UNIVERSIDAD DE GRANADA

Metaheurísticas

Temario de la asignatura

Ismael Sallami Moreno

Recursos Ingeniería Informática y Ade

Licencia

Metaheurísticas

Ismael Sallami Moreno

https://elblogdeismael.github.io

Índice general

1 Introducción a las metaheurísticas **9**

1.1 El problema de optimización **9**

1.2 Complejidad de los problemas **9**

1.3 Algoritmos aproximados **10**

1.4 Concepto de metaheurística **11**

1.5 Clasificación **12**

2 Optimización y búsqueda en inteligencia artificial **15**

2.1 Modelos de optimización en inteligencia artificial **15**

2.2 Manejo de restricciones **15**

2.3 Búsqueda por entornos **16**

2.4 Trayectorias frente a poblaciones **18**

3 Metaheurísticas basadas en poblaciones **19**

3.1 Concepto y elementos **19**

3.2 Algoritmos genéticos **20**

3.3 Programación genética **22**

3.4 Evolución diferencial **22**

3.5 Estrategias de evolución **23**

3.6 Aplicación a problemas **23**

4 Algoritmos meméticos **25**

4.1 Hibridación de metaheurísticas **25**

4.2 Algoritmos meméticos **25**

4.3 Las decisiones de diseño **26**

4.4 Control de la diversidad **27**

4.5 Un caso completo **28**

5 Metaheurísticas basadas en trayectorias 29

5.1 El inconveniente de la búsqueda local 29

5.2 Enfriamiento simulado 29

5.3 Búsqueda tabú 31

5.4 Trayectorias múltiples 32

5.5 Búsqueda por entornos variables 33

5.6 Comparación 34

6 Metaheurísticas basadas en adaptación social 35

6.1 Introducción a la adaptación social 35

6.2 Cooperación de agentes en problemas de optimización 35

6.3 Algoritmos basados en colonias de hormigas 36

6.4 Algoritmos basados en nubes de partículas 37

6.5 Aplicación a problemas 38

7 Aspectos avanzados en metaheurísticas 41

7.1 Diversidad frente a convergencia 41

7.2 Modelos evolutivos con equilibrio explícito 42

7.3 Múltiples soluciones: nichos 43

7.4 Nuevos algoritmos bioinspirados 43

7.5 Metaheurísticas paralelas 44

8 Aprendizaje evolutivo: problemas y modelos 45

8.1 El aprendizaje como problema de optimización 45

8.2 Cuándo compensa una metaheurística 45

8.3 Modelos de aprendizaje basados en metaheurísticas 46

8.4 Aprendizaje evolutivo de reglas 47

8.5 Aprendizaje evolutivo de parámetros y modelos 47

8.6 Un experimento bien planteado 48

9 Prácticas y seminarios 49

9.1 Los dos problemas del banco de pruebas 49

9.2 P1 · Técnicas de búsqueda local 50

9.3 P2 · Poblaciones: genéticos y meméticos 50

9.4 P3 · Trayectorias: enfriamiento, multiarranque, GRASP e ILS 51

9.5 Prácticas de pizarra 51

9.6 Seminarios 51

Capítulo 1

Introducción a las metaheurísticas

Tema 1 del programa. Qué hace que un problema de optimización sea intratable, qué se puede ofrecer cuando el óptimo está fuera de alcance y qué es exactamente una metaheurística.

1.1 El problema de optimización

Un problema de optimización queda descrito por tres piezas:

Pieza	Notación	Qué es
Espacio de búsqueda	S	el conjunto de todas las soluciones posibles
Función objetivo	$f : S \to \mathbb{R}$	lo que mide la calidad de una solución
Restricciones	$s \in F \subseteq S$	qué soluciones son admisibles

Resolverlo es encontrar

$$s^* = \arg\min_{s \in F} f(s)$$

para un problema de minimización, o el $\arg\max$ si se maximiza. Los dos son el mismo problema: minimizar f equivale a maximizar $-f$, así que el temario habla de minimización y lo demás se traduce cambiando el signo.

Dos distinciones que condicionan todo lo que sigue:

- **Combinatorio o continuo.** En el combinatorio S es finito —permutaciones, subconjuntos, asignaciones— y en el continuo es un subconjunto de $\mathbb{R}^n$. El primero se recorre; el segundo se muestrea.
- **Óptimo global u óptimo local.** s^* es global si $f(s^*) \leq f(s)$ para todo $s \in F$, y local si la desigualdad solo vale dentro de un entorno de s^*. Casi todo lo que este temario construye existe para salir de los locales.

1.2 Complejidad de los problemas

La razón por la que hacen falta las metaheurísticas es que la mayoría de los problemas de optimización interesantes no admiten un algoritmo exacto eficiente.

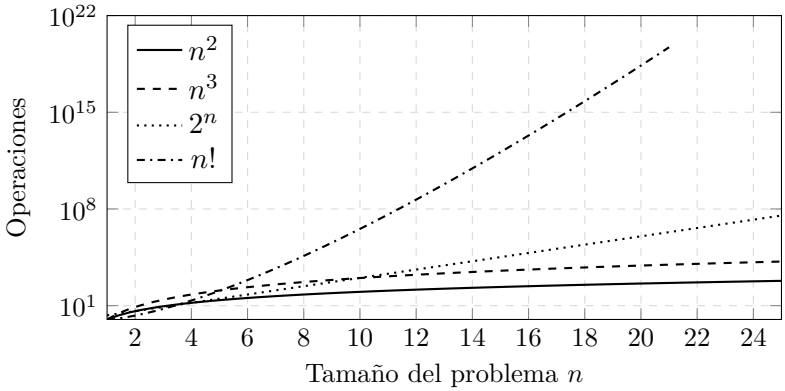
Clase	Qué significa
P	resolubles en tiempo polinómico por una máquina determinista
NP	una solución candidata se **verifica** en tiempo polinómico
NP-completo	está en NP y todo problema de NP se reduce a él en tiempo polinómico
NP-duro	todo problema de NP se reduce a él, pero puede no estar en NP

Un problema de optimización cuya versión de decisión es NP-completa es NP-duro. Y para un NP-duro no se conoce ningún algoritmo exacto de coste polinómico; si alguien encontrara uno, tendría P = NP.

El caso de referencia es el **viajante de comercio**: dadas n ciudades y las distancias entre ellas, encontrar el recorrido más corto que las visita todas una vez. El espacio de búsqueda tiene $(n-1)!/2$ recorridos distintos.

n	Recorridos	Tiempo a 10^9 por segundo
10	$1{,}8 \times 10^5$	instantáneo
20	$6{,}1 \times 10^{16}$	2 años
30	$4{,}4 \times 10^{30}$	$1{,}4 \times 10^{14}$ años

Treinta ciudades es un problema pequeño, y la enumeración exhaustiva ya no termina. La ramificación y poda recorta el árbol, pero su peor caso sigue siendo exponencial y basta una instancia adversa para alcanzarlo.



La escala del eje vertical es logarítmica, así que una recta es crecimiento exponencial. Ninguna mejora de la máquina cambia la forma de esas dos últimas curvas: duplicar la velocidad del procesador añade **una** ciudad al tamaño abordable por enumeración.

1.3 Algoritmos aproximados

Cuando el óptimo no se puede garantizar, se renuncia a él a cambio de tiempo. Las opciones no son equivalentes:

Tipo	Qué garantiza	Qué cuesta
Exacto	el óptimo	tiempo exponencial en el peor caso
Aproximación	una cota sobre el error, del tipo $f(s) \leq \rho f(s^*)$	polinómico, pero solo existe para algunos problemas
Heurística	nada demostrable	rápido, y depende del problema
Metaheurística	nada demostrable	ajustable: más tiempo, mejor solución

Un **algoritmo de aproximación** es el caso cómodo: para el viajante con distancias métricas, el algoritmo de Christofides garantiza una solución a lo sumo un 50 % peor que la óptima. El problema es que esas garantías no existen para la mayoría de los problemas reales, y cuando existen la cota suele ser demasiado holgada para servir de algo.

Una **heurística** es una regla razonable, atada a un problema concreto: en el viajante, «ir siempre a la ciudad no visitada más cercana». Se programa en veinte líneas, corre en un instante y no ofrece ninguna garantía; en instancias adversas puede quedarse muy lejos.

La diferencia práctica entre las dos últimas filas está en **cómo se comportan al darles más tiempo**. Una heurística constructiva termina y devuelve lo que ha construido: correrla el doble de tiempo no mejora nada. Una metaheurística usa el tiempo extra para seguir explorando, y su curva de calidad frente a tiempo es creciente. Por eso la pregunta que ordena la asignatura no es «¿cuál es el mejor algoritmo?», sino «¿qué calidad se alcanza con el presupuesto de tiempo que hay?».

1.4 Concepto de metaheurística

Una metaheurística es una **estrategia de alto nivel** que guía a otras heurísticas subordinadas en la exploración del espacio de búsqueda. El prefijo *meta* es justamente eso: no resuelve el problema, organiza a quien lo resuelve.

Sus rasgos:

- **Independiente del problema.** El esquema es el mismo para el viajante que para el entrenamiento de una red neuronal. Lo que cambia son las piezas que se le enchufan: representación, función objetivo y operadores.
- **No exacta.** No garantiza el óptimo ni una cota sobre la distancia a él.
- **Estocástica**, en casi todos los casos. Dos ejecuciones con la misma entrada dan resultados distintos, y por eso los resultados se informan como media y desviación típica sobre varias semillas, nunca como un número suelto.
- **Guiada por un compromiso** entre explorar el espacio y explotar lo que ya se ha encontrado.

1.4.1 Diversificación y intensificación

Es el eje sobre el que se explica toda la asignatura.

	Diversificación	Intensificación
Qué hace	visitar regiones nuevas del espacio	refinar lo que ya se tiene

	Diversificación	Intensificación
Si sobra	la búsqueda se vuelve aleatoria y no converge	se queda atrapada en el primer óptimo local
Ejemplo	reinicio desde una solución aleatoria	búsqueda local sobre la mejor solución

Todo lo que viene después son formas distintas de repartir el presupuesto entre las dos. El enfriamiento simulado lo hace con una temperatura que decrece; la búsqueda tabú, con una memoria que prohíbe deshacer lo recién hecho; los algoritmos genéticos, con la mutación de un lado y la selección del otro.

1.4.2 Componentes que hay que fijar

Antes de aplicar cualquier metaheurística a un problema hay que decidir cuatro cosas, y el temario vuelve a ellas en cada tema:

1. **Representación.** Cómo se codifica una solución: vector binario, permutación, vector real, árbol. Determina qué operadores son posibles.
2. **Función objetivo.** Cómo se evalúa. Su coste manda en el diseño: si evaluar es caro, el número de evaluaciones es el presupuesto real.
3. **Operadores.** Cómo se genera una solución nueva a partir de las que hay: vecindario, cruce, mutación.
4. **Criterio de parada.** Número de evaluaciones, tiempo, o estancamiento durante un número de iteraciones.

