

Algoritmo de recocido simulado (cristalización simulada, templeado simulado o enfriamiento simulado)

Un algoritmo de ascensión de colinas que nunca hace movimientos cuesta abajo hacia estados con un valor inferior (o más alto dependiendo si se quiere maximizar o minimizar) garantiza ser incompleto, porque puede estancarse en un máximo local. En contraste, un camino puramente aleatorio es completo, pero sumamente ineficaz. Por lo tanto, parece razonable intentar combinar la ascensión de colinas con un camino aleatorio de algún modo que produzca tanto eficacia como completitud.

El algoritmo de recocido simulado es ese algoritmo. En metalurgia, el temple es el proceso utilizado para templar o endurecer metales y cristales calentándolos a una temperatura muy alta y luego gradualmente enfriarlos, así permite al material fundirse en un estado cristalino de baja energía. Para entender el algoritmo, cambiemos el punto de vista de la ascensión de colinas al gradiente descendiente (es decir, minimizando el costo). Imaginemos la tarea de colocar una pelota de ping-pong en la grieta más profunda en una superficie desigual. Si dejamos caer la pelota en un lugar arbitrario, se podría detener en un mínimo local. El truco es agitar con bastante fuerza para echar la pelota de los mínimos locales, pero no tanta para sacarla del mínimo global. La solución del recocido simulado debe comenzar sacudiendo con fuerza (a temperatura muy alta) y luego gradualmente reducir la intensidad de la sacudida (a temperatura baja).

El bucle interno del algoritmo de recocido simulado es bastante similar a la ascensión en colinas. En vez de escoger al mejor movimiento, sin embargo, escoge un movimiento aleatorio. Si el movimiento mejora la situación, es siempre aceptado. Por otra parte, el algoritmo acepta el movimiento con una probabilidad menos que uno. La probabilidad disminuye exponencialmente con la maldad de movimiento (la cantidad ΔE por la que se empeora la evaluación). La probabilidad también disminuye cuando la temperatura T baja: los malos movimientos son más probables al comienzo cuando la temperatura es alta, y se hacen más improbables cuando T disminuye. Se puede demostrar que si el esquema disminuye T bastante despacio, el algoritmo encontrará un óptimo global con probabilidad cercana a uno.

función TEMPLE-SIMULADO(*problema*, *esquema*) **devuelve** un estado solución

entradas: *problema*, un problema

esquema, una aplicación desde el tiempo a «temperatura»

variables locales: *actual*, un nodo

siguiente, un nodo

T , una «temperatura» controla la probabilidad de un paso hacia abajo

$actual \leftarrow \text{HACER-NODO}(\text{ESTADO-INICIAL}[\textit{problema}])$

para $t \leftarrow 1$ **a** ∞ **hacer**

$T \leftarrow \textit{esquema}[t]$

si $T = 0$ **entonces devolver** *actual*

siguiente \leftarrow un sucesor seleccionado aleatoriamente de *actual*

$\Delta E \leftarrow \text{VALOR}[\textit{siguiente}] - \text{VALOR}[\textit{actual}]$

si $\Delta E \leq 0$ **entonces** $actual \leftarrow \textit{siguiente}$

en caso contrario $actual \leftarrow \textit{siguiente}$ sólo con probabilidad $e^{\Delta E/T}$

Figura 4.14 Algoritmo de búsqueda de temple simulado, una versión de la ascensión de colinas estocástico donde se permite descender a algunos movimientos. Los movimientos de descenso se aceptan fácilmente al comienzo en el programa de templadura y luego menos, conforme pasa el tiempo. La entrada del *esquema* determina el valor de T como una función de tiempo.

