- Aceptar vecinos que no mejoran: estos enfoques permiten medidas que degradan la solución actual. Es posible desplazar la cuenca de atracción de un óptimo local dado. El recocido simulado y la búsqueda tabú son representantes populares de esta clase de algoritmos. El recocido simulado fue el primer algoritmo que abordó explícitamente la pregunta "¿por qué deberíamos considerar sólo movimientos cuesta abajo?"

- Cambio de vecindad: Esta clase de enfoques consiste en cambiar la estructura de vecindad durante la búsqueda. Por ejemplo, este enfoque se utiliza en estrategias de búsqueda de vecindarios variables.
- Cambiar la función objetivo o los datos de entrada del problema: En esta clase, el problema se transforma perturbando los datos de entrada del problema, la función objetivo o las restricciones, con la esperanza de resolver más eficientemente el problema original. Este enfoque se ha implementado en la búsqueda local guiada, las estrategias de suavizado y los métodos de ruido. Los dos últimos enfoques pueden verse como enfoques que cambian el panorama del problema a resolver.

2.4 RECOCIDO SIMULADO

El recocido simulado aplicado a problemas de optimización surge del trabajo de S. Kirkpatrick et al. [464] y V. Cerny [114]. En estos trabajos pioneros, SA se ha aplicado a la partición de gráficos y al diseño VLSI. En la década de 1980, SA tuvo un gran impacto en el campo de la búsqueda heurística por su simplicidad y eficiencia en la resolución de problemas de optimización combinatoria. Luego, se ha ampliado para abordar problemas de optimización continua [204.512.596].

SA se basa en los principios de la mecánica estadística según los cuales el proceso de recocido requiere calentar y luego enfriar lentamente una sustancia para obtener una estructura cristalina fuerte. La resistencia de la estructura depende de la velocidad de enfriamiento de los metales. Si la temperatura inicial no es suficientemente alta o se aplica un enfriamiento rápido, se obtienen imperfecciones (estados metaestables). En este caso, el sólido que se enfría no alcanzará el equilibrio térmico a cada temperatura. Los cristales fuertes se desarrollan mediante un enfriamiento cuidadoso y lento. El algoritmo SA simula los cambios de energía en un sistema sometido a un proceso de enfriamiento hasta converger a un estado de equilibrio (estado congelado estacionario). Este esquema fue desarrollado en 1953 por Metropolis [543].

La tabla 2.4 ilustra la analogía entre el sistema físico y el problema de optimización. La función objetivo del problema es análoga al estado energético del sistema. Una solución del problema de optimización corresponde a un estado del sistema.
Las variables de decisión asociadas con una solución del problema son análogas a las posiciones moleculares. El óptimo global corresponde al estado fundamental del sistema. Encontrar un mínimo local implica que se ha alcanzado un estado metaestable.

SA es un algoritmo estocástico que permite, bajo algunas condiciones, la degradación de una solución. El objetivo es escapar de los óptimos locales y así retrasar la convergencia. SA es un algoritmo sin memoria en el sentido de que el algoritmo no

TABLA 2.4 Analogía entre el sistema físico y el
Problema de optimizacion

Sistema físico	Problema de optimizacion
Estado del sistema	Solución
Posiciones moleculares	Variables de decisión
Energía	Función objetiva
Estado fundamental	Solución óptima global
Estado metaestable	Óptimo local
Enfriamiento rápido	Busqueda local
Temperatura	Parámetro de control T
Recocido cuidadoso	Recocido simulado

utilizar cualquier información recopilada durante la búsqueda. A partir de una solución inicial, SA procede en varias iteraciones. En cada iteración, se genera un vecino aleatorio. mueve eso mejorar la función de costos siempre se aceptan. De lo contrario, se selecciona el vecino. con una probabilidad dada que depende de la temperatura actual y de la cantidad de degradación E de la función objetivo. E representa la diferencia en el valor objetivo (energía) entre la solución actual y la vecina generada.
solución. A medida que avanza el algoritmo, la probabilidad de que se acepten dichos movimientos disminuye (figura 2.25). Esta probabilidad sigue, en general, la distribución de Boltzmann:

$$P(mi, T) = mi \quad - \frac{f(s)' - f(s)}{t}$$

Utiliza un parámetro de control, llamado temperatura, para determinar la probabilidad de aceptar soluciones que no mejoran. A un nivel particular de temperatura, muchos ensayos son explorado. Una vez que se alcanza un estado de equilibrio, la temperatura disminuye gradualmente

