Inteligencia Artificial
Búsqueda local

Primavera 2007

profesor: Luigi Ceccaroni

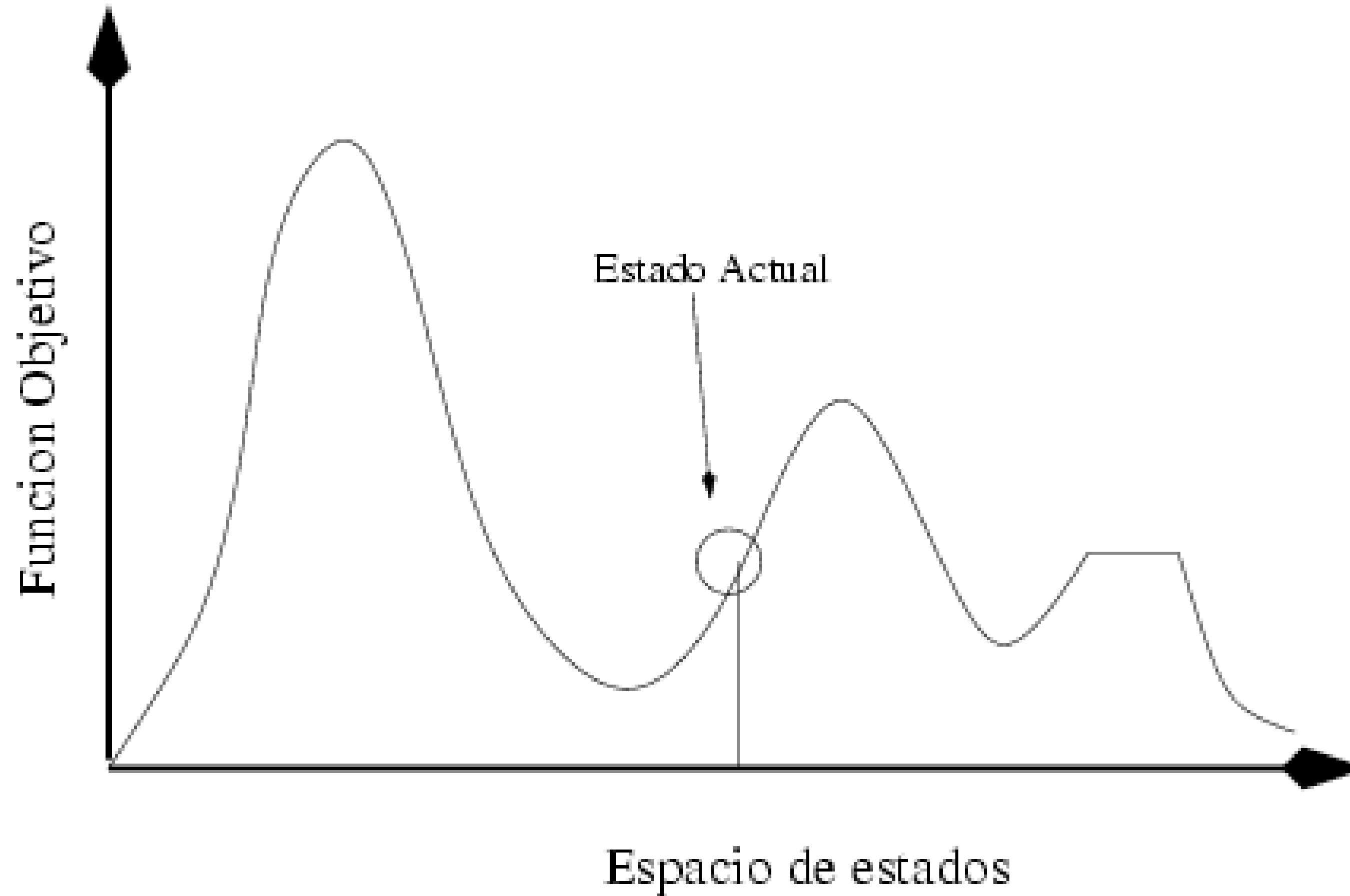
Búsqueda local

- En la búsqueda local (BL), **se empieza de una configuración inicial** (generalmente aleatoria) y se hacen pequeños cambios (a través de operadores) hasta alcanzar un estado desde el cual no se puede alcanzar ningún estado mejor.
- Las técnicas de BL son propensas a encontrar óptimos locales que no son la mejor solución posible. El óptimo global es generalmente imposible de alcanzar en un tiempo limitado, por el tamaño del espacio de soluciones.
  - Los algoritmos no pueden hacer una exploración sistemática.
- Los métodos usados en BL son conocidos como **meta-heurísticas** u optimización local.