De las cuatro, la representación es la que más decisiones arrastra. Codificar un recorrido del viajante como permutación hace natural el intercambio de dos ciudades y absurdo el cruce en un punto, que produciría recorridos con ciudades repetidas.

1.5 Clasificación

Las metaheurísticas del programa se ordenan por cuántas soluciones mantienen vivas a la vez:

Familia	Soluciones activas	Temas	Ejemplos
Basadas en trayectorias	una	5	enfriamiento simulado, búsqueda tabú, GRASP, ILS
Basadas en poblaciones	muchas	3, 6	algoritmos genéticos, evolución diferencial, hormigas, nubes de partículas
Híbridas	ambas	4	algoritmos meméticos

Y por otros dos criterios que reaparecen:

- **Con memoria o sin ella.** La búsqueda tabú recuerda de forma explícita por dónde ha pasado; el enfriamiento simulado no recuerda nada.
- **Inspiradas en la naturaleza o no.** Los algoritmos genéticos, las hormigas y las nubes de partículas toman su esquema de un fenómeno natural. La inspiración ayuda a explicar el algoritmo y **no** es un argumento a favor de su rendimiento: lo que decide es la comparación experimental.

El programa sigue este orden: primero poblaciones (tema 3), luego su hibridación con búsqueda local (tema 4), después trayectorias (tema 5) y por último la adaptación social (tema 6). Los temas 7 y 8 son los aspectos avanzados y una aplicación completa.

Los tratamientos generales de este marco están en Talbi, 2009, Chopard y Tomassini, 2018 y Du y Swamy, 2016, y el catálogo de problemas y aplicaciones en Pardalos y Resende, 2002.

Capítulo 2

Optimización y búsqueda en inteligencia artificial

Tema 2 del programa. Dónde encajan las metaheurísticas dentro de los modelos de optimización de la inteligencia artificial, qué es un entorno y qué separa a las técnicas de trayectoria de las de población.

2.1 Modelos de optimización en inteligencia artificial

La inteligencia artificial trata la optimización de tres formas distintas, y conviene no confundirlas porque resuelven problemas diferentes.

Modelo	Qué se busca	Ejemplo
Búsqueda en espacio de estados	un **camino** desde el estado inicial a uno meta	A*, Dijkstra, el 8-puzle
Satisfacción de restricciones	una asignación que cumpla **todas** las restricciones	coloreado de grafos, sudoku
Optimización	la solución de **mejor valor**	viajante, mochila, asignación

La diferencia con la búsqueda en espacio de estados es la que más cuesta: allí importa cómo se llega, porque la solución **es** la secuencia de operadores. Aquí el camino es desechable, y lo único que se devuelve es la solución final. Eso permite saltar de una solución a otra sin ninguna relación entre ellas, algo que A* no puede hacer.

La satisfacción de restricciones es el caso límite en el que la función objetivo es binaria. Se puede tratar como optimización tomando como objetivo el número de restricciones incumplidas y buscando el mínimo, que es cero. Es una técnica útil: convierte un problema de «sí o no» en uno con gradiente, y así una metaheurística puede acercarse por pasos.

2.2 Manejo de restricciones

Casi ningún problema real es libre: el espacio admisible F es un subconjunto propio de S. Cuatro maneras de tratarlo, de más a menos deseable:

Estrategia	Cómo funciona	Cuándo
Representación cerrada	codificar de forma que toda solución sea válida	cuando es posible; siempre preferible
Reparación	corregir la solución inválida tras generarla	cuando la corrección es barata
Penalización	sumar a f un término proporcional a la violación	restricciones blandas
Rechazo	descartar y volver a generar	solo si las inválidas son raras

La **representación cerrada** es la mejor y la más olvidada. En el viajante, codificar como permutación garantiza que toda solución es un recorrido válido; no hace falta comprobar nada. En la mochila, un vector binario no la cierra —se puede exceder la capacidad— y ahí toca reparar o penalizar.

La **penalización** transforma el problema restringido en uno libre:

$$f'(s) = f(s) + \sum_i \lambda_i \,\text{máx}(0, g_i(s))$$

donde $g_i(s) > 0$ indica que la restricción i se incumple. El ajuste de las λ_i decide el comportamiento: si son pequeñas, la búsqueda se instala en la zona no admisible porque le sale rentable; si son grandes, no puede cruzarla y pierde los atajos que pasan por ella. Una penalización creciente con el tiempo —permisiva al principio, severa al final— evita los dos extremos.

El **rechazo** es el peor de los cuatro y el más tentador de programar. Si el espacio admisible es una fracción pequeña de S, casi todo el presupuesto se gasta generando soluciones que se tiran.

2.3 Búsqueda por entornos

Es el mecanismo sobre el que se construye todo el tema 5, y aparece dentro de los algoritmos meméticos del tema 4.

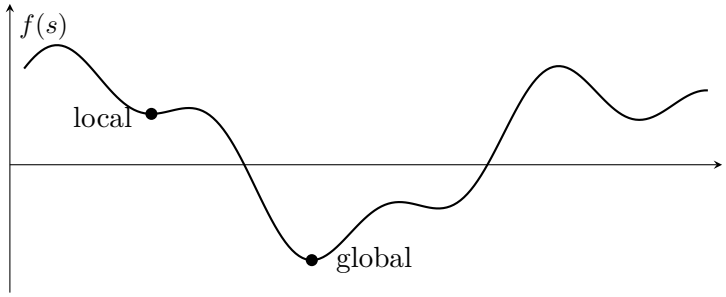
Un **entorno** es una función $N : S \to 2^S$ que asigna a cada solución el conjunto de las que se alcanzan con un cambio elemental. Ese cambio se llama **movimiento**, y el entorno queda definido por él.

Representación	Movimiento típico	Tamaño de $N(s)$
Binaria, n bits	invertir un bit	n
Permutación de n	intercambiar dos posiciones	$n(n-1)/2$
Permutación de n	invertir un segmento (2-opt)	$n(n-1)/2$
Real, n dimensiones	sumar ruido gaussiano	infinito

La elección del movimiento **es** el diseño del algoritmo. Dos entornos distintos sobre el mismo problema dan paisajes distintos, con óptimos locales distintos, y uno puede ser mucho mejor que el otro. En el viajante, el 2-opt —que invierte un tramo del recorrido— funciona mejor que el intercambio de dos ciudades porque elimina los cruces del recorrido, que es donde está el desperdicio.

2.3.1 El paisaje de la función objetivo

Con un entorno fijado, el espacio de búsqueda deja de ser un conjunto y pasa a ser un **paisaje**: soluciones vecinas, cada una a su altura $f(s)$.



Espacio de búsqueda

Tres propiedades del paisaje deciden qué técnica conviene:

- **Rugosidad.** Cuántos óptimos locales hay y cómo de profundos son. Un paisaje liso se resuelve con búsqueda local; uno muy rugoso necesita mecanismos de escape.
- **Correlación entre vecinos.** Si soluciones parecidas tienen valores parecidos, la información local sirve para guiar. Si no, la búsqueda equivale a un muestreo aleatorio y ninguna metaheurística ayuda.
- **Distribución de los óptimos.** Si los buenos óptimos locales están agrupados —el «gran valle»— compensa intensificar cerca de los mejores encontrados. Es la hipótesis sobre la que descansan GRASP y los algoritmos meméticos.

2.3.2 Búsqueda local

El algoritmo elemental: desde una solución, moverse a un vecino mejor mientras lo haya.

```
funcion busqueda_local(s):
    repetir:
        s' = elegir_de(N(s))
        si f(s') < f(s):
            s = s'
        si no:
            devolver s
```

Dos formas de elegir, con consecuencias distintas:

Estrategia	Qué hace	Coste por paso	Comportamiento
El mejor	evalúa todo $N(s)$ y toma el mínimo	$\|N(s)\|$	pasos largos, pocos
El primero	toma el primer vecino que mejora	variable, menor en media	pasos cortos, muchos

La segunda suele ganar cuando el entorno es grande, porque no paga la exploración completa. Y con ella el **orden de exploración importa**: recorrer siempre el entorno en el mismo orden sesga la búsqueda, así que se aleatoriza.

La búsqueda local termina siempre en un óptimo local, y ahí se queda. El resto de la asignatura es una colección de respuestas a esa limitación:

Respuesta	Cómo	Tema
Aceptar empeoramientos con probabilidad	enfriamiento simulado	5
Prohibir deshacer lo recién hecho	búsqueda tabú	5
Reiniciar desde otro punto	multiarranque, GRASP, ILS	5
Trabajar con muchas soluciones a la vez	algoritmos genéticos	3
Cambiar de entorno al atascarse	búsqueda por entornos variables	5

2.4 Trayectorias frente a poblaciones

Es la división que ordena el resto del programa.

	Trayectorias	Poblaciones
Soluciones vivas	una	muchas
Cómo avanza	movimientos en el entorno	recombinación y selección
Fuerte en	intensificar	diversificar
Débil en	escapar de óptimos locales	refinar la mejor solución
Memoria	del recorrido reciente	implícita, en la población
Coste por iteración	bajo	alto

Las de trayectoria bajan rápido y se atascan. Las de población cubren el espacio y tardan mucho en pulir. Puestas así, la conclusión del tema 4 se ve venir: los **algoritmos meméticos** son una población que ejecuta búsqueda local sobre sus miembros, y llevan años siendo lo mejor que se conoce en la mayoría de los problemas combinatorios.

Una advertencia que conviene tener presente desde este tema: el teorema de **no hay comida gratis** dice que, promediando sobre **todas** las funciones objetivo posibles, cualquier par de algoritmos de búsqueda tiene el mismo rendimiento. No existe la metaheurística mejor en general. Lo que existe es la que mejor explota la estructura de una familia concreta de problemas, y por eso la asignatura compara siempre sobre bancos de pruebas definidos.

Los modelos de búsqueda por entornos y su análisis están en Talbi, 2009 y Du y Swamy, 2016; el tratamiento del paisaje de la función objetivo, en Chopard y Tomassini, 2018.

Capítulo 3

Metaheurísticas basadas en poblaciones

Tema 3 del programa. Mantener muchas soluciones a la vez, combinarlas entre sí y dejar que la selección haga el resto.