Datos iniciales y parámetros a ser definidos para poder inicializar el algoritmo:

Temperatura inicial (T0)

La temperatura inicial T0 debe ser una temperatura que permita casi (o todo) movimiento, es decir que la probabilidad de pasar del estado i al j sea muy alta, sin importar la diferencia VALOR(j) – VALOR(i). Esto es que el sistema tenga un alto grado de libertad. En problemas como el del agente viajero, donde la entrada son los nodos de un grafo y las soluciones posibles son distintas formas de recorrer estos nodos, puede tomarse T0 proporcional a la raíz cuadrada de la cantidad de nodos. En general se toma un valor T0 que se cree suficientemente alto y se observa la primera etapa para verificar que el sistema tenga un grado de libertad y en función de esta observación se ajusta T0.

Solución inicial

En todas las versiones, el sistema debe ser “derretido” antes de implementar el algoritmo. Esto es que la solución factible inicial debería ser una solución tomada al azar del conjunto de soluciones factibles. En algunos problemas esto puede hacerse utilizando números pseudo-aleatorios provistos por una maquina. Pero en muchos casos ya es problemático encontrar una solución, por lo que es imposible tomar una al azar. En estos casos se puede implementar un algoritmo voraz tipo búsqueda local para buscar una solución factible y se toma esta como i0.

Factor de enfriamiento (esquema[t])

$T_{sig} = \alpha T$ (factor de enfriamiento geométrico, $\alpha < 1$, muy cercano a 1)

$T_{sig} = 1 / (1 + \beta T)$ (donde β es un real positivo cercano a cero)

Búsqueda por haz local

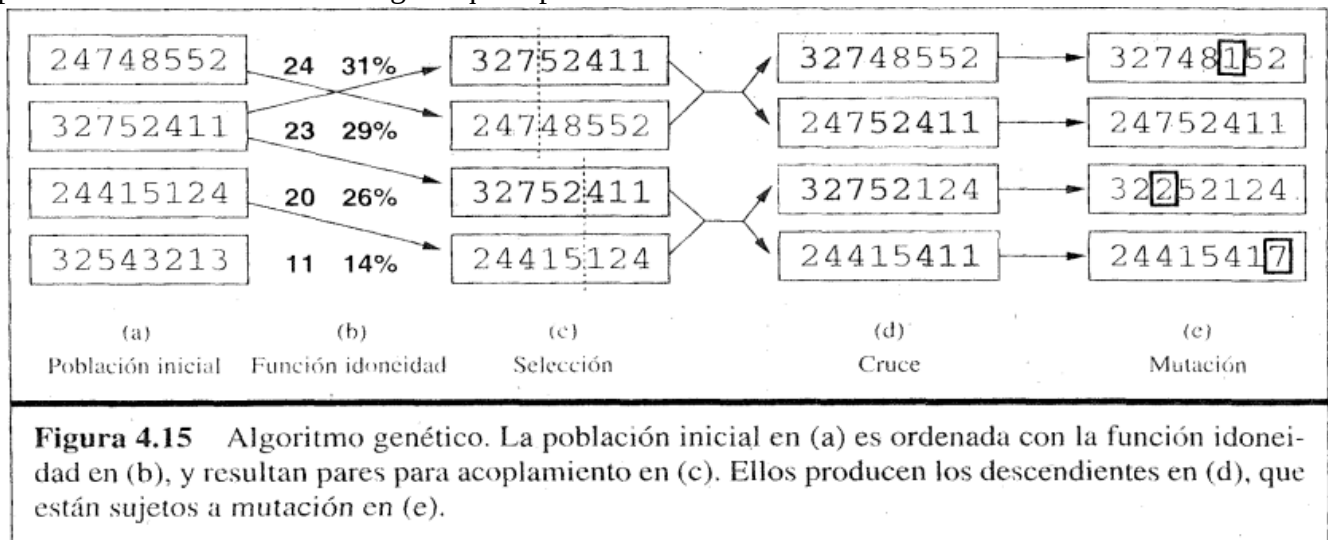
Una alternativa a la búsqueda de ascensión de colinas es guardar la pista de k estados (no sólo uno). Comienza con estados generados aleatoriamente. En cada paso, se generan todos los vecinos o sucesores de los k estados. Si alguno es objetivo, detenemos el algoritmo. Por otra parte, se seleccionan los k mejores sucesores de la lista completa y repetimos.