Búsqueda local

- Hay una **función heurística** que evalúa la calidad de la solución, pero que no está necesariamente ligada a un coste.
- La función heurística se usará para podar el espacio de búsqueda (soluciones que no merece la pena explorar).
- **No se suele** guardar historia del camino recorrido (el gasto de memoria es mínimo).
- La **falta total de memoria** puede suponer un problema (bucles).

Búsqueda local



Búsqueda de ascensión de colinas

- Ascensión de colinas (AdC) simple:
  - Se busca una **cualquier** operación que suponga una mejora respecto al estado actual.
- Ascensión de colinas por máxima pendiente (*steepest-ascent hill climbing*, *gradient search*):
  - Se selecciona el **mejor** movimiento (no el primero de ellos) que suponga mejora respecto al estado actual.

Ascensión de colinas: algoritmo informal

1. Set L to be a list of the initial nodes in the problem, sorted by their expected distance to the goal. Nodes expected to be close to the goal should precede those that are farther from it.
2. Let n be the first node on L. If L is empty, fail.
3. If n is a goal node, stop and return it and the path from the initial node to n.
4. Otherwise, remove n from L. Sort n's children by their expected distance to the goal, label each child with its path from the initial node, and add the children to the front of L. Return to step 2.

Ascensión de colinas: algoritmo formal

```
Algoritmo Hill Climbing
 Actual= Estado_inicial
 fin = falso
 Mientras ¬fin hacer
    Hijos= generar_sucesores(Actual)
    Hijos= ordenar_y_eliminar_peores(Hijos, Actual)
    si  ¬vacio?(hijos) entonces Actual= Escoger_mejor(Hijos)
    si no fin=cierto
 fMientras
fAlgoritmo
```

- Sólo se consideran los descendientes cuya función de estimación es mejor que la del padre (poda del espacio de búsqueda).
- Se puede usar una pila y guardar los hijos mejores que el padre para poder volver atrás, pero en general el coste es prohibitivo.

Ascensión de colinas

- Las características de la función heurística determinan la calidad del resultado y la rapidez de la búsqueda.
- Problemas:
  - **Máximo local**. Todos los vecinos tienen función heurística peor.
  - **Meseta**. Todos los vecinos tienen la misma función heurística que el nodo actual.
  - **Crestas**: Las crestas causan una secuencia de máximos locales que hace muy difícil la navegación para los algoritmos avaros.

Ascensión de colinas

- Soluciones:
  - Volver a un nodo anterior y seguir el proceso en otra dirección (prohibitivo en espacio).
  - Reiniciar la búsqueda en otro punto.
  - Aplicar dos o más operadores antes de decidir el camino.
  - Hacer AdC en paralelo.
    - Ejemplo: dividir el espacio de búsqueda en regiones y explorar las más prometedoras.

El problema de las 8-reinas

- Los algoritmos de BL típicamente usan una **formulación de estados completa**.
  - Cada estado tiene a ocho reinas sobre el tablero, una por columna.
- La función sucesor devuelve todos los estados posibles generados moviendo una reina a otro cuadrado en la misma columna.
  - Cada estado tiene 8 x 7 = 56 sucesores.

El problema de las 8-reinas

- La función de costo heurística *h* es el número de pares de reinas que se atacan la una a la otra, directa o indirectamente.
  - Problema de minimización
- El mínimo global de esta función es cero.
  - Ocurre sólo en soluciones perfectas.
- Estado inicial: cualquiera.

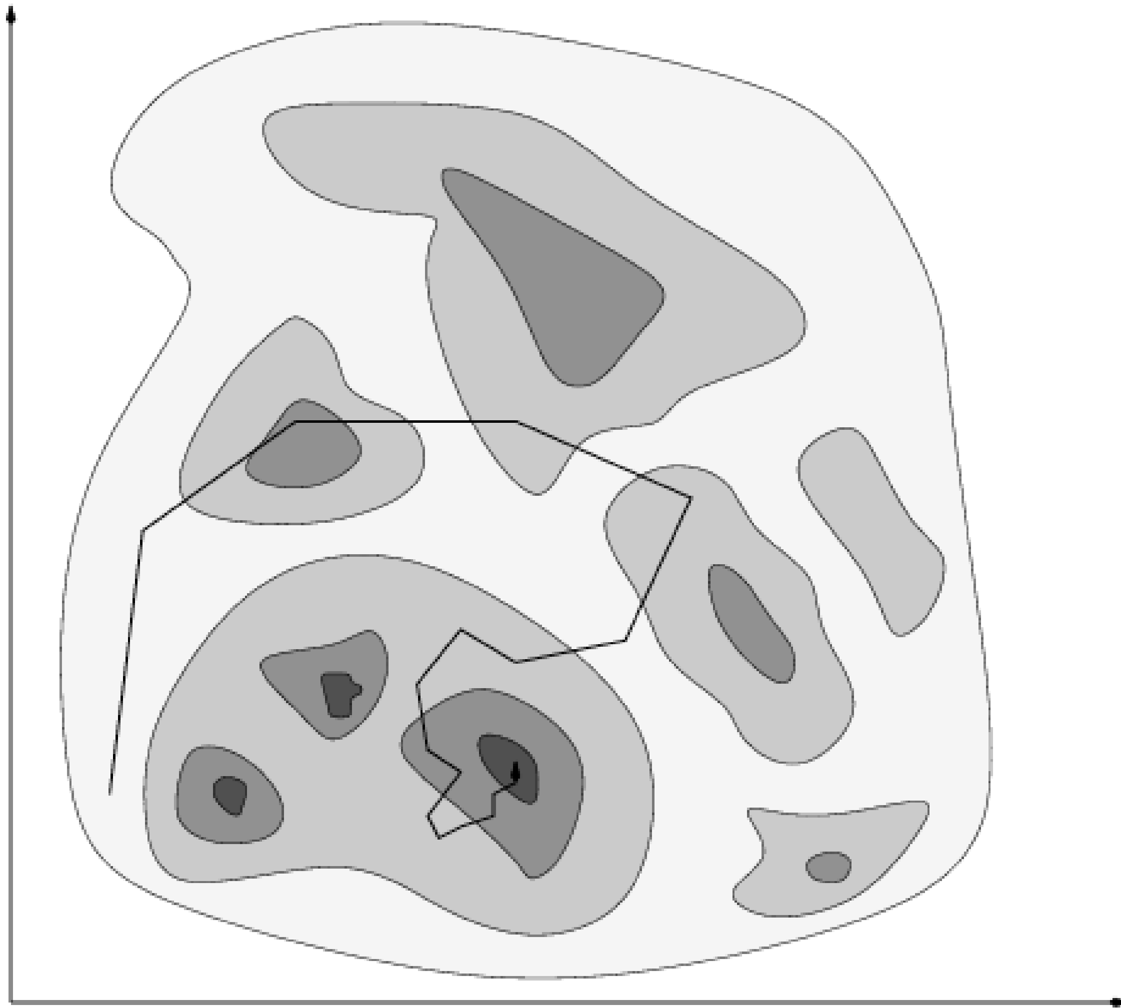
Otros algoritmos de búsqueda local

- Existen otros algoritmos inspirados en analogías físicas y biológicas:
  - **Temple simulado**: ascensión de colinas estocástica inspirada en el proceso de enfriamiento de metales
  - **Algoritmos genéticos:** ascensión de colinas paralela inspirada en los mecanismos de selección natural
- Ambos mecanismos se aplican a problemas reales con bastante éxito.

Temple simulado

- Es un algoritmo de ascensión de colinas estocástica:
  - Elegimos un sucesor de entre todos los posibles según una distribución de probabilidad.
  - El sucesor puede ser peor.
  - Se hacen pasos aleatorios por el espacio de soluciones.
- Inspirado en el proceso físico de enfriamiento controlado (cristalización, templado de metales):
  - Se calienta un metal a alta temperatura y se enfría progresivamente de manera controlada.
  - Si el enfriamiento es adecuado se obtiene la estructura de menor energía (mínimo global).

Búsqueda de temple simulado



Temple simulado: metodología

- Se identifican los elementos del problema de búsqueda con los del problema físico:
  - **Temperatura**: parámetro de control principal
  - **Energía**: función heurística sobre la calidad de la solución f(n)
  - **Función que determina la elección de un estado sucesor**: F(Δf ,T), depende de la temperatura y la diferencia entre la calidad de los nodos
    - A menor temperatura menor probabilidad de elegir sucesores peores
  - **Estrategia de enfriamiento**: parámetros que determinan el número de iteraciones de la búsqueda, disminución de la temperatura y número de pasos para cada temperatura

Temple simulado: algoritmo básico

Partimos de una temperatura

Mientras la temperatura no sea cero **hacer**

/* Paseo aleatorio por el espacio de soluciones */

Para un numero prefijado de iteraciones **hacer**

E_{nuevo}=Generamos_sucesor(E_{actual})

si $F(f(E_{actual})- f(E_{nuevo}),T) > 0$ **entonces** $E_{actual}=E_{nuevo}$

fPara

Disminuimos la temperatura

fMientras

Temple simulado: aplicación

- Adaptable a problemas de optimización combinatoria (configuración óptima de elementos) y continua (punto óptimo en un espacio N-dimensional)
- Indicado para problemas grandes en los que el óptimo esta rodeado de muchos óptimos locales
- Indicado para problemas en los que encontrar una heurística discriminante es difícil (una elección aleatoria es tan buena como otra cualquiera)
- Aplicaciones: *travelling salesman problem* (TSP), diseño de circuitos *very large scale integration* VLSI
- Problema: determinar los valores de los parámetros requiere experimentación

Temple simulado - Ejemplo - TSP

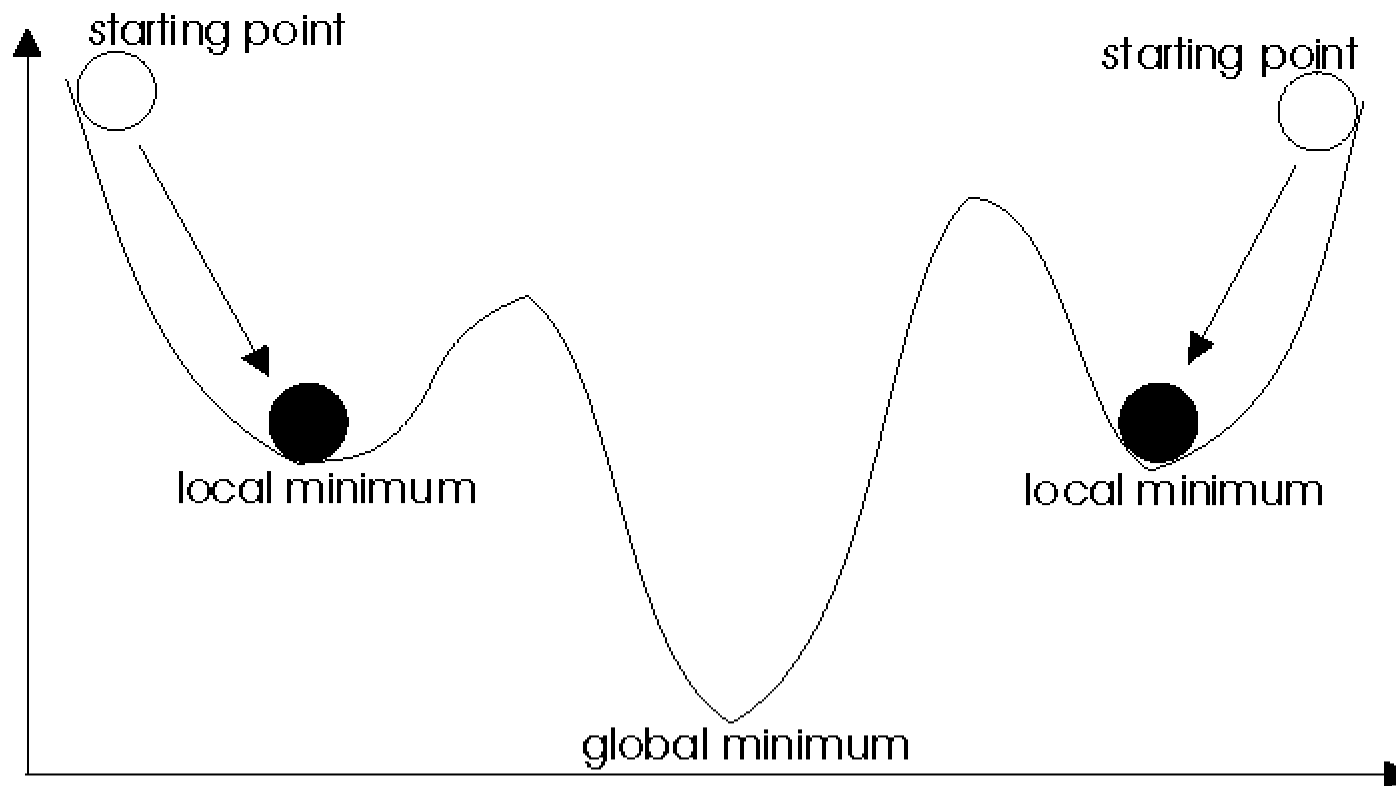
- Viajante de comercio (TSP): Espacio de búsqueda N!
- Definimos posibles transformaciones de una solución (operadores): Inversiones, traslaciones, intercambios
- Definimos la función de energía (Suma de distancia entre ciudades, según el orden de la solución)

$$E = \sum_{i=1}^{N} \sqrt{(x_i - x_{i+1}) + (y_i - y_{i+1})^2} + \sqrt{(x_N - x_1) + (y_N - y_1)^2}$$

- Definimos una temperatura inicial (experimentación)
- Determinamos cuantas iteraciones hacemos para cada temperatura y como disminuimos la temperatura

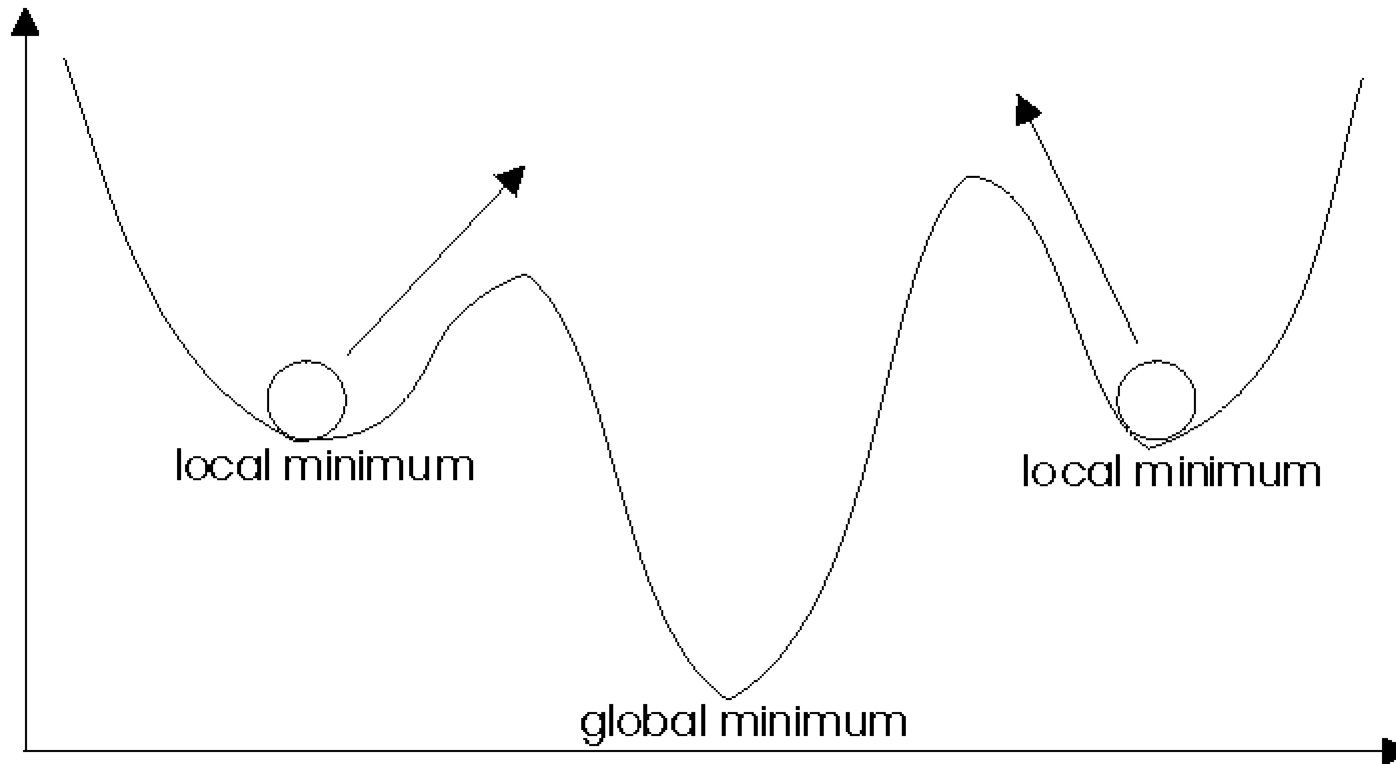
Temple simulado: resumen (1)

- **Propósito**: evitar el problema de los máximos (o mínimos) locales de la ascensión de colinas (AdC).



Temple simulado: resumen (2)

- **Solución**: en la técnica de AdC, ocasionalmente, dar un paso en una dirección diferente de la donde la tasa de cambio es máxima.



Temple simulado: resumen (3)

- **Idea principal**: los pasos dados en la dirección aleatoria no reducen la habilidad para encontrar un máximo global.
- **Desventaja**: es probable que estos pasos incrementen el tiempo de ejecución del algoritmo.
- **Ventaja**: es posible que estos pasos permitan bajar de una pequeña colina.
- **Temperatura**: es el descriptor que determina (a través de una función de probabilidad) la amplitud de los pasos, largos al principio y luego cada vez más cortos.
  - Cuando la amplitud del paso aleatorio es suficientemente pequeña para no permitir bajar de la colina que se está considerando, se puede decir que el resultado del algoritmo está *templado*.

Temple simulado: resumen (4)

- Es posible demostrar que, si la *temperatura* del algoritmo se reduce muy lentamente, se encontrará un máximo global con probabilidad cerca de uno:
  - Valor de la función en el máximo global = *m*
  - Valor de la función en el mejor máximo local = *l* < *m*
  - Habrá alguna temperatura *t* suficientemente grande para permitir bajar del máximo local pero no del máximo global
  - Dado que la temperatura se reduce muy lentamente, el algoritmo trabajará lo suficiente con una temperatura cerca de *t*, hasta que finalmente encontrará y subirá al máximo global y se quedará allí porque no podrá bajar.
- **Conclusión**: cuando se está resolviendo un problema de búsqueda, ocasionalmente habría que examinar un nodo que parece sustancialmente peor que el mejor nodo que se encuentra en la lista L de nodos abiertos.

Búsqueda por haz local (1)

- Guardar sólo un nodo en memoria puede parecer una reacción extrema al problema de limitación de memoria.
- El algoritmo de búsqueda por haz local guarda la pista de *k* nodos.
  - Comienza con *k* estados generados aleatoriamente.
  - En cada paso se generan todos los sucesores de los *k* estados.
  - Se comprueba si alguno es un objetivo.
  - Si no, se seleccionan los *k* mejores sucesores de la lista completa y se repite el proceso.

Búsqueda por haz local (2)

- ¡Es diferente de ejecutar *k* reinicios aleatorios en paralelo en vez de en secuencia!
  - Si un estado genera varios sucesores buenos, el algoritmo rápidamente abandona las búsquedas infructuosas y mueve sus recursos allí donde se hace la mayor parte del progreso.
- En su forma más simple, puede sufrir una carencia de diversidad entre los *k* estados (concentrados en una pequeña región del espacio de estados) y volverse en poco más que una versión cara de la AdC.

Búsqueda de haz estocástica

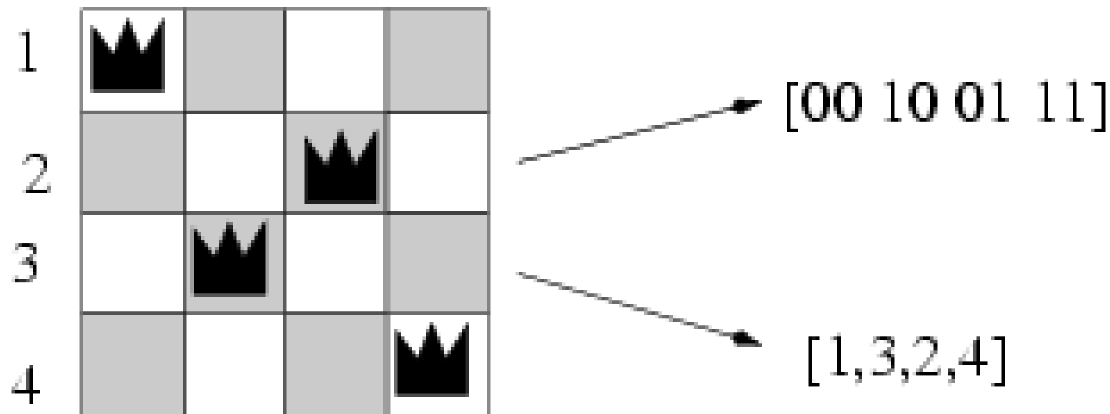
- En vez de elegir los *k* mejores sucesores, escoge a *k* sucesores **aleatoriamente**:
  - **probabilidad** de elegir a un sucesor como una **función creciente** del valor de la **función de idoneidad**
- Parecido con el proceso de **selección natural**: los *sucesores* (descendientes) de un *estado* (organismo) pueblan la siguiente generación según su *valor* (idoneidad, salud, adaptabilidad).

Algoritmos genéticos (AGs)

- Son una variante de la búsqueda de haz estocástica en que se combinan **dos** estados padres.
- Analogía entre búsqueda local y evolución por selección natural:
  - Los estados corresponden a **individuos**.
  - Una **función de idoneidad/calidad/evaluación** indica/mide la bondad/calidad de los estados.
  - Combinando buenos estados se obtienen estados mejores.

AGs: codificación

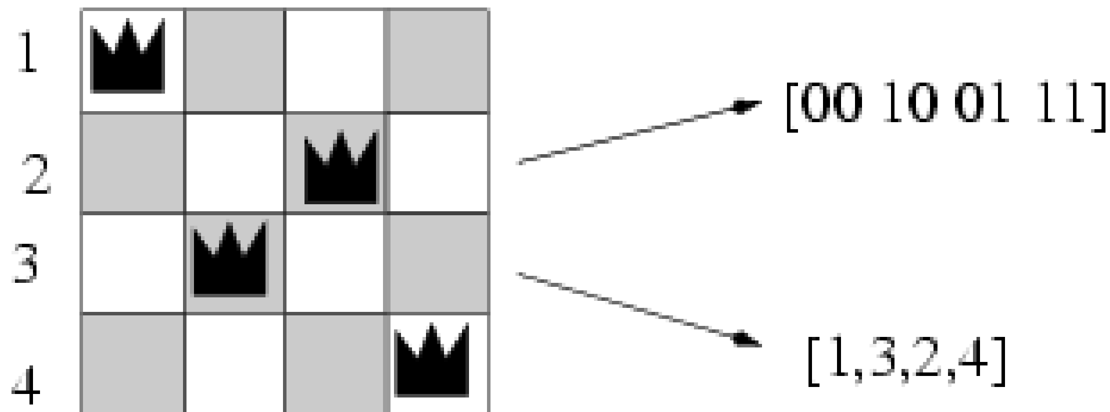
- Definición de las características de los **individuos**:
  - Cada individuo se representa como una cadena sobre un alfabeto finito (comúnmente, una cadena de 0s y 1s)



- La codificación define el tamaño del espacio de búsqueda y el tipo de operadores de combinación necesarios.

AGs: codificación

- Si el estado debe especificar las posición de *n* reinas, cada una en una columna de *n* cuadrados, se requieren $n * \log_2 n$ bits (en el ejemplo = 8).
- El estado podría también representarse como *n* dígitos [1,*n*].

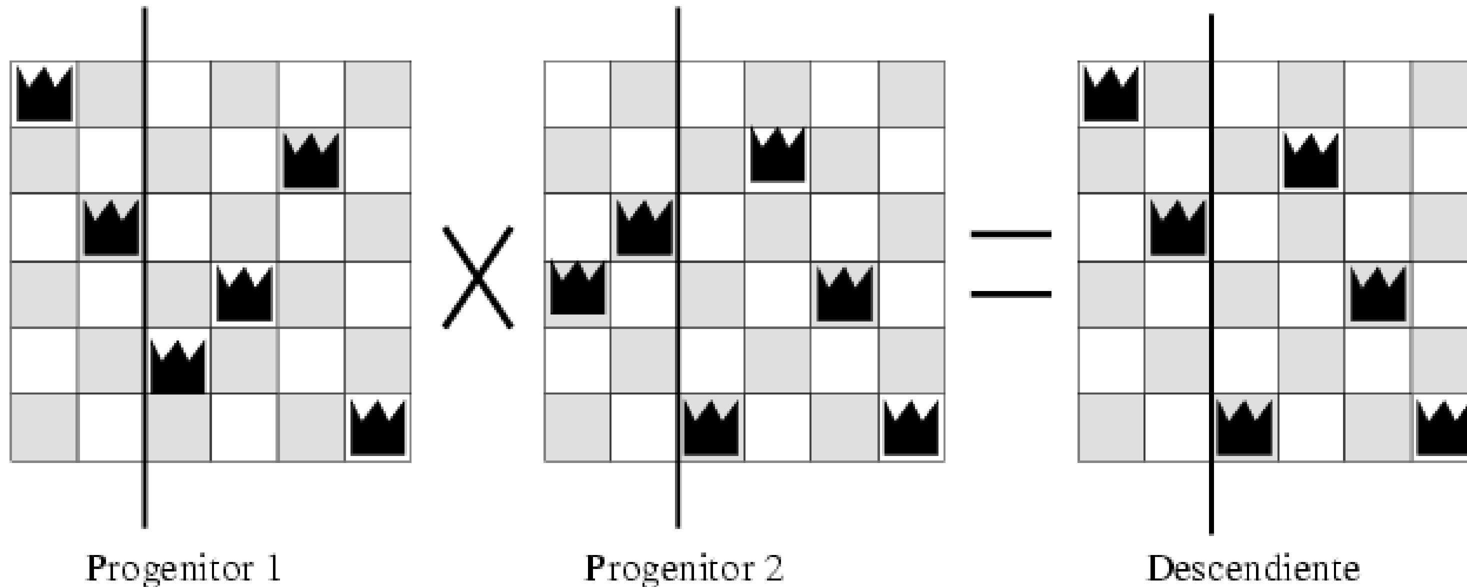


AGs: función de idoneidad

- En la producción de la siguiente generación de estados, para cada estado se calcula la **función de idoneidad**.
- Una función de idoneidad debería devolver valores más altos para estados mejores.
  - Para el problema de las *n* reinas se puede utilizar el número de pares de reinas que no se atacan.
  - En el caso de 4 reinas, la función de idoneidad tiene un valor de 6 para una solución.

AGs: operadores

- La combinación de individuos se realiza mediante operadores de **cruce**.
- El operador básico es el cruce por un punto:
  - Se elige aleatoriamente un punto de la codificación.
  - Los descendientes se crean cruzando las cadenas paternales en el punto de cruce.



Operadores

- En el ejemplo, el hijo consigue las dos primeras columnas del primer padre y las columnas restantes del segundo padre.
- Cada paso es una generación de individuos.
  - El tamaño de la **población** en general se mantiene constante (N).
- Existen otras posibilidades:
  - Cruce en dos puntos
  - Intercambio aleatorio de bits
  - Operadores *ad hoc* según la representación

Mutación

- Analogía con la combinación de genes:
  - A veces la información de parte de ellos cambia aleatoriamente.
- Básicamente, la **mutación** consiste en cambiar el signo de cada bit (si se trata con una cadena binaria) con cierta probabilidad:
  - Cada posición está sujeta a una mutación aleatoria con una pequeña probabilidad independiente.

AGs: combinación

- Los AGs comienzan con una población de *k* estados generados aleatoriamente.
- Para pasar a la siguiente población debemos elegir que individuos se han de combinar (*población intermedia*), por ejemplo:
  - Cada individuo se elige con probabilidad proporcional a su función de idoneidad.
  - Se establecen N *torneos* aleatorios entre parejas de individuos y se eligen los que ganan en cada torneo.
  - Se define un ranking lineal entre individuos según su función de idoneidad.
- En la *población intermedia,* siempre habrá individuos que aparezcan más de una vez e individuos que no aparezcan.

AGs: pasos de ejecución

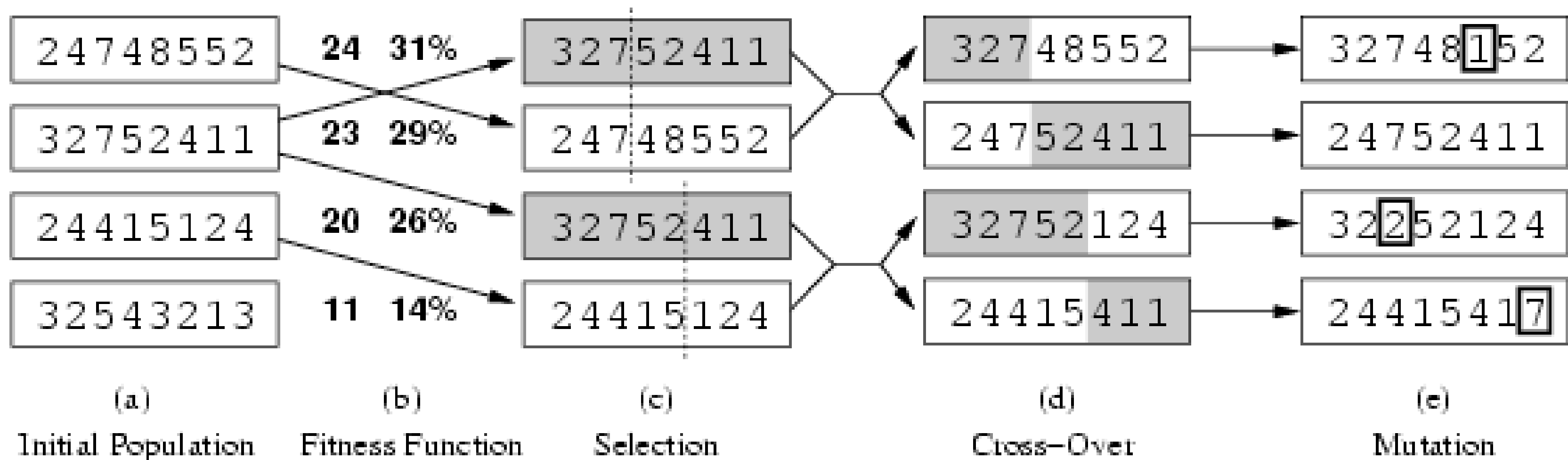
1. Se escogen N individuos de la población actual para la población intermedia.
2. Se emparejan los individuos y para cada pareja:
   - con una probabilidad (P_cruce) se aplica el operador de cruce a los individuos y se obtienen dos nuevos individuos;
   - con una probabilidad (P_mutación) se mutan los nuevos individuos.
3. Estos individuos forman la nueva población.

- El procedimiento se itera hasta que la población converge o pasa un número específico de iteraciones.

AGs: ejemplo de las 8 reinas

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7 / 2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29%
- Etc.

AGs: ejemplo de las 8 reinas



- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7 / 2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29%
- Etc.

AGs: aplicación

- Son aplicables casi a cualquier tipo de problema.
- Permiten abordar problemas para los que no se dispone de una función heurística adecuada.
- Por lo general serán peores que un algoritmo clásico con una buena heurística.
- Dificultades:
  - codificación de los estados
  - determinación de los parámetros del algoritmo:
    - tamaño de la población
    - iteraciones
    - probabilidad de cruce y mutación