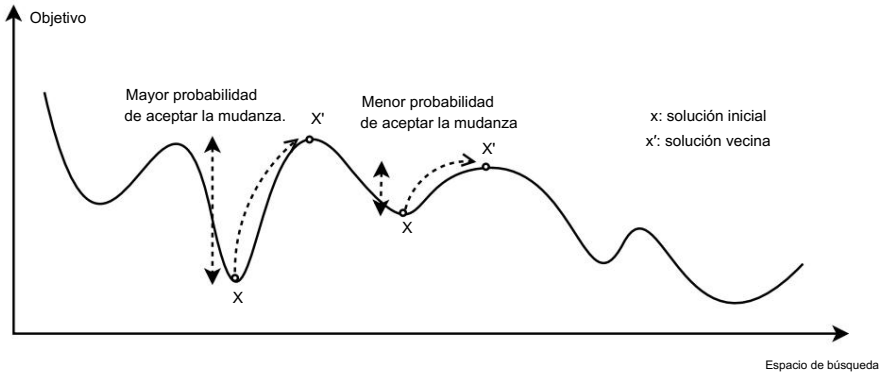


FIGURA 2.25 Recocido simulado que se escapa del óptimo local. Cuanto mayor sea la temperatura, cuanto más significativa sea la probabilidad de aceptar una peor jugada. A una temperatura dada, la Cuanto menor sea el aumento de la función objetivo, más significativa será la probabilidad de aceptar el movimiento. Siempre se acepta una mejor jugada.

de acuerdo con un programa de enfriamiento tal que se acepten pocas soluciones que no mejoren al final de la búsqueda. El algoritmo 2.3 describe la plantilla del algoritmo SA.

Algoritmo 2.3 Plantilla de algoritmo de recocido simulado.

```
Entrada: Programa de enfriamiento.
s = s0 ; / Generación de la solución inicial /
T = Tmáx ; / Temperatura inicial /
Repetir
    Repetir / A una temperatura fija /
        Generar un vecino aleatorio s' ;
        mi = f (s') − f (s) ;
        Si E ≤ 0 Entonces s = s' / Acepta la solución vecina /
        De lo contrario , aceptar s' con una probabilidad e^(−mi/T) ;
    Hasta la condición de equilibrio
    / por ejemplo, un número determinado de iteraciones ejecutadas a cada temperatura T /
    T = g(T ); / Actualización de temperatura /
Hasta que se cumplan los criterios de parada / por ejemplo, T < Tmín /
Resultado: Mejor solución encontrada.
```

Ejemplo 2.23 Ilustración del algoritmo SA. Maximicemos la continuidad función $f(x) = x^3 - 60x^2 + 900x + 100$. Una solución x se representa como una cadena de 5 bits. El barrio consiste en girar un poco al azar. El máximo global de esta función es 01010 (x = 10, f (x) = 4100). El primer escenario parte de la solución. 10011 (x = 19, f (x) = 2399) con una temperatura inicial T0 igual a 500 (Tabla 2.5). El segundo escenario parte de la misma solución 10011 con una temperatura inicial T0 igual a 100 (Tabla 2.6). La temperatura inicial no es lo suficientemente alta y el algoritmo se queda atascado en los óptimos locales.

Además de la solución actual, la mejor solución encontrada desde el comienzo de la búsqueda se almacena. Pocos parámetros controlan el progreso de la búsqueda, que son los temperatura y el número de iteraciones realizadas a cada temperatura.

El análisis teórico de la convergencia asintótica de SA está bien desarrollado [478]. La búsqueda puede modelarse mediante una cadena de Markov, donde el siguiente estado depende sólo

TABLA 2.5 Primer Escenario T = 500 y Solución Inicial (10011)

t	Mover	Solución	F	F	¿Mover?	Nueva solución vecina
500	1	00011	2287	112	Sí	00011
450	3	00111	3803	<0	Sí	00111
405	5	00110	3556	247	Sí	00110
364,5	2	01110	3684	<0	Sí	01110
328	4	01100	3998	<0	Sí	01100
295,2	3	01000	3972	decisión	Sí	01000
265,7	4	01010	4100	<0	Sí	01010
239.1	5	01011	4071	29	Sí	01011
215.2	1	11011	343	3728	No	01011

TABLA 2.6 Segundo Escenario: T = 100 y Solución Inicial (10011). Cuando
La temperatura no es lo suficientemente alta, el algoritmo se atasca

t	Mover solución		F	F	¿Mover?	Nueva solución vecina
100		00011	2287	112	No	10011
90	1	10111	1227	1172	No	10011
81	3	10010	2692	<0	Sí	10010
72,9	5 2	11010	516	2176	No	10010
65,6	4	10000	3236	<0	Sí	10000
59	3	10100	2100	1136	Sí	10000

sobre el estado actual. Existe una garantía de convergencia hacia la solución óptima:

$$\Pr(sM \quad R) \rightarrow 1 \text{ cuando } M \rightarrow \infty$$

donde R representa el conjunto de soluciones óptimas globales y sM la solución en la iteración
M bajo el siguiente programa de enfriamiento lento:

$$Tk = \frac{\hat{W}}{\text{registro k}}$$

donde Ŵ es una constante. En la práctica, tal programa de enfriamiento es inútil porque
es una convergencia asintótica; es decir, la convergencia se obtiene después de un infinito
número de iteraciones. Sin embargo, se necesita mucho más trabajo en el análisis de finitos.
rendimiento del tiempo [267].

