

Inteligencia Artificial

Tema 4. Optimización

José Manuel Fuertes García
jmf@ujaen.es

Departamento de Informática
Universidad de Jaén



27 de marzo de 2017

Objetivos

- ▶ Entender en qué consiste un problema de optimización
- ▶ Conocer el funcionamiento general de un algoritmo de búsqueda local
- ▶ Entender el funcionamiento de algoritmos de optimización de colinas, temple simulado, haz local y algoritmos genéticos
- ▶ Conocer ventajas e inconvenientes de cada uno de los enfoques
- ▶ Obtener la capacidad de aplicar este tipo de algoritmos para mejorar el comportamiento de los agentes

Índice

Introducción

Ascensión de colinas

Temple simulado

Haz Local

Algoritmos Genéticos

Introducción

- ▶ Hay problemas para los que no disponemos de una expresión analítica de la solución
- ▶ **Optimización**: Búsqueda de una solución lo suficientemente buena
- ▶ El camino para alcanzarla es irrelevante
- ▶ Partimos de una(s) solución(es) candidata(s) inicial(es), que vamos mejorando
- ▶ Disponemos únicamente de una función que evalúa la calidad de la solución
- ▶ **Algoritmos de búsqueda local**: funcionan con un solo estado actual y se mueven sólo a los vecinos del estado

Ventajas e inconvenientes de los algoritmos de búsqueda local

Ventajas:

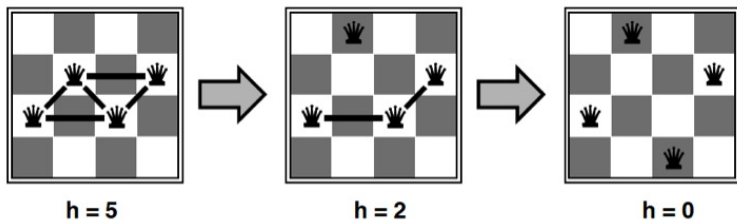
- ▶ Consumen poca memoria
- ▶ Pueden encontrar soluciones satisfactorias en espacios de estados grandes o infinitos, incluso continuos

Inconvenientes:

- ▶ El tamaño del espacio de soluciones en general no permite obtener el óptimo
- ▶ Los algoritmos no pueden hacer una exploración sistemática
- ▶ La falta total de memoria puede producir ciclos

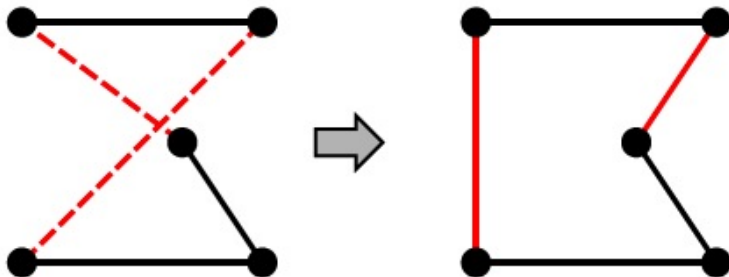
Ejemplo: mejora iterativa local para el problema de las 4-reinas

Mejora iterativa: mover una reina a una posición en la que se disminuya el número de conflictos



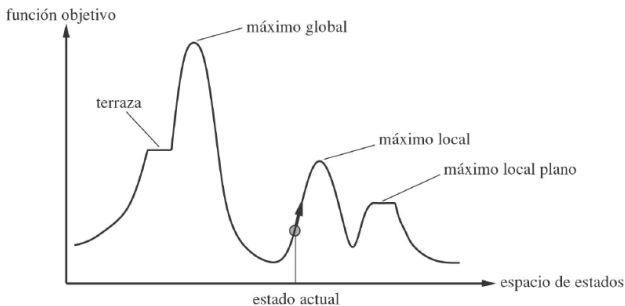
Ejemplo: mejora iterativa local para un problema de rutas (TSP)

Mejora iterativa: realizar cambios de pares de aristas para disminuir el costo



Algoritmos de búsqueda local y el paisaje de espacio de estados

- ▶ Para entender la búsqueda local es útil considerar el **paisaje de espacio de estados**, en el que la posición viene dada por el estado y la elevación por el valor de la función coste u objetivo
- ▶ Los algoritmos de búsqueda exploran este paisaje