3.1 Concepto y elementos

Una metaheurística basada en poblaciones sustituye la solución única por un conjunto $P = \{s_1, \ldots, s_\mu\}$ que evoluciona en paralelo. El esquema general:

```
funcion evolutivo():
    P = poblacion_inicial()
    evaluar(P)
    mientras no parada:
        Padres = seleccionar(P)
        Hijos  = recombinar(Padres)
        Hijos  = mutar(Hijos)
        evaluar(Hijos)
        P = reemplazar(P, Hijos)
    devolver mejor(P)
```

Cinco piezas, y cada una es una decisión de diseño:

Pieza	Qué controla
Población inicial	de dónde se parte y cuánta diversidad hay al empezar
Selección	qué soluciones se reproducen: presión selectiva
Recombinación (cruce)	cómo se mezcla la información de dos padres
Mutación	cómo se introduce variación nueva
Reemplazamiento	qué población sobrevive: presión selectiva otra vez

La **diversidad de la población** es el recurso que hay que administrar. Al principio sobra y al final falta, y cuando se agota la población colapsa: todos los individuos son iguales, el cruce deja de producir nada nuevo y el algoritmo se convierte en una búsqueda local cara. Eso se llama **convergencia prematura**, y es el fallo más común de esta familia.

3.2 Algoritmos genéticos

El caso canónico. Codifican la solución como un **cromosoma** y aplican selección, cruce y mutación.

3.2.1 Representación

Codificación	Cuándo	Ejemplo
Binaria	subconjuntos, decisiones sí/no	mochila
Entera	asignaciones a categorías	coloreado de grafos
Permutación	orden o secuencia	viajante, planificación
Real	parámetros continuos	ajuste de modelos

La binaria fue la original y arrastra un problema conocido: con codificación binaria estándar, números contiguos pueden diferir en muchos bits —de 7 (`0111`) a 8 (`1000`) cambian los cuatro—, así que una mutación pequeña en el cromosoma es un salto enorme en el espacio real. El **código Gray** lo corrige haciendo que enteros consecutivos difieran en un solo bit.

3.2.2 Selección

Decide qué individuos se reproducen. Su parámetro real es la **presión selectiva**: cuánto favorece a los mejores.

Operador	Cómo	Presión
Ruleta	probabilidad proporcional a la aptitud	depende de la escala de f
Torneo de tamaño k	se eligen k al azar y gana el mejor	crece con k
Por ranking	probabilidad según la posición, no el valor	estable

La **ruleta** tiene dos defectos que la han retirado de la práctica: si un individuo tiene una aptitud muy superior, acapara la reproducción y la población colapsa en pocas generaciones; y cuando los valores se igualan al final, la selección se vuelve casi aleatoria justo cuando hace falta discriminar. Además no se puede usar directamente en minimización ni con valores negativos.

El **torneo** no tiene ninguno de esos problemas: solo compara, así que le da igual la escala de f y funciona igual en minimización. $k = 2$ es la elección habitual; $k = 3$ o 4 acelera la convergencia y arriesga perder diversidad.

3.2.3 Cruce

Combina dos padres en uno o dos hijos. El operador tiene que ser **coherente con la representación**, y es donde más fallos de diseño se cometen.

Para representación binaria:

```
Padre 1:  1 0 1 | 1 0 0 1
Padre 2:  0 1 1 | 0 1 1 0
          ------+--------
```

```
Hijo 1:   1 0 1 | 0 1 1 0
Hijo 2:   0 1 1 | 1 0 0 1
```

Es el cruce en un punto. El de dos puntos intercambia un tramo central, y el **uniforme** decide bit a bit con probabilidad 1/2: este último es el que más mezcla y el que menos respeta los bloques contiguos.

Para permutaciones, el cruce en un punto **no vale**: produce recorridos con ciudades repetidas y otras ausentes. Hacen falta operadores específicos:

Operador	Qué conserva
PMX, cruce parcialmente mapeado	las posiciones absolutas de un tramo
OX, cruce por orden	el orden relativo de las ciudades
Cruce por ciclos	las posiciones que forman ciclos entre los padres

Cuál conviene depende de dónde esté la información del problema. En el viajante lo que importa son las **aristas**, no las posiciones, así que OX funciona mejor que PMX; en problemas de planificación, donde importa la posición absoluta en el calendario, es al revés.

Para codificación real, el **cruce BLX-α** genera cada gen del hijo en el intervalo

$$[\, c_{\text{mín}} - \alpha I,\ c_{\text{máx}} + \alpha I \,], \qquad I = c_{\text{máx}} - c_{\text{mín}}$$

con $c_{\text{mín}}$ y $c_{\text{máx}}$ los valores de los dos padres. Con $\alpha = 0$ el hijo queda entre los padres y la población se contrae; con $\alpha \approx 0{,}5$ puede salirse del intervalo, y ese margen es lo que mantiene la diversidad viva.

3.2.4 Mutación

Introduce variación que el cruce no puede generar. Es lo que impide que un valor perdido en toda la población quede perdido para siempre.

Representación	Mutación	Probabilidad habitual
Binaria	invertir un bit	$1/n$ por gen
Permutación	intercambiar dos posiciones, o invertir un tramo	0,1 por individuo
Real	sumar ruido gaussiano $\mathcal{N}(0, \sigma)$	$1/n$ por gen

$1/n$ por gen significa que se cambia **un gen por individuo en media**, que es el valor por defecto razonable. Subirlo mucho convierte el algoritmo en una búsqueda aleatoria; bajarlo a cero lo deja sin salida cuando la población converge.

3.2.5 Reemplazamiento

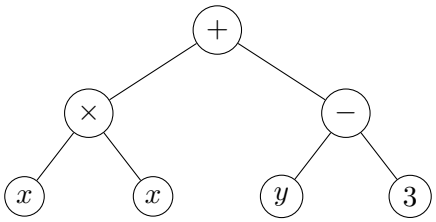
Decide la población siguiente. Dos esquemas, y la diferencia es grande:

Esquema	Cómo	Efecto
Generacional	los hijos sustituyen a toda la población	más exploración, más lento
Estacionario	se generan dos hijos y compiten con los peores	más explotación, converge antes

En el generacional se pierde al mejor individuo si sus hijos salen peores. El **elitismo** lo evita conservando de forma explícita al mejor de cada generación, y es prácticamente obligatorio: sin él, la mejor solución encontrada puede desaparecer y el algoritmo empeora con el tiempo.

3.3 Programación genética

La misma maquinaria con una representación distinta: el individuo es un **árbol de expresión**, no un vector.



El árbol de la figura representa $x \times x + (y - 3)$. Los nodos internos salen del conjunto de **funciones** —operadores aritméticos, condicionales— y las hojas del de **terminales**: variables y constantes.

El cruce intercambia subárboles entre dos padres, y la mutación sustituye un subárbol por otro generado al azar. Su problema característico es el **crecimiento descontrolado**: los árboles se hinchan con ramas que no cambian la salida, la evaluación se vuelve lenta y el resultado deja de ser legible. Se combate limitando la profundidad o penalizando el tamaño en la función objetivo.

Se usa cuando lo que se busca **es** una expresión: regresión simbólica, descubrimiento de fórmulas, control. Con longitud fija basta un algoritmo genético.

3.4 Evolución diferencial

Para optimización continua, y en ese terreno es de lo mejor que hay. Su operador sustituye el cruce clásico por una **combinación de diferencias entre individuos**.

Para cada individuo x_i de la población se construye un vector mutado tomando otros tres al azar:

$$v_i = x_{r_1} + F \cdot (x_{r_2} - x_{r_3}), \qquad F \in [0, 2]$$

y luego se cruza con x_i gen a gen con probabilidad CR. El hijo sustituye al padre solo si es mejor.

Lo que hace especial al esquema es que **el tamaño del paso lo fija la propia población**. Al principio los individuos están dispersos, las diferencias $x_{r_2} - x_{r_3}$ son grandes y los saltos también; según converge, las diferencias se encogen y el algoritmo pasa solo a refinar. No hay que programar ningún calendario: la adaptación es automática, y es la razón de su buen comportamiento sin ajuste fino.

Parámetro	Rango habitual	Qué controla
F	0,5–0,9	amplitud del salto
CR	0,1–0,9	cuántos genes se toman del mutado
μ	$10n$	tamaño de población, con n dimensiones

CR alto conviene cuando las variables interaccionan; CR bajo, cuando el problema es casi separable y compensa mover pocas coordenadas a la vez.

3.5 Estrategias de evolución

La otra familia clásica de optimización continua. Su rasgo propio es que **la mutación se auto-adapta**: el individuo lleva consigo la desviación típica σ con la que se muta, y esa σ también evoluciona.

La notación (μ, λ) indica que μ padres generan λ hijos y la población siguiente sale **solo de los hijos**; $(\mu + \lambda)$ indica que compiten padres e hijos, lo que la hace elitista. La primera puede empeorar de una generación a la siguiente, y eso le permite escapar de óptimos locales.

La versión moderna, CMA-ES, estima además la matriz de covarianzas de la distribución de mutación, con lo que aprende la forma del valle en el que está y alinea los saltos con él. Es la referencia contra la que se comparan los algoritmos de optimización continua.

3.6 Aplicación a problemas

Fijar la representación y los operadores es todo el trabajo de diseño:

Problema	Representación	Cruce	Mutación
Mochila	binaria, n objetos	uniforme	invertir un bit
Viajante	permutación de n ciudades	OX	invertir un tramo
Ajuste de parámetros	vector real	BLX-α	gaussiana
Coloreado de grafos	entera, un color por nodo	uniforme	cambiar el color de un nodo
Selección de características	binaria, un bit por atributo	uniforme	invertir un bit

En la mochila el cruce uniforme produce soluciones que exceden la capacidad, así que hace falta reparar: quitar objetos por orden creciente de valor por unidad de peso hasta que quepa. Es la estrategia de reparación del tema 2, y es preferible a penalizar porque devuelve siempre soluciones admisibles.

El tratamiento completo de los algoritmos evolutivos está en Eiben y Smith, 2015, y su comparación con el resto de familias en Talbi, 2009 y Du y Swamy, 2016.

Capítulo 4

Algoritmos meméticos

Tema 4 del programa. Poner una búsqueda local dentro de un algoritmo de población, que es la combinación que mejor funciona en la mayoría de los problemas combinatorios.

4.1 Hibridación de metaheurísticas

El tema 2 dejó la comparación planteada: las técnicas de trayectoria intensifican bien y se atascan; las de población diversifican bien y refinan mal. Hibridar es juntar las dos y quedarse con lo bueno de cada una.

