10. Algoritmo Simulated Annealing

Originalmente descrito por Kirkpatrick, Gellat y Vecci [17] para resolver problemas discretos NP-Hard, es un método de optimización heurístico iterativo que intenta obtener la solución buscada mediante la minimización de una función objetivo. Después de un número concreto de iteraciones consigue una solución suficientemente buena. Es ideal para funciones no monótonas donde otros métodos pueden quedarse atrapados en algún mínimo local.

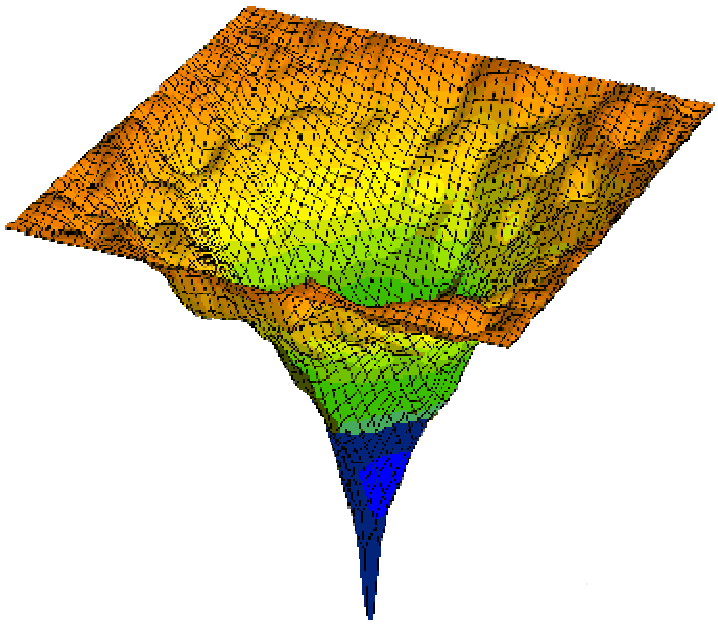

Figura 33 Función no monótona con múltiples mínimos locales.

El método toma su nombre del ordenamiento natural que experimentan las moléculas de una aleación metálica al irse enfriando muy lentamente hasta formar un cristal perfecto. Si la pérdida de temperatura no es lo suficientemente gradual aparecen desperfectos que quedan atrapados en la red cristalina. Estos desperfectos equivaldrían a los mínimos locales.

La idea consiste en realizar una búsqueda aleatoria sobre el espacio de soluciones posibles de un determinado problema. Cada solución es evaluada al ser computada mediante una Función Objetivo (FO). La solución final es aquella que ha conseguido un valor mínimo en la FO. Simulated annealing (SA) no es la única técnica para resolver problemas de optimización, existen otros métodos de búsqueda aleatoria como Tabu Search [18], Algoritmos Genéticos [19], Particle Swarm [20], etc.

10.1 Principio de funcionamiento

SA es un algoritmo que sigue las condiciones de enfriamiento de un material cuando se vuelve sólido desde estado gaseoso con una reducción quasi-estática de su temperatura. Durante el enfriamiento la temperatura va descendiendo y la energía térmica media de las partículas (soluciones) también se va reduciendo hasta el momento en el que alcanzan el mínimo. Al final el material es sólido con la mínima energía estructural. La implementación práctica fue formulada Huang, Romeo y Sangiovanni-Vincentelli [21], donde cinco elementos están presentes en la búsqueda heurística:

1. La temperatura inicial que marca el punto de inicio del sistema.
2. Un criterio para el descenso de la temperatura para obtener el valor de la siguiente iteración.
3. Un criterio de aceptación dependiente de la temperatura de forma que una solución pueda ser aceptada aunque sea peor que alguna previa. Esto permite que el algoritmo no se quede atrapado en un mínimo local. A bajas temperaturas, únicamente se aceptan soluciones mejores de forma que el algoritmo se vuelve gradencial al final del proceso.
4. Un criterio de equilibrio que decida cuándo se va a computar la nueva temperatura.
5. Una condición de stop basada en que el valor de la función sea lo suficientemente bajo o que la temperatura haya descendido lo suficiente.

A continuación se presenta el algoritmo en el siguiente pseudo-código.

```
T=Tinit; Accepted=0; Rejected=0;
While(Frozen_condition(T, Accepted, Rejected))
     S_next=Select_Solution(S_act,T);
     C_next=Eval_objective_func(S_next)
     If(Accept(C_next,C_act,T)=True) then
          S_act= S_next; Cact=C_next;
          Accepted++;
     Else
          Rejected++;
     End if;
     If(Equilibrium_condition(Accepted,Rejected))=True) then
          T=Decrease(T); Accepted=0; Rejected=0;
     End if;
End while;
```

La salida proporcionada por el algoritmo es la mejor solución encontrada que se encuentra almacenada en la variable S_{act}. **Tinit** es la temperatura inicial mientras que T es la temperatura en la iteración actual que controla el proceso de optimización.

