Genetic Algorithms

Chapter 3

GA Quick Overview

- Most widely known type of EA
- Developed: USA in the 1970's
- Early names: J. Holland, K. De Jong, D. Goldberg
- Typically applied to:
  - discrete optimization (Boolean, integer, permutation, etc.)
- Attributed features:
  - not too fast
  - good heuristic for combinatorial (組合) problems (eg. TSP)
- Special Features:
  - Traditionally emphasizes combining information from good parents (crossover)
  - many variants, e.g., reproduction models, operators

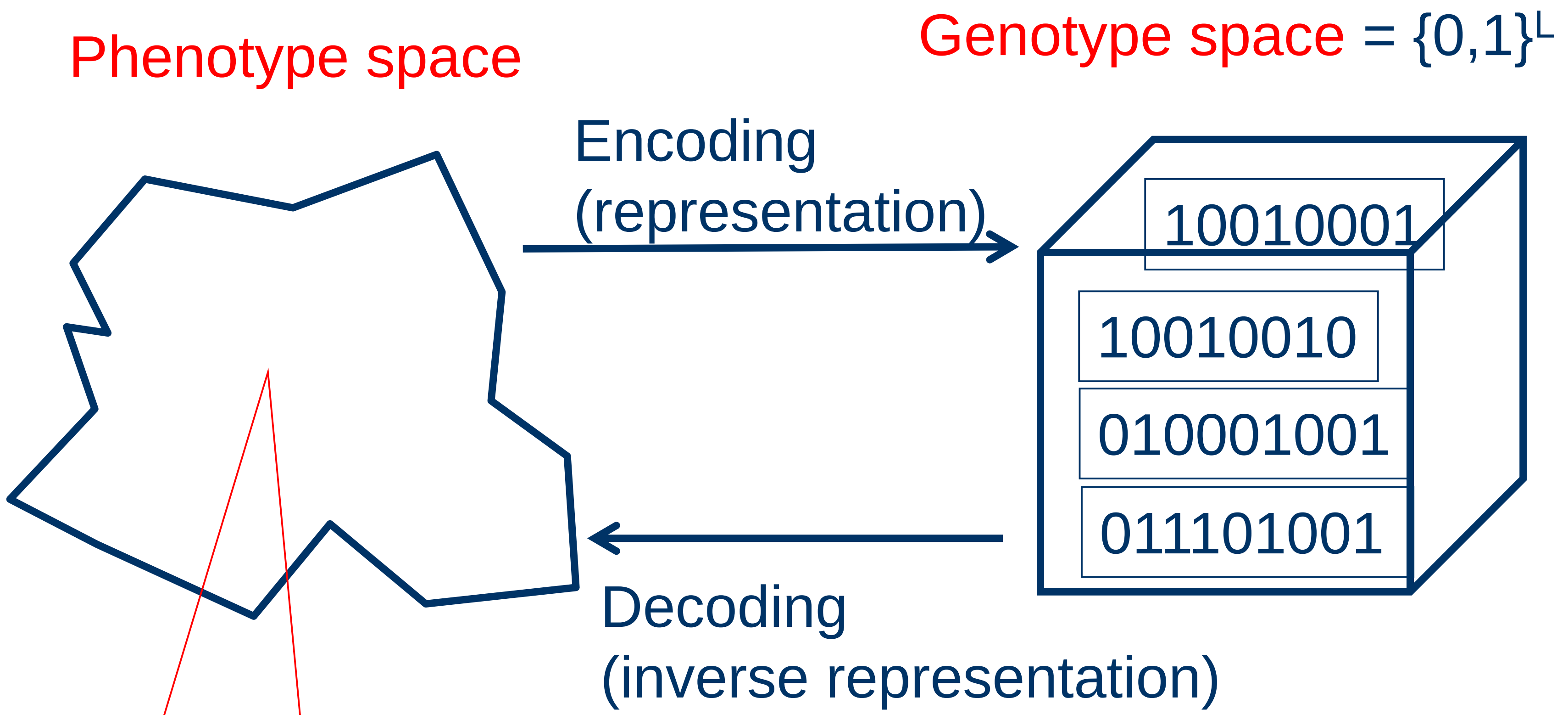
Genetic algorithms

- Holland’s original GA is now known as the **Simple Genetic Algorithm** (**SGA**)
- Other GAs use different:
  - Representations (binary, integer, real-value, permutation)
  - Mutations
  - Crossovers
  - Selection mechanisms
- Most widely known

SGA Technical Summary Tableau

Representation	Binary strings
Recombination	1-point (N-point or uniform)
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate (對應適應值)
Survivor selection	All children replace parents
Speciality	Emphasis on crossover (Xover)

Representation



- Not always continuous!
- Can be discrete
- can be non-convex (非凸)

Representation

- “Right” representation for the problem being solved.
- Representation is closely related to variation operators (recombination & mutation).
- Representation (Genotype to Phenotype mapping) should be as **natural** as possible for the problem under consideration.

- Bit string (genotype consists simply of a bit string) has been mistakenly used independently of the problem to be solved.
- Bit string: Perfect match for Boolean decision problem, but not for other problems.
- Ex: 80-bit chromosome represents
  - ➢Ten 8-bit integers, or
  - ➢Five 16-bit real numbers
- Why not use integer or real-valued representations directly? Better results can be obtained!!

- Another problem: Different bits have different significance for bit string?

0110← 0111→1000

➔Use grey code (**minimum-change code**)

Grey code: Hamming distance =1
(任兩相鄰編碼差距只有一位元)

0000 0001 0011 0010 0110 0111 0101 0100

0110← 0111→1000

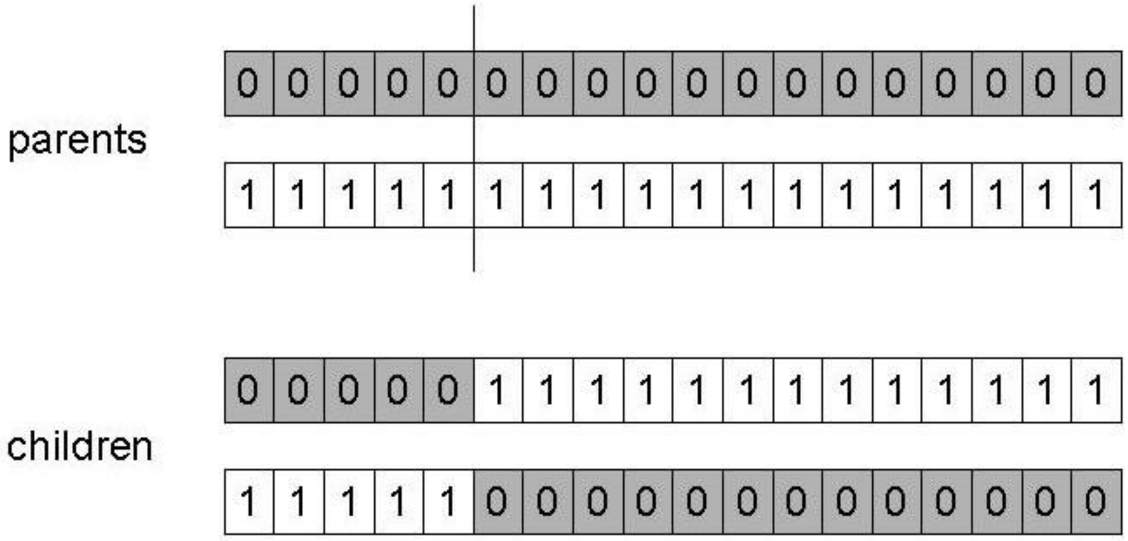
Prefer equal chance of changing 7→8 and 7→6, by independent bit-flip.

SGA reproduction cycle

1. Select parents for the mating pool
   (size of mating pool = population size)
2. Shuffle the mating pool (搞亂、弄混、洗牌)
3. For each consecutive pair apply crossover with probability p_c , otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
5. Replace the whole population with the resulting offspring (i.e., aged based)

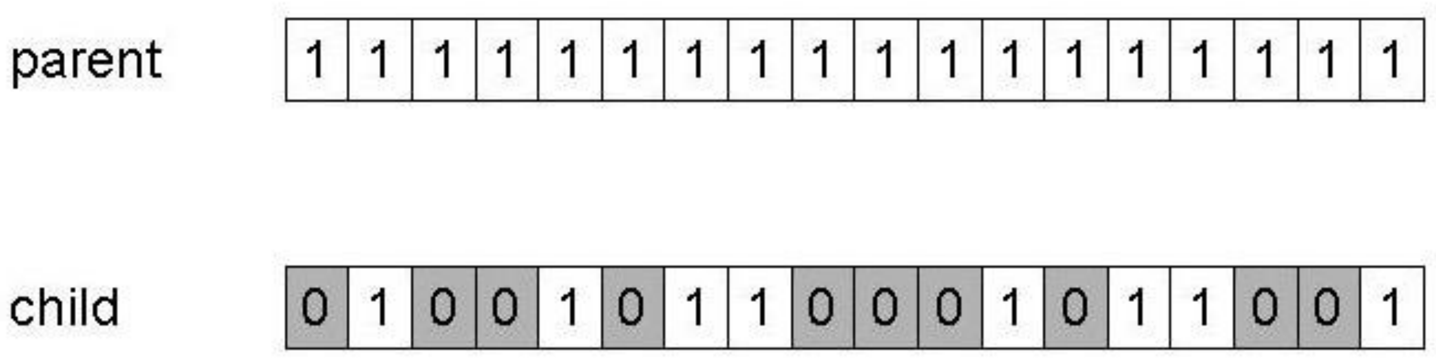
SGA operators: 1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- p_c typically in range (0.6, 0.9) or [0.5 1.0]



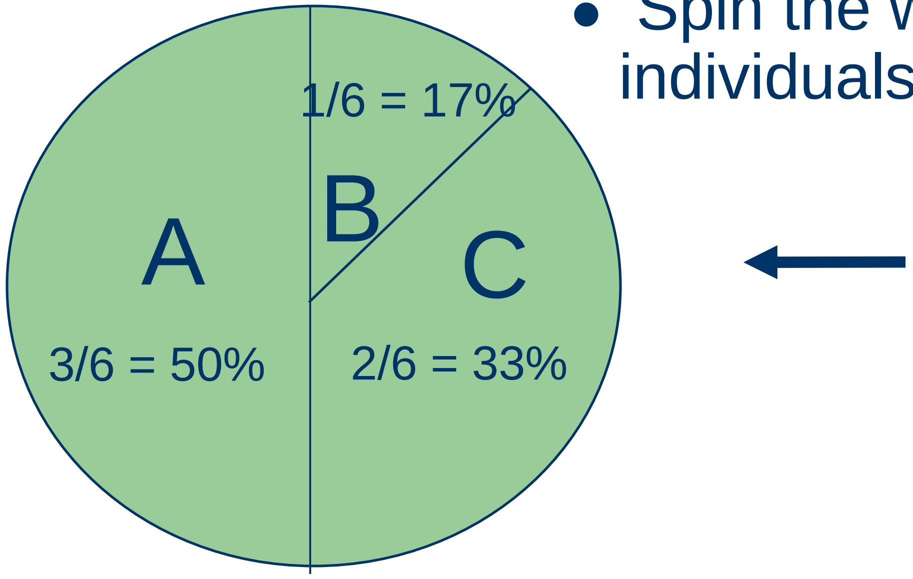
SGA operators: mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between (1/pop_size) and (1/ chromosome_length)