Talbi clasifica las hibridaciones por dónde se produce la unión:

Tipo	Cómo	Ejemplo
De bajo nivel, **integrativa**	una técnica se convierte en operador de la otra	algoritmo memético
De alto nivel, **secuencial**	una se ejecuta después de la otra	genético y luego búsqueda local sobre el resultado
De alto nivel, **paralela**	varias corren a la vez e intercambian soluciones	modelo de islas
Con métodos exactos	la metaheurística usa un solucionador exacto en un subproblema	matheurísticas

La secuencial es la barata y la que menos aporta: una pasada de búsqueda local al final mejora el resultado un poco, pero no cambia cómo ha buscado el algoritmo. La integrativa sí, porque **cambia el espacio en el que la población evoluciona**.

4.2 Algoritmos meméticos

Un algoritmo memético es un algoritmo de población en el que cada individuo, o algunos, ejecutan una búsqueda local antes de entrar en la población.

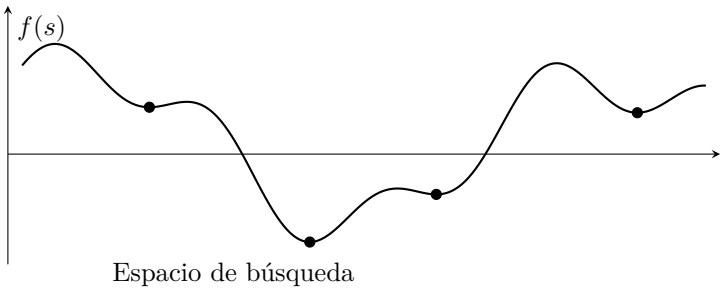
```
funcion memetico():
    P = poblacion_inicial()
    P = optimizar_localmente(P)
    mientras no parada:
        Padres = seleccionar(P)
        Hijos  = recombinar(Padres)
        Hijos  = mutar(Hijos)
        Hijos  = optimizar_localmente(Hijos)     <- la unica linea nueva
```

```
        P = reemplazar(P, Hijos)
    devolver mejor(P)
```

El nombre viene de *mema*: la unidad de información cultural que, a diferencia del gen, el individuo puede mejorar durante su vida antes de transmitirla. Es una analogía, no un argumento; lo que sostiene a estos algoritmos son los resultados experimentales.

4.2.1 Qué cambia de verdad

La consecuencia importante no es que las soluciones sean mejores. Es que **la población deja de vivir en S y pasa a vivir en el conjunto de los óptimos locales**.



Espacio de búsqueda

Los cuatro puntos marcados son los óptimos locales del paisaje. Tras la búsqueda local, todo individuo cae en uno de ellos, y el algoritmo genético trabaja sobre esos cuatro estados en vez de sobre el continuo. El espacio efectivo se reduce muchísimo, y esa reducción es la ganancia real.

De ahí sale también su riesgo característico: si la búsqueda local es agresiva, todos los individuos caen en el mismo óptimo y la población pierde la diversidad en las primeras generaciones. **El memético converge antes, y por eso hay que vigilarlo más.**

4.3 Las decisiones de diseño

Cuatro preguntas, y las respuestas cambian mucho el resultado.

4.3.1 A cuántos individuos se aplica

Estrategia	Cómo	Comentario
A todos	cada hijo se optimiza	máxima explotación, coste altísimo
A una fracción p_{LS}	se elige al azar con probabilidad p_{LS}	el compromiso habitual, con $p_{LS} \approx 0{,}1$
Solo a los mejores	se ordena y se optimiza el 10 % superior	intensifica donde ya hay calidad
Solo al mejor	una vez por generación	apenas cambia el comportamiento

Aplicarla a todos es lo primero que se prueba y casi nunca es lo mejor: el coste por generación se multiplica por el de la búsqueda local, y con presupuesto fijo de evaluaciones eso significa muchas menos generaciones.

4.3.2 Cuánto se optimiza

La búsqueda local puede llevarse hasta el óptimo local o cortarse antes.

- **Completa**: se itera hasta que ningún vecino mejora. Da el máximo refinamiento y el máximo coste, y es la que colapsa la diversidad.
- **Truncada**: un número fijo de iteraciones o de evaluaciones. Deja al individuo a medio camino, conserva diversidad y suele rendir mejor con presupuesto limitado.

4.3.3 Qué entorno usa

No tiene por qué ser el mismo que usaría una búsqueda local sola. Un entorno pequeño y barato dentro de un memético suele batir a uno grande, porque el algoritmo hace muchas más llamadas.

4.3.4 Cómo se devuelve el resultado

Dos variantes, y la diferencia es conceptual:

Variante	Qué se guarda en la población
Lamarckiana	la solución optimizada; el cromosoma se reescribe
Baldwiniana	el cromosoma original, con la aptitud del optimizado

La lamarckiana propaga la mejora a los descendientes y converge más rápido. La baldwiniana solo cambia la presión selectiva —premia a los individuos que están cerca de un buen óptimo— y conserva más diversidad. En la práctica se usa casi siempre la lamarckiana; la baldwiniana aparece cuando la convergencia prematura es el problema dominante.

4.4 Control de la diversidad

Como el memético converge deprisa, casi siempre hace falta un mecanismo que sostenga la diversidad:

Mecanismo	Cómo
Reinicio	si la población se estanca, se regenera conservando al mejor
Cruce entre distintos	se emparejan individuos alejados entre sí
Reemplazo del más parecido	el hijo sustituye al individuo más próximo, si es mejor
Nichos	se penaliza a los individuos con muchos vecinos cercanos

El **reemplazo del más parecido** es el más efectivo y el más barato de programar: mantiene el tamaño de población y evita que dos copias de la misma solución convivan. Necesita una medida de distancia entre soluciones, que en representación binaria es la de Hamming y en permutaciones es el número de posiciones distintas.

4.5 Un caso completo

Un memético para el viajante, con las decisiones tomadas:

Elemento	Elección	Por qué
Representación	permutación	cierra las restricciones
Selección	torneo binario	inmune a la escala de f
Cruce	OX	conserva el orden relativo, que es donde está la información
Mutación	invertir un tramo	coherente con el 2-opt
Búsqueda local	2-opt truncado a 100 movimientos	elimina cruces del recorrido
p_{LS}	0,1	evita que el coste por generación se dispare
Reemplazo	estacionario, del más parecido	sostiene la diversidad

La coherencia entre la mutación y el entorno de la búsqueda local no es un detalle: si la mutación intercambia dos ciudades y el 2-opt invierte tramos, el 2-opt deshace la mutación en su primer movimiento y el operador deja de servir para nada. Es el fallo más habitual al montar un memético, y no lo señala ningún error: el algoritmo funciona, simplemente rinde como si no tuviera mutación.

El marco de la hibridación y su taxonomía están en Talbi, 2009, y el análisis de los algoritmos meméticos dentro de la computación evolutiva en Eiben y Smith, 2015 y Du y Swamy, 2016.

Capítulo 5

Metaheurísticas basadas en trayectorias

Tema 5 del programa. Una sola solución que se mueve por el espacio, y los distintos mecanismos para que ese recorrido no muera en el primer óptimo local.

5.1 El inconveniente de la búsqueda local

La búsqueda local del tema 2 termina cuando ningún vecino mejora, y ese punto es un óptimo local respecto del entorno elegido. Nada garantiza que sea el global, y en paisajes rugosos suele estar lejos.

Dos formas de medir el daño:

- **Calidad**: en instancias del viajante de tamaño medio, el 2-opt desde una solución aleatoria se queda entre un 5 y un 8 % por encima del óptimo, y no baja de ahí por mucho que se repita el mismo descenso.
- **Varianza**: dos ejecuciones desde puntos de partida distintos acaban en óptimos locales distintos, con diferencias grandes entre ellos. La solución depende más del azar inicial que del algoritmo.

Las respuestas de este tema se agrupan en tres ideas:

Idea	Cómo	Algoritmos
Aceptar empeoramientos	con una probabilidad que decrece	enfriamiento simulado
Prohibir volver atrás	con memoria explícita	búsqueda tabú
Empezar otra vez	desde otro punto, mejor elegido	multiarranque, GRASP, ILS

5.2 Enfriamiento simulado

Toma su esquema del recocido de los metales: se calienta el material y se enfría despacio, de modo que los átomos tengan tiempo de reordenarse en una estructura de energía baja. La analogía es fiel en un punto concreto: **a temperatura alta se aceptan estados peores, y a temperatura baja casi no**.

```
funcion enfriamiento_simulado(s0, T0):
    s = s0;  mejor = s0;  T = T0
    mientras no parada:
        repetir L veces:
```

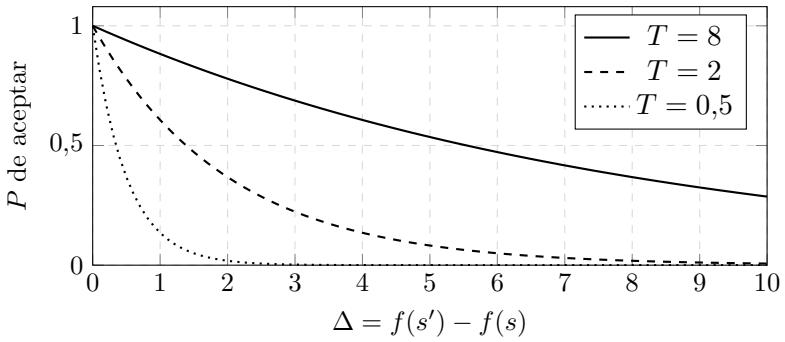
```
            s' = vecino_aleatorio(s)
            delta = f(s') - f(s)
            si delta < 0 o aleatorio() < exp(-delta / T):
                s = s'
                si f(s) < f(mejor): mejor = s
        T = enfriar(T)
    devolver mejor
```

El criterio de aceptación es el de Metropolis: un empeoramiento de tamaño Δ se acepta con probabilidad

$$P(\Delta, T) = e^{-\Delta/T}$$

Dos lecturas de esa fórmula, y las dos importan:

- **A T alta**, el exponente es casi cero y la probabilidad casi uno: se acepta casi todo y la búsqueda es prácticamente aleatoria.
- **A T baja**, la probabilidad se hunde y solo pasan los movimientos que mejoran: el algoritmo degenera en búsqueda local.
- **Para T fija**, un empeoramiento pequeño se acepta mucho más que uno grande. Esto es lo que distingue al enfriamiento simulado de aceptar al azar: discrimina por cuánto se empeora, no solo por si se empeora.



5.2.1 El esquema de enfriamiento

Es el parámetro que decide el rendimiento.

Esquema	Fórmula	Comentario
Geométrico	$T_{k+1} = \alpha T_k$, con $\alpha \in [0{,}8, 0{,}99]$	el habitual, por simple
Lineal	$T_{k+1} = T_k - \beta$	enfría demasiado deprisa al final
Cauchy	$T_k = T_0/(1+k)$	más lento en la cola
Logarítmico	$T_k = T_0/\log(1+k)$	garantiza el óptimo, en tiempo infinito

El logarítmico es el único con garantía teórica de convergencia al óptimo global, y es inútil en la práctica: exige más iteraciones que enumerar el espacio. Lo que se usa es el geométrico, ajustando α al presupuesto disponible.