Índice

Introducción

Ascensión de colinas

Temple simulado

Haz Local

Algoritmos Genéticos

Optimización mediante ascensión de colinas

Se mueve en dirección del valor creciente y termina cuando se alcanza un óptimo local

Variantes:

- ▶ **Escalada simple:** se busca cualquier operación que mejore la calidad
- ▶ **Escalada estocástica:** se escoge aleatoriamente un movimiento ascendente
- ▶ **Escalada por máxima pendiente:** selecciona el mejor movimiento

Algoritmo de búsqueda Ascensión de colinas (*escalada por máxima pendiente*)

función *Ascensión-Colinas* (*problema*) **devuelve** un estado que es un máximo local

entradas: *problema* (un problema)

variables locales: *actual* (un nodo), *vecino* (un nodo)

actual \leftarrow Hacer-Nodo(Estado-Inicial[*problema*])









bucle hacer

vecino \leftarrow sucesor de valor más alto de *actual*

si Valor[*vecino*] \leq Valor[*actual*] **entonces devolver** Estado[*actual*]

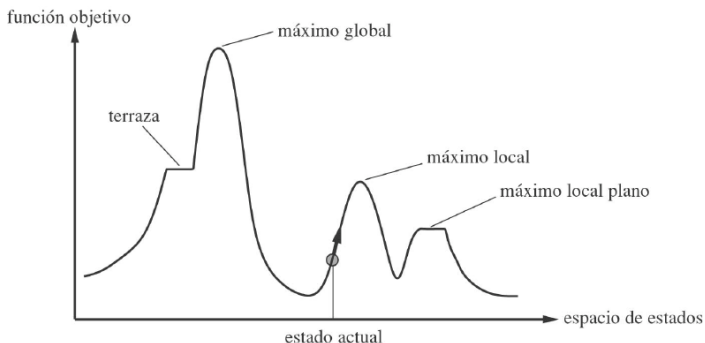
actual \leftarrow *vecino*

El problema de las 8-reinas

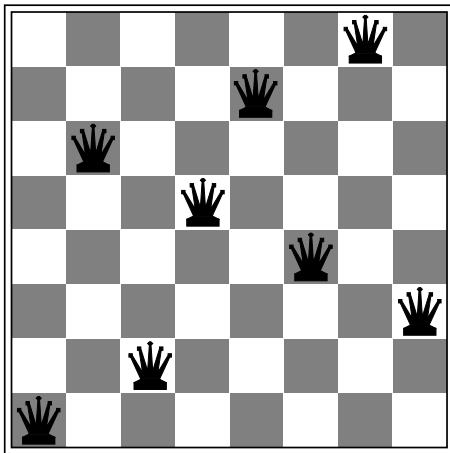
18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14		13	16	13	16
	14	17	15		14	16	16
17		16	18	15		15	
18	14		15	15	14		16
14	14	13	17	12	14	12	18

Características de la ascensión de colinas

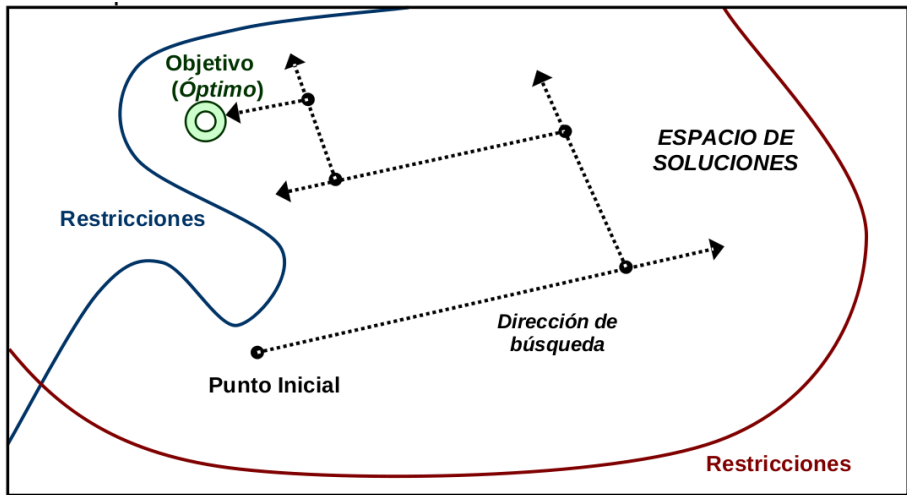
- ▶ Las características de la función de ajuste determinan el éxito y rapidez de la optimización
- ▶ La estrategia del algoritmo hace que la búsqueda pueda acabar en un punto donde la solución sólo sea la óptima *aparentemente*