SGA operators: Selection

- Main idea: better individuals get higher chance
  - Chances proportional to fitness
  - Implementation: roulette wheel (輪盤) technique
    - Assign to each individual a part of the roulette wheel
    - Spin the wheel *n* times to select n individuals



fitness(A) = 3

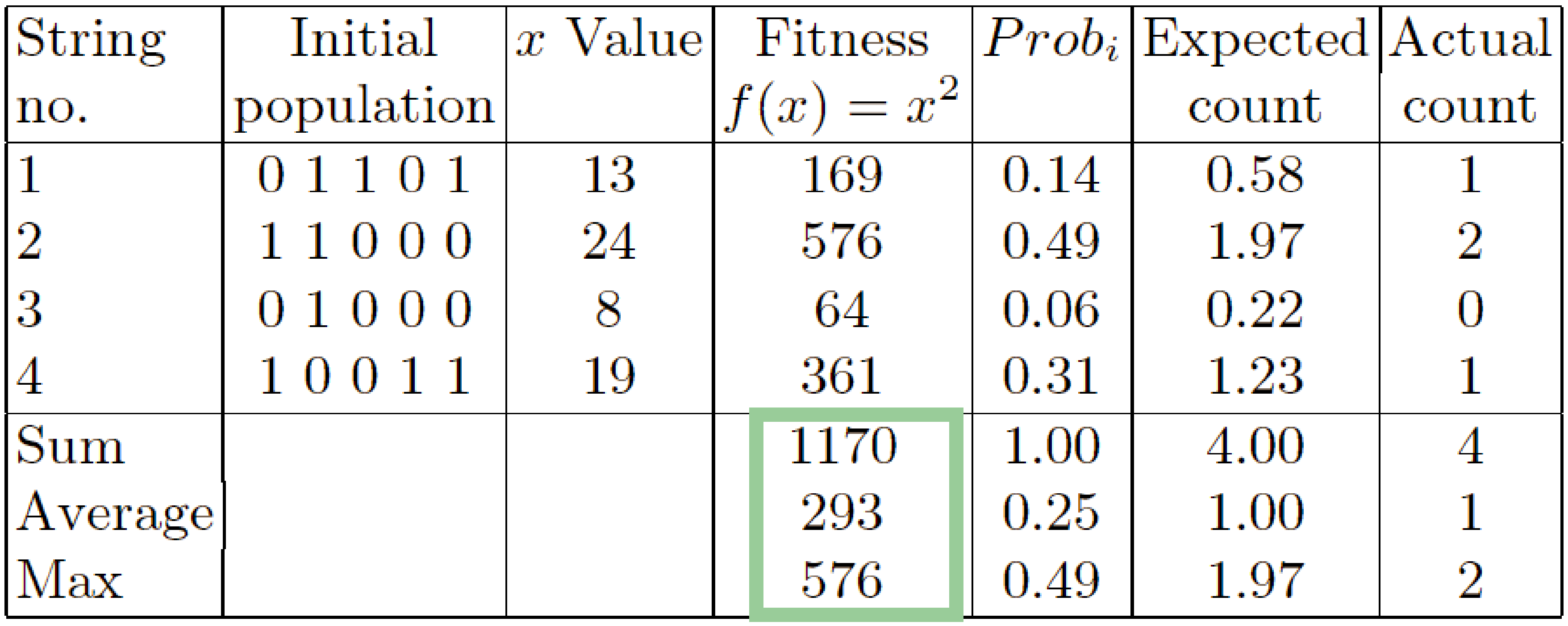
fitness(B) = 1

fitness(C) = 2

An example after Goldberg ‘89 (1)

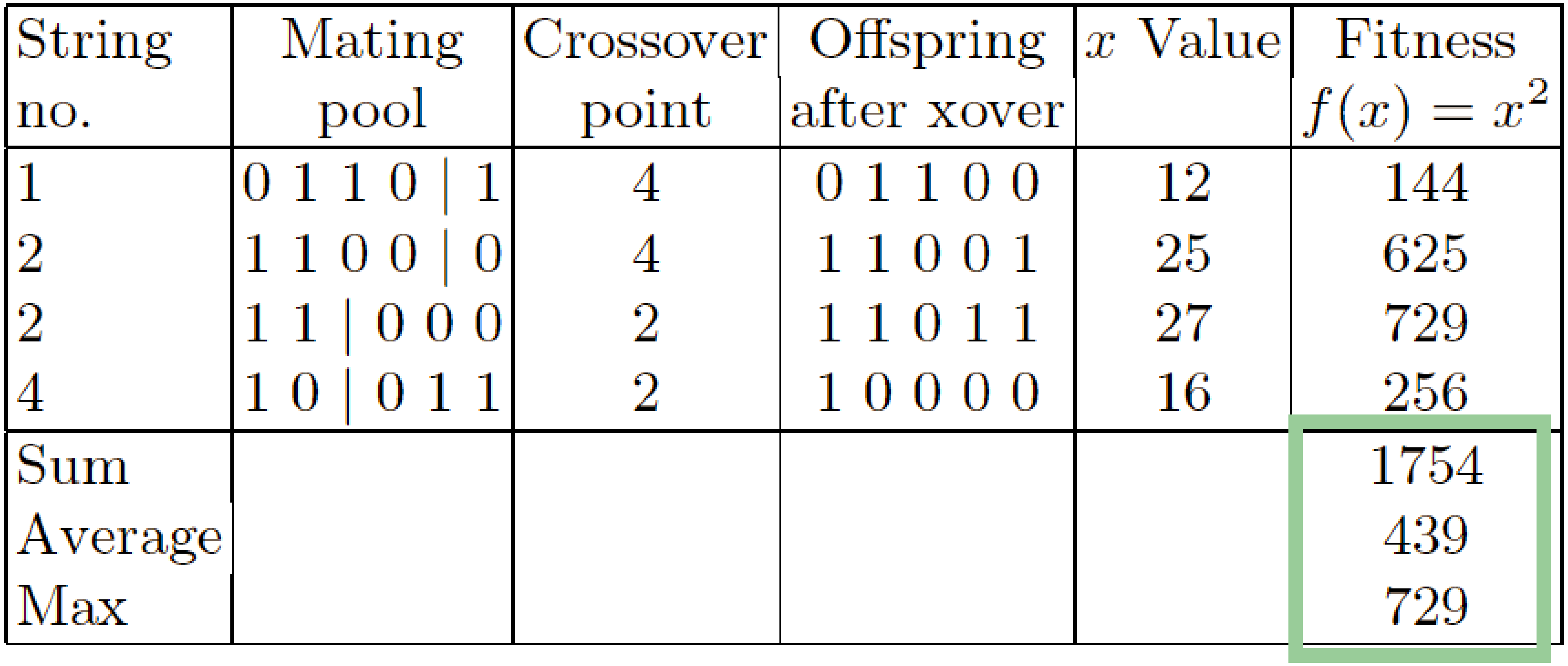
- Simple problem: $\max\{x^2\}$ over $\{0,1,\ldots,31\}$
- GA approach:
  - Representation: binary code, e.g. 01101 $\rightarrow$ 13
  - Population size: 4
  - 1-point Xover, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- We show one generational cycle done by hand

x^2 example: selection



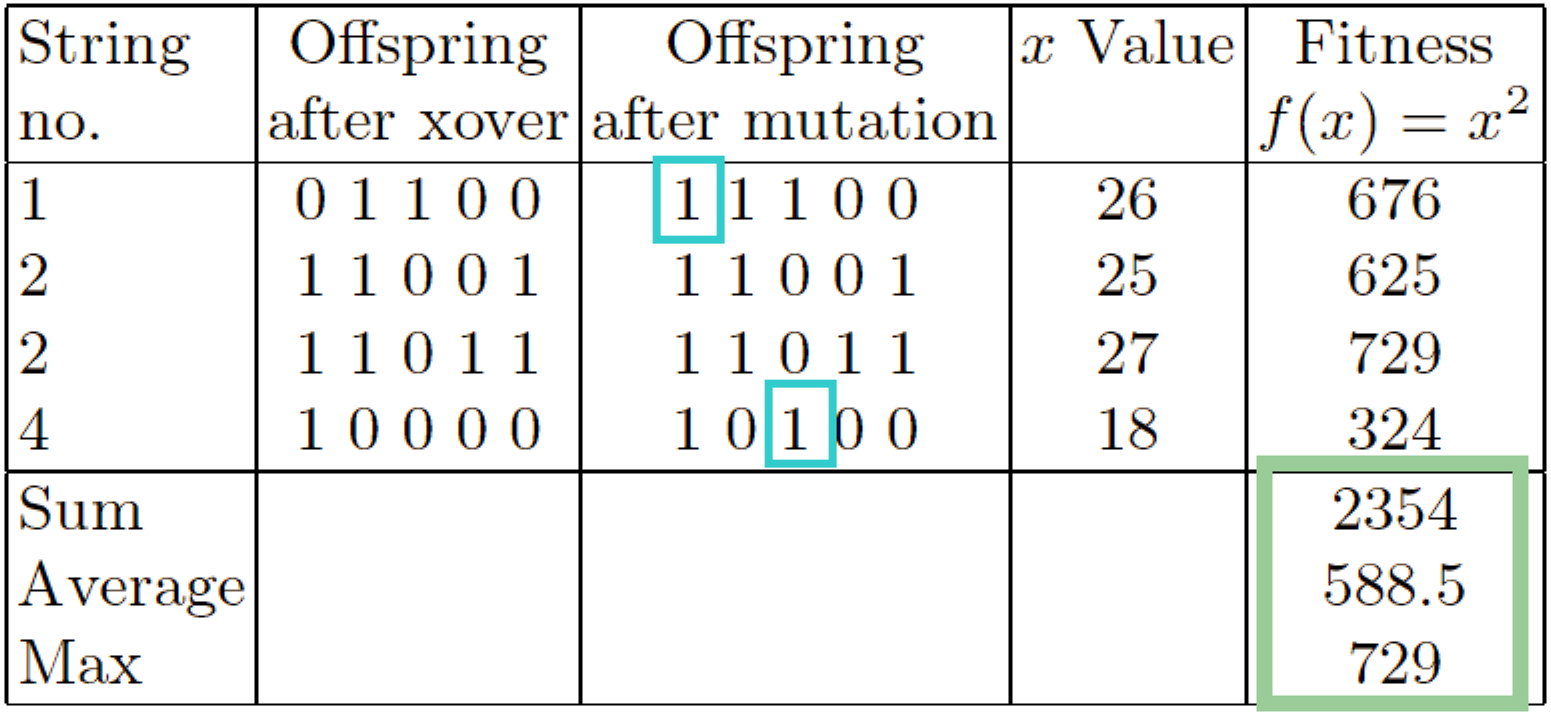
String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

