Algoritmos Geneticos

AGs repaso rapido

- Desarrollados en los EEUU en los 70s
- primeros promotores J. Holland, K. DeJong, D. Goldberg
- Tipicamente aplicados a optimisacion discreta
- Propiedades atribuidas:
  - no son muy rapidos
  - buenas heuristicas para problemas combinatoriales
- Propiedades especiales:
  - hacen incapie en combinar buenas soluciones de los padres por medio del crossover
  - muchas variantes: dependen de los operadores de seleccion de padres y de generacion siguiente

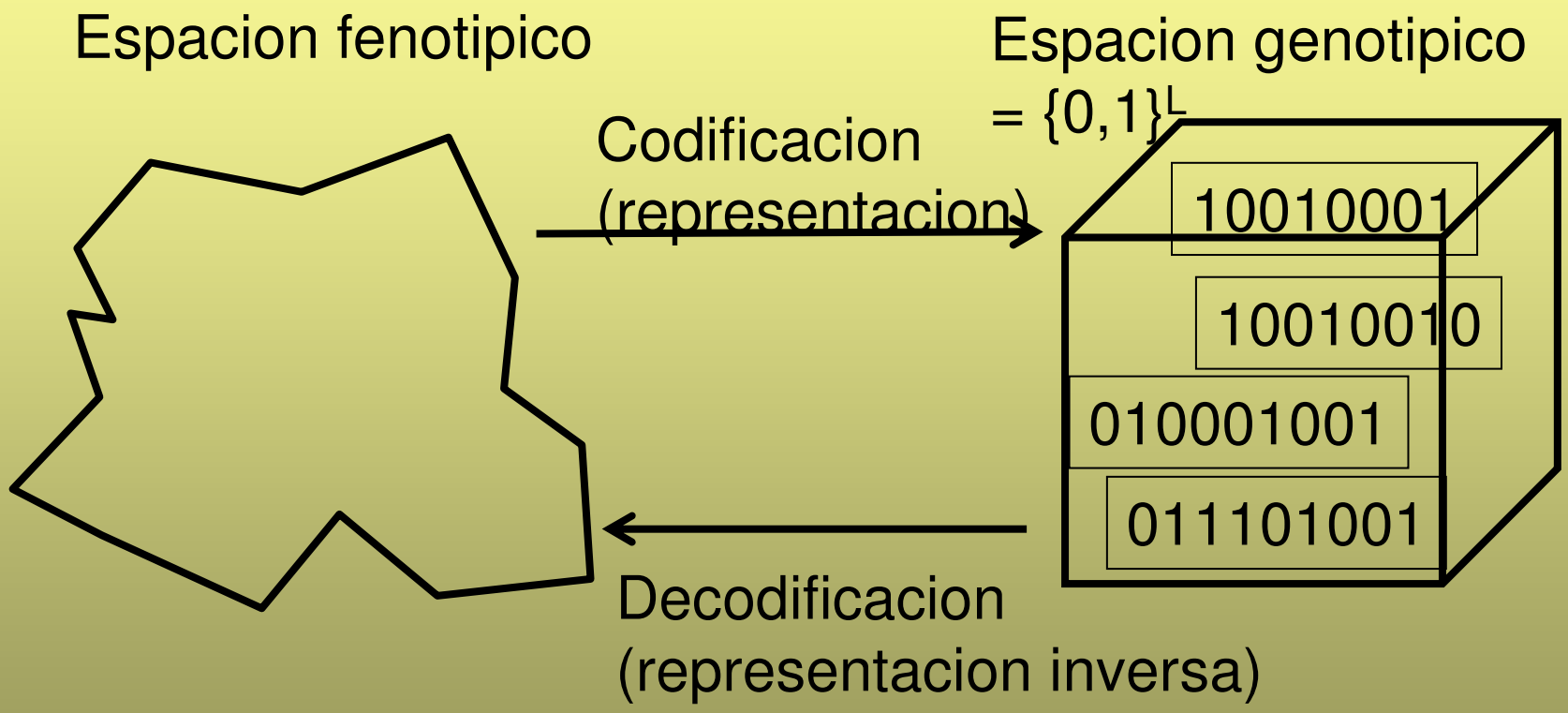
Algoritmos Geneticos

- Los algoritmos originales de J. Hollan se conocen ahora como los Algoritmos Geneticos Simples
- Otros algoritmos geneticos usan distintas:
  - Representaciones
  - Mutaciones
  - Crossovers
  - Mecanismos de seleccion

Tableau tecnica de los AGS

Representacion	cadenas binarias
Recombinacion	N-point o uniforme
Mutacion	“Bitwise bit-flipping” con probabilidad fija
Seleccion de padres	Proporcional al fitness
Seleccion de sobrevivientes	Los hijos reemplazan a los padres
Especialidad	Enfasis en el crossover