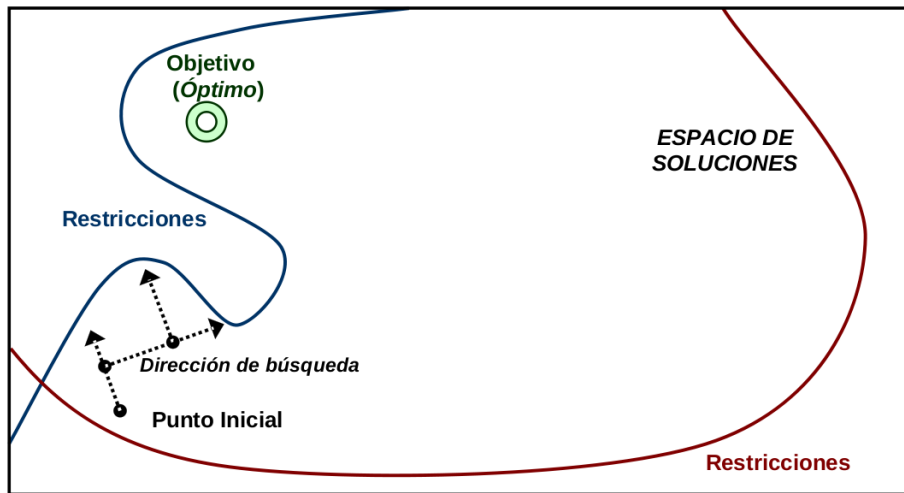
El problema de las 8-reinas. Un mínimo local



Proceso de optimización



Proceso de optimización



¿Cómo *escapar* de óptimos locales?

- ▶ Hacer backtracking a un nodo anterior y seguir el proceso en otra dirección (prohibitivo en espacio)
- ▶ Reiniciar la búsqueda en otro punto buscando mejorar la solución actual
- ▶ Aplicar dos o más operaciones antes de decidir el camino
- ▶ Hacer varias optimizaciones en paralelo (p.ej. dividir el espacio de búsqueda en regiones y explorar las más prometedoras)

Índice

Introducción

Ascensión de colinas

Temple simulado

Haz Local

Algoritmos Genéticos

Optimización mediante *Temple Simulado*

- ▶ Un algoritmo de ascensión de colinas nunca hace movimientos hacia estados con un valor inferior y eso hace que sea incompleto
- ▶ Un algoritmo aleatorio es completo pero sumamente ineficaz
- ▶ **Idea:** escapar de los máximos locales permitiendo algunos movimientos *malos*, pero disminuyendo gradualmente su frecuencia
- ▶ Inspirado en el proceso físico de enfriamiento controlado (cristalización, templado de metales)
 - ▶ Se calienta un metal/disolución a alta temperatura y se enfría progresivamente de manera controlada
 - ▶ Si el enfriamiento es adecuado se obtiene la estructura de menor energía (mínimo global)

Optimización mediante *Temple Simulado*

- ▶ Es un algoritmo de ascensión de colinas estocástico (elegimos un sucesor de entre todos los posibles según una distribución de probabilidad, el sucesor podría ser peor)
- ▶ Hacemos paseos aleatorios por el espacio de soluciones
- ▶ Existe un parámetro T (temperatura) que controla la probabilidad de moverse a estados peores
- ▶ Se puede probar que si T disminuye suficientemente despacio, entonces la búsqueda de temple simulado encontrará un óptimo global con probabilidad próxima a 1
- ▶ Se ha utilizado ampliamente para resolver problemas de distribución VLSI, planificaciones de líneas aéreas, etc.

Esquema básico *Temple simulado*

función *Temple-Simulado*(*problema*, *esquema*) devuelve un estado solución

entradas: *problema* (un problema), *esquema* (describe la variación de la temperatura en el tiempo)

variables locales: *actual* (un nodo), *siguiente* (un nodo), T (una temperatura, controla la probabilidad de un paso hacia abajo)

$actual \leftarrow \text{Hacer-Nodo}(\text{Estado-Inicial}[problema])$

para $t \leftarrow 1$ a ∞ **hacer**

$T \leftarrow esquema[t]$

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

siguiente \leftarrow un sucesor seleccionado aleatoriamente de *actual*

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

si $\Delta E > 0$

entonces $actual \leftarrow siguiente$

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

Índice

Introducción

Ascensión de colinas

Temple simulado

Haz Local

Algoritmos Genéticos

Optimización por *Haz Local*

- ▶ Guarda la pista de k estados (no sólo uno)
- ▶ Comienza con k estados generados aleatoriamente
- ▶ En cada iteración se generan todos los vecinos de todos los k estados
- ▶ Si el ajuste de alguno es lo suficientemente bueno es un objetivo, paramos el algoritmo
- ▶ En caso contrario, se seleccionan los k mejores estados de la lista completa y repetimos
 - ▶ No se trata de ejecutar k reinicios aleatorios y ejecutar en paralelo
- ▶ En la búsqueda por haz local la información útil se pasa entre los k hilos paralelos
- ▶ **Variante:** Optimización de haz estocástica, que elige k sucesores aleatoriamente, siendo la probabilidad de elegir un sucesor una función creciente de su medida de calidad

Esquema básico *Beam search*

función *Beam-search* (problema, k) **devuelve** solución

Comenzar con k estados generados aleatoriamente

repetir

Generar todos los sucesores de todos los k estados

Si cualquiera de ellos es una solución **entonces** devolver solución **en caso contrario** seleccionar los k mejores sucesores

Índice

Introducción

Ascensión de colinas

Temple simulado

Haz Local

Algoritmos Genéticos

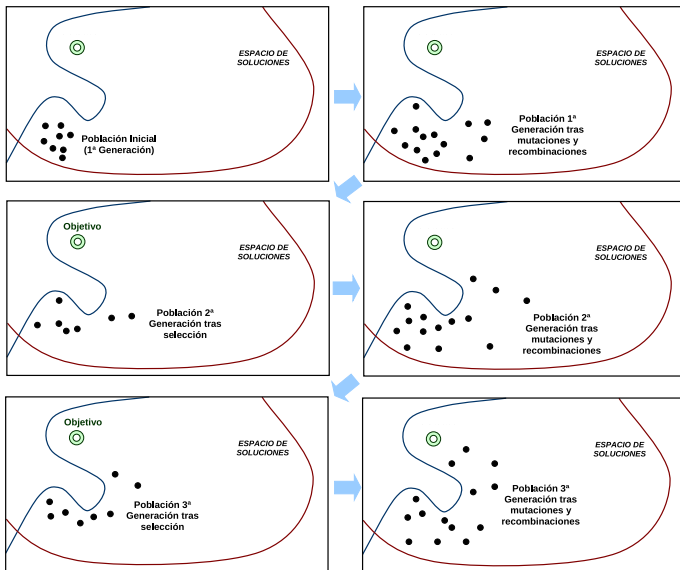
Algoritmos Genéticos

- ▶ Se basan en los mecanismos de la evolución natural
- ▶ Características del Proceso Evolutivo:
 - ▶ Una entidad o individuo tiene la habilidad de reproducirse
 - ▶ Hay una población de tales individuos que son capaces de reproducirse
 - ▶ Existe alguna variedad, o diferencia, entre los individuos que se reproducen
 - ▶ Algunas diferencias en la habilidad para sobrevivir en el entorno están asociadas con esa variedad
- ▶ Está compuesta por modelos de evolución basados en poblaciones cuyos elementos representan soluciones a problemas
- ▶ La simulación de este proceso en un ordenador resulta ser una técnica de optimización probabilística

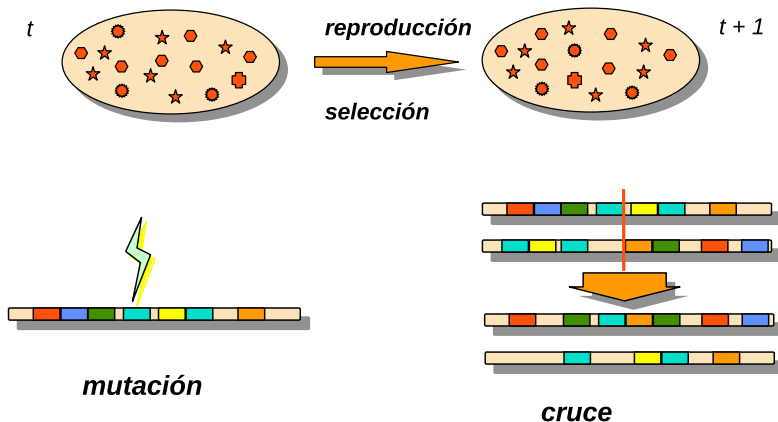
Funcionamiento básico de los Algoritmos Genéticos

- ▶ Mantiene una población de posibles soluciones para el problema
- ▶ Cada individuo se caracteriza por su información genética
- ▶ Se realizan modificaciones sobre la población
- ▶ Se seleccionan, a través de una función de adaptación, qué individuos se mantendrán en generaciones futuras y cuáles se eliminarán
- ▶ La población evoluciona a través de las mejores regiones del espacio de búsqueda mediante la modificación y selección

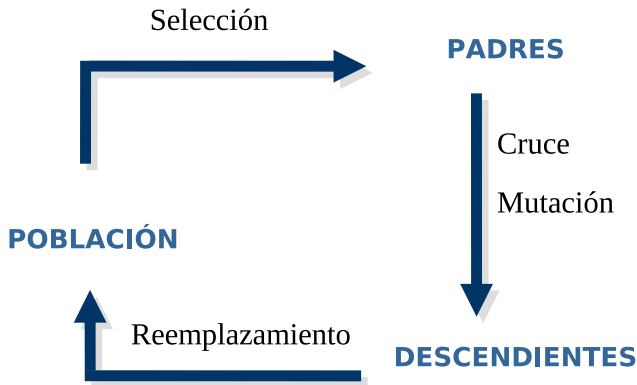
Funcionamiento básico de los Algoritmos Genéticos



Componentes de un Algoritmo Genético



El ciclo de la evolución



Conceptos básicos

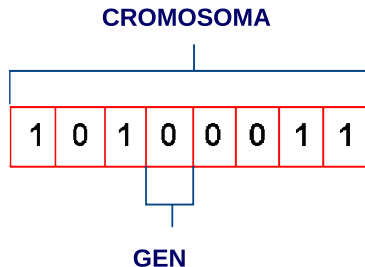
Genotipo y Fenotipo

- ▶ **Genotipo**: Valor de los genes
- ▶ **Fenotipo**: Manifestación externa (interpretación) de la información genética

Cromosoma, Gen y Alelos

- ▶ **Cromosoma**: Vector que codifica un individuo
- ▶ **Gen**: Elemento mínimo de un cromosoma (puede ser un bit)
- ▶ **Alelos**: Valores que puede tomar cada gen

Ejemplo



- ▶ **Genotipo:** 10100011.
- ▶ **Fenotipo:** Entero, real, secuencia, etc.
- ▶ Otros genotipos: números reales, permutaciones de elementos, etc.

Proceso de selección

- ▶ Se debe garantizar que los mejores individuos tienen una mayor probabilidad de ser padres (reproducirse) que los individuos menos buenos
- ▶ Debemos ser cuidadosos para dar una oportunidad de reproducirse a los individuos menos buenos. Éstos pueden incluir material genético útil en el proceso de reproducción.
- ▶ Esta idea define la **presión selectiva** que determina en qué grado la reproducción está dirigida por los mejores individuos

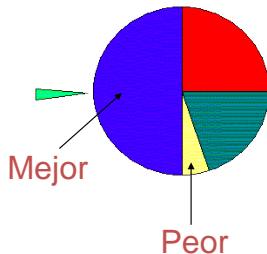
Ejemplos esquemas de selección

Ejemplo 1: Selección proporcional

- El número de veces que un individuo debe reproducirse es

$$ps_i = \frac{f_i}{\sum_j f_j}$$

- Los mejores individuos tienen:
 - Más espacio en la ruleta
 - Más probabilidad de ser seleccionados



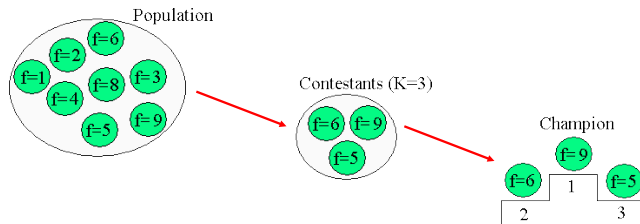
Ejemplos esquemas de selección

Ejemplo 2: Selección por torneo

Para cada padre a seleccionar:

- Escoger aleatoriamente k individuos, con reemplazamiento
- Seleccionar el mejor de ellos

k se denomina **tamaño del torneo**. A mayor k , mayor presión selectiva y viceversa.



Mutación y cruce

Cruce:

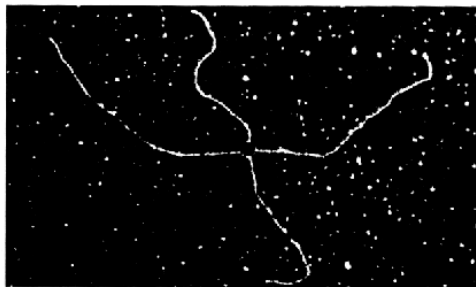
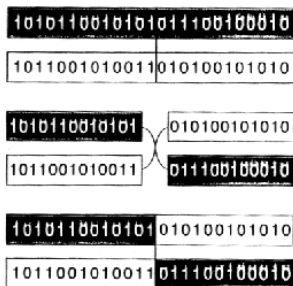
- ▶ Debe producir cromosomas válidos
- ▶ Debe diseñarse de acuerdo a la representación
- ▶ Posibilidades:
 - ▶ Tomar algunos genes de cada padre
 - ▶ Calcular valores medios entre algunos genes de los padres
 - ▶ ...

Mutación:

- ▶ Un bit es alterado con probabilidad p
- ▶ Se intercambian dos bits escogidos aleatoriamente
- ▶ Se suma a un gen (representado por un número real) un valor aleatorio
- ▶ ...

Ejemplos de operadores de cruce

Imagen clásica (John Holland) que introduce el operador de cruce



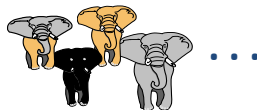
CROSSOVER is the fundamental mechanism of genetic rearrangement for both real organisms and genetic algorithms.

Chromosomes line up and then swap the portions of their genetic code beyond the crossover point.

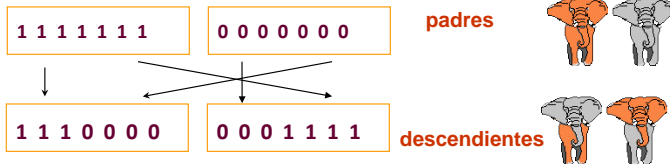
Ejemplos de operadores de cruce

Ejemplo 1 de operador de cruce

Población:

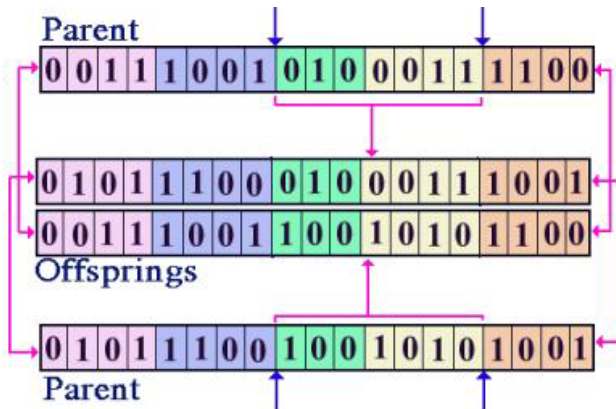


Cada cromosoma se corta en n partes que son recombinadas. (Ejemplo para $n = 1$).



Ejemplos de operadores de cruce

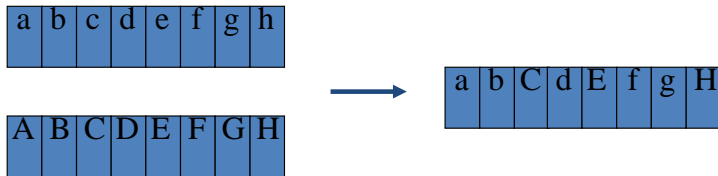
Ejemplo 2: operador de cruce en dos puntos para representación binaria



Ejemplos de operadores de cruce

Ejemplo 3: operador de cruce para representación real

Recombinación discreta (cruce uniforme): dados 2 padres se crea un descendiente como sigue:



Ejemplos de operadores de cruce

Ejemplo 4: operador de cruce para representación real

Recombinación aritmética (cruce aritmético):

a	b	c	d	e	f
---	---	---	---	---	---

A	B	C	D	E	F
---	---	---	---	---	---



$(a+A)/2$	$(b+B)/2$	$(c+C)/2$	$(d+D)/2$	$(e+E)/2$	$(f+F)/2$
-----------	-----------	-----------	-----------	-----------	-----------

Ejemplo de operador de mutación

Ejemplo de operador de mutación para representación binaria

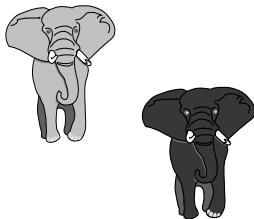
antes

1 1 1 1 1 1 1

después

1 1 1 0 1 1 1

gen mutado

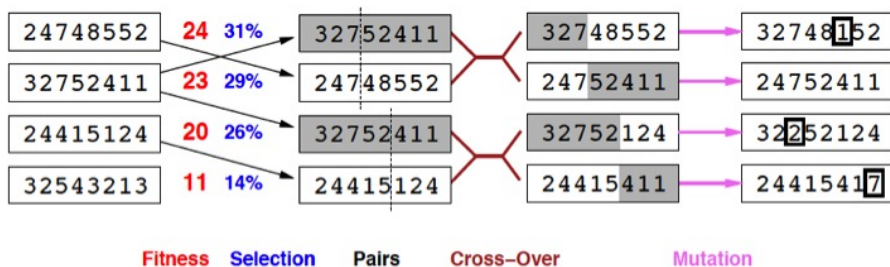


La mutación ocurre con una probabilidad p_m para cada gen

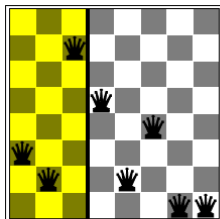
Proceso de reemplazo

- ▶ Hay que decidir cómo los nuevos cromosomas reemplazan a los viejos
- ▶ Pueden emplearse métodos aleatorios o determinísticos
- ▶ Podemos conservar el(los) mejor(es) cromosoma(s) de cada generación (**elitismo**)
- ▶ Modelo **generacional**: cada nueva generación reemplaza a la anterior
- ▶ Modelo **estacionario**: se van seleccionando parejas de individuos, que se reproducen y son reemplazados por sus descendientes

Ejemplo: esquema algoritmo genético

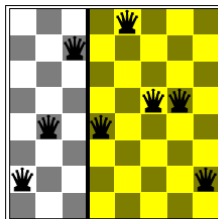


Ejemplo: 8-reinas



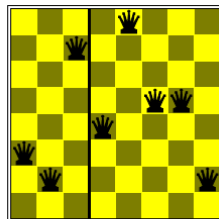
672 47588

+



752 51447

=



672 51447

Bibliografía recomendada

- ▶ S.J. Russell, P. Norvig. Inteligencia Artificial. Un enfoque moderno. Prentice Hall 2004.
- ▶ A.E. Eiben, J.E. Smith. Introduction to evolutionary computing. Springer, 2003.
- ▶ D.E. Goldberg. Genetic algorithms in search, optimization and machine learning. Addison Wesley, 1989.

Algunos enlaces:

- ▶ Vídeo 1: Algoritmo genético aplicado al problema del viajante de comercio
- ▶ Video 2: Algoritmo genético aplicado a planificación en un robot
- ▶ Vídeo 3: Algoritmos genéticos y redes neuronales para aprender a andar
- ▶ Vídeo 4: Un ejemplo de algoritmos genéticos y robocode
- ▶ Artículo: Genetic Algorithms Computer programs that *evolve* in ways that resemble natural selection can solve complex problems even their creators do not fully understand by John H. Holland