X^2 example: 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 example: mutation



String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	$\boxed{1}$ 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 $\boxed{1}$ 0 0	18	324
Sum				2354
Average				588.5
Max				729

The simple genetic algorithm (SGA)

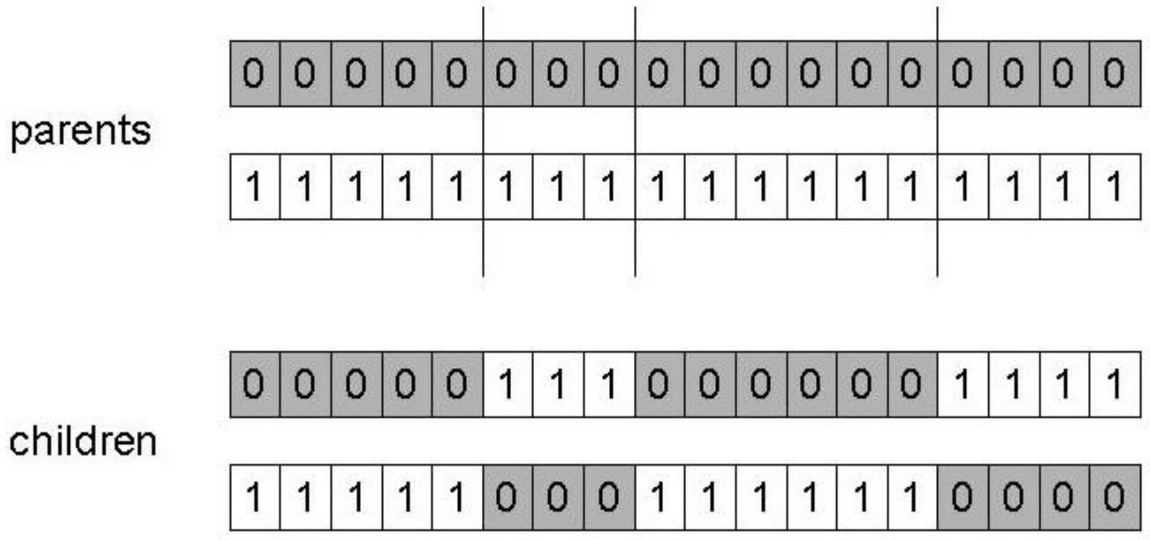
- Has been subject of many (early) studies
  - still often used as benchmark (benchmark method/algorithm) for novel GAs
- Shows many shortcomings, e.g.
  - Representation is too restrictive (un-natural for real numbers and integers)
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations with close fitness values
  - Lose diversity if there is a excessively large fitness

Alternative Crossover Operators

- Performance with 1-Point Crossover depends on the order that variables occur in the representation
  - more likely to keep together genes that are near each other
  - Can never keep together genes from opposite ends of string (for combinatorial problems)
  - This is known as *Positional Bias*
  - Can be exploited if we know about the structure of our problem, but this is not usually the case

N-point crossover

- Choose *N* random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1 point (still some positional bias)



Uniform crossover

- Treat each gene independently by making a random choice as to which parent it should be inherited
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position

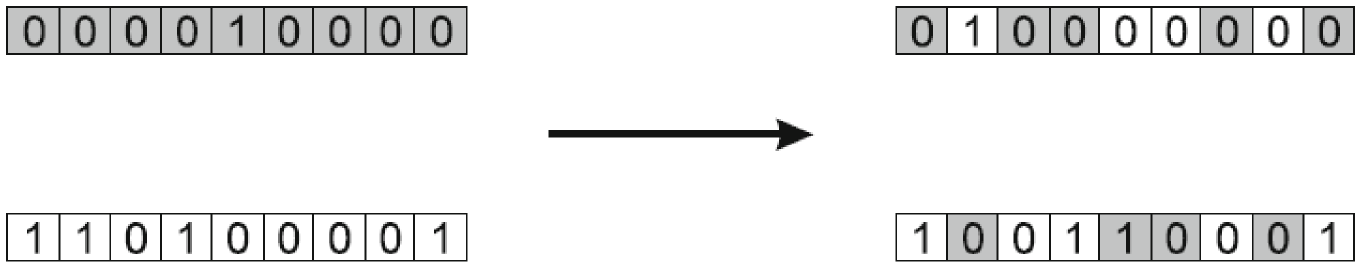


Fig. 4.3. Uniform crossover. The array [0.3, 0.6, 0.1, 0.4, 0.8, 0.7, 0.3, 0.5, 0.3] of random numbers and $p = 0.5$ were used to decide inheritance for this example.

➔The general nature of the algorithms and the **No Free Lunch (NFL)** theorem make it impossible to state that one or the other of these operators performs best on **any** given problem!

➔The understanding of the operations and types of bias exhibited by different recombination operators can be valuable when designing an algorithm for a particular problem.

Crossover OR mutation?

- Decade long debate: which one is better / necessary / main-background ?

- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have another role
  - **mutation-only-EA is possible, xover-only-EA would not work**

Crossover OR mutation? (cont'd)

Viewpoint 1:

Exploration : The generation of new individuals in as yet untested regions of the search space (探索)

Exploitation: The concentration of the search in the vicinity of known good solution (鑽探)

➔There is co-operation AND competition between them

Viewpoint 2:

Exploration and exploitation are the two cornerstones (基石) of problem solving by search. The common opinion about evolutionary algorithms is that they explore the search space by the (genetic) search operators, while exploitation is done by selection.

➔ This opinion is, however, questionable???

Crossover OR mutation? (cont'd)

Viewpoint 3:

Exploration : Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

Viewpoint 4:

Most crossover operators generate individuals in the exploitation zones. In this way, the crossover operator implements a depth search or exploitation, leaving the breadth search or exploration for the mutation operator.

Viewpoint 5:

- Crossover is explorative, it makes a *big* jump to an area somewhere “**in between**” two (parent) areas
- Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of) the parent

  Ex: 0000➔ 1000, exploitable???

Viewpoint 6:

The Exploitation vs Exploration Dilemma

Local search:

- Exploitation of fit individuals.

fitter individuals should be close to fit individuals.

Global search:

- Exploration of unknown regions of the search space.

find fittest individuals requires to possibly visit the whole space.

Too much exploitation $\Longrightarrow$ hill-climbing, Premature convergence toward some local optimum

Too much exploration $\Longrightarrow$ random walk, NO convergence

Viewpoint 7:

In EAs, the selection process, operators (e.g., crossover and mutation), and population size establish a balance between the exploration and exploitation of the search space [6]. The selection process drives search towards the regions of the best individuals. Hence, exploitation is done by selection. However, Bäck [1] showed that the selection processes can control the level of exploration or exploitation by varying selection pressure. Higher selection pressure pushes the search towards more exploitation and lower selection pressure urges the search towards more exploration. A mutation operator randomly modifies individuals, with a given probability, and thus increases the structural diversity of the population. From this point of view, the mutation operator is more an exploration operator. Such an operator helps to recover the genetic diversity lost during the selection phase and to explore new solutions avoiding premature convergence. Conversely, mutation can also be seen as an exploitation operator, because most of the genetic material is preserved. However, note that in some EAs (e.g., evolution strategies) mutation has a much bigger exploration role than in genetic algorithms. The crossover operator combines two or more parents to generate better offspring. Such a combination can be derived from the idea that the exchange of information between good individuals will generate even better offspring. From this point of view, the crossover operator is more an exploitation operator. However, a good crossover operator should also generate individuals in the exploration zone. Directing the evolutionary process towards exploration or exploitation is also possible by population resizing [9]. With bigger population size, the search space is more explored than with smaller population size. Therefore, good balance between exploration and exploitation in EAs is achieved by selection, good mutation and crossover operators and by determining parameters (e.g., *pm*, *pc*, tournament size, population size), which control mutation, crossover, and selection, respectively.

Liu, "Entropy-Driven Parameter Control for Evolutionary Algorithms," *Informatica* 31 (2007) 41–50.

Crossover OR mutation? (cont'd)

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- To hit the optimum you often need a '*lucky*' mutation
- Mutation becomes more important relative to crossover as the population loses diversity.

Other representations

- Gray coding of integers (still binary chromosomes)
- Integers
- Floating-point (real-valued) variables
  - ➔Nowadays it is generally accepted that it is better to encode numerical variables directly as real-valued numbers
- Permutation

Gray coding

- Problem: Coding number in binary have different significance
- Gray coding is a mapping that means that small changes in the genotype cause small changes in the phenotype (unlike binary coding).
- “Smoother” genotype-phenotype mapping makes life easier for the GA
- Hamming distance = 1
- Chance of changing a 7 (0111, Grey code 0100) into 8 (1000, Grey code 1100) equal to that changing it to a 6 (0110, Grey code 0101).

Gray coding

Integer	0	1	2	3	4	5	6	7
Standard	000	001	010	011	100	101	110	111
Gray	000	001	011	010	110	111	101	100

Integer representations

- Some problems naturally have integer variables, e.g. image processing parameters
- Others take *categorical* values from a fixed set e.g. {blue, green, yellow, pink}; {East, West, South, North}
- N-point / uniform crossover operators work

Mutation operators for Integer representations

- By extending bit-flipping mutation
- Work with mutation probability p_m
- Random resetting: a new value is chosen at random from the set of permissible values
- Creep mutation: add a small value to each gene with probability

Recombination for Integer representations

- Make no sense using "blending" allele values (it creates non-integer numbers)
- Use the same set of operators as for binary representation (1-point, N-point, uniform crossover)