La **temperatura inicial** se fija de modo que al principio se acepte una fracción dada de los empeoramientos, típicamente el 80 o el 90 %. Se estima muestreando movimientos al azar, calculando

el Δ medio y despejando T_0 de $e^{-\bar{\Delta}/T_0} = 0{,}9$. Fijarla a ojo es la causa más común de que el algoritmo no funcione: demasiado alta y malgasta la mitad del presupuesto haciendo un paseo aleatorio; demasiado baja y es una búsqueda local desde el primer paso.

El número de iteraciones por temperatura, L, debe ser proporcional al tamaño del entorno, para que a cada temperatura le dé tiempo a alcanzar el equilibrio.

5.3 Búsqueda tabú

La otra respuesta clásica, y de naturaleza opuesta: en vez de aceptar empeoramientos al azar, **se mueve siempre al mejor vecino aunque empeore**, y usa memoria para no volver por donde ya ha pasado.

```
funcion busqueda_tabu(s0):
    s = s0;  mejor = s0;  T = lista_vacia
    mientras no parada:
        s' = mejor_vecino_no_tabu(s, T)
        si f(s') < f(mejor):            # criterio de aspiracion
            mejor = s'
        T.insertar(movimiento(s, s'))
        T.eliminar_los_mas_antiguos()
        s = s'
    devolver mejor
```

Sin la lista tabú, «moverse siempre al mejor vecino» produce un ciclo: desde un óptimo local el mejor vecino es el que empeora menos, y desde él el mejor vecino vuelve a ser el óptimo local. La búsqueda oscila entre dos puntos para siempre. La **lista tabú** lo rompe prohibiendo deshacer los últimos movimientos.

5.3.1 Qué se guarda en la lista

No la solución completa —comparar soluciones enteras es caro y ocupa mucho—, sino el **atributo del movimiento**. En el viajante, las dos ciudades intercambiadas; en representación binaria, el índice del bit invertido.

Eso hace la lista barata y trae un efecto secundario: prohibir un atributo prohíbe también soluciones que nunca se han visitado. Es una pérdida asumida, y el **criterio de aspiración** la compensa: un movimiento tabú se permite si lleva a una solución mejor que la mejor conocida, porque entonces no puede estar reincidiendo.

La **longitud de la lista** —el *tenor*— controla el equilibrio:

Tenor	Efecto
Corto	intensifica; riesgo de ciclos largos
Largo	diversifica; puede prohibir toda la región buena
Variable	se alarga al estancarse y se acorta al mejorar

Un valor de referencia es $\sqrt{n}$ con n el tamaño del problema, ajustado después de forma empírica.

5.3.2 Memoria a largo plazo

La lista tabú es memoria a corto plazo. Una búsqueda tabú completa añade dos mecanismos más:

- **Intensificación**: se registra la frecuencia con que cada atributo aparece en las buenas soluciones y se reinicia desde una construida con los más frecuentes.
- **Diversificación**: se penalizan los atributos más usados, forzando a la búsqueda a regiones que no ha visitado.

Las dos comparten estructura, una tabla de frecuencias, y se disparan cuando el algoritmo lleva muchas iteraciones sin mejorar.

5.4 Trayectorias múltiples

La tercera respuesta: en vez de arreglar un recorrido, hacer varios.

5.4.1 Multiarranque básico

Repetir búsqueda local desde soluciones aleatorias y quedarse con la mejor. Es la línea base contra la que hay que comparar cualquier técnica de este tema: si una metaheurística no bate al multiarranque con el mismo presupuesto, no aporta nada.

Su defecto es que **no aprende**: cada arranque ignora todo lo que descubrieron los anteriores. Las dos técnicas que siguen son dos formas de corregirlo.

5.4.2 GRASP

Greedy Randomized Adaptive Search Procedure. Cada iteración construye una solución con un voraz aleatorizado y luego la mejora con búsqueda local.

```
funcion grasp():
    mejor = nulo
    mientras no parada:
        s = construir_voraz_aleatorizado()
        s = busqueda_local(s)
        si f(s) < f(mejor): mejor = s
    devolver mejor
```

La fase de construcción es lo característico. En cada paso se ordenan los elementos candidatos por su coste voraz y se forma la **lista restringida de candidatos** —los mejores según un umbral— de la que se elige uno **al azar**:

$$\mathrm{LRC} = \{\, e : c(e) \leq c_{\text{mín}} + \alpha\,(c_{\text{máx}} - c_{\text{mín}}) \,\}$$

El parámetro α interpola entre los dos extremos:

α	Comportamiento
0	voraz puro: la misma solución siempre, y una sola iteración útil
1	aleatorio puro: equivale al multiarranque básico
0,1–0,3	el rango útil: soluciones buenas y distintas entre sí

Su ventaja sobre el multiarranque es que parte de soluciones ya razonables, así que la búsqueda local tiene menos trabajo y termina en óptimos locales mejores.

5.4.3 ILS

Iterated Local Search. En vez de empezar de cero, **perturba la mejor solución encontrada** y vuelve a optimizar.

```
funcion ils(s0):
    s = busqueda_local(s0)
    mientras no parada:
        s' = perturbar(s)
        s' = busqueda_local(s')
        s  = aceptar(s, s')
    devolver s
```

Todo depende del tamaño de la perturbación, y el equilibrio es estrecho:

- **Demasiado pequeña**: la búsqueda local deshace la perturbación y devuelve al mismo óptimo local. El algoritmo se queda quieto sin que nada lo indique.
- **Demasiado grande**: la solución resultante no conserva nada de la anterior y el ILS degenera en multiarranque.

Un valor razonable en el viajante es el **doble puente**, que corta el recorrido en cuatro tramos y los reordena: es un movimiento que el 2-opt **no puede deshacer**, que es justo la propiedad que se necesita.

El criterio de aceptación fija cuánto se intensifica: aceptar solo si mejora es lo más intensificador; aceptar siempre convierte el recorrido en un paseo aleatorio por óptimos locales; aceptar con el criterio de Metropolis del enfriamiento simulado es el punto intermedio.

5.4.4 ILS híbrida

Sustituir la búsqueda local del ILS por el enfriamiento simulado. Se obtiene un algoritmo que escapa de óptimos locales en dos escalas: la fina, dentro de cada enfriamiento, y la gruesa, con la perturbación. Es de las combinaciones más efectivas para problemas combinatorios con presupuesto medio.

5.5 Búsqueda por entornos variables

La idea complementaria: un óptimo local **lo es respecto de un entorno concreto**. Si al atascarse se cambia de entorno, deja de serlo.

```
funcion vns(s, entornos N1..Nk):
    mientras no parada:
        i = 1
        mientras i <= k:
            s' = aleatorio_de(Ni(s))
            s' = busqueda_local(s')
            si f(s') < f(s):
                s = s';  i = 1           # se vuelve al primero
            si no:
                i = i + 1                # se pasa al siguiente
    devolver s
```

Los entornos se ordenan de menor a mayor. Mientras haya mejora se trabaja con el más pequeño, que es el más barato, y solo al agotarlo se pasa a uno mayor. Volver a N_1 tras cada mejora es lo

que mantiene el coste bajo.

5.6 Comparación

Algoritmo	Memoria	Escapa por	Parámetros críticos
Búsqueda local	no	no escapa	entorno
Enfriamiento simulado	no	aceptar empeoramientos	T_0, α, L
Búsqueda tabú	sí, explícita	prohibir el retorno	tenor, aspiración
Multiarranque	no	reiniciar	ninguno
GRASP	no	reiniciar mejor	α de la LRC
ILS	implícita, en la solución actual	perturbar	tamaño de perturbación
VNS	no	cambiar de entorno	los entornos y su orden

Ninguno domina a los demás en todos los problemas, y ese es el contenido práctico del teorema de no hay comida gratis. Lo que decide es la comparación experimental sobre el problema concreto, con las mismas evaluaciones para todos y varias semillas por algoritmo.

El desarrollo detallado de estas técnicas está en Talbi, 2009 y Chopard y Tomassini, 2018; su análisis comparado en Du y Swamy, 2016, y el catálogo de aplicaciones en Pardalos y Resende, 2002.

Capítulo 6

Metaheurísticas basadas en adaptación social

Tema 6 del programa. Poblaciones de agentes simples que no se recombinan, sino que se coordinan: cada uno decide con información local y el comportamiento útil aparece en el conjunto.

6.1 Introducción a la adaptación social

Las técnicas del tema 3 mezclan soluciones con un operador de cruce. Las de este tema no: los agentes **se influyen** unos a otros, y la solución emerge de esa influencia.

	Evolutivas	Adaptación social
Unidad	cromosoma	agente
Cómo se comunica la información	cruce entre padres	señal compartida o memoria colectiva
Qué se hereda	material genético	nada; el agente persiste
Fuente de la mejora	selección	cooperación

La **inteligencia de enjambre** es el nombre general del fenómeno: agentes con reglas triviales y visión local producen un comportamiento colectivo que ninguno de ellos podría producir solo. Las dos realizaciones del programa son las colonias de hormigas y las nubes de partículas.

6.2 Cooperación de agentes en problemas de optimización

Los agentes se coordinan de dos maneras, y la distinción explica las dos familias:

- **Comunicación indirecta**, a través del entorno. Un agente modifica el medio y otro lee esa modificación más tarde. Se llama **estigmergia**, y es el mecanismo de las hormigas: la feromona depositada en el suelo.
- **Comunicación directa**, entre agentes. Cada uno conoce la posición de los demás o de los mejores. Es el mecanismo de las nubes de partículas.

La indirecta tiene una propiedad que la hace apta para problemas combinatorios: la información persiste en el medio y se **acumula**, así que varias generaciones de agentes contribuyen al mismo rastro. La directa es más rápida y adecuada para espacios continuos, donde «acercarse a otro agente» tiene sentido geométrico.

6.3 Algoritmos basados en colonias de hormigas

Las hormigas encuentran el camino más corto entre el nido y la comida sin verlo entero. Cada una deja feromona al pasar; los caminos cortos se recorren antes, así que acumulan feromona más deprisa, y eso atrae a más hormigas. El rastro se refuerza solo.