Además de los problemas de diseño comunes para las metaheurísticas S, como la definición
del barrio y la generación de la solución inicial, los principales temas de diseño
específicos de SA son

- La función de probabilidad de aceptación: Es el elemento principal de SA que permite
  Se seleccionarán vecinos que no mejoren.
- El programa de enfriamiento: El programa de enfriamiento define la temperatura en cada paso
  del algoritmo. Tiene un papel esencial en la eficiencia y la eficacia
  del algoritmo.

Las siguientes secciones presentan una guía práctica en la definición de la función de probabilidad
de aceptación y el programa de enfriamiento en SA.

2.4.1 Aceptación de movimiento

El sistema puede escapar de los óptimos locales debido a la aceptación probabilística de un
vecino que no mejora. La probabilidad de aceptar un vecino que no mejora es
proporcional a la temperatura T e inversamente proporcional al cambio de la
función objetivo e.

La ley de la termodinámica establece que a la temperatura T , la probabilidad de un aumento en la energía de magnitud, E, está dada por P(E, T) = exp(−E/kt) donde k es una constante conocida como constante de Boltzmann. Entonces, la probabilidad de aceptación de un movimiento que no mejora es

$$P(E,T) = \exp\left(\frac{-\delta E}{kt}\right) > R$$

donde E es el cambio en la función de evaluación, T es la temperatura actual y R es un número aleatorio uniforme entre 0 y 1.

A altas temperaturas, la probabilidad de aceptar movimientos peores es alta. Si T = ∞, se aceptan todos los movimientos, lo que corresponde a un paseo local aleatorio en el paisaje. A bajas temperaturas, la probabilidad de aceptar movimientos peores disminuye. Si T = 0, no se aceptan movimientos peores y la búsqueda es equivalente a la búsqueda local (es decir, escalar una colina). Además, la probabilidad de aceptar un gran deterioro en la calidad de la solución disminuye exponencialmente hacia 0 según la distribución de Boltzmann.

2.4.2 Programa de enfriamiento

El programa de enfriamiento define para cada paso del algoritmo i la temperatura Ti . Tiene un gran impacto en el éxito del algoritmo de optimización SA. De hecho, el rendimiento del SA es muy sensible a la elección del programa de refrigeración.

Los parámetros a considerar al definir un programa de enfriamiento son la temperatura inicial, el estado de equilibrio, una función de enfriamiento y la temperatura final que define los criterios de parada. A continuación se proporciona una guía que trata sobre la inicialización de cada parámetro.

2.4.2.1 Temperatura inicial Si la temperatura inicial es muy alta, la búsqueda será más o menos una búsqueda local aleatoria. De lo contrario, si la temperatura inicial es muy baja, la búsqueda será más o menos una primera mejora del algoritmo de búsqueda local. Por lo tanto, tenemos que equilibrar estos dos procedimientos extremos. La temperatura inicial no debe ser demasiado alta para realizar una búsqueda aleatoria durante un período de tiempo, pero sí lo suficientemente alta como para permitir traslados a un estado casi vecino.

Hay tres estrategias principales que se pueden utilizar para abordar este parámetro:

- Aceptar todos: la temperatura inicial se establece lo suficientemente alta como para aceptar a todos los vecinos durante la fase inicial del algoritmo [464]. El principal inconveniente de esta estrategia es su alto coste computacional.
- Desviación de aceptación: La temperatura inicial se calcula mediante kσ utilizando experimentos preliminares, donde σ representa la desviación estándar de la diferencia entre los valores de las funciones objetivo y k = −3/ln(p) con la probabilidad de aceptación de p, que es mayor que 3σ [392].

• Relación de aceptación: La temperatura inicial se define de manera que la temperatura de aceptación relación de distancia de soluciones mayor que un valor predeterminado a0

$$T0 = \frac{+}{\ln(m1(a0 - 1)/m2 + a0)}$$

donde m1 y m2 son el número de soluciones que se reducirán y aumentarán en los experimentos preliminares, respectivamente, y + es el promedio de los valores de la función objetivo aumentados [2]. Por ejemplo, la temperatura inicial debe inicializarse de tal manera que la tasa de aceptación esté en el intervalo [40%, 50%].

2.4.2.2 Estado de equilibrio Para alcanzar un estado de equilibrio a cada temperatura, se debe aplicar un número suficiente de transiciones (movimientos). La teoría sugiere que el número de iteraciones a cada temperatura podría ser exponencial al tamaño del problema, lo cual es una estrategia difícil de aplicar en la práctica. El número de iteraciones debe establecerse de acuerdo con el tamaño de la instancia del problema y particularmente proporcional al tamaño de la vecindad |N(s)|. El número de transiciones visitadas puede ser el siguiente:

• Estática: en una estrategia estática, el número de transiciones se determina antes de que comience la búsqueda. Por ejemplo, se explora una proporción dada y de la vecindad N(s) . Por lo tanto, el número de vecinos generados a partir de una solución s es y · |N(s)|. Cuanto más significativa sea la relación y, mayor será el coste computacional y mejores serán los resultados.

• Adaptativo: El número de vecinos generados dependerá de las características de la búsqueda. Por ejemplo, no es necesario alcanzar el estado de equilibrio a cada temperatura. Se pueden utilizar algoritmos de recocido simulados en desequilibrio: el programa de enfriamiento se puede aplicar tan pronto como se genere una solución vecina mejorada. Esta característica puede resultar en la reducción del tiempo de cálculo sin comprometer la calidad de las soluciones obtenidas [107].