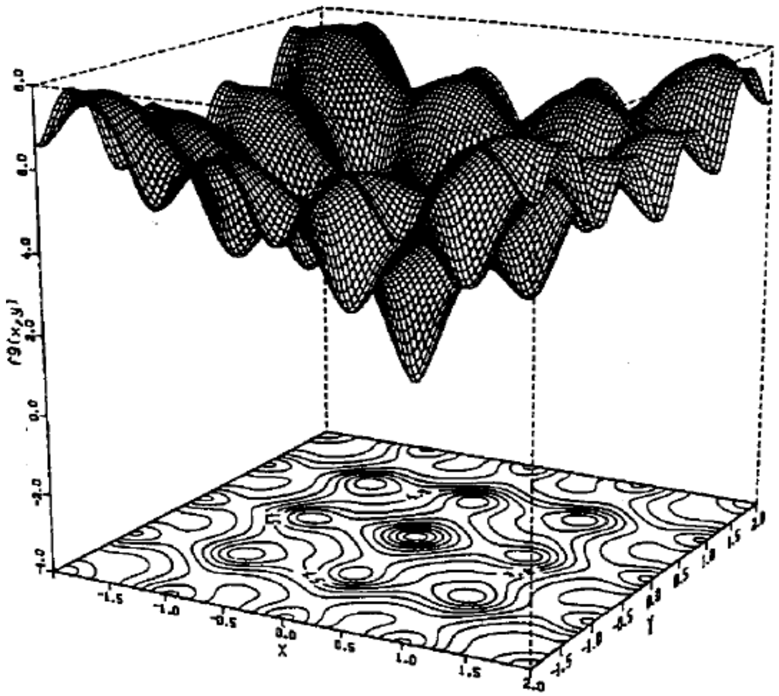
Real-valued problems

- Genes come from a continuous rather than discrete distribution
- Many problems occur as real-valued problems, e.g. continuous parameter optimization *f* : $\Re^n \rightarrow \Re$
- Illustration: Ackley's function (often used in EC)

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

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

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



Mapping real values on bit strings

- $z \in [x,y] \subseteq \Re$ represented by $\{a_1,\ldots,a_L\} \in \{0,1\}^L$

- $[x,y] \rightarrow \{0,1\}^L$ must be invertible (one phenotype per genotype)
- $\Gamma: \{0,1\}^L \rightarrow [x,y]$ defines the representation

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

$$x_i = a_i + decimal(1001\ldots001_2) \cdot \frac{b_i - a_i}{2^{m_i}-1}$$

- Only 2^L values out of infinite are represented
- L determines possible maximum precision of solution
- High precision → long chromosomes (slow evolution)

Floating point mutations 1

General scheme of floating point mutations

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

- Uniform mutation:

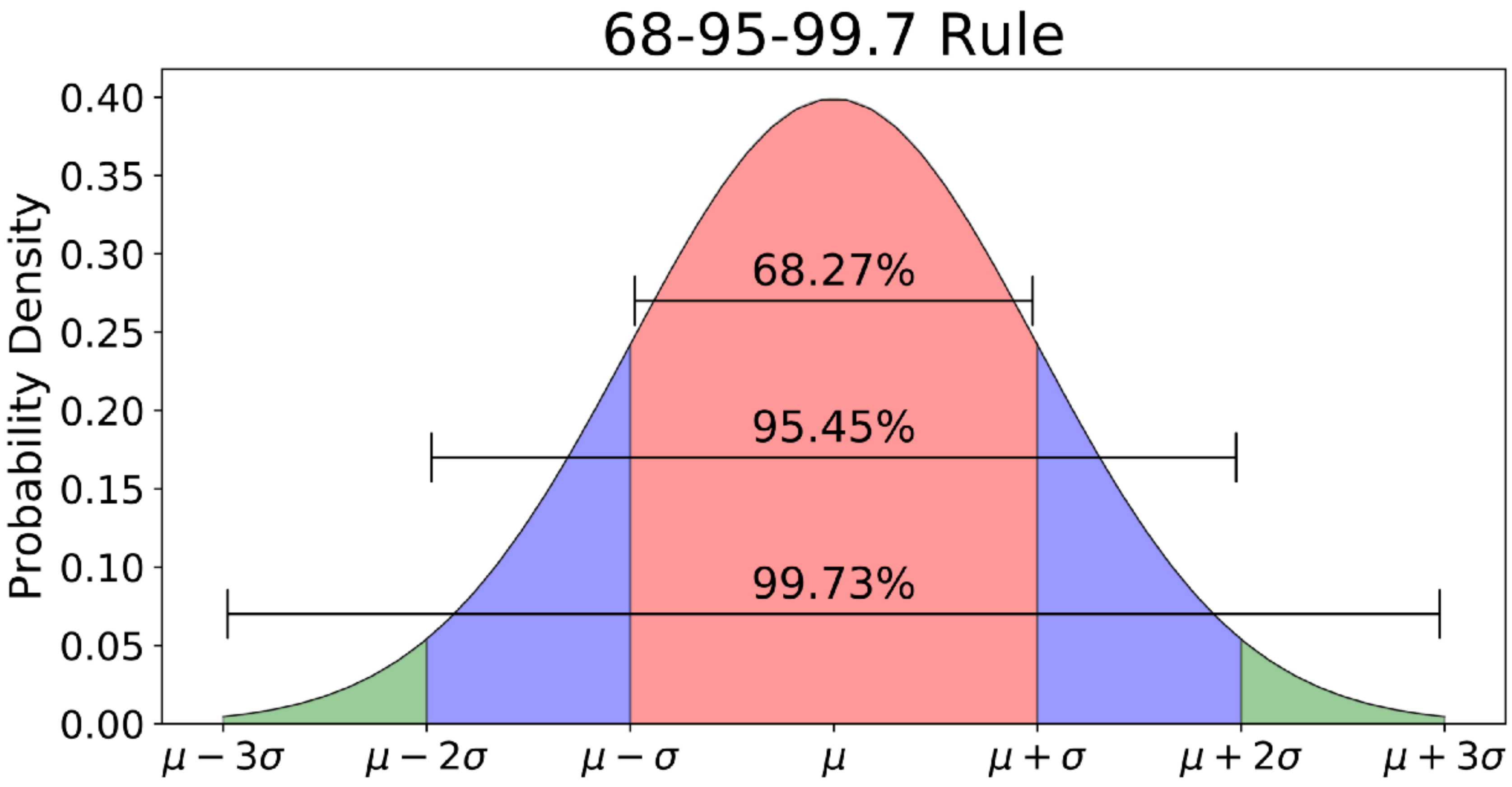
  x_i' drawn randomly (uniform) from $[LB_i, UB_i]$

- Analogous to bit-flipping (binary) or random resetting (integers)

Floating point mutations 2

- Non-uniform mutations:
  - Many methods proposed, such as time-varying range of change etc.
  - Most schemes are probabilistic but usually only make a small change to value
  - Most common method is to add random deviation to each variable separately, taken from $N(0, \sigma)$ Gaussian distribution and then curtail to range

    $x'_i = x_i + N(0,\sigma)$
  - Standard deviation σ, mutation step size, controls amount of change (2/3 of deviations will lie in range ($- \sigma$ to $+ \sigma$)


68-95-99.7 Rule
Probability Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
68.27%
95.45%
99.73%
$\mu - 3\sigma$
$\mu - 2\sigma$
$\mu - \sigma$
μ
$\mu + \sigma$
$\mu + 2\sigma$
$\mu + 3\sigma$

Non-uniform mutation

$$s_v^t = < v_1, v_2, \cdots, \boxed{v_k}, \cdots, v_m >$$

$$s_v^{t+1} = < v_1, v_2, \cdots, \boxed{v_k^{'},} \cdots, v_m >$$

$$v_k^{'} = \begin{cases} v_k + \Delta(t, UB - v_k), & \text{if a random digit is } 0 \\ v_k - \Delta(t, v_k - LB) & \text{if a random digit is } 1 \end{cases}$$

$$\Delta(t, y) = y(1 - r^{(1-\frac{t}{T})^b})$$

r is a random number from [0..1]

T is the max generation number

b is a system parameter determining the degree of dependency on iteration no. (commonly *b*=5)

$$\Delta(t, y) = y(1 - r^{(1-\frac{t}{T})^b})$$

➔ returns a value in [0, *y*-*r*],

Ex: $r = 0.1,\ b = 2$

If $t/T = 0.001$, $\Delta = y * 0.9$ (very initial stage)

If $t/T = 0.01$, $\Delta = y * 0.43$ (early stage),

If $t/T = 0.5$, $\Delta = y * 0.16$ (mid stage)

If $t/T = 0.9$, $\Delta = y * 0.07$ (final stage) ,

- Fine tuning capability of the system
- $\Delta(t, y)$ returns a value in [0, *y*], ← $\Delta(t, y) = y(1 - r^{(1-\frac{t}{T})^b})$
- $\Delta(t, y)$ probability close to 0 as *t* increases
- Search the space uniformly initially (when *t* is small), and very locally at later stage

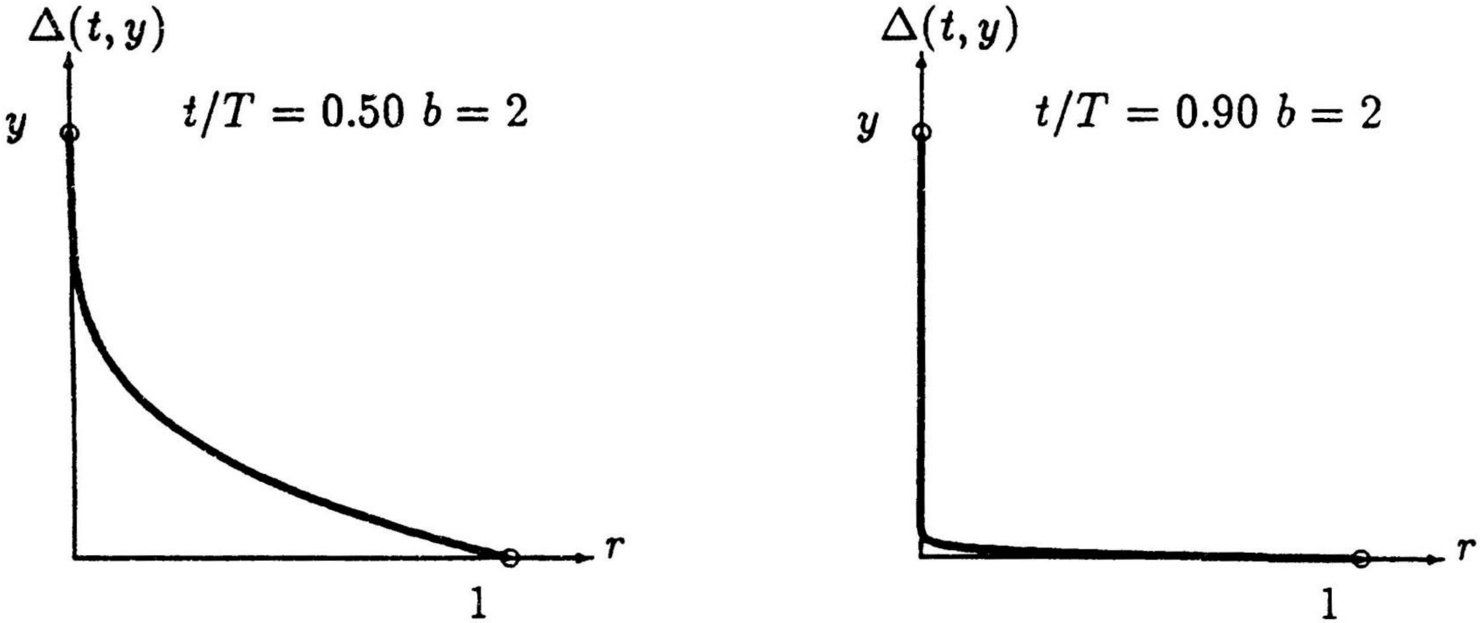


Fig. 6.1. $\Delta(t, y)$ for two selected times

Crossover operators for real-valued GAs

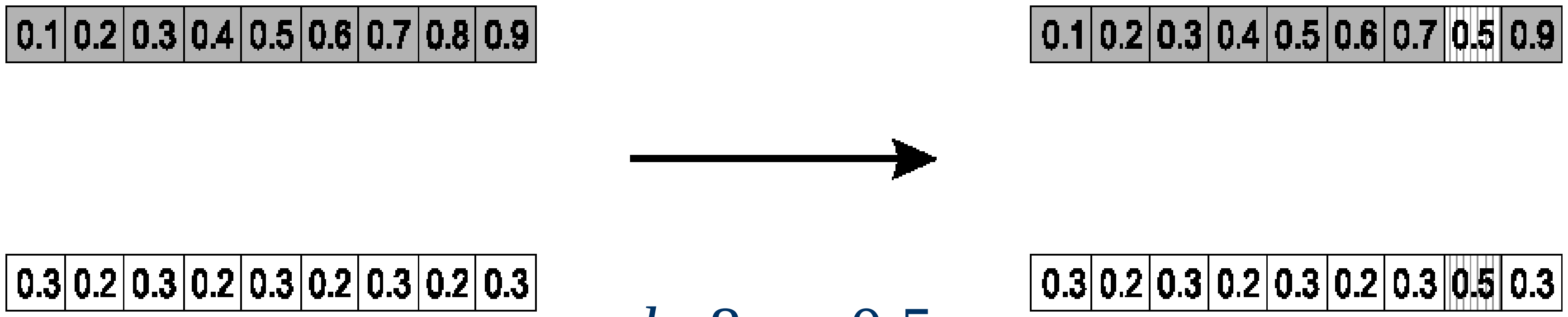
- Discrete recombination :
  - each allele value in offspring *z* comes from one of its parents (*x*,*y*) with equal probability: $z_i = x_i$ or y_i
  - Could use *N*-point or uniform
  - Give only new combinations of existing float numbers
  - Disadvantage: Only mutation can insert new values into the population
- Intermediate recombination (a.k.a. *arithmetic* recombination)
  - Create new allele value in the offspring
  - exploits idea of creating children “between” parents, i.e. blending (混合)
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ where $\alpha : 0 \leq \alpha \leq 1$.
  - The parameter α can be:
    - constant: uniform arithmetical crossover (often α =0.5)
    - variable (e.g. depend on the age of the population)
    - picked at random every time

Single arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n\rangle$
- Pick a single gene (k) at random, and take average of the parents
- child$_1$ is:

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

- child$_2$: the same way with x and y reversed



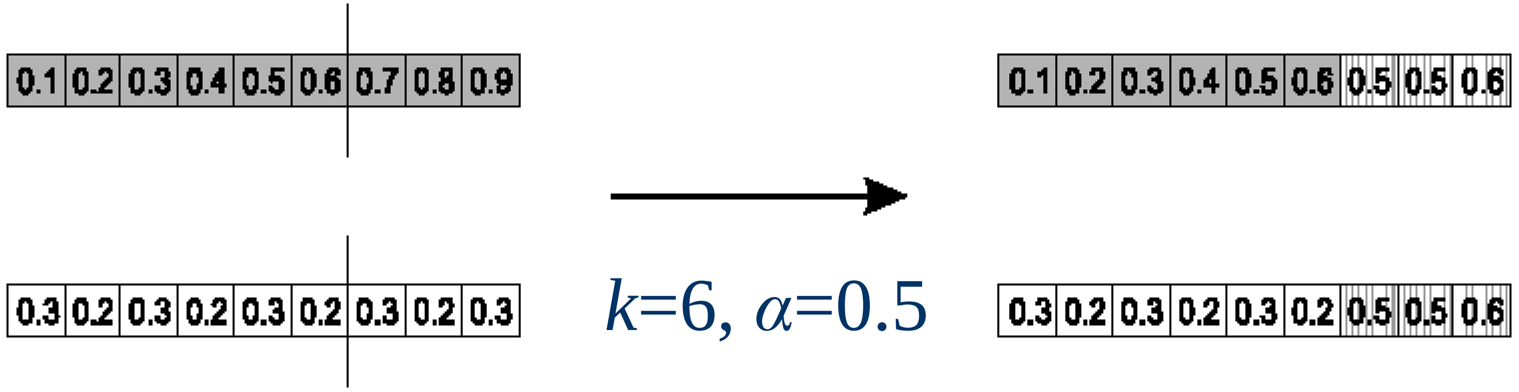
k=8, α=0.5

Simple arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Pick random gene (k) after this point mix values
- child$_1$ is:

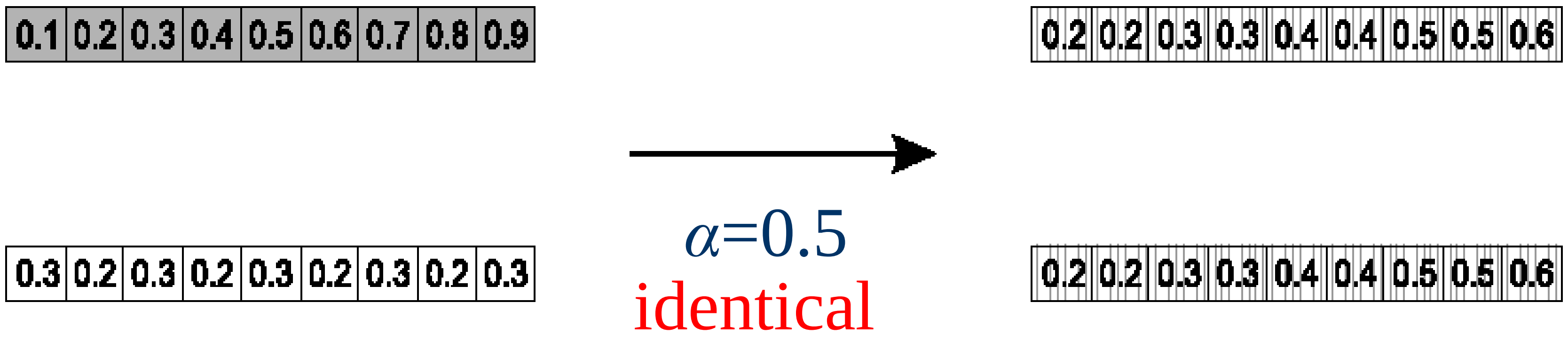
$$\left\langle x_1, \ldots, x_k, \boxed{\alpha \cdot y_{k+1} + (1-\alpha)\cdot x_{k+1}, \ldots, \alpha \cdot y_n + (1-\alpha)\cdot x_n} \right\rangle$$

- child2 : the same way with x and y reversed. e.g. with $\alpha = 0.5$



Whole arithmetic crossover

- **Most commonly used**
- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- child$_1$ is: $a \cdot \bar{x} + (1-a) \cdot \bar{y}$
- child$_2$ is: $a \cdot \bar{y} + (1-a) \cdot \bar{x}$
- Take the weighted sum of the two parental alleles **for each gene**



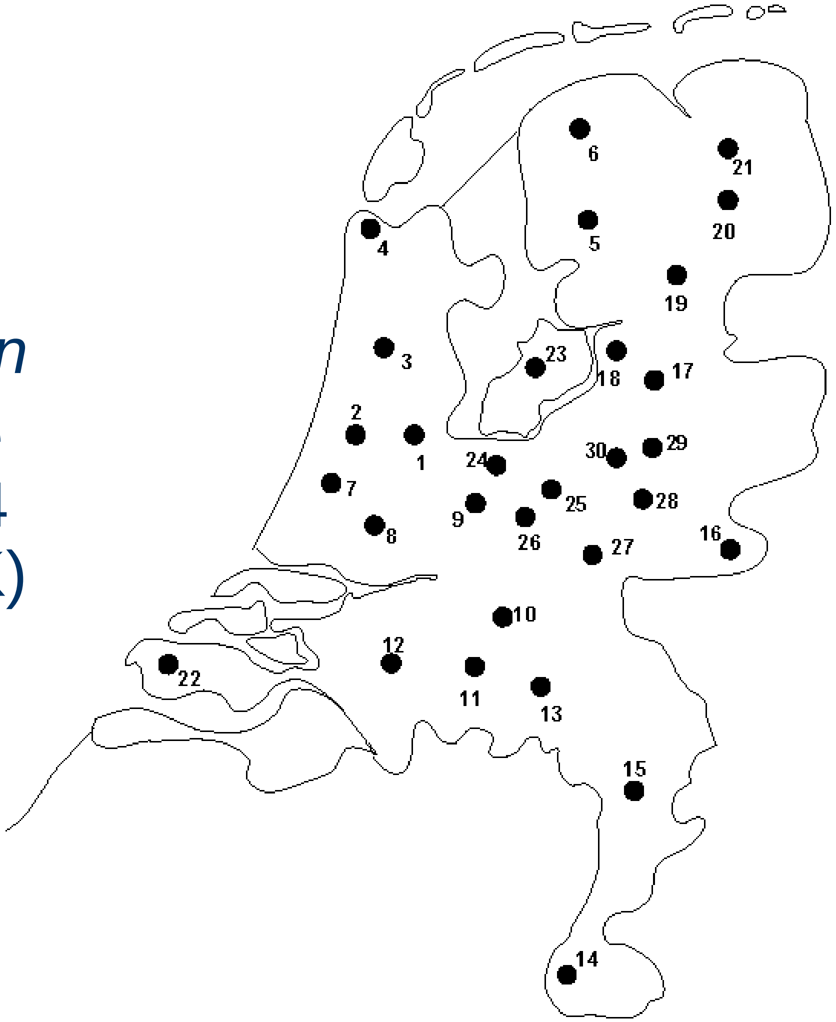
Permutation Representations

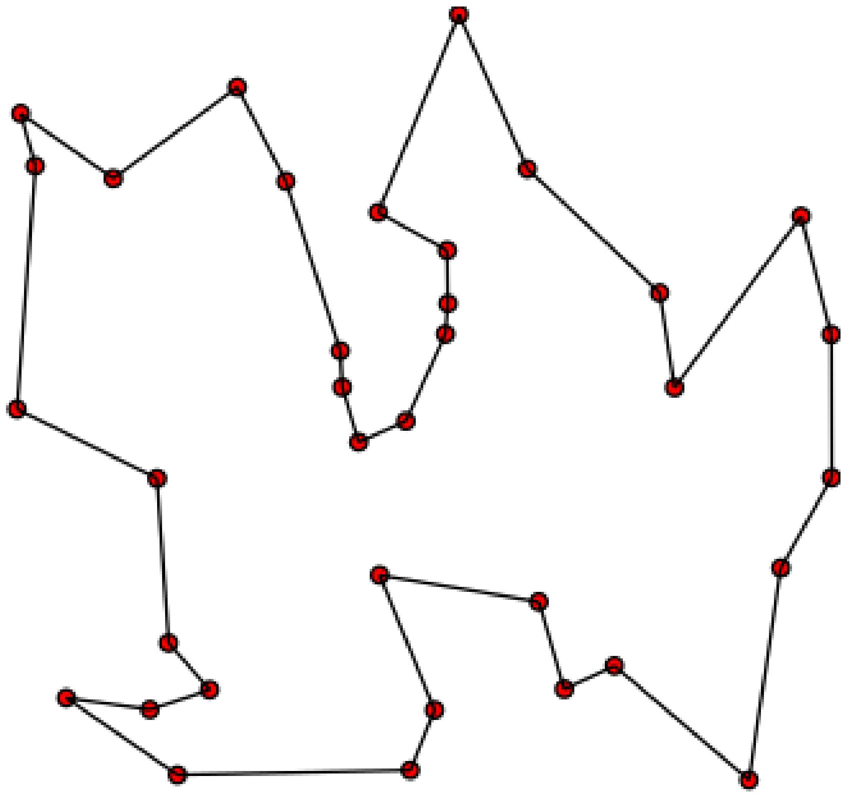
- Decide on the order in which a sequence of events should occur
- Ordering / sequencing problems form a special type
- Task is arranging some objects in a certain order
  - Example: job shop scheduling: important thing is which elements occur before others (order)
  - Example: Travelling Salesman Problem (TSP) : important thing is which elements occur next to each other (adjacency)
- These problems are generally expressed as a permutation:
  - if there are *n* variables then the representation is as a list of *n* integers, each of which occurs exactly once
- Integer representation is NOT valid.

Permutation representation: TSP example

- Problem:
  - Given *n* cities
  - Find a complete tour with **minimal length**
- Encoding:
  - Label the cities 1, 2, … , *n*
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:

  for 31 cities there are **30!** ≈ **10^{32}** possible tours



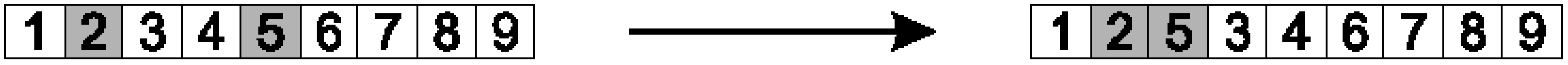


Mutation operators for permutations

- Normal mutation operators lead to inadmissible (不允許) solutions
  - e.g. bit-wise mutation : let gene *i* have value *j*
  - changing to some other value *k* would mean that *k* occurred twice and *j* no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

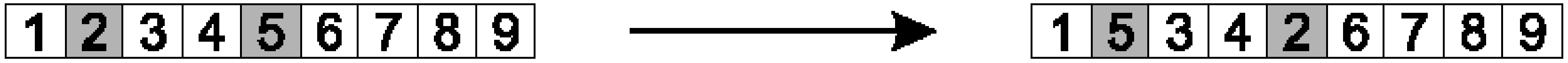
Insert Mutation for permutations

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information



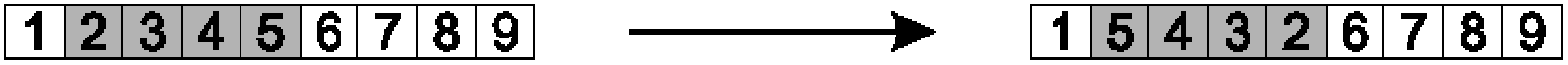
Swap mutation for permutations

- Pick two alleles at random and swap their positions
- Preserves most of adjacency information, disrupts order more



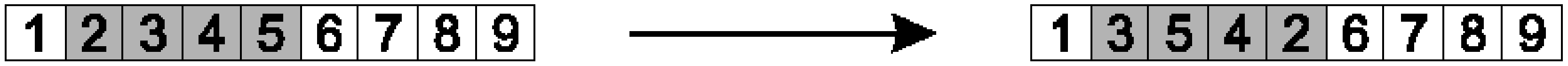
Inversion mutation for permutations

- Pick two alleles at random and then invert the substring between them.
- Preserves most adjacency information but disruptive of order information



Scramble mutation for permutations

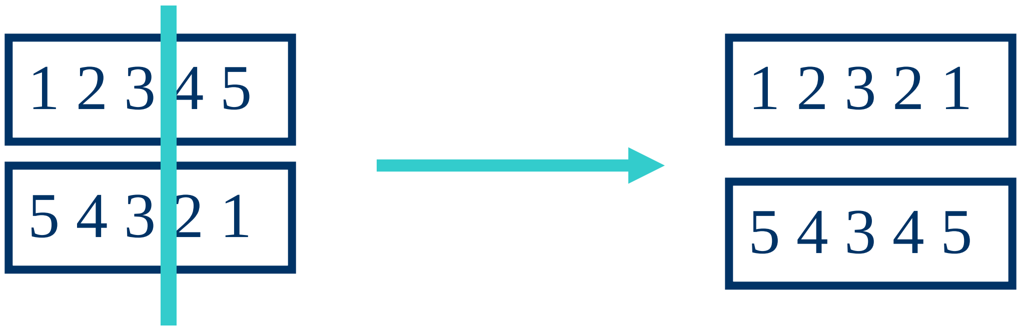
- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions



Note: subset does not have to be contiguous (連續)

Crossover operators for permutations

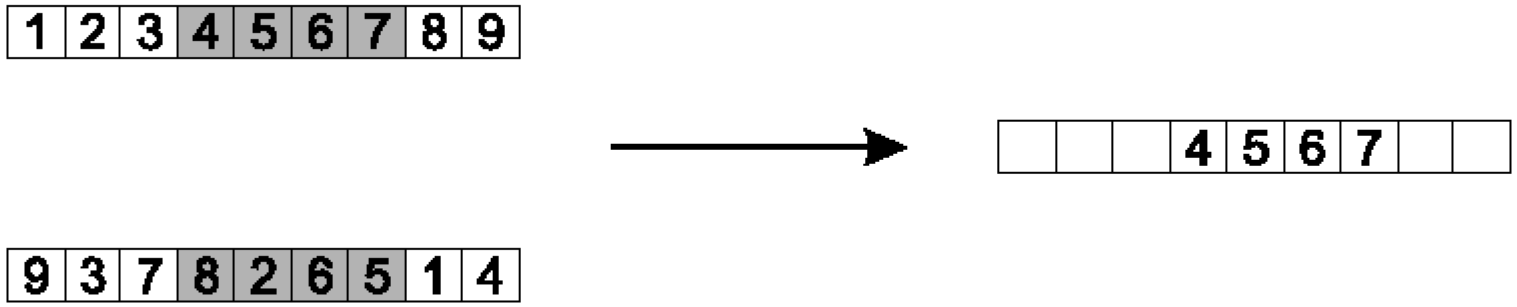
- "Normal" crossover operators will often lead to inadmissible solutions



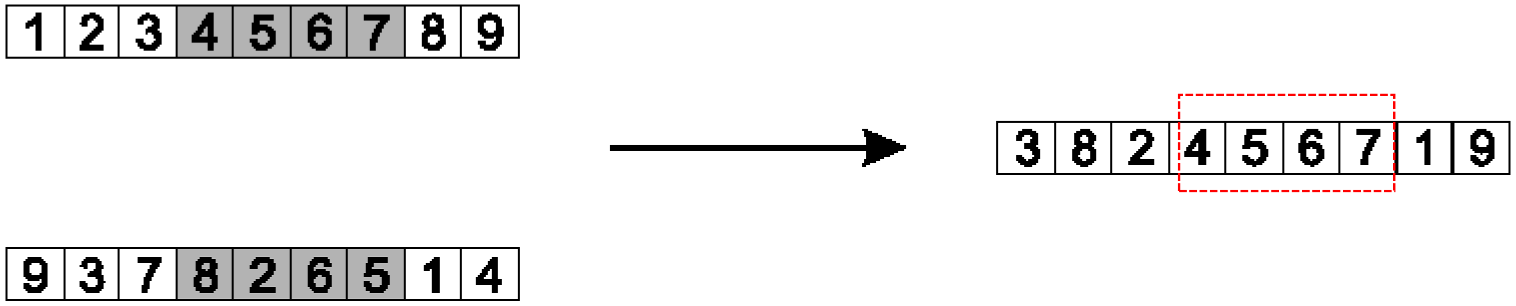
- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Order crossover example

- Copy randomly selected set from first parent



- Copy rest from second parent in order 1,9,3,8,2



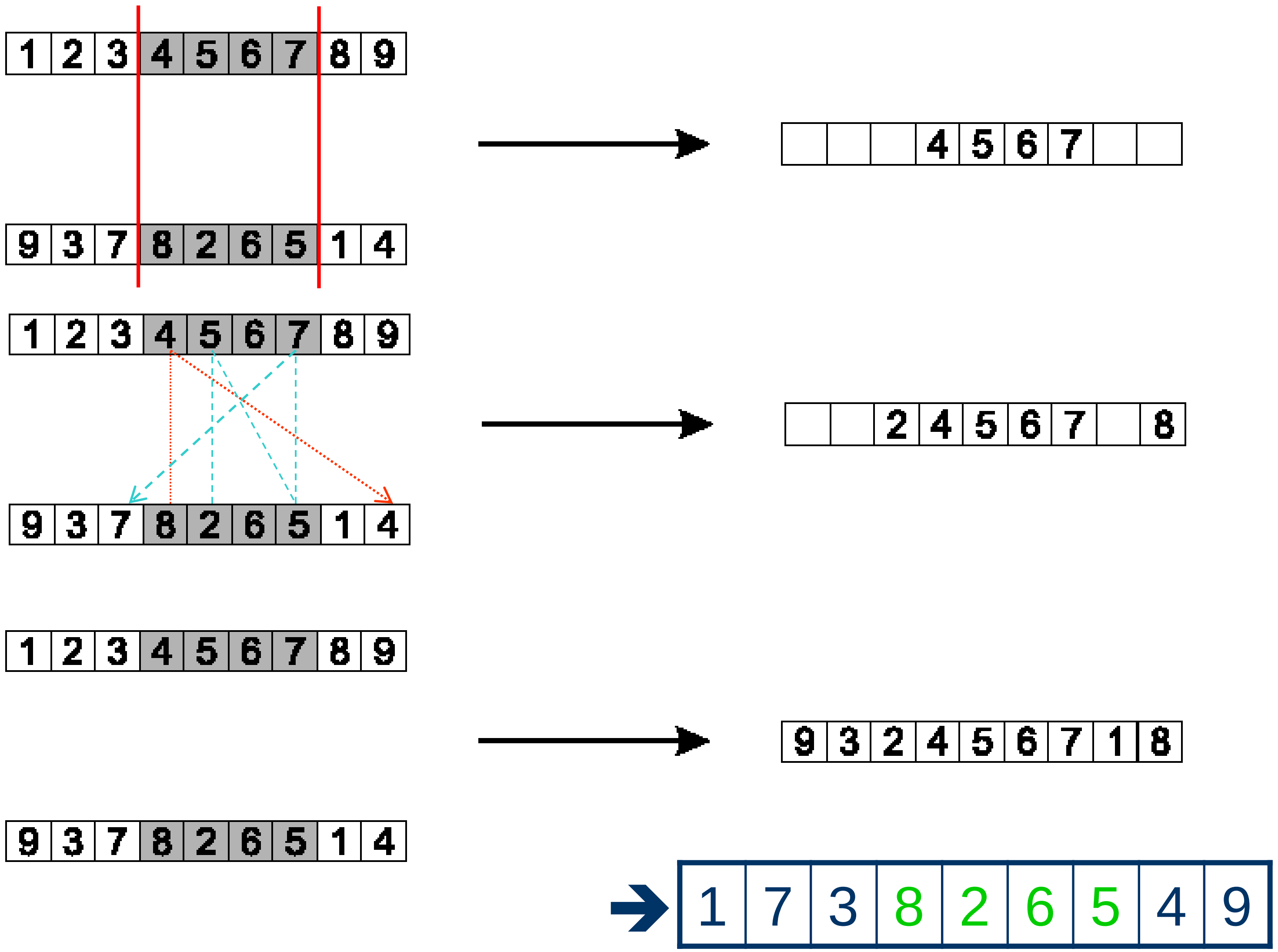
The second one: 347 8265 91

Order crossover

- Idea is to preserve relative order that elements occur
- Informal procedure:
  1. Choose an arbitrary part from the first parent
  2. Copy this part to the first child
  3. Copy the numbers that are not in the first part, to the first child:
     - starting right from cut point of the copied part,
     - using the **order** of the second parent
     - and wrapping around at the end
  4. Analogous for the second child, with parent roles reversed

Partially Mapped Crossover (PMX) example

- Step 1
- Step 2
- Step 3



Partially Mapped Crossover (PMX)

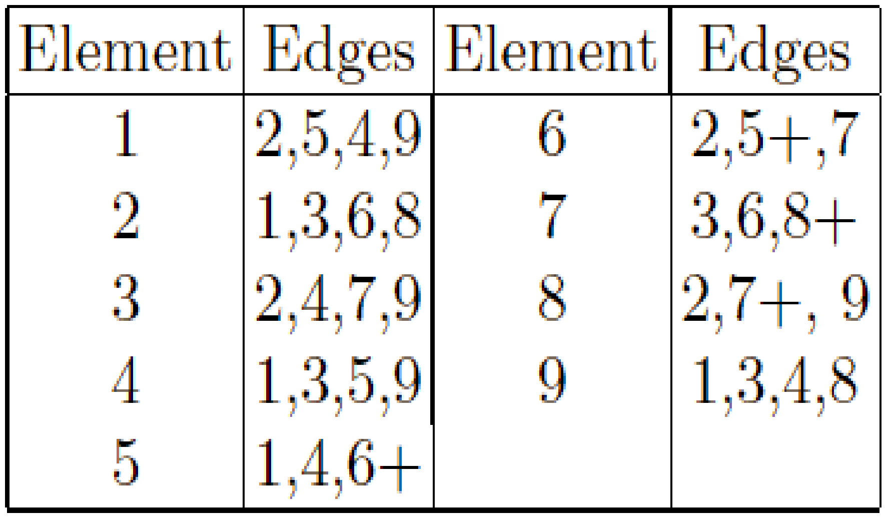
Informal procedure for parents P1 and P2:

1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1
4. Place *i* into the position occupied *j* in P2, since we know that we will not be putting *j* there (as is already in offspring)
5. If the place occupied by *j* in P2 has already been filled in the offspring *k*, put *i* in the position occupied by *k* in P2
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

Second child is created analogously

Edge Recombination

- Offspring should be created as far as possible using only edges that are present in one or more parents.
- Works by constructing an adjacency table listing which edges are present in the two parents, if an edge is common to both, mark with a +
- e.g. [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]



Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

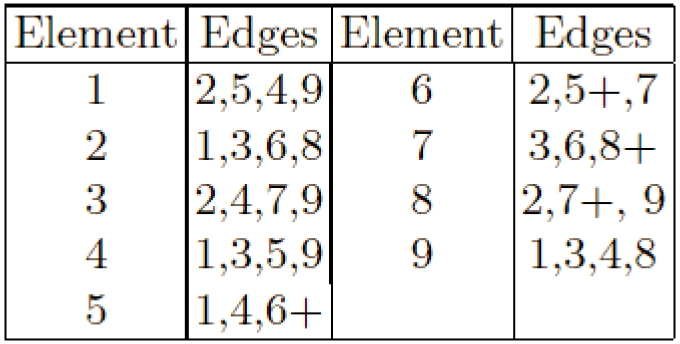
Edge Recombination 2

Informal procedure once edge table is constructed

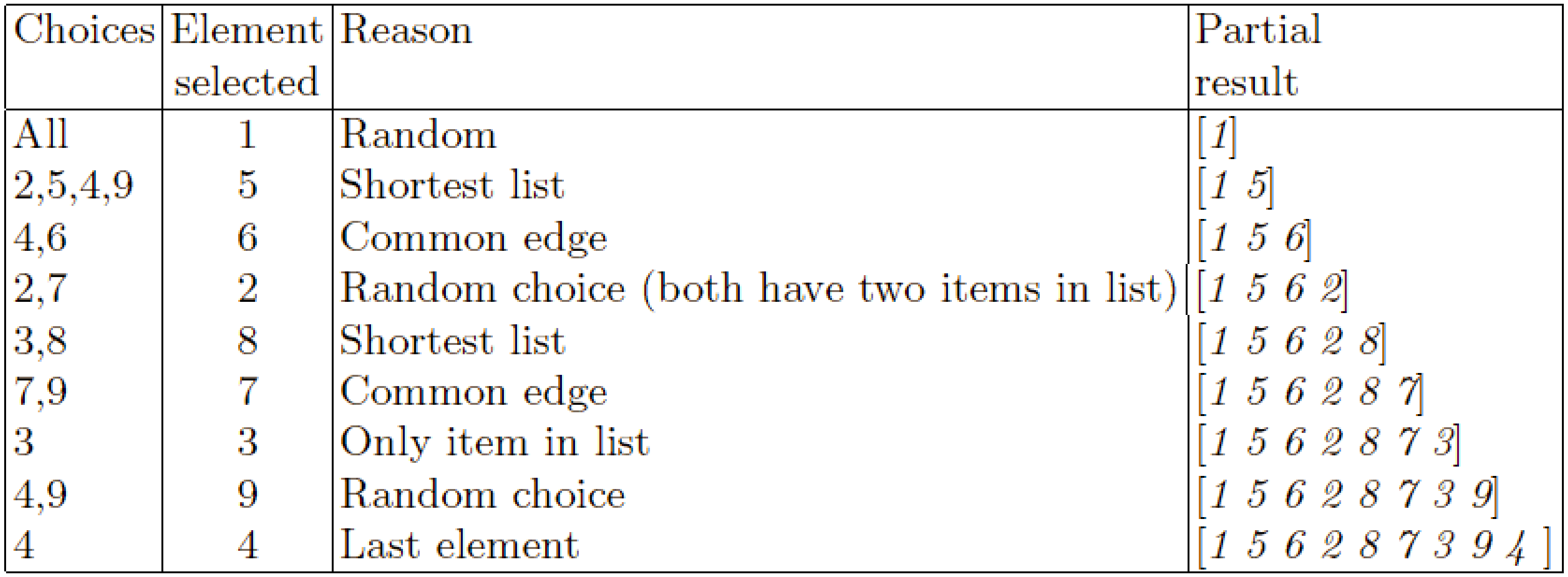
1. Pick an initial element at random and put it in the offspring
2. Set the variable current element = entry
3. Remove all references to current element from the table
4. Examine list for current element:
   - If there is a common edge, pick that to be next element
   - Otherwise pick the entry in the list which itself has the shortest list
   - Ties are split at random
5. In the case of reaching an empty list:
   - Examine the other end of the offspring is for extension
   - Otherwise a new element is chosen at random

Edge Recombination example

e.g. [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]



Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		



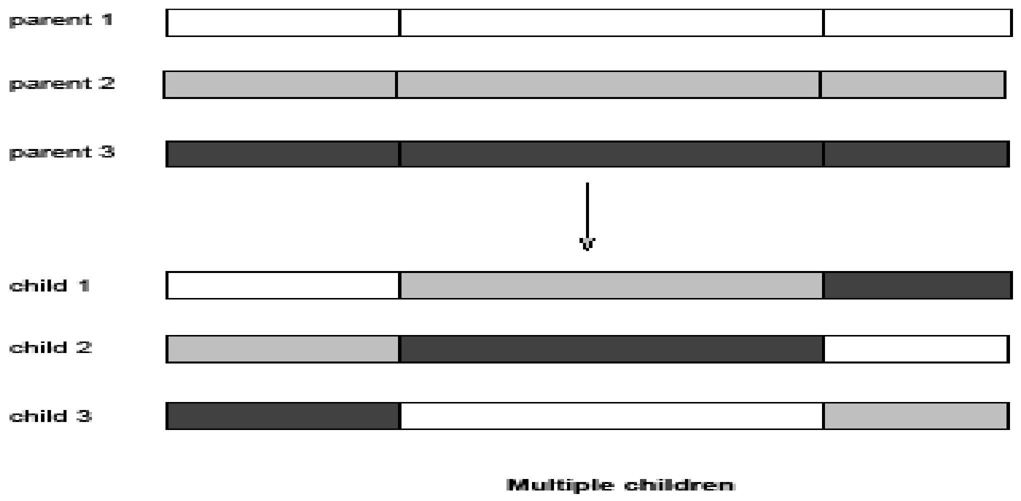
Choices	Element selected	Reason	Partial result
All	1	Random	*[1]*
2,5,4,9	5	Shortest list	*[1 5]*
4,6	6	Common edge	*[1 5 6]*
2,7	2	Random choice (both have two items in list)	*[1 5 6 2]*
3,8	8	Shortest list	*[1 5 6 2 8]*
7,9	7	Common edge	*[1 5 6 2 8 7]*
3	3	Only item in list	*[1 5 6 2 8 7 3]*
4,9	9	Random choice	*[1 5 6 2 8 7 3 9]*
4	4	Last element	*[1 5 6 2 8 7 3 9 4]*

Multiparent recombination

- Recall that we are not constricted by the practicalities of nature
- Noting that mutation uses 1 parent, and "traditional" crossover uses 2, the extension to *a*>2 is natural to examine
- Been around since 1960s, still rare but studies indicate useful
- Do not exist in biology!
- Three main types:
  - Based on allele frequencies, e.g., p-sexual voting generalising uniform crossover
  - Based on segmentation and recombination of the parents, e.g., diagonal crossover generalising N-point crossover
  - Based on numerical operations on real-valued alleles, e.g., center of mass crossover, generalising arithmetic recombination operators

Real-Valued or Floating-Point Representation: Multi-parent recombination, type 1

- Idea: segment and recombine parents
- Example: diagonal crossover for *n* parents:
  - Choose *n-1* crossover points (same in each parent)
  - Compose *n* children from the segments of the parents in along a “diagonal”, wrapping around



- This operator generalises 1-point crossover

Simplex crossover

An interesting attempt to combine genetic algorithms with the Simplex Method resulted in the ternary *simplex crossover* (Bersini and Seront 1992) . If x^1, x^2, x^3 are the three parents sorted in decreasing order of fitness, then the simplex crossover generates one child x by the following two rules.

(i) If $x_i^1 = x_i^2$ then $x_i = x_i^1$;
(ii) if $x_i^1 \neq x_i^2$ then $x_i = x_i^3$ with probability p and $x_i = 1 - x_i^3$ with probability $1 - p$.

Using the value $p = 0.8$, the simplex GA performed better than the standard GA on the DeJong functions. The authors remark that applying a modified crossover on more than three parents "is worth to try".

Global recombination

Given a population of μ individuals global recombination

creates one offspring x by the following mechanism.

$$x_i = \begin{cases} x_i^{S_i} \text{ or } x_i^{T_i} & \textit{global discrete recombination} \\ x_i^{S_i} + \chi_i \cdot (x_i^{T_i} - x_i^{S_i}) & \textit{global intermediate recombination} \end{cases}$$

where the two parents x^{S_i}, x^{T_i} $(S_i, T_i \in \{1, \ldots, \mu\})$ are redrawn for each i anew

Performance of muti-parent recombination

- Increasing the arity (no of inputs) of recombination has a positive effect on the performance of EA.
- However, not always positive
- Depends on the type of recombination and problem at hand
- Experimental investigations show that using more than two parents is advantageous in many cases.

➔ Worth trying!

Population Models

- SGA uses a Generational model:
  - Whole population is affected
  - each individual survives for exactly one generation
  - the entire set of parents is replaced by the offspring
- At the other end of the scale are Steady-State models:
  - Partial population is affected
  - one (λ out of μ) offspring is generated per generation,
  - one (λ out of μ) member of population replaced,
- Generation Gap
  - the percentage of the population replaced: λ / μ
  - 1.0 for SGA
  - $1/\mu$ for SSGA (Steady State Genetic Algorithm (SSGA)), where μ is the population size (only one member is replaced!)

Fitness Based Competition

- Selection can occur in two places:
  - Selection from current generation to take part in mating (parent selection)
  - Selection from (parents + offspring) to go into next generation (survivor selection)
- Selection operators work on whole individuals
  - i.e. they are representation-independent
- Distinction between selection
  - operators: define selection probabilities
  - algorithms: define how probabilities are implemented

Implementation example: SGA

- Expected number of copies of an individual *i*

$$E(n_i) = f(\mathrm{i}) / \langle \mathrm{f} \rangle$$

(μ = pop. size, f(i) = fitness of i, ⟨f⟩ avg. fitness in pop.)

- **Roulette wheel algorithm** (RWS) :
  - Given a probability distribution, spin a 1-armed wheel *n* times to make *n* selections
  - No guarantees on actual value of n_i
- Baker's SUS (**Stochastic Universal Sampling**) algorithm:
  - Only one random is chosen for the whole selection process
  - *n* evenly spaced arms on wheel and spin once
  - Guarantees $floor(E(n_i)) \leq n_i \leq ceil(E(n_i))$
  - exhibit no bias by choosing them at evenly spaced intervals

```
BEGIN
  /*  Given the cumulative probability distribution a */
  /*  and assuming we wish to select $\lambda$ members of the mating pool */
  set $current\_member = 1$;
  WHILE ( $current\_member \leq \lambda$ ) DO
    Pick a random value $r$ uniformly from $[0, 1]$;
    set $i = 1$;
    WHILE (  $a_i < r$ ) DO
      set $i = i + 1$;
    OD
    set mating_pool[$current\_member$] = parents[$i$];
    set $current\_member = current\_member + 1$;
  OD
END
```

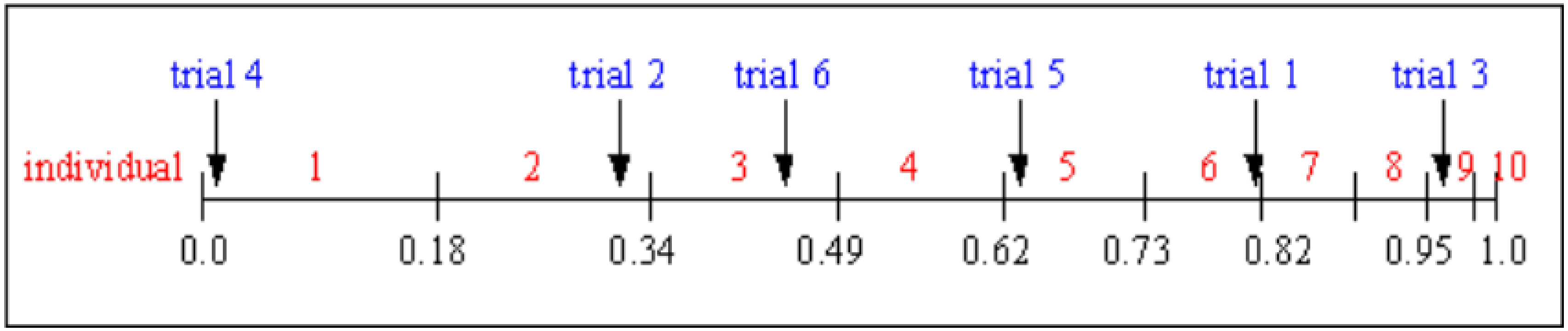
Fig. 5.1. Pseudocode for the roulette wheel algorithm

Number of individual	1	2	3	4	5	6	7	8	9	10	11
fitness value	2.0	1.8	1.6	1.4	1.2	1.0	0.8	0.6	0.4	0.2	0.0
selection probability	0.18	0.16	0.15	0.13	0.11	0.09	0.07	0.06	0.03	0.02	0.0

➔The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness. A random number is generated and the individual whose segment spans the random number is selected.

sample of 6 random numbers:

```
0.81, 0.32, 0.96, 0.01, 0.65, 0.42.
```



After selection the mating population consists of the individuals:

```
1, 2, 3, 5, 6, 9.
```

Fitness-Proportionate Selection (FPS)

- Problems include
  - Highly fit (outstanding) members can rapidly take over if rest of population is much less fit: ➔Premature Convergence
  - At end of runs when fitnesses are similar, lose selection pressure (almost uniformly random; no selection pressure)
  - Highly susceptible (易受影響) to function transposition

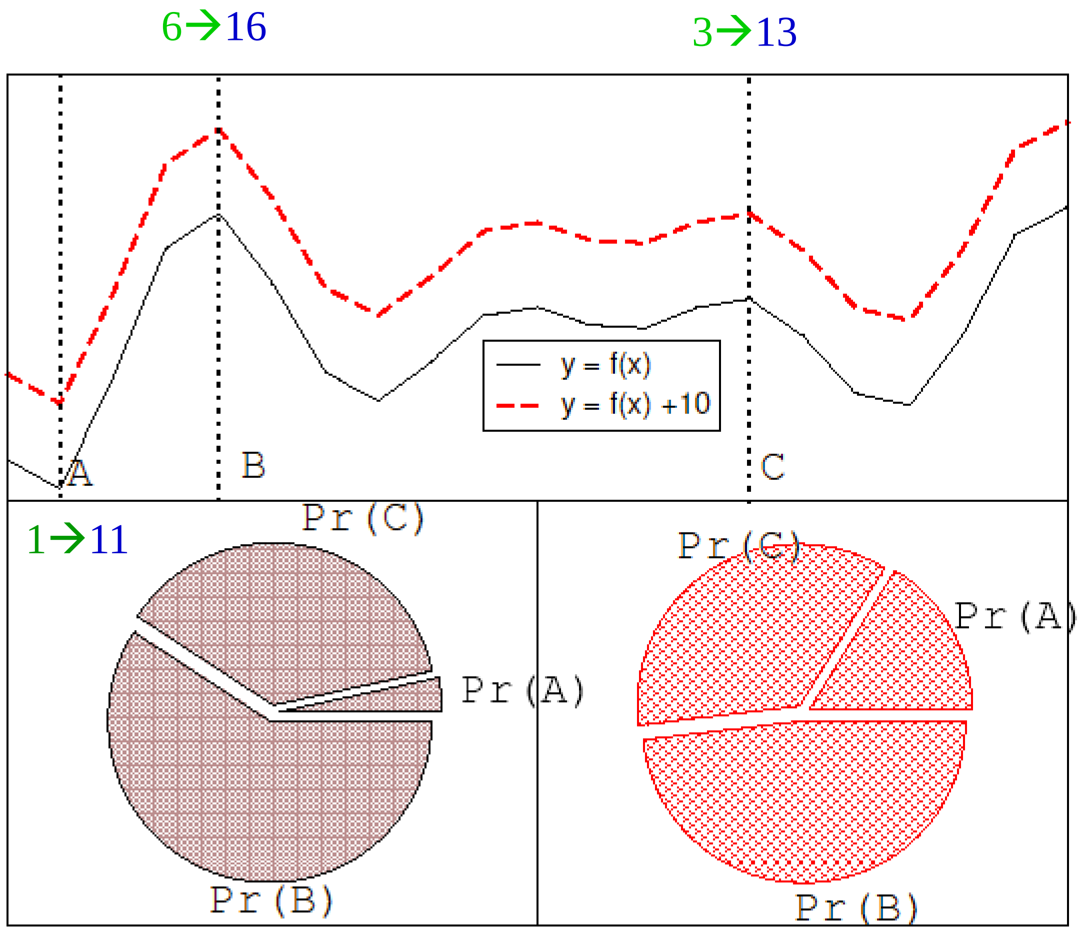
Function transposition for FPS

1. Selective advantage of B is reduced after transposition!

2. What if +100?

3. susceptible!

(易受影響的)



- Scaling to prevent the last two problems

  - Windowing: $f'(i) = f(i) - \beta$

    where β is the worst fitness (value of the least-fit member) in this generation (or, in the last n generations by taking average)

  - Sigma Scaling:

    $f'(i) = \max(f(i) – (\langle f \rangle - c \bullet \sigma_f), 0.0)$