Frozen_condition es una función que detiene el algoritmo cuando no se encuentra ningún avance a baja temperatura. **Select_Solution** es una función que genera una nueva solución compatible con el problema, normalmente a partir de la última solución procesada. **Eval_objective_func** computa el coste de la nueva solución en la función objetivo. Este coste, en términos del algoritmo, representa la energía de la solución actual. **Accept** es el núcleo del algoritmo. La nueva solución será aceptada si su coste es menor que el de la solución anterior (gradencial) y en caso contrario según una función de probabilidad que depende de la temperatura actual. Esto permite al algoritmo escapar de los mínimos locales y poder seguir buscando el mínimo global. **Equilibrium_condition** detecta cuándo la energía media de las soluciones es estable para permitir a la función **Decrease** que compute el nuevo valor de la temperatura.

Se ha comprobado que en condiciones ideales, con un descenso gradual y muy suave de la temperatura y una gran cantidad de pruebas en cada nivel de temperatura que garanticen la estabilidad del sistema, la solución final alcanza el mínimo global de la función objetivo. La principal ventaja de utilizar este algoritmo es que es posible escapar de los mínimos locales tal y como ya ha sido comentado anteriormente. Debido a la naturaleza periódica de las ecuaciones, el uso de algoritmos gradenciales no es viable. Por el contrario, el coste computacional requerido para alcanzar una solución es muy alto. Esto hace inviable el uso de sistemas que vayan calculando las soluciones online.

En el planteamiento propuesto, las soluciones están almacenadas como datos de tablas que han sido calculados previamente offline. Es necesario mencionar que la implementación del algoritmo es un programa relativamente corto y que no requiere mucha memoria para su ejecución. Sin embargo, requiere muchos ciclos de ejecución para completar una búsqueda dada su naturaleza heurística. Aunque puede ejecutarse en un PC de sobremesa es preferible utilizar una arquitectura hardware paralelo para poder lanzar varias veces el mismo proceso y explorar distintas condiciones del problema de forma simultánea. En este caso concreto se ha utilizado el servicio de la Universidad de Sevilla de procesamiento paralelo para obtener parte de los cálculos.

A continuación se muestran algunos detalles técnicos de la implementación:

- **Tinit** en aplicaciones prácticas es un valor fijo. En este caso se le ha dado el valor 1000.

- **Select_Solution** es una de las funciones más importantes del algoritmo. Básicamente es una función que genera una nueva solución α' a partir de la solución previa α dependiendo del valor de la temperatura en ese instante T. La nueva solución se obtiene con la siguiente expresión:

$$\alpha' = \alpha + T/Tinit*rnd()$$

  Donde rnd() proporciona un número aleatorio en el rango $[0,\pi/2]$. La nueva solución debe estar en una hiperesfera de radio T/Tinit. Otra restricción viene impuesta por el tiempo mínimo que es necesario respetar entre dos conmutaciones seguidas de los interruptores. El intervalo temporal de seguridad lo proporciona el fabricante y se convierte a radianes una vez fijada la frecuencia del armónico fundamental que se quiere conseguir.
- **Eval_objective_func** computa el valor de la función objetivo para la solución actual α. Su salida C es el coste de esa solución.
- **Accept** es la otra parte esencial del algoritmo. Decide si se acepta la nueva solución comparándola con la solución previa. La expresión es:

$$\text{Accept} = \text{TRUE, if } (C' < C)$$
$$\text{Accept} = \text{TRUE, if } (C' > C) \text{ and } rnd() < e^{-(C'-C)/(T*K_0)}$$
$$\text{Accept} = \text{FALSE in the rest of the cases}$$

  Donde rnd() genera números aleatorios en el rango [0,1], T es la temperatura en la iteración actual y K_0 es un parámetro de procesado que debe ser ajustado dependiendo de la naturaleza concreta del problema considerado. En general se puede fijar como Cinit/Tinit donde Cinit es el coste de una solución obtenida en alta temperatura.

- Criterio de disminución de la temperatura. Teóricamente debería ser implementado usando una función exponencial de forma que a baja temperatura vaya decreciendo a menor velocidad que cuando la temperatura es alta. En la implementación realizada se ha modelado como una simple escala lineal con coeficientes diferentes según tres rangos de temperatura:

$$0.7 \quad \text{Si} \quad T > 100$$
$$0.9 \quad \text{Si} \quad 10 < T < 100$$
$$0.95 \quad \text{Si} \quad T < 10$$

- Condición de equilibrio. Se alcanza cuando el valor medio de los costes de las distintas soluciones encontradas para una determinada temperatura es estable. Cuando se alcanza dicho valor medio se reduce la temperatura de nuevo.
- Condición de parada. Es el criterio final necesario para interrumpir al algoritmo. En esta implementación se ha establecido como criterio el que T alcance una determinada temperatura. Otro podría ser que después de un determinado número de intentos no se consiga encontrar una solución mejor.