Trasladado a un algoritmo, para el viajante:

```
funcion aco():
    inicializar tau[i][j] = tau0
    mientras no parada:
        para cada hormiga k:
            construir un recorrido eligiendo cada ciudad segun tau y eta
        evaporar: tau[i][j] = (1 - rho) * tau[i][j]
        depositar: tau[i][j] += suma de las contribuciones de las hormigas
    devolver el mejor recorrido encontrado
```

6.3.1 La regla de transición

Una hormiga situada en la ciudad i elige la siguiente j entre las no visitadas con probabilidad

$$p_{ij} = \frac{\tau_{ij}^{\alpha}\,\eta_{ij}^{\beta}}{\sum_{l \in \text{no visitadas}} \tau_{il}^{\alpha}\,\eta_{il}^{\beta}}$$

donde τ_{ij} es la feromona en la arista y $\eta_{ij} = 1/d_{ij}$ es la **información heurística**: el inverso de la distancia, que favorece a las ciudades cercanas.

Parámetro	Qué pondera	Si vale 0
α	la feromona: la experiencia acumulada	el algoritmo es un voraz aleatorizado sin memoria
β	la heurística: el coste inmediato	se ignora la distancia y la convergencia es muy lenta

Valores de referencia: $\alpha = 1$ y $\beta \in [2, 5]$. La heurística pesa más que la feromona al principio, cuando el rastro aún no dice nada.

6.3.2 Evaporación

$$\tau_{ij} \leftarrow (1 - \rho)\,\tau_{ij} + \Delta\tau_{ij}$$

La evaporación es el mecanismo de olvido, y sin ella el algoritmo no funciona: los primeros recorridos, que son malos, acumularían feromona indefinidamente y la colonia quedaría fijada en ellos. ρ controla la velocidad del olvido; valores entre 0,01 y 0,1 son los habituales.

El depósito $\Delta\tau_{ij}$ es proporcional a la calidad del recorrido que usó esa arista, típicamente Q/L_k con L_k la longitud del recorrido de la hormiga k.

6.3.3 Variantes

Variante	Qué cambia	Efecto
Sistema de hormigas (AS)	todas las hormigas depositan	el original; converge despacio
Sistema de colonias (ACS)	solo la mejor deposita, más evaporación local	mucho más rápido
MAX-MIN	la feromona se acota entre $\tau_{\text{mín}}$ y $\tau_{\text{máx}}$	evita el estancamiento

MAX-MIN resuelve el fallo característico de la familia: cuando una arista acumula tanta feromona que su probabilidad se acerca a uno, todas las hormigas construyen el mismo recorrido y el algoritmo deja de explorar. Acotar por arriba lo impide, y acotar por abajo garantiza que ninguna arista queda descartada del todo.

El campo natural de aplicación son los problemas donde la solución **se construye por partes** y hay una noción de arista o componente: viajante, enrutamiento de vehículos, asignación cuadrática, planificación de tareas.

6.4 Algoritmos basados en nubes de partículas

Particle Swarm Optimization. Está pensado para espacios continuos y su analogía es el vuelo coordinado de una bandada: cada individuo ajusta su rumbo según a dónde va él y a dónde va el grupo.

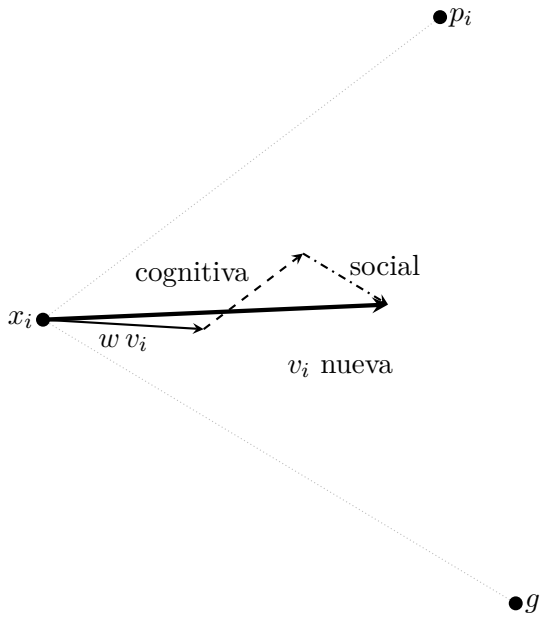
Cada partícula tiene una posición x_i y una velocidad v_i, y recuerda dos cosas: su mejor posición histórica p_i y la mejor del enjambre g.

$$v_i \leftarrow w\, v_i + c_1 r_1 (p_i - x_i) + c_2 r_2 (g - x_i)$$

$$x_i \leftarrow x_i + v_i$$

con r_1, r_2 aleatorios uniformes en $[0, 1]$. Los tres sumandos son tres fuerzas:

Término	Nombre	Qué hace
$w\, v_i$	inercia	mantiene el rumbo; controla la exploración
$c_1 r_1 (p_i - x_i)$	componente cognitiva	tira hacia lo mejor que ha visto la partícula
$c_2 r_2 (g - x_i)$	componente social	tira hacia lo mejor que ha visto el enjambre



6.4.1 Ajuste

Parámetro	Valor habitual	Qué pasa si crece
w	0,9 → 0,4, decreciente	la nube se dispersa y no converge
c_1	2	cada partícula va a lo suyo; poca cooperación
c_2	2	todas caen sobre g y la nube colapsa
$v_{\text{máx}}$	una fracción del rango	sin límite, las partículas se salen del espacio

La inercia decreciente es el equivalente al esquema de enfriamiento del tema 5: exploración al principio, explotación al final. Sin límite de velocidad la nube diverge, y sin el límite ni la inercia decreciente el algoritmo no converge en absoluto.

La **topología del vecindario** decide cuánta información circula. Con la topología global —todas las partículas ven a la mejor del enjambre— la convergencia es rápida y el riesgo de caer en un óptimo local, alto. Con topologías locales, donde cada partícula solo ve a unas pocas vecinas, la información se propaga despacio y la diversidad dura más.

6.5 Aplicación a problemas

Problema	Técnica	Por qué
Viajante, enrutamiento	hormigas	la solución se construye arista a arista
Asignación cuadrática	hormigas	hay componentes discretas con coste asociado

Problema	Técnica	Por qué
Ajuste de parámetros continuos	nubes de partículas	el espacio es $\mathbb{R}^n$ y la geometría tiene sentido
Entrenamiento de redes neuronales	nubes de partículas	igual, y muchas dimensiones
Planificación de tareas	hormigas o meméticos	según sea de orden o de asignación

Aplicar nubes de partículas a un problema combinatorio exige redefinir qué significan la posición, la velocidad y la suma, y las adaptaciones que se han propuesto rara vez baten a las hormigas o a un memético en su terreno.

El tratamiento canónico de las colonias de hormigas está en Dorigo y Stützle, 2004; las nubes de partículas y el resto de técnicas bioinspiradas, en Du y Swamy, 2016 y Chopard y Tomassini, 2018.

Capítulo 7

Aspectos avanzados en metaheurísticas

Tema 7 del programa. El equilibrio entre diversidad y convergencia, qué hacer cuando no basta una sola solución, y qué aporta el paralelismo.

7.1 Diversidad frente a convergencia

Es el eje del tema 1, tratado ahora con instrumentos.

Una población pierde diversidad de forma natural: la selección premia a los mejores, sus descendientes se parecen a ellos y con las generaciones todos convergen al mismo punto. Si eso ocurre **antes** de que el algoritmo haya explorado lo suficiente, se llama **convergencia prematura** y el resultado es un óptimo local mediocre.

7.1.1 Cómo se mide

No se puede controlar lo que no se mide, y la diversidad hay que medirla en cada generación:

Medida	Cómo se calcula	Cuándo
Distancia media entre pares	media de $d(s_i, s_j)$ sobre todos los pares	general, coste $O(\mu^2)$
Distancia media al centroide	media de $d(s_i, \bar{s})$	continuo, coste $O(\mu)$
Entropía por gen	entropía de la distribución de valores en cada posición	binaria y entera
Varianza de la aptitud	varianza de f sobre la población	barata, y engañosa

La última es la más fácil y la menos fiable: **soluciones muy distintas pueden tener la misma aptitud**. Una varianza de aptitud nula no prueba que la población haya colapsado, y una alta no prueba que no. Cuando la medida importa, se mide sobre el genotipo.

7.1.2 Mecanismos de control

Mecanismo	Qué hace	Coste
Elitismo moderado	conserva solo al mejor, no al 10 %	ninguno

Mecanismo	Qué hace	Coste
Reemplazo del más parecido	evita duplicados en la población	una distancia por hijo
Compartición de aptitud	penaliza a los individuos con muchos vecinos	$O(\mu^2)$
Aclarado	dentro de un radio, solo los mejores conservan su aptitud	$O(\mu^2)$
Reinicio	se regenera la población conservando a los mejores	una generación
Mutación adaptativa	sube la probabilidad de mutación al estancarse	ninguno

La **compartición de aptitud** divide la aptitud de cada individuo por el número de vecinos dentro de un radio σ_{share}:

$$f'(s_i) = \frac{f(s_i)}{\sum_j \text{sh}(d(s_i, s_j))}, \qquad \text{sh}(d) = \begin{cases} 1 - (d/\sigma_{\text{share}})^\gamma & d < \sigma_{\text{share}} \\ 0 & \text{en otro caso} \end{cases}$$

Un individuo rodeado de copias suyas ve su aptitud dividida entre muchos, así que la selección deja de favorecerlo. El resultado es que la población se reparte entre los picos del paisaje en vez de amontonarse en uno. El precio es fijar σ_{share}, que exige saber a qué distancia están los picos, que es justo lo que no se sabe.

7.2 Modelos evolutivos con equilibrio explícito

Algunos modelos no controlan la diversidad con un añadido, sino que la estructuran en el propio esquema.

Modelo de islas. La población se parte en subpoblaciones que evolucionan aisladas y cada cierto número de generaciones intercambian individuos. Cada isla converge a su propia región, y la migración transfiere lo bueno sin homogeneizarlo todo.

Parámetro	Qué controla
Número de islas	cuántas regiones se exploran a la vez
Frecuencia de migración	cada cuántas generaciones se intercambia
Tasa de migración	cuántos individuos viajan
Topología	qué isla manda a cuál: anillo, malla, completa

Migrar demasiado a menudo convierte el conjunto en una sola población grande y anula la ventaja. Migrar demasiado poco deja islas independientes que no cooperan.

Modelo celular. Cada individuo ocupa una posición en una malla y solo se cruza con sus vecinos inmediatos. La información buena se propaga por difusión, y esa lentitud es la que sostiene la diversidad. Es la **estructuración espacial** frente a la temporal: en vez de repartir el presupuesto a lo largo del tiempo —enfriar, decrecer la inercia—, se reparte en el espacio de la población.