Se puede utilizar otro enfoque adaptativo que utilice tanto las peores como las mejores soluciones encontradas en el bucle interno del algoritmo. Sea fl (resp. fh) el valor de función objetivo más pequeño (resp. más grande) en el bucle interno actual. El siguiente número de transiciones L se define de la siguiente manera:

$$L = LB + \quad LB \cdot F-$$

donde x es el mayor entero menor que x, F− = 1 − exp(−(fh − fl)/fh), y LB es el valor inicial del número de transiciones [25].

2.4.2.3 Enfriamiento En el algoritmo SA, la temperatura disminuye gradualmente tal que

$$Ti > 0, \quad i$$

y

$$\lim_{i \to \infty} Ti = 0$$

Siempre existe un compromiso entre la calidad de las soluciones obtenidas y la velocidad del programa de enfriamiento. Si se disminuye lentamente la temperatura se obtienen mejores soluciones pero con un tiempo de cálculo más significativo. La temperatura T se puede actualizar de diferentes formas:

- Lineal: en el programa lineal trivial, la temperatura T se actualiza de la siguiente manera: $T = T - \beta$, donde β es un valor constante especificado. Por lo tanto, tenemos

$$Ti = T0 - i \times \beta$$

donde Ti representa la temperatura en la iteración i. •

Geométrico: En el cronograma geométrico, la temperatura se actualiza usando el fórmula

$$T = \alpha T$$

donde $\alpha \in]0, 1[$. Es la función de enfriamiento más popular. La experiencia ha demostrado que α debería estar entre 0,5 y 0,99.

- Logarítmico: Se utiliza la siguiente fórmula:

$$Ti = \frac{T0}{\log(yo)}$$

Este programa es demasiado lento para ser aplicado en la práctica, pero tiene la propiedad de demostrar la convergencia hacia un óptimo global [303].

- Disminución muy lenta: la principal desventaja de un programa de enfriamiento es el uso de una gran cantidad de iteraciones a unas pocas temperaturas o una pequeña cantidad de iteraciones a muchas temperaturas. Un programa de enfriamiento muy lento como

$$Ti+1 = \frac{Ti}{1 + \beta Ti}$$

puede usarse [521], donde $\beta = T0 - TF/(L - 1)T0TF$ y TF es la temperatura final. Sólo se permite una iteración a cada temperatura en esta función decreciente muy lenta.

- No monótono: los programas de enfriamiento típicos utilizan temperaturas monótonas. Se pueden sugerir algunos esquemas de programación no monótonos en los que la temperatura aumenta nuevamente [390]. Esto fomentará la diversificación en el espacio de búsqueda. Para algunos tipos de entornos de búsqueda, el cronograma óptimo es no monótono [352].

- Adaptativo: La mayoría de los programas de enfriamiento son estáticos en el sentido de que el programa de enfriamiento está definido completamente a priori. En este caso, el programa de enfriamiento es "ciego" a las características del panorama de búsqueda. En un programa de enfriamiento adaptativo, la tasa decreciente es dinámica y depende de cierta información obtenida durante la búsqueda [402]. Se puede utilizar un programa de enfriamiento dinámico donde se lleva a cabo una pequeña cantidad de iteraciones a altas temperaturas y una gran cantidad de iteraciones a bajas temperaturas.

2.4.2.4 Condición de detención En cuanto a la condición de detención, la teoría sugiere una temperatura final igual a 0. En la práctica, se puede detener la búsqueda cuando la probabilidad de aceptar un movimiento es insignificante. Se podrán utilizar los siguientes criterios de parada:

- Alcanzar una temperatura final TF es el criterio de parada más popular. Esta temp-
  La temperatura debe ser baja (p. ej., Tmin = 0,01).
- Lograr un número predeterminado de iteraciones sin mejorar la mejor solución encontrada [675].
  - Lograr un número

predeterminado de veces que se acepte un porcentaje de vecinos en cada temperatura; es decir, un contador aumenta en 1 cada vez que se completa una temperatura con un porcentaje menor de movimientos aceptados que un límite predeterminado y se reinicia a 0 cuando se encuentra una nueva mejor solución. Si el contador alcanza un límite predeterminado R, el algoritmo SA se detiene [420].

En comparación con la búsqueda local, SA sigue siendo simple y fácil de implementar. Da buenos resultados para un amplio espectro de problemas de optimización: los históricos como el diseño de TSP y VLSI en diferentes dominios de aplicación. Se puede encontrar una buena encuesta sobre SA en las referencias [1,489,733].

2.4.3 Otros métodos similares

En la literatura se han propuesto otros métodos similares de recocido simulado, como la aceptación de umbral, el algoritmo del gran diluvio, el viaje de registro a registro y los algoritmos demoníacos (fig. 2.26). El principal objetivo en el diseño de estos algoritmos inspirados en SA es acelerar la búsqueda del algoritmo SA sin sacrificar la calidad de las soluciones.

