmo termina y devuelve la mejor solución encontrada hasta ese momento.



1 → Func BusquedaLocalIterada(nCiudades, matrizDistancias, iteraciones):  
2   → mejorSolGlobal = GeneraSolAleatoria(nCiudades) → Sol. inicial  
3   mejorDistanciaGlobal = EvaluarSol(mejorSolGlobal, matrizDistancias)  
4  
5   Para cada \_ en rango(iteraciones):  
6   → solModificada = Perturbacion(mejorSolGlobal)  
7   mejorSolLocal, mejorDistanciaLocal = BusquedaLocal(solModificada, matrizDistancias)  
8  
9   Si mejorDistanciaLocal < mejorDistanciaGlobal:  
10    mejorSolGlobal = Copiar(mejorSolLocal)  
11    mejorDistanciaGlobal = mejorDistanciaLocal  
12  
13   Devolver mejorSolGlobal, mejorDistanciaGlobal  
14 Fin Func  
15

termino