7.3 Múltiples soluciones: nichos

Hasta aquí el objetivo era una solución. Hay casos en que hacen falta varias:

- El problema tiene **varios óptimos globales equivalentes** y conviene conocerlos todos, porque el criterio real de elección no está en f.
- Se busca **robustez**: una solución algo peor pero en una región amplia aguanta mejor los cambios que otra óptima en un pico estrecho.
- El problema es **multiobjetivo** y no hay un único óptimo, sino un frente.

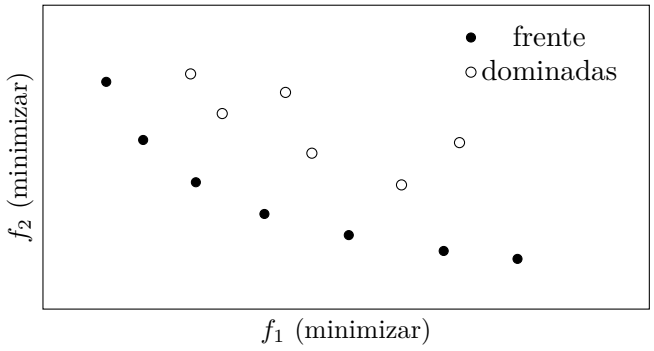
Las técnicas de nichos mantienen subpoblaciones en picos distintos:

Técnica	Cómo
Compartición de aptitud	penaliza la concentración, como arriba
Aclarado	dentro de un radio, solo los k mejores conservan su aptitud
Apiñamiento determinista	el hijo compite con el padre más parecido, no con el peor
Especiación	la población se agrupa y cada grupo evoluciona por separado

El **apiñamiento determinista** es el más simple y de los más efectivos: no necesita radio ni ningún parámetro nuevo, solo una distancia.

7.3.1 Optimización multiobjetivo

Con varios objetivos en conflicto no existe una solución mejor que todas. Se compara por **dominancia**: a domina a b si a es al menos igual de bueno en todos los objetivos y estrictamente mejor en alguno. Las soluciones no dominadas por ninguna otra forman el **frente de Pareto**.



Un algoritmo multiobjetivo persigue dos cosas a la vez: acercarse al frente y **repartirse a lo largo de él**, porque un frente concentrado en un extremo no le sirve a quien tiene que elegir. NSGA-II lo consigue ordenando por niveles de dominancia y desempatando por distancia de apiñamiento, que premia a las soluciones en zonas poco pobladas del frente.

7.4 Nuevos algoritmos bioinspirados

En las últimas dos décadas se han publicado decenas de metaheurísticas nuevas con metáforas de lo más variado. Conviene mirarlas con criterio:

- **La metáfora no es el algoritmo.** Bajo la analogía casi siempre hay una población, un

operador de perturbación y una selección. Muchas propuestas resultan ser, tras quitar el vocabulario, variantes de la evolución diferencial o de las nubes de partículas.
- **La comparación tiene que ser justa**: mismo número de evaluaciones, mismas instancias, varias semillas y contraste estadístico. Comparar por iteraciones en vez de por evaluaciones favorece a quien más hace por iteración.
- **Contra algoritmos actuales.** Batir a un genético de los años noventa no dice nada; hay que compararse con CMA-ES o con evolución diferencial bien ajustada.

Es la aplicación práctica del teorema de no hay comida gratis: si un algoritmo nuevo gana en todas partes, casi siempre lo que falla es el protocolo experimental.

7.5 Metaheurísticas paralelas

El paralelismo aporta dos cosas distintas, y conviene no mezclarlas: **acelerar** lo mismo, o **buscar mejor**.

Modelo	Qué se reparte	Qué se gana
Paralelización de la evaluación	las evaluaciones de la función objetivo	velocidad; el algoritmo no cambia
Paralelización del entorno	los vecinos de una búsqueda local	velocidad
Modelo de islas	subpoblaciones independientes	velocidad **y** calidad
Modelo celular	la malla de individuos	velocidad **y** calidad
Ejecuciones independientes	varias corridas con semillas distintas	robustez frente al azar

Las dos primeras son paralelismo puro: el resultado es el mismo que en secuencial, solo que antes. Compensan cuando evaluar es caro, que es el caso habitual en problemas reales —una simulación, un entrenamiento— y ahí la ganancia es casi lineal en el número de procesadores.

Las dos siguientes **cambian el algoritmo**, y por eso pueden dar mejores soluciones que la versión secuencial con el mismo número total de evaluaciones. Es la estructuración espacial frente a la temporal otra vez: repartir la población en el espacio produce un comportamiento que ninguna planificación temporal reproduce.

La medida de referencia es la aceleración $S_p = T_1/T_p$ con p procesadores. Su límite lo pone la ley de Amdahl: si una fracción β del algoritmo es intrínsecamente secuencial, la aceleración no pasa de $1/\beta$ por muchos procesadores que se añadan. En el modelo de islas la parte secuencial es la migración, así que migrar poco es bueno también por esta razón.

El tratamiento completo del paralelismo en metaheurísticas está en Alba, 2005, y los modelos de diversidad y multiobjetivo en Eiben y Smith, 2015 y Talbi, 2009.

Capítulo 8

Aprendizaje evolutivo: problemas y modelos

Tema 8 del programa. Plantear el aprendizaje automático como un problema de optimización y resolverlo con metaheurísticas.

8.1 El aprendizaje como problema de optimización

Aprender de datos es buscar, dentro de un espacio de modelos, el que mejor explica un conjunto de ejemplos. Escrito con la notación del tema 1:

Pieza	En aprendizaje
Espacio de búsqueda S	los modelos posibles: pesos, reglas, estructuras
Función objetivo f	el error sobre los datos, más un término de complejidad
Restricciones	la forma admisible del modelo

$$f(m) = \text{error}(m, D) + \lambda\,\text{complejidad}(m)$$

El segundo término es la **regularización**, y sin él el proceso encuentra modelos que memorizan los datos de entrenamiento y fallan con datos nuevos. Es la forma que toma aquí el sobreajuste: el algoritmo optimiza exactamente lo que se le pide, y si lo que se le pide es el error de entrenamiento, eso es lo que minimiza.

De ahí sale la regla de trabajo: **el error que guía la búsqueda se calcula sobre entrenamiento y el que se informa sobre un conjunto de prueba separado**, que el algoritmo no ha visto nunca. Un resultado medido sobre los datos con los que se optimizó no dice nada.

8.2 Cuándo compensa una metaheurística

Los métodos clásicos de aprendizaje usan el gradiente, que es mucho más eficiente cuando se puede calcular. Las metaheurísticas entran en los casos en que no:

Situación	Por qué falla el gradiente
La función objetivo no es derivable	no hay gradiente que calcular
El espacio es discreto o mixto	seleccionar atributos, elegir estructura
Hay muchos óptimos locales	el descenso se queda en el primero

Situación	Por qué falla el gradiente
El objetivo es una simulación o una caja negra	no se conoce la forma analítica
Hay varios objetivos en conflicto	precisión frente a interpretabilidad

Y la contrapartida, que conviene decir: **con un problema derivable y convexo, una metaheurística es la herramienta equivocada**. El descenso de gradiente lo resuelve en órdenes de magnitud menos tiempo y con garantía.

8.3 Modelos de aprendizaje basados en metaheurísticas

Tres niveles, según qué parte del modelo se optimiza:

Nivel	Qué se busca	Espacio
Parámetros	los valores del modelo, con la estructura fija	continuo
Estructura	la forma del modelo	discreto
Preprocesado	qué datos y qué atributos entran	binario

8.3.1 Selección de características

El caso más limpio y el más usado. Con n atributos hay 2^n subconjuntos, así que la enumeración deja de ser viable enseguida.

- **Representación**: vector binario de n bits, uno por atributo.
- **Función objetivo**: la precisión del clasificador entrenado solo con los atributos seleccionados, penalizando el número de atributos.
- **Operadores**: cruce uniforme y mutación por bit.

$$f(v) = \text{precisión}(v) - \lambda \frac{|v|}{n}$$

El coste está en la evaluación: cada individuo exige entrenar y validar un modelo. Con validación cruzada de cinco particiones, una población de 50 durante 100 generaciones son 25 000 entrenamientos. Aquí la paralelización de la evaluación del tema 7 no es una mejora, es lo que hace el experimento posible.

Este planteamiento se llama **envolvente** porque envuelve al clasificador y lo trata como una caja negra. Encuentra subconjuntos mejores que los métodos de filtro, que puntúan cada atributo por separado, porque tiene en cuenta las interacciones entre atributos; y es mucho más caro.

8.3.2 Optimización de parámetros

Ajustar los pesos de un modelo cuya estructura está fija: los pesos de una red neuronal, los parámetros de una función de pertenencia, los coeficientes de un controlador.

Con redes neuronales, entrenar los pesos con un algoritmo evolutivo se llama **neuroevolución**. Frente a la retropropagación:

	Retropropagación	Neuroevolución
Necesita gradiente	sí	no
Función de activación	derivable	cualquiera
Óptimos locales	se atasca	escapa
Velocidad	mucho mayor	mucho menor
Puede optimizar la topología	no	sí

Para redes grandes la retropropagación gana con diferencia. La neuroevolución tiene su sitio en redes pequeñas, en aprendizaje por refuerzo donde no hay una señal de error derivable, y cuando lo que se busca **es** la topología además de los pesos.

8.4 Aprendizaje evolutivo de reglas

Un modelo basado en reglas del tipo

```
SI temperatura ALTA Y humedad BAJA ENTONCES clase = riesgo
```

es legible por una persona, y esa es su razón de ser: en dominios donde hay que justificar la decisión —médico, financiero, legal— un modelo interpretable vale más que uno algo más preciso que no lo sea.

El espacio de las bases de reglas es discreto, enorme y sin gradiente. Es terreno natural para los algoritmos evolutivos, y hay tres formas de codificarlo:

Enfoque	Un individuo es	Ventaja	Problema
Pittsburgh	una base de reglas completa	la aptitud mide lo que interesa	cromosomas largos y variables
Michigan	una sola regla	cromosomas cortos	la aptitud de una regla no mide la base
IRL	una regla por ejecución, iterando	control del proceso	las reglas no se optimizan juntas

El enfoque de **Pittsburgh** evalúa lo correcto —el rendimiento del conjunto— a costa de manejar cromosomas de longitud variable, lo que obliga a operadores de cruce específicos. El de **Michigan** es barato pero arrastra un problema de fondo: una regla individual buena no garantiza que la base lo sea, porque lo que importa es cómo se cubren entre ellas. **IRL**, aprendizaje iterativo de reglas, extrae una regla, marca los ejemplos que cubre y repite; es un voraz, y como todo voraz puede dejar un residuo difícil al final.