Representacion

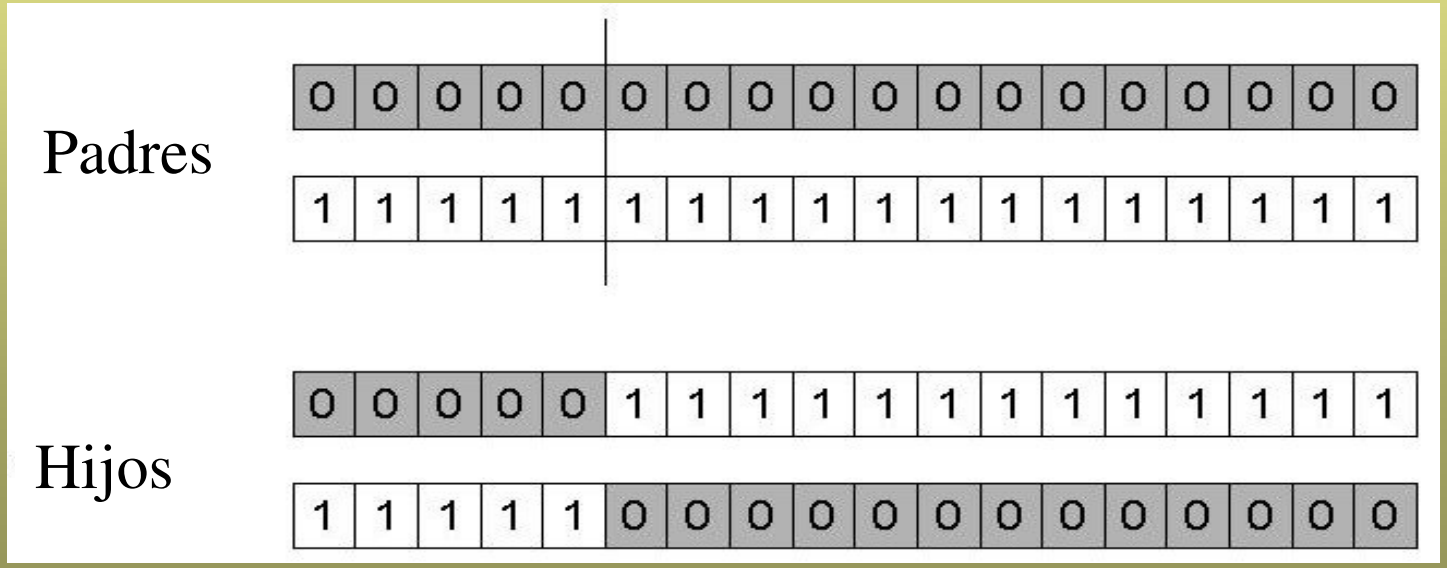


Ciclo de reproduccion

1. Seleccionar padres para el conjunto reproductivo (mating pool, |mating pool| = tamanio de la poblacion)
2. mezclar el mating pool
3. Para cada par consecutivo del mating pool aplicar crossover con probabilidad P_c, sino copiar padres
4. Para cada nuevo individuo aplicar mutacion (bit-flip con probabilidad p_m independiente en cada bit)
5. Reemplazar toda la poblacion con los nuevos individuos

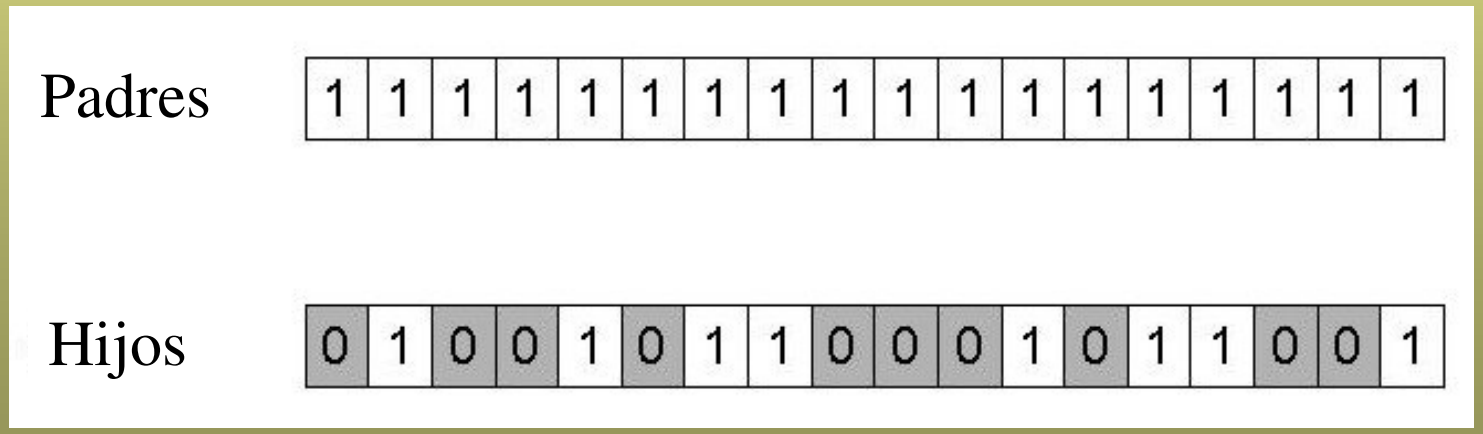
Operadores del AGs: 1-point crossover

- Seleccionar un punto al azar en ambos padres
- Cortar los padres en este punto
- Crear hijos intercambiando las colas
- P_c tipicamente en el rango (0.6, 0.9)



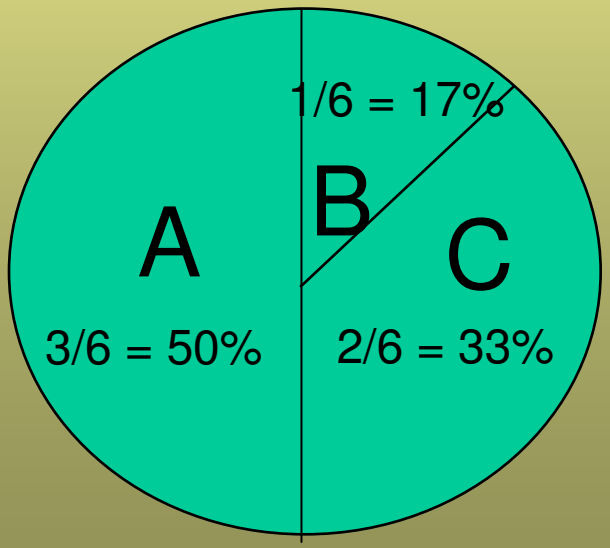
Operadores del AGs: mutacion

- Alterar cada gene con una probabilidad independiente p_m
- p_m se llama la probabilidad de mutacion
  - Tipicamente entre [1/pop_size, 1/tamanio_cromosoma]



Operadores del AGs: Seleccion

- Idea principal:los mejores individuos tienen mas chances
  - Chaces proporcionales al fitness
  - Implementacion: tecnica de la ruleta
  - Asignar a cada individuo una parte de la ruleta proporcional a su fitness
    - » rotar la ruleta n veces para obtener n individuos



fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

Un ejemplo de Goldberg ‘89 (1)

- Problema simple: max x^2 sobre {0,1,…,31}
- Enfoque AG:
  - Representacion: codigo binario, 01101 $\leftrightarrow$ 13
  - Tamanio poblacion: 4
  - 1-point xover, bitwise mutacion
  - Seleccion de ruleta
  - Inicializacion al azar
- Ejecutamos a mano una generacion:

x^2 ejemplo: seleccion

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

X^2 ejemplo: crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

X^2 ejemplo: mutacion

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

The simple GA

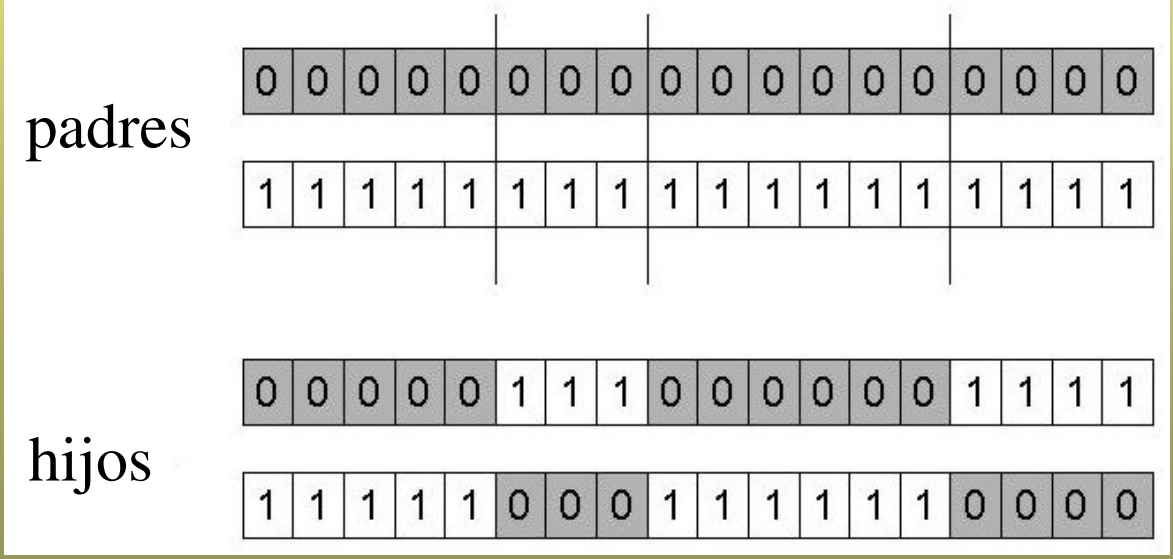
- Ha sido ampliamente estudiado
  - aun se lo usa como benchmark
- Tiene muchas limitaciones
  - La representacion es muy restrictiva
  - Mutacion y Crossover solo aplicable a representaciones binarias y enteras
  - El mecanismo de seleccion es sensitivo a poblaciones que convergen y tienen valores de fitness muy cercanos
  - El mecanismo de reemplazo se puede mejorar con un mecanismo de reemplazo explicito

Operadores de crossover alternativos

- La performa del operador 1-point depende del orden en el que las variables aparecen en el algoritmo
  - mas probable que mantenga genes que se encuentran cerca
  - No puede mantener juntos genes que se encuentran en los extremos
  - Esto se conoce como *Positional Bias*
  - Puede ser explotado si se conoce la estructura del problema pero generalmente esto no ocurre.

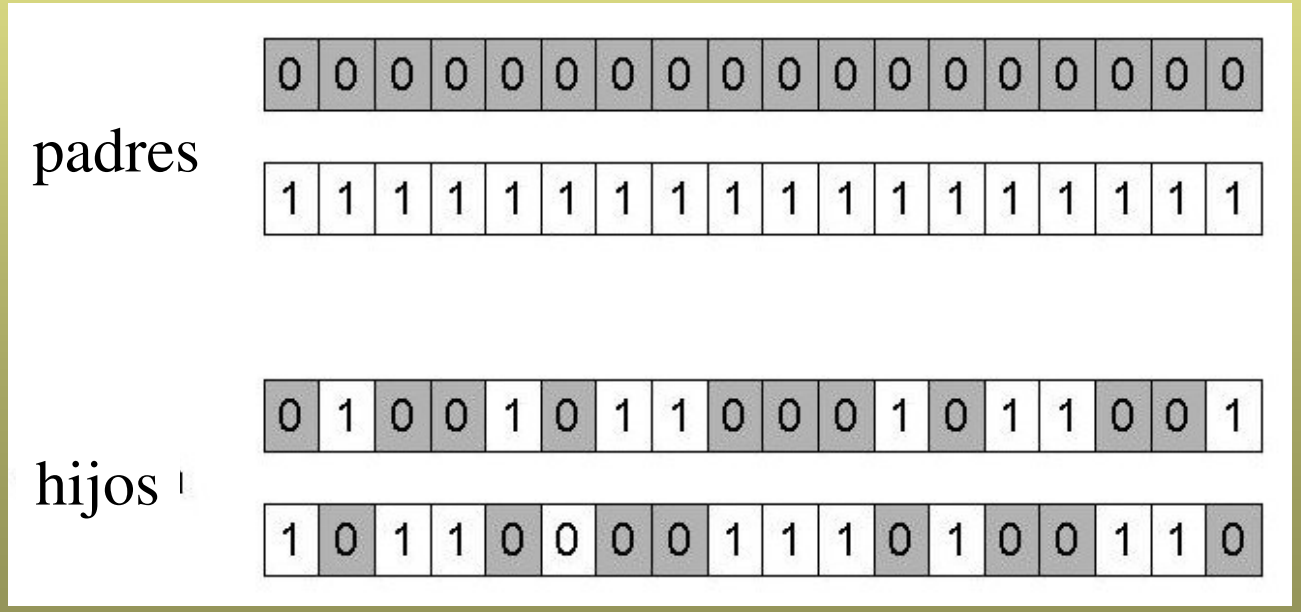
Crossover de n-puntos

- Elegir n puntos al azar
- Cortar en esos puntos
- Pegar las partes alternando entre los padres
- Generalizacion del crossover 1-punto (aun existe bias posicional)



Crosover uniforme

- Asignar “caras” a un padre y “sello” al otro
- Tirar una moneda para cada gen del hijo
- Hacer una copia inversa para el segundo hijo
- Herencia es independiente de la posicion



Crossover O mutaxion? (1)

- Debate de mas de una decada: cual es mejor / necesario / principal-secundario

- Acuerdo parcial:
  - depende del problema pero:
  - en general es bueno tener ambos
  - cada uno tiene roles distintos
  - AE solo con mutacion puede funcionar, solo con crossover en general devolvera un optimo local

Crossover O mutaxion? (2)

Exploracion: obtiene informacion del espacio de busqueda

Explotacion: refina la busqueda en cierta region prometedora del espacio

Crossover O mutaxion? (3)

- Solo, el crossover puede combinar la informacion de dos padres
- Sola, la mutacion puede introducir nuevos alelos
- Crossover no cambia la frecuencia de los alelos en la poblacion
- Para llegar al optimo generalmente uno necesita a “lucky strike”

Otras representaciones

- "Gray coding" de enteros (bitstrings)
  - Es un mapeo que hace que pequenias variaciones en el genotipo produscan pequenias variaciones en el fenotipo de un individuo.
  - Esto le hace la vida mas facil al AG

Hoy en dia se acepta que es mejor codificar las variables numericas directamente como:

- enteros o
- valores en punto flotante.

Representaciones Enteras

- Algunos problemas tienen variables enteras/reales de forma natural (procesado de imagenes, plegado de proteinas, etc)
- Otros tienen valores categoricos {blanco, rojo, verde, etc}
- N-point / uniform funcionan igual
- hay que extender la mutacion para que “gateen”
  - “gatear” permite moverse a valores similares en un conjunto categorico

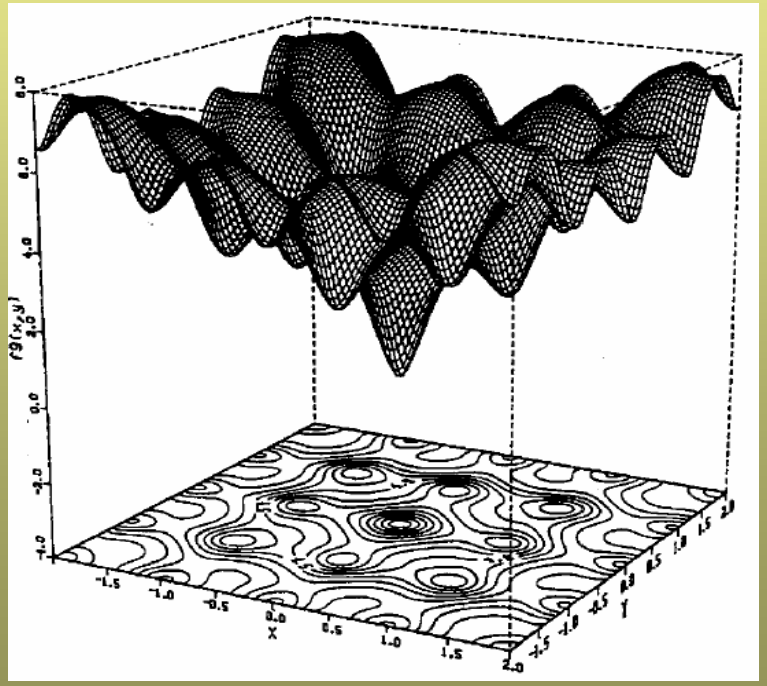
Problemas con valores reales

- Optimizacion de parametros continuos $f: \Re^n \rightarrow \Re$
- Ejemplo: Funcion de Hackley (usado como benchmark)

$$f(\overline{x}) = -c_1 \cdot exp\left(-c_2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)$$

$$-exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} cos(c_3 \cdot x_i)\right) + c_1 + 1$$

$$c_1 = 20,\ c_2 = 0.2,\ c_3 = 2\pi$$



Mapeando valores reales a bit strings

$z \in [x,y] \subseteq \mathfrak{R}$ representado por $\{a_1,\ldots,a_L\} \in \{0,1\}^L$

- $[x,y] \rightarrow \{0,1\}^L$ debe ser invertible (un fenotipo por genotipo)
- Γ: $\{0,1\}^L \rightarrow [x,y]$ define la representacion

$$\Gamma(a_1,\ldots,a_L) = x + \frac{y-x}{2^L-1} \cdot \left(\sum_{j=0}^{L-1} a_{L-j} \cdot 2^j\right) \in [x,y]$$

- Se discretiza el espacio real y solo 2^L valores se representan
- L la precision maxima de una solucion
- Mas precision → cromosomas mas largos → mas lenta la obtencion de buenos designoides

Mutaciones para punto flotante (1)

Esquema general de mutacion:

$$\bar{x} = \langle x_1, \ ..., x_l \rangle \rightarrow \bar{x}' = \langle x_1', ..., x_l' \rangle$$
$$x_i, x_i' \in [LB_i, UB_i]$$

Mutacion uniforme:

Analogo a bit-flip (binario) o reseteo al azar (entero)

x_i' drawn randomly (uniform) from $[LB_i, UB_i]$

Mutaciones para punto flotante (2)

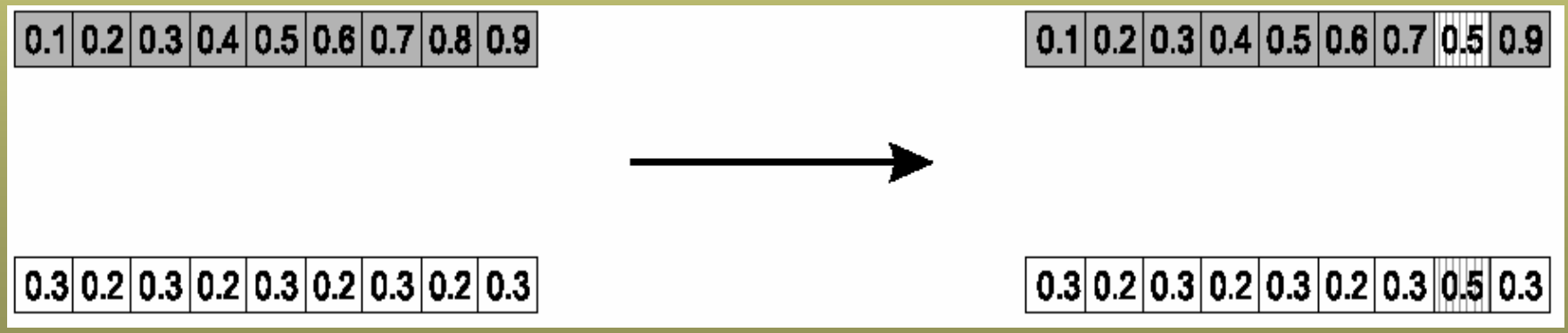
- Mutaciones no uniformes:
  - se propusieron muchos metodos, por ejemplo donde los UB and LB cambian con el tiempo.
  - La mayoria son esquemas probabilisticos pero en general producen pequenios cambios
  - Lo mas comun es modificar cada variable mediante un sampling en una distribucion Gausiana $N(0, \sigma)$
  - La desviacion estandard σ controla la cantidad de cambio, por ejemplo 2/3 de las desviaciones van a parar al rango $[-\sigma, +\sigma]$

Crossover para AGs con valores reales

- Discrete:
  - cada alelo z del hijo deriva de uno de los padres (x,y) con igual probabilidades: $z_i = x_i$ o y_i
  - Puede usar crosover de n-puntos (o uniforme)
- Intermedio
  - se basa en la idea de crear hijos en medio de los padres (una combinacion convexa)
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ donde $\alpha : 0 \leq \alpha \leq 1.$
  - El parametro α puede ser:
    - constante: crosover aritmetico uniforme
    - variable (depende de la edad de la poblacion)
    - al azar

Crosover aritmetico simple

- Padres: $\langle x_1,\ldots,x_n \rangle$ **y** $\langle y_1,\ldots,y_n \rangle$
- Elegir al azar un gen (*k*),
- Hijo$_1$ es: $\left\langle x_1, ..., x_k, \alpha \cdot y_k + (1-\alpha) \cdot x_k, ..., x_n \right\rangle$
- Reverso para el otro “pibe” con $\alpha = 0.5$

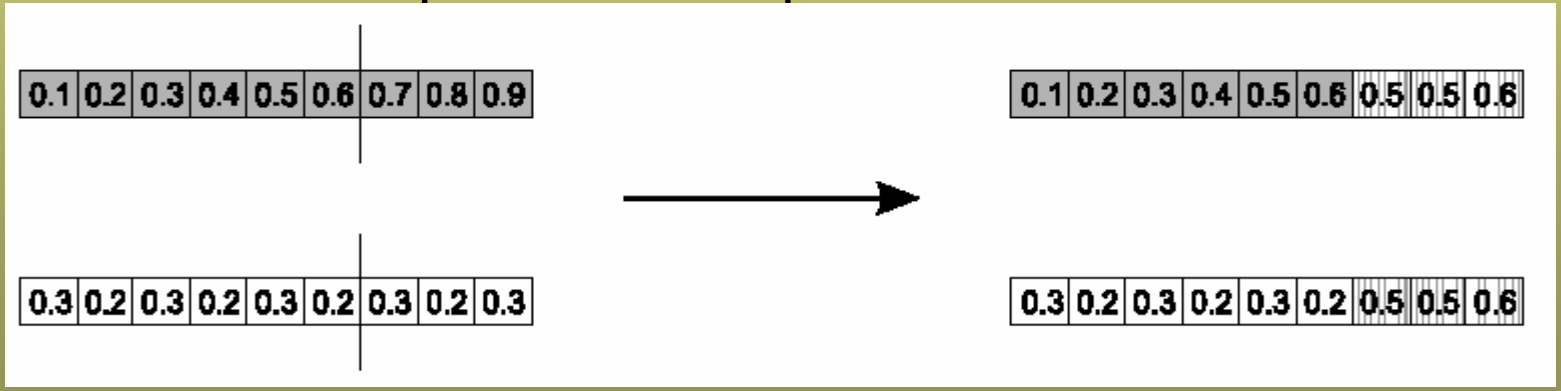


Simple crosover aritmetico

- Padres: $\langle x_1,\ldots,x_n \rangle$ y $\langle y_1,\ldots,y_n \rangle$
- Elegir un punto al azar *(k)* despues de ese punto mezclar
- $Hijo_1$ es:

$$\left\langle x_1,\ldots,x_k,\alpha\cdot y_{k+1}+(1-\alpha)\cdot x_{k+1},\ldots,\alpha\cdot y_n+(1-\alpha)\cdot x_n\right\rangle$$

- El reverso para el otro “pibe” con $\alpha = 0.5$

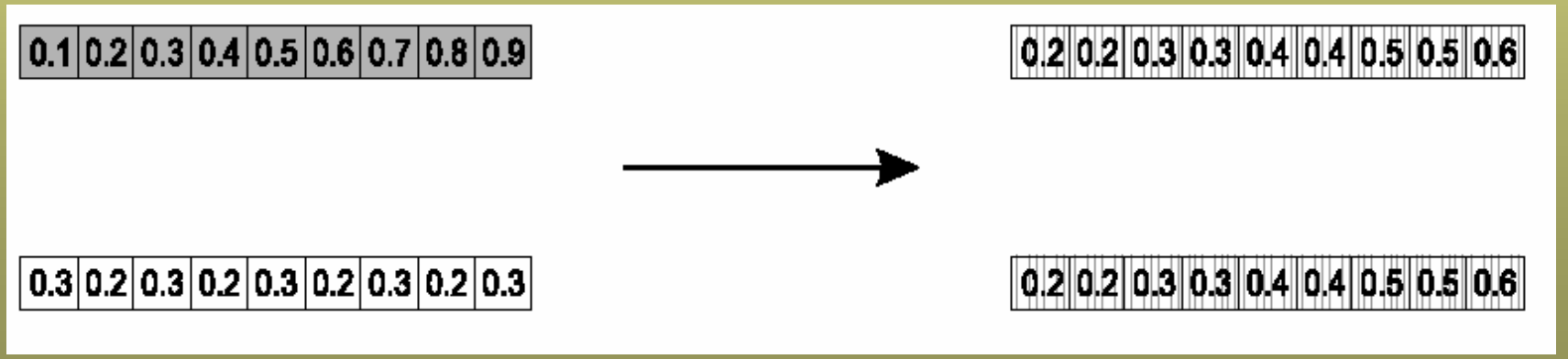


Crosover aritmetico completo

- Mas comunmente usado
- Padres: $\langle x_1,\ldots,x_n \rangle$ y $\langle y_1,\ldots,y_n \rangle$
- $Hijo_1$ es:

$$a \cdot \bar{x} + (1-a) \cdot \bar{y}$$

- Reverse para el otro “pibe” con $\alpha = 0.5$



Representacion de Permutaciones

- Problemas de ordenamiento/secuenciamiento

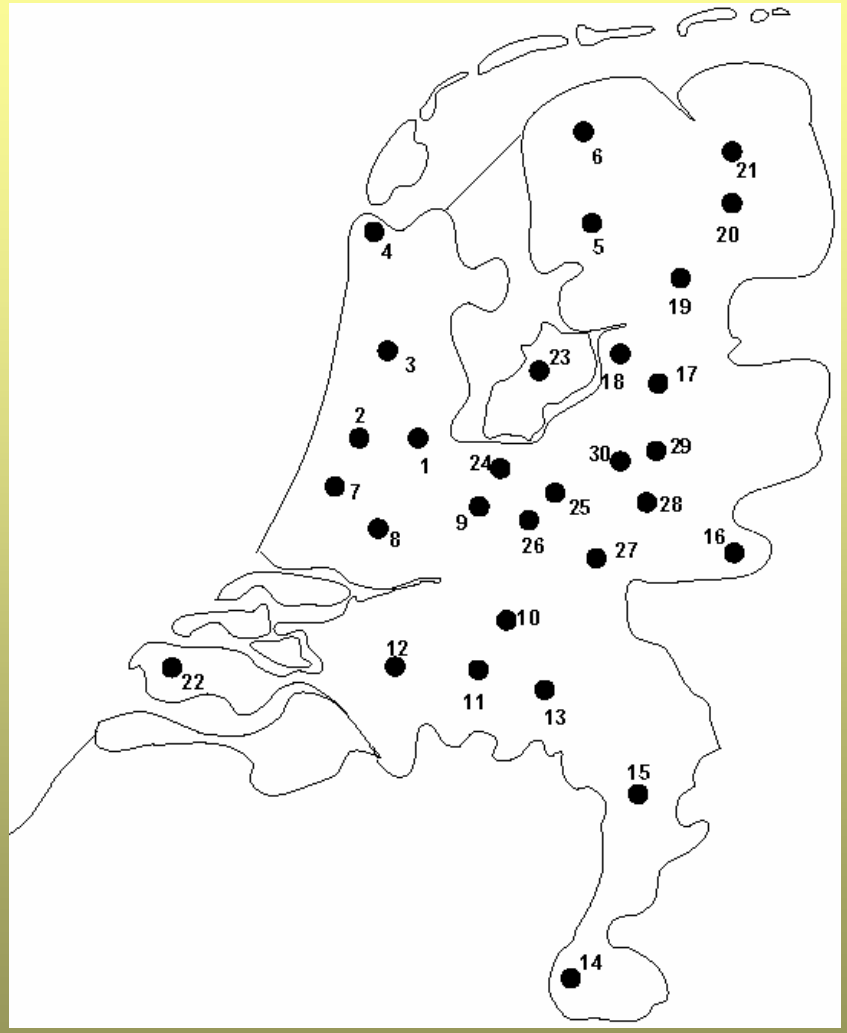
- La tarea consiste en encontrar un ordenamiento de objetos:
  - Ejemplo: algoritmos de ordenamiento (que objetos ocurren delante de cuales otros objetos)
  - Ejemplo: Travelling Salesman Problem (TSP) : lo importante es que objetos ocurren cerca de otros objetos (adyacencia)

- Estos problemas se expresan como permutaciones:
  - si existen *n* variables entonces la representacion es una lista de *n* enteros, cada uno ocurriendo exactamente una vez

El TSP

- Problema:
  - dadas n ciudades
  - encontrar un tour hamiltoneano de longitud minima
- Codificacion:
  - etiquetar las ciudades 1, 2, … , *n*
  - un tour hamiltoneano es una permutacion (n =4 [1,2,3,4], [3,4,2,1] son OK)
- El espacio de busqueda es enorme:

  para 30 ciudades hay 30! ≈ 10^{32} tours posibles



Mutaciones para permutaciones

- la mutacion normal produce permutaciones invalidas ya que se repetirian etiquetas
- Tienen que cambiar al menos dos alelos
- La probabilidad de mutacion se refiere en estos casos a las chances de modificar el genotipo no a las chances de modificar un gene solamente

Mutacion de Insercion para Permutaciones

- Elegir al azar dos alelos
- Mover el segundo a continuacion del primero y hacer un shift del resto
- Notar que esto preserva casi toda la informacion de adyacencias (orden)

Mutacion de intercambio

- Elegir dos alelos e intercambiar sus valores
- Preserva casi toda la informacion de adyacencia (4 links rotos) pero el orden se modifica mas (en el casi anterior se rompian tres links)

Mutacion de Inversion para Permutaciones

- Elegir dos puntos del genotipo e invertir la subcadena encerrada entre esos puntos.
- Preserva casi toda la informacion, rompe solo dos lins

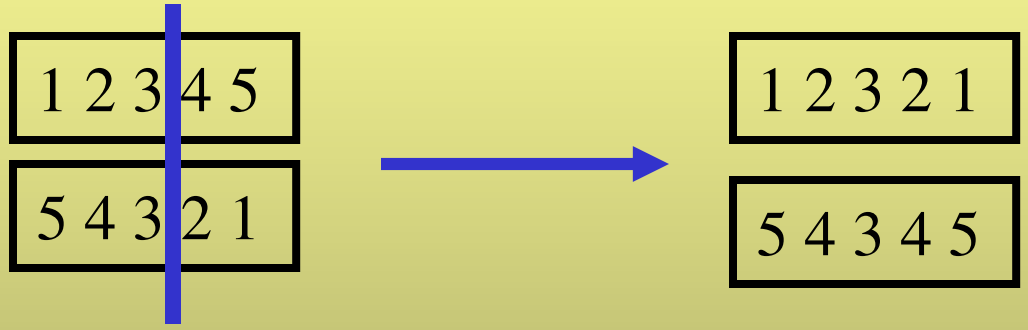
Mutacion “scramble” para Permutaciones

- Elegir un subset de alelos
- reacomodar al azar los alelos en el subconjunto

(el subconjunto no tiene porque ser contiguo)

Operadores de Cruzamiento para Permutaciones

- Los crosovers estandar generalmente conllevan a soluciones inadmisibles.



- Se han propuestos varios operadores para permutaciones para tratar de preservar informacion entre padres e hijos

Partially Mapped Crossover (PMX)

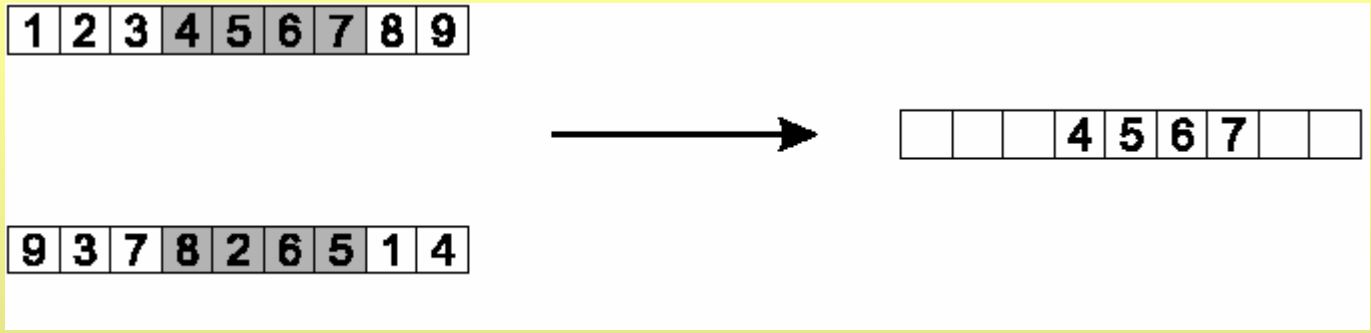
Informal procedure for parents P1 and P2:

1. Copiar un segmento al azar de P1
2. Comenzando en el primer punto de crossover buscar elementos en ese segmento de P2 que no han sido copiados
3. Para cada uno de esos *i* buscar en el hijo que elementos *j* han sido copiados en su lugar
4. Poner *i* en la posicion ocupada por *j* en P2, porque sabemos que no *pondremos j* alli ya que j ya fue copiado.
5. Si el lugar ocupado por *j* en P2 ya ha sido ocupado en el hijo en *k*, poner*i* en la posicion ocupada por *k* en P2
6. El resto puede ser copiado de P2.

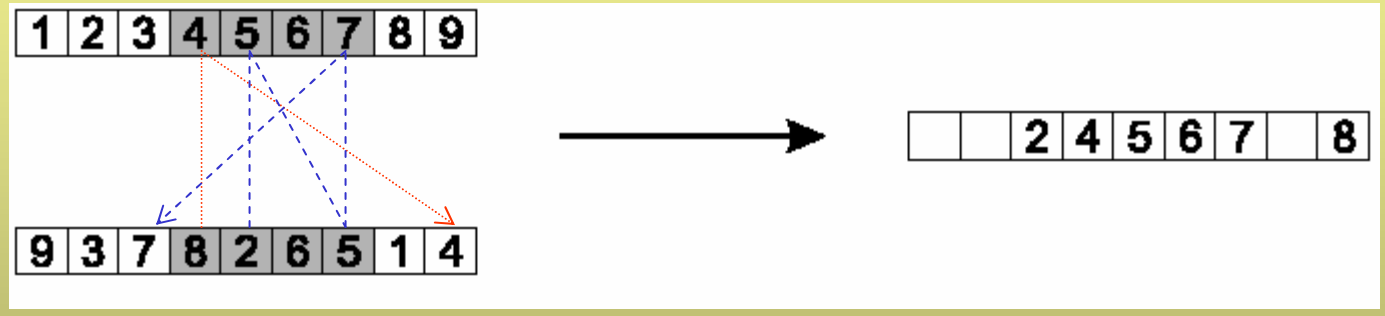
Se hace lo mismo para el otro hijo

PMX ejemplo

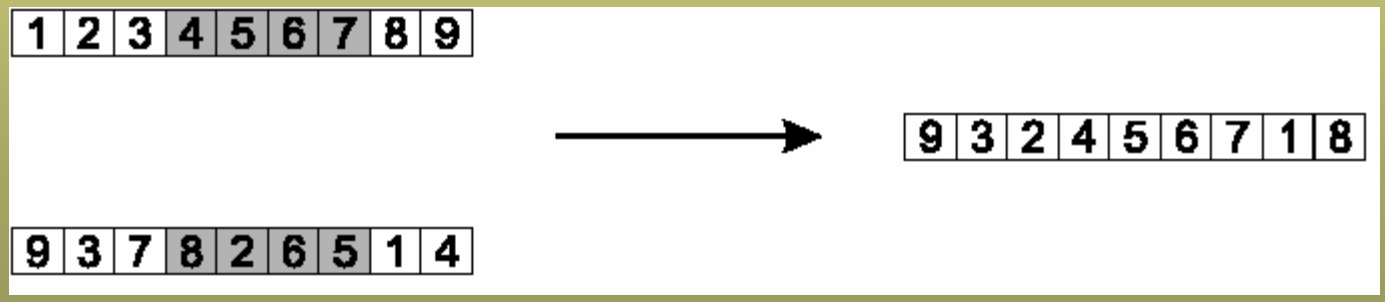
- Paso 1



- Paso 2



- Paso 3



Ciclo crossover

Idea basica:

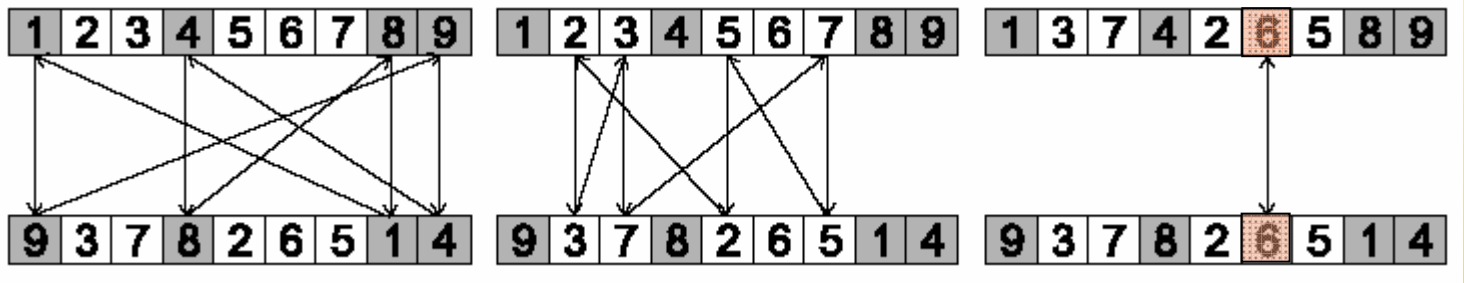
Cada alelo viene de un padre *junto con su posicion*.

Procedimiento informal:

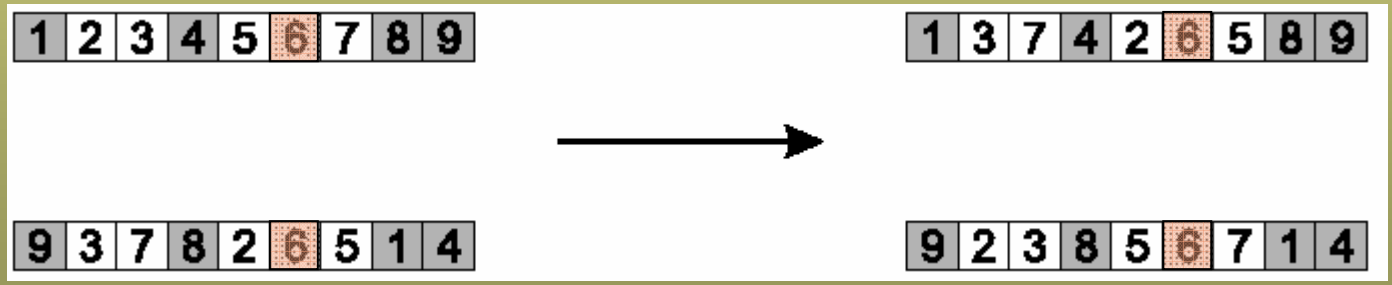
1. Armar un ciclo de alelos de P1 de la siguiente manera:
   (a) Comenzar con el primer alelo de P1.
   (b) Mirar al alelo en *la misma posicion* en P2.
   (c) Ir a la posicion con el *mismo alelo* en P1.
   (d) Agregar este alelo al ciclo.
   (e) Repetir b hasta d hasta arrivar al primer alelo de P1.
2. Poner los alelos del ciclo en el primer hijo en las posiciones que vienen del primer padre.
3. Tomar el siguiente ciclo del otro padre

Ejemplo

- Paso 1: identificar ciclos



- Paso 2: copiar ciclos alternativos en los hijos



•Existen muchos otros crossovers: Edge Crosover, Sex crosover,DPX, etc

•En mi tesis de doctorado “Studies on the theory and design space of memetic algorithms” doy ejemplos de mutaciones y crossovers con algunas estimaciones de tamanios de vecindarios inducidos, complejidades e inclusiones.

Recombinacion de Multi-Parental

- No estamos restringidos como en la naturaleza
- La mutacion usa aridad 1 y el crosover “natural” 2.
- No muy populares pero ciertos indicios de que puede ser util una aridad mayor a 2
- Tres tipos principales:
  - Basado en la frecuencia de los alelos, por ejemplo, “p-sexual voting” generaliza el uniform crossover
  - Basado en la segmentacion y recombinacion de los padres, por ejemplo, “diagonal crossover” generaliza el n-point crossover
  - Basados en operaciones aritmeticas sobre los alelos, por ejemplo, crosover de centro de masa, crosover aritmetico generalizado, etc

Modelos de Poblacion

- El AGs usa un modelo “generacional”:
  - cada individuo sobrevive solo una generacion
  - todo el conjunto de padres es reemplazado por los hijos
- En el otro extremo estan los “Steady-State” AGs:
  - se genera un solo hijo por generacion,
  - se reemplaza un solo miembro de la poblacion por este hijo
- Gap generacional
  - la proporcion de la poblacion reemplazada
  - 1.0 para AGs, 1/|poblacion| para SSAGs

Competicion Basada en el Fitness

- La seleccion puede ocurrir en dos lugares:
  - Seleccion en la generacion actual para tomar parte del entrecruzamiento
  - Seleccion entre padres + hijos para sobrevivir a la siguiente generacion
- Los operadores de seleccion operan sobre individuos enteros , esto es, son independientes de la representacion

Ejemplos con AGs

- Numero esperado de copias del individuo *i*

$$E(n_i) = \mu \cdot f(i) / \langle f \rangle$$

(μ = |poblacion|, f(i) = fitness de i, ⟨f⟩ fitness promedio en la pob.)

- Algoritmo de la ruleta:
  - Dada una distribucion de probabilidades, rotar una ruleta de 1 brazo *n* veces para hacer *n* selecciones
  - No hay garantias sobre el numero n_i
- el algoritmo SUS de Baker:
  - *n* brazos equidistantes y se rota la ruleta solo una vez
  - Garantiza piso$(E(n_i)) \leq n_i \leq techo(E(n_i))$

Seleccion Proporcional al Fitnes

- Problemas incluyen
  - Un individuo muy "fit" puede monopolizar la poblacion muy rapido: convergencia prematura
  - Al final de las ejecuciones cuando la poblacion esta muy convergida pierde presion selectiva
- El escalado del fitness puede compesar este problema:
  - "Windowing": $f'(i) = f(i) - \beta^t$
    - donde β es el peor fitness en la(s) ultima(s) generacion(nes)
  - "Sigma Scaling": $f'(i) = max(f(i) - (\langle f \rangle - c \cdot \sigma_f), 0.0)$
    - donde c es constante, usualmente 2.0

Seleccion Basada en el Rango

- Trata de contrarestar los problemas de la seleccion proporcional al fitnes al basarse en rangos relativos y no valores absolutos
- Ranquea poblacion del mejor al peor, donde el mejor tiene rango μ y el peor 1
- Esto incurre en un costo computacional extra ya que se debe reordenar la poblacion

Ranking lineal

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

- Parametrizada por el factor *s:* $1.0 < s \leq 2.0$
  - mide la ventaja del mejor individuo
  - en un AGs es el numero de hijos que se le alocan
- Ejemplo con 3 individuos (Psel = 1-Plin)

	Fitness	Rank	P_{selFP}	$P_{selLR} \quad (s=2)$	$P_{selLR} \quad (s=1.5)$
A	1	1	0.1	0	0.167
B	5	2	0.5	0.67	0.5
C	4	2	0.4	0.33	0.33
Sum	10		1.0	1.0	1.0

Ranking Exponencial

$$P_{exp-rank}(i) = \frac{1 - e^{-i}}{c}$$

- El ranking lineal es limitado a la presion selectiva
- El ranking exponencial puede alocar mas de dos copias al mejor individuo
- Se normaliza el factor constance c de acuerdo al tamanio de la poblacion

Seleccion de Torneo(1)

- Todos los metodos anteriores necesitan informacion global de la poblacion
  - puede ser un cuello de botella en implementaciones paralelas
  - depende de una funcion externa de fitnes que podria no existir/ser conocida (jugadores designoids)
- Procedimiento informal:
  - Elegir *k* miembros de la poblacion y seleccionar el mejor
  - Repetir para seleccionar mas individuo

Seleccion de Torneo(2)

- La probabilidad de elegir i depende de:
  - El ranking de i
  - El tamanio del sample k
    - cuanto mas grande sea k mayor sera la presion selectiva
  - Si los contrincantes se eligen con/sin reemplazo
    - seleccionando sin reemplazo aumenta la presion selectiva
  - Si el mejor siempre gana o depende de una probabilidad p
- Para $k = 2$, el tiempo para que el mejor individuo monopolice la poblacion es igual al ranking lineal con $s = 2 \bullet p$

Seleccion de Sobrevivientes

- La mayoria de los metodos anteriores son usados en la seleccion de padres
- La seleccion de sobrevivientes se puede dividir en dos enfoques:
  - Seleccion basada en la edad
    - ejemplo AGs
    - en un AGSS “first-in-first-out” (borra el mas viejo)
  - Seleccion basada en el fitnes
    - usa alguno de los metodos mencionados

Dos Casos Especiales

- Elitismo
  - Ampliamente usado en ambos AG generacionales
  - Siempre preserva una copia del mejor individuo encontrado hasta ahora
- "Borrar el Peor"
  - Del algoritmo de Whitley's que origino al steady state GA, tambien usaba linear ranking para la seleccion de padres
  - Monopolizacion rapida : usar con poblaciones grandes o impedir duplicaciones