2.4.3.1 Aceptación de umbral La aceptación de umbral puede verse como la variante determinista del recocido simulado [228]. TA escapa de los óptimos locales al aceptar soluciones que no son peores que la solución actual por más de un umbral dado Q. Una función de aceptación determinista se define de la siguiente manera:

$$Pi((s, s')) = \begin{matrix} 1 \text{ si } Qi \geq (s, s') \\ 0 \text{ en caso contrario} \end{matrix}$$

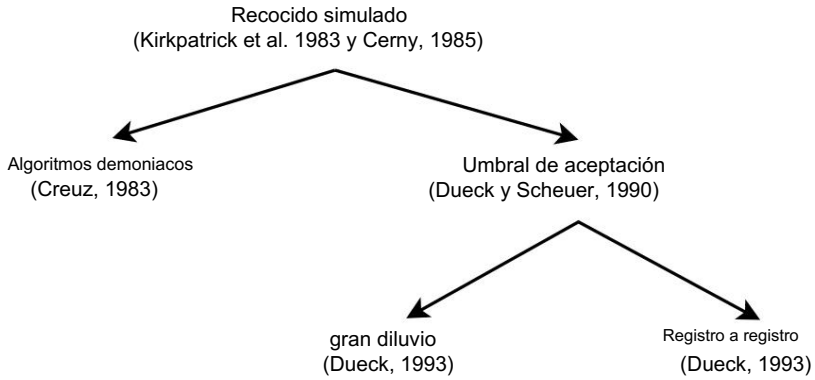


FIGURA 2.26 Genealogía de algoritmos basados en recocido simulado.

donde Qi es el valor umbral en la iteración i y (s, s′) es el cambio en la función de evaluación entre la solución actual sy las soluciones vecinas . El parámetro umbral en′TA opera de manera similar a la temperatura en el recocido simulado.
El algoritmo 2.4 describe la plantilla del algoritmo TA. El número de vecinos generados en cada iteración se fija a priori. El umbral Q se actualiza siguiendo cualquier programa de recocido.

Algoritmo 2.4 Plantilla de algoritmo de aceptación de umbral.

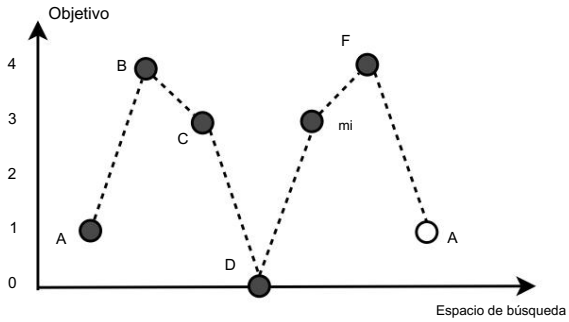
```
Entrada: Umbral de recocido.
s = s0 ; /      Generación de la solución inicial
    / Q = Qmax ; /      Umbral inicial
    /
    Repetir Repetir /      En un umbral fijo      /
      Generar un vecino aleatorio ′      norte(s) ;
        mi = f (s ′ ) − f (s) ; Si
      E ≤ Q Entonces s = s Hasta la ′ /    Aceptar la solución vecina      /
    condición de equilibrio, por ejemplo,
        un número determinado de iteraciones ejecutadas en cada umbral Q/Q =          /
    g(Q) ; /      Actualización de umbral      /
 Hasta que se cumplan los criterios de      por ejemplo, Q ≤ Qmín    /
parada / Salida: Se encontró la mejor solución.
```

TA es un algoritmo rápido en comparación con SA porque la generación de números aleatorios y funciones exponenciales consume una cantidad significativa de tiempo computacional. La literatura informa algunas mejoras de rendimiento en comparación con el algoritmo de recocido simulado en la resolución de problemas de optimización combinatoria como el problema del viajante [228,565].

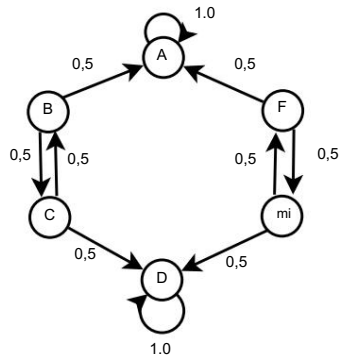
En términos de convergencia asintótica, las propiedades teóricas de TA son similares a las del algoritmo SA [28].

El umbral Q se actualiza según un programa de recocido. Debe establecerse como una función escalonada determinista no creciente en el número de iteraciones i. El umbral disminuye en cada iteración y luego alcanza el valor 0 después de un número determinado de iteraciones.

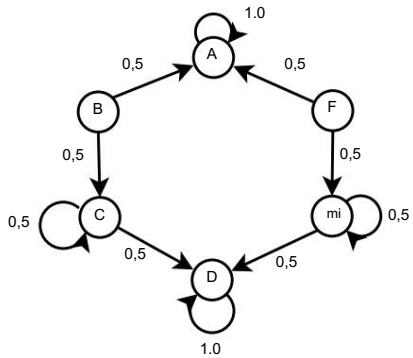
Ejemplo 2.24 Programa de umbral no monótono. Este sencillo ejemplo ilustrará que el cronograma umbral óptimo para resolver un problema dado puede ser no monótono [390]. Consideremos el panorama unidimensional de un problema que se muestra en la figura 2.27. El espacio de búsqueda se compone de seis soluciones y cada solución tiene dos vecinos. El número de iteraciones del algoritmo se fija en M. El programa óptimo se define como aquel que maximiza la probabilidad de encontrar la solución óptima global a partir de cualquier solución inicial aleatoria. Para muchos valores de M, se ha demostrado que el programa de umbral óptimo no es monótono con umbrales negativos. Por ejemplo, para M = 9, el programa óptimo es (0, −3, 3, −1, −3, 3, 3, −1) y la probabilidad de encontrar el óptimo global es 0,8372.



(a) Panorama de búsqueda



(b) Probabilidades de transición para T = 2



(c) Probabilidades de transición para T =0

FIGURA 2.27 Programa de umbral óptimo no monótono [390]. (a) Representa el panorama de búsqueda del problema. (b) (Resp. (c)) representa las probabilidades de transición para T = 2(T = 0).

Ejemplo 2.25 Umbral adaptativo no monótono: la aceptación del viejo soltero.
En el antiguo algoritmo de aceptación de solteros, el umbral cambia dinámicamente (hacia arriba o hacia abajo) según el historial de búsqueda [390]. Después de cada fracaso en la aceptación de una nueva solución, el criterio de aceptación se relaja aumentando ligeramente el umbral Qi 14: Qi+1 = Qi + incr(Qi). Por lo tanto, después de muchos fracasos sucesivos, el umbral será lo suficientemente grande como para escapar de los óptimos locales. Por el contrario, después de cada aceptación de una solución vecina, el umbral se reduce de modo que el algoritmo se vuelve más agresivo al avanzar hacia un óptimo local Qi+1 = Qi − decr(Qi).
Las funciones decr e incr se basan en los siguientes factores:

- El tamaño del barrio |N|. Este valor tiene un impacto en la "alcanzabilidad" entre soluciones (es decir, el diámetro y la multiplicidad de caminos en el espacio de búsqueda). •
- La antigüedad de la solución actual, que es el número de iteraciones desde la aceptación del último movimiento. Valores más grandes de edad implican una mayor probabilidad de que la solución actual sea un óptimo local y que el umbral deba aumentar. • La
- cantidad de tiempo restante M − i, donde M representa el número total de iteraciones e i la iteración actual. La actualización del umbral dependerá de la proporción del tiempo utilizado i/M.
- El valor umbral actual Qi .

El algoritmo 2.5 describe la plantilla del algoritmo OBA.

Algoritmo 2.5 Plantilla del algoritmo de aceptación de solteros antiguos.

```
s = s0 ; /     Generación de la solución inicial     /
Q0 = 0 ; /     Umbral inicial     / edad
= 0 ;
 Para i = 0 a M − 1 Hacer
     Generar un vecino aleatorio
     Si f (s ′) < f (s) + Qi Entonces s = s ′norte(s) ; ; edad = 0 ;
     De lo contrario edad = edad + 1;
     Qi+1 = (edad/a)^b − 1 × × 1 − (i/METRO)^c / Actualización de umbral /
final para
Resultado: Mejor solución encontrada.
```

El umbral se actualiza de la siguiente manera:

$$Qi+1 = \left(\frac{edad}{a}\right)^{b} - 1 \times \times 1 - \left(\frac{i}{METRO}\right)^{c}$$

Siempre que edad = 0, el algoritmo establece el umbral en el valor más negativo permitido (es decir, -), lo que le da al algoritmo más "ambición" para mejorar rápidamente. por negativo

14Esta es la motivación para el nombre “aceptación del viejo soltero”.

valores del umbral, el algoritmo puede preferir un buen movimiento de mejora a un movimiento de mejora aleatorio. Luego, el umbral aumenta desde este valor negativo hasta que se produce la aceptación del siguiente movimiento.

Para edad > 0, la regla de actualización permite que la tasa de crecimiento umbral aumente con la edad. Los parámetros a, byc brindan la capacidad de ajustar la tasa de crecimiento incr(Qi) de la siguiente manera: representa la granularidad de la actualización, a ajusta el umbral de crecimiento mediante un factor multiplicativo, b permite una tasa de crecimiento de ley de potencia , y c ajusta un factor de "dumping" heurístico 1 − (i/M) utilizado para escalar la magnitud de Qi como i −→ M.

2.4.3.2 Viaje de registro a registro Este algoritmo también es un algoritmo de optimización determinista inspirado en el recocido simulado [229]. El algoritmo acepta una solución vecina que no mejora con un valor objetivo menor que RECORD menos una desviación D. RECORD representa el mejor valor objetivo de las soluciones visitadas durante la búsqueda. El límite disminuye con el tiempo a medida que mejora el valor objetivo REGISTRO de la mejor solución encontrada. El algoritmo 2.6 describe la plantilla del algoritmo RRT.

Algoritmo 2.6 Plantilla del algoritmo de viaje de registro a registro.

```
Entrada: Desviación D > 0. s =
s0 ; /     Generación de la solución inicial     / RECORD =
f (s) ; /     Iniciando RECORD     / Repetir Generar un
vecino
     aleatorio s Si f (s Si RECORD > f (s  ′ ;
     Hasta  ′) < REGISTRO + D Entonces s = s          ′ ; /     Aceptar la solución vecina     /
     que se cumplan los          ′) Entonces REGISTRO = f (s      ′) ; /     Actualización de REGISTRO     /
 criterios de detención Salida: Se
encontró la mejor solución.
```

El algoritmo RRT tiene la ventaja de depender de un solo parámetro, el valor de DESVIACIÓN. Un valor pequeño para la desviación producirá malos resultados en un tiempo de búsqueda reducido. Si la desviación es alta, se producen mejores resultados después de un tiempo computacional importante.

2.4.3.3 Algoritmo del gran diluvio El algoritmo del gran diluvio fue propuesto por Dueck en 1993 [229]. La principal diferencia con el algoritmo SA es la función de aceptación determinista de soluciones vecinas. La inspiración del algoritmo GDA proviene de la analogía de la dirección que tomaría un escalador en un gran diluvio para mantener sus pies secos. Encontrar el óptimo global de un problema de optimización15 puede considerarse como encontrar el punto más alto de un paisaje. Como llueve incesantemente y sin cesar, el nivel del agua aumenta. El algoritmo nunca avanza más allá del

15 Supongamos aquí un problema de maximización.

nivel de agua. Explorará el área descubierta del paisaje para alcanzar el óptimo global.

El algoritmo 2.7 describe la plantilla del algoritmo GDA en un contexto de minimización. Se acepta una solución vecina generada si el valor absoluto de la función objetivo es menor que el valor límite actual, denominado nivel. El valor inicial del nivel es igual a la función objetivo inicial. El parámetro de nivel en GDA funciona de manera similar a la temperatura en SA. Durante la búsqueda, el valor del nivel disminuye monótonamente. La disminución de la reducción es un parámetro del algoritmo.

Algoritmo 2.7 Plantilla del algoritmo del gran diluvio.

Entrada: Nivel L.
s = s0 ; // Generación de la solución inicial
Elige la velocidad de lluvia ARRIBA ; / Elija el nivel inicial ARRIBA > 0 /
del agua NIVEL ; Repita Generar un vecino
aleatorio s ′ ; Si f (s ′) < NIVEL Entonces s = s ′ Acepta la solución vecina / NIVEL = NIVEL − ARRIBA ; / actualiza el nivel del agua /
Hasta que se cumplan los criterios de parada. Salida: Se encontró la mejor solución.

El algoritmo del gran diluvio necesita ajustar un solo parámetro, el valor UP que representa la velocidad de la lluvia. La calidad de los resultados obtenidos y el tiempo de búsqueda dependerán únicamente de este parámetro. Si el valor del parámetro UP es alto, el algoritmo será rápido pero producirá resultados de mala calidad. De lo contrario, si el valor UP es pequeño, el algoritmo generará resultados relativamente mejores en un tiempo computacional mayor. Un ejemplo de regla que se puede utilizar para definir el valor del parámetro UP puede ser el siguiente [229]: un valor menor al 1% de la brecha promedio entre la calidad de la solución actual y el nivel del agua.

2.4.3.4 Algoritmos Demon Desde 1998 se han propuesto muchas metaheurísticas S basadas en el algoritmo demon (DA) (ver Ref. [165]) [824]. El algoritmo demoníaco es otro algoritmo basado en recocido simulado que utiliza funciones de aceptación computacionalmente más simples.

El algoritmo 2.8 describe el algoritmo del demonio principal. La función de aceptación se basa en el valor energético del demonio (crédito). El demonio se inicializa con un valor dado D. Se acepta una solución no mejorada si el demonio tiene más energía (crédito) que la disminución del valor objetivo. Cuando un algoritmo DA acepta una solución de mayor valor objetivo, el valor de cambio del objetivo se acredita al demonio. De la misma manera, cuando un algoritmo DA acepta una solución mejoradora, la disminución del valor objetivo se debita del demonio.

Algoritmo 2.8 Plantilla del algoritmo del demonio.

```
Entrada: valor inicial del demonio
D s = s0 ; /    Generación de la solución inicial     /
Repetir
      Generar un vecino aleatorio        ′ ;
        mi = f (s  ′) − f (s) ;
      Si E ≤ D Entonces
            ′s = s     ; /    Aceptar la solución vecina     /
            re = re - mi ; / Hasta     Actualización del valor del demonio    /
 que se cumplan los criterios de detención
Salida: Se encontró la mejor solución.
```

La función de aceptación de los algoritmos demoníacos es computacionalmente más simple que en SA. Requiere una comparación y una resta, mientras que en SA requiere una función exponencial y una generación de un número aleatorio. Además, los valores de los demonios varían dinámicamente en el sentido de que la energía (crédito) depende de las soluciones visitadas (cadena de Markov) durante la búsqueda, mientras que en SA y TA la temperatura (umbral) no se reduce dinámicamente. En efecto, la energía absorbida y liberada por el demonio depende principalmente de las soluciones aceptadas.

En la literatura se pueden encontrar diferentes variantes del algoritmo DA [610,824]. Se diferencian por el programa de recocido de la función de aceptación:

- Algoritmo de demonio acotado: este algoritmo impone un límite superior D0 para el crédito del demonio. Por lo tanto, una vez que el crédito del demonio es mayor que el límite superior, no se recibe ningún crédito incluso si se generan soluciones mejoradas.

- Algoritmo del demonio recocido: En este algoritmo, se utiliza un programa de recocido similar al de recocido simulado para disminuir el crédito del demonio. El crédito del demonio jugará el mismo papel que la temperatura en el recocido simulado.

- Algoritmo demoníaco acotado y aleatorio: se introduce un mecanismo de búsqueda aleatorio en el algoritmo BDA. El crédito del demonio se reemplaza con una variable aleatoria gaussiana normal, donde la media es igual al crédito del demonio (Dm) y una desviación estándar especificada Dsd. Por tanto, la energía asociada con el demonio será D = Dm+ ruido gaussiano.
- Algoritmo demoníaco recocido aleatorio: se introduce el mismo mecanismo de búsqueda aleatoria del algoritmo RBDA que en el algoritmo ADA.

La tabla 2.7 ilustra las especificidades de las diferentes variantes de los algoritmos demoníacos.

En comparación con el recocido simulado, la aplicación de algoritmos demoníacos a problemas académicos y de la vida real muestra una calidad de resultados competitiva en un tiempo de búsqueda reducido [610,824]. Además, son muy fáciles de diseñar e implementar y necesitan ajustar pocos parámetros.

TABLA 2.7 Variantes de algoritmos demoníacos y sus partes específicas

Algoritmo	Especificidad
BDA	Valor inicial del demonio (límite superior): D0
	Actualización del valor del demonio: si D > D0, entonces D = D0
ADA	Actualización del valor del demonio: cronograma de recocido
RBDA y RADA	Valor inicial del demonio: media Dm
	Función de aceptación: D = Dm+ ruido gaussiano
	Actualización del valor del demonio: Dm = Dm − E

2.5 BÚSQUEDA TABÚ

Glover propuso el algoritmo de búsqueda tabú [323]. En 1986, señaló la aleatorización controlada en SA para escapar de los óptimos locales y propuso un algoritmo determinista [322]. En un trabajo paralelo, Hansen [364] propuso un enfoque similar denominado “ascenso más pronunciado/descenso más suave”. En la década de 1990, el algoritmo de búsqueda tabú se hizo muy popular para resolver problemas de optimización de forma aproximada.
Hoy en día es una de las metaheurísticas S más extendidas. El uso de la memoria, que almacena información relacionada con el proceso de búsqueda, representa la característica particular de la búsqueda tabú.

TS se comporta como un algoritmo LS más pronunciado, pero acepta soluciones que no mejoran para escapar de los óptimos locales cuando todos los vecinos son soluciones que no mejoran. Por lo general, todo el vecindario se explora de manera determinista, mientras que en SA se selecciona un vecino aleatorio. Al igual que en la búsqueda local, cuando se encuentra un vecino mejor, reemplaza la solución actual. Cuando se alcanza un óptimo local, la búsqueda continúa seleccionando un candidato peor que la solución actual. La mejor solución en el vecindario se selecciona como la nueva solución actual incluso si no mejora la solución actual.
La búsqueda tabú puede verse como una transformación dinámica del barrio. Esta política puede generar ciclos; es decir, las soluciones visitadas anteriormente podrían seleccionarse nuevamente.

Para evitar ciclos, TS descarta los vecinos que han sido visitados previamente. Memoriza la trayectoria de búsqueda reciente. La búsqueda tabú gestiona una memoria de las soluciones o movimientos aplicados recientemente, lo que se denomina lista tabú. Esta lista tabú constituye la memoria a corto plazo. En cada iteración de TS, se actualiza la memoria a corto plazo.
Almacenar todas las soluciones visitadas requiere tiempo y espacio. De hecho, tenemos que comprobar en cada iteración si una solución generada no pertenece a la lista de todas las soluciones visitadas.
La lista tabú suele contener un número constante de movimientos tabú. Normalmente, los atributos de los movimientos se almacenan en la lista tabú.

Al introducir el concepto de funciones de solución o funciones de movimiento en la lista tabú, se puede perder cierta información sobre la memoria de búsqueda. Podemos rechazar soluciones que aún no se han generado. Si un movimiento es “bueno”, pero es tabú, ¿aun así lo rechazamos? La lista tabú puede ser demasiado restrictiva; una solución no generada puede estar prohibida.
Sin embargo, para algunas condiciones, llamadas criterios de aspiración, se pueden aceptar soluciones tabú. Las soluciones vecinas admisibles son aquellas que no son tabú o cumplen los criterios de aspiración.