A primera vista, una búsqueda por haz local con k estados podría parecerse a ejecutar k veces la ascensión de colinas con estados diferentes. Sin embargo esto no es así, Si ejecutamos k veces el algoritmo de ascensión de colinas con estados diferentes (aún de manera paralela), la información de cada ejecución es independiente. En la búsqueda por haz local, la información útil es compartida por los k estados generados. Por ejemplo, si un estado genera varios sucesores buenos y los otros k-1 estados generan sucesores malos, entonces el efecto es que el primer estado “llama” a los demás, indicando que la solución probablemente esta de ese lado.

Algoritmos genéticos.

Un algoritmo genético realiza una analogía a la selección natural, tomando en cuenta la reproducción sexual y la mutación. Los algoritmos genéticos comienzan con un conjunto de k estados generados aleatoriamente, llamados población. Cada estado, o individuo, está representado como una cadena sobre un alfabeto finito (comúnmente 0s y 1s). Por ejemplo, un estado de 8 reinas debe especificar las posiciones de las ocho reinas en cada columna con 8 posibles posiciones. Para representarlo en 0s y 1s se requieren de 3 bits para representar dígitos del 1 al 8 y con 8 reinas se necesita en total una cadena de 24 bits para representar una posible solución. También podemos usar el dígito como tal, entonces tendríamos una cadena de 8 dígitos que representa una posible solución. Estas

representaciones se comportarían de manera diferente en el algoritmo. La figura 4.15(a) muestra una población de 4 cadenas de 8 dígitos que representan estados de 8 reinas.



En el resto de la figura 4.15 se muestra la producción de la siguiente generación de estados. En (b) cada estado se evalúa con la función de evaluación o función de idoneidad. Dicha función debería devolver valores más altos para estados mejores, así que para el problema de las 8 reinas utilizaremos el número de pares de reinas que NO se atacan, para una solución debería ser 28. Los valores de los cuatro estados son 24, 23, 20 y 11. En este ejemplo tomaremos la probabilidad de ser elegido para la reproducción directamente proporcional al resultado de la función de idoneidad.

En (c), se seleccionan dos pares, de manera aleatoria, para la reproducción, de acuerdo con las probabilidades en (b). Notemos que un individuo se selecciona dos veces y uno ninguna. Para que cada par se aparee, se elige aleatoriamente un punto de cruce de las posiciones en la cadena. En la figura los puntos de cruce están después del tercer dígito en el primer par y después del quinto en el segundo par.

En (d), los descendientes se crean cruzando las cadenas parentales en el punto de cruce. Por ejemplo, el primer hijo del primer par consigue los tres primeros dígitos del primer estado y los dígitos restantes del segundo estado, mientras que el segundo hijo consigue los tres primeros dígitos del segundo estado y el resto del primer estado. En la figura 4.16 se puede ver el cruce de la primera pareja de estos.

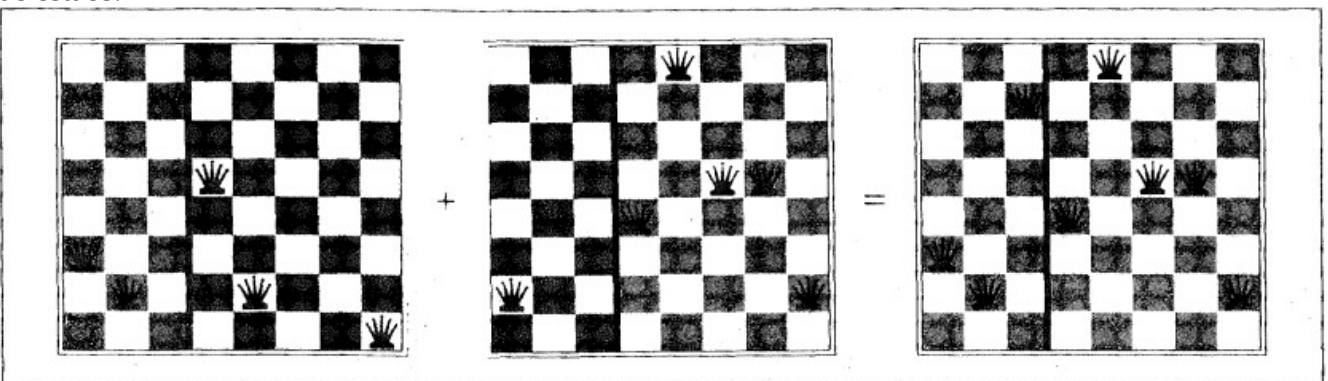


Figura 4.16 Los estados de las ocho reinas correspondientes a los dos primeros padres de la Figura 4.15(c) y el primer descendiente de Figura 4.15(d). Las columnas sombreadas se pierden en el paso de la transición y las columnas no sombreadas se mantienen.

Finalmente en (e), cada posición está sujeta a la mutación aleatoria con una pequeña probabilidad independiente. Un dígito fue transformado en el primer, tercer, y cuarto descendiente.

Al igual que el algoritmo de recocido simulado, los algoritmos genéticos dan grandes saltos al inicio, cuando la población es muy diversa, y poco a poco se van centrando en una solución. En la figura 4.17 se describe un algoritmo genético.

función ALGORITMO-GENÉTICO(*población*, IDONEIDAD) **devuelve** un individuo

entradas: *población*, un conjunto de individuos

IDONEIDAD, una función que mide la capacidad de un individuo

repetir

nueva_población \leftarrow conjunto vacío

bucle para *i* **desde** 1 **hasta** TAMAÑO(*población*) **hacer**

x \leftarrow SELECCIÓN-ALEATORIA(*población*, IDONEIDAD)

y \leftarrow SELECCIÓN-ALEATORIA(*población*, IDONEIDAD)

hijo \leftarrow REPRODUCIR(*x*,*y*)

si (probabilidad aleatoria pequeña) **entonces** *hijo* \leftarrow MUTAR(*hijo*)

añadir *hijo* a *nueva_población*

población \leftarrow *nueva_población*

hasta que algún individuo es bastante adecuado, o ha pasado bastante tiempo

devolver el mejor individuo en la *población*, de acuerdo con la IDONEIDAD

función REPRODUCIR(*x*,*y*) **devuelve** un individuo

entradas: *x*,*y*, padres individuales



n \leftarrow LONGITUD(*x*)

c \leftarrow número aleatorio de 1 a *n*





devolver AÑADIR(SUBCADENA(*x*, 1, *c*), SUBCADENA(*y*, *c* + 1, *n*))

Figura 4.17 Algoritmo genético. El algoritmo es el mismo que el de la Figura 4.15, con una variación: es la versión más popular; cada cruce de dos padres produce sólo un descendiente, no dos.

Que pasa si representamos las cadenas en bits, tomemos las casillas del 0 al 7 en lugar de 1 a 8

[2 1 | 6 4 1 3 0 0]  [2 1 6 3 7 4 4 1]
 [1 3 | 6 3 7 4 4 1]  [1 3 6 4 1 3 0 0]

Ahora con 0s y 1s (con la particularidad de

[0 1 | 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0]  [0 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 0 0 0 0 1]
 [2 1 6 4 1 3 0 0]  [3 3 6 3 7 4 4 1]
 [0 0 | 1 0 1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 0 0 0 0 1]  [0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0]
 [1 3 6 3 7 4 4 1]  [0 1 6 4 1 3 0 0]