En el caso de reglas difusas hay que decidir además qué se optimiza: solo la parte antecedente, solo las funciones de pertenencia, o las dos a la vez. Lo último se llama **aprendizaje simultáneo** y es lo que da mejores resultados, a costa de un espacio de búsqueda mucho mayor.

8.5 Aprendizaje evolutivo de parámetros y modelos

El nivel más alto: dejar que el algoritmo decida también la **forma** del modelo.

- **Selección de modelo**: elegir entre árbol, red neuronal o máquina de vectores soporte, y su configuración.
- **Optimización de hiperparámetros**: profundidad del árbol, número de capas, tasa de aprendizaje, constante de regularización.
- **Búsqueda de arquitecturas neuronales**: qué capas y cómo se conectan.
- **AutoML**: la cadena completa —preprocesado, selección de atributos, modelo e hiperparámetros— optimizada a la vez.

La dificultad propia de este nivel es que el espacio es **mixto y condicional**: mezcla variables continuas, enteras y categóricas, y algunas solo tienen sentido si otra toma cierto valor —el número de capas solo importa si se eligió una red—. Los operadores tienen que respetar esa condicionalidad, o la mitad de los individuos serán configuraciones sin sentido.

Y una advertencia que vale para todo el tema: cada evaluación entrena un modelo, así que el presupuesto es de decenas o centenares de evaluaciones, no de millones. En ese régimen conviene una población pequeña, una búsqueda local que aproveche cada evaluación, y un modelo sustituto que estime la aptitud de los candidatos malos sin llegar a entrenarlos.

8.6 Un experimento bien planteado

Reuniendo lo que la asignatura pide en cada tema, así se compara un algoritmo:

1. **Instancias**: un banco de pruebas público y reconocido, no instancias propias.
2. **Presupuesto**: el mismo número de **evaluaciones** para todos los algoritmos. Comparar por tiempo mezcla el algoritmo con la implementación; comparar por iteraciones favorece al que más hace en cada una.
3. **Repeticiones**: varias ejecuciones con semillas distintas, porque son algoritmos estocásticos.
4. **Estadísticos**: media y desviación típica, no el mejor de las ejecuciones. El mejor de treinta corridas mide la suerte.
5. **Contraste**: pruebas no paramétricas —Wilcoxon para dos algoritmos, Friedman con un test *post hoc* para varios—, porque los resultados no suelen ser normales y las muestras son pequeñas.
6. **Parámetros**: los mismos criterios de ajuste para todos, ajustados sobre instancias distintas de las de la comparación.

El punto 4 es donde más se falla, y el 6 el que más resultados invalida: ajustar el algoritmo propio a fondo y dejar el de referencia con sus valores por defecto produce una ventaja que no es del algoritmo.

Los modelos de aprendizaje evolutivo y su encaje en la computación evolutiva están en Eiben y Smith, 2015 y Du y Swamy, 2016; el planteamiento como optimización y las aplicaciones, en Talbi, 2009 y Pardalos y Resende, 2002.

Capítulo 9

Prácticas y seminarios

El temario práctico de la guía docente: tres prácticas de laboratorio, las prácticas de pizarra y los cinco seminarios. El código de las tres prácticas está en el repositorio `metaheuristics`, con quince algoritmos implementados sobre dos problemas.

9.1 Los dos problemas del banco de pruebas

Las prácticas no cambian de problema en cada entrega: se resuelven los mismos dos con técnicas distintas, que es lo que permite comparar.

Selección de cartera de Markowitz. Repartir un capital entre n activos minimizando el riesgo para una rentabilidad exigida:

$$\min_{w} \; w^{\top}\Sigma\, w \quad \text{sujeto a} \quad \sum_{i} w_i = 1, \quad \sum_{i} \mu_i w_i \geq R, \quad w_i \geq 0$$

con Σ la matriz de covarianzas de los rendimientos y μ el vector de rendimientos esperados. Es continuo y restringido; sin restricciones adicionales tiene solución analítica, y lo que lo hace duro es añadir la **cardinalidad**: obligar a que como mucho k activos tengan peso no nulo convierte el problema en mixto y NP-duro.

Las restricciones se tratan con las estrategias del tema 2: la de suma uno se cierra en la representación normalizando el vector, y la de rentabilidad mínima se penaliza.

Banco de funciones CEC'17. El conjunto de referencia para optimización continua: treinta funciones —unimodales, multimodales, híbridas y compuestas— definidas en $[-100, 100]^D$ con el óptimo desplazado y rotado a propósito.

El desplazamiento y la rotación no son un adorno. Sin ellos el óptimo cae en el origen, y un algoritmo que tienda al centro del dominio acierta sin buscar nada. La rotación además elimina la separabilidad, con lo que optimizar coordenada a coordenada deja de funcionar.

Grupo	Qué mide
Unimodales (1–3)	velocidad de convergencia
Multimodales simples (4–10)	capacidad de escapar de óptimos locales
Híbridas (11–20)	comportamiento con subconjuntos de variables de distinta naturaleza
Compuestas (21–30)	robustez global

El protocolo del banco fija el presupuesto en $10\,000 \cdot D$ evaluaciones y exige 51 ejecuciones por función, informando el error respecto del óptimo conocido. Es la concreción de las reglas experimentales del tema 8.

9.2 P1 · Técnicas de búsqueda local

La primera práctica implementa lo del tema 2 y el principio del 5, que es la base sobre la que se mide todo lo demás.

Algoritmo	Qué aporta
Búsqueda aleatoria	la línea base absoluta: si un algoritmo no la bate, está mal
Búsqueda local del primer mejor	el descenso, con exploración aleatorizada del entorno
Búsqueda local del mejor	el descenso completo, más caro por paso
Multiarranque básico	repetir el descenso desde puntos distintos

Lo que se comprueba aquí, y condiciona el resto del curso:

- La búsqueda local mejora enormemente sobre la aleatoria en pocas evaluaciones y luego **se planta**. La curva de calidad frente a evaluaciones se aplana, y esa meseta es el óptimo local.
- El primer mejor suele batir al mejor con presupuesto fijo, porque hace muchos más pasos con las mismas evaluaciones.
- El multiarranque mejora poco: cada arranque parte de cero e ignora todo lo descubierto.

Sobre la cartera, la búsqueda local en el continuo se implementa con mutación gaussiana sobre los pesos seguida de normalización, que es la forma de mantener la representación cerrada.

9.3 P2 · Poblaciones: genéticos y meméticos

La segunda cubre los temas 3 y 4. Las variantes que se comparan:

Variante	Configuración
AGG	genético generacional con elitismo
AGE	genético estacionario
AM-(10,1.0)	memético, búsqueda local a toda la población cada 10 generaciones
AM-(10,0.1)	memético, al 10 % elegido al azar
AM-(10,0.1mej)	memético, al 10 % mejor

Los tres meméticos aíslan la decisión del tema 4 sobre a cuántos individuos aplicar la búsqueda local, con todo lo demás igual. Es la forma correcta de medirla: si se cambian dos cosas a la vez, el resultado no atribuye la diferencia a ninguna.

Lo que se observa:

- Los meméticos baten a los genéticos puros con claridad, y la diferencia crece con el tamaño del problema.
- Aplicar la búsqueda local a toda la población es lo peor de los tres: gasta el presupuesto en refinar individuos mediocres.
- El estacionario converge antes que el generacional y con más riesgo de estancarse; el elitismo en el generacional no es opcional.

9.4 P3 · Trayectorias: enfriamiento, multiarranque, GRASP e ILS

La tercera cubre el resto del tema 5:

Algoritmo	Qué se compara
Enfriamiento simulado	el esquema de enfriamiento y T_0
Búsqueda multiarranque básica	la línea base de reinicio
GRASP	construcción voraz aleatorizada más descenso
ILS	perturbación sobre la mejor solución
ILS híbrida	perturbación más enfriamiento simulado

El resultado que suele salir es que el ILS y su versión híbrida quedan por delante, porque conservan parte de la estructura de la mejor solución en vez de tirarla en cada reinicio. Y que el enfriamiento simulado depende mucho más de sus parámetros que el resto: mal calibrado queda por debajo del multiarranque.

Aviso sobre el repositorio. La práctica 3 no compila tal como está publicada: su `CMakeLists.txt` incluye un directorio `common/` que nunca llegó a subirse. Está documentado en el README del repositorio y en su lista de fallos conocidos, y ahí se queda: el criterio del repositorio es afirmar los defectos de lo entregado, no reescribirlo después.

9.5 Prácticas de pizarra

La guía las plantea aparte de las de laboratorio, y son las que fuerzan a diseñar en vez de programar. Dos bloques:

Problemas clásicos. Dado un problema —viajante, mochila, coloreado, asignación cuadrática, horarios—, decidir las cuatro piezas del tema 1: representación, función objetivo, operadores y criterio de parada. La parte que más discusión da es la primera, porque arrastra todas las demás.

Problemas con restricciones. El mismo ejercicio cuando el espacio admisible es una fracción del total. Se decide entre representación cerrada, reparación, penalización y rechazo, con el orden de preferencia del tema 2. Un caso típico: en horarios, «dos asignaturas del mismo curso no pueden solaparse» se puede cerrar en la representación si se codifica por franjas, y hay que penalizar si se codifica por asignaturas.

9.6 Seminarios

Seminario	Contenido
S1	ejemplos de resolución con metaheurísticas y software disponible
S2	optimización con metaheurísticas basadas en búsqueda local
S3	optimización con metaheurísticas basadas en poblaciones

Seminario	Contenido
S4	técnicas basadas en trayectorias simples y múltiples
S5	manejo de restricciones

El S1 recorre las bibliotecas que evitan reimplementar lo estándar —jMetal, DEAP, ECJ, Optuna para hiperparámetros—, y conviene tomarlo en serio por una razón concreta: una implementación propia de un algoritmo de referencia mal ajustada convierte cualquier comparación en un resultado falso a favor del algoritmo nuevo, que es el error que el tema 8 señala como el que más resultados invalida.

Las implementaciones de referencia y su discusión están en Talbi, 2009 y Eiben y Smith, 2015; los problemas de cartera y su formulación, en Pardalos y Resende, 2002.

Bibliografía

Alba, E. (2005). *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley & Sons.

Chopard, B., & Tomassini, M. (2018). *An Introduction to Metaheuristics for Optimization*. Springer.

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. The MIT Press.

Du, K.-L., & Swamy, M. N. S. (2016). *Search and Optimization by Metaheuristics*. Springer.

Eiben, A. E., & Smith, J. E. (2015). *Introduction to Evolutionary Computing* (2.ª ed.). Springer.

Pardalos, P. M., & Resende, M. G. C. (2002). *Handbook of Applied Optimization*. Oxford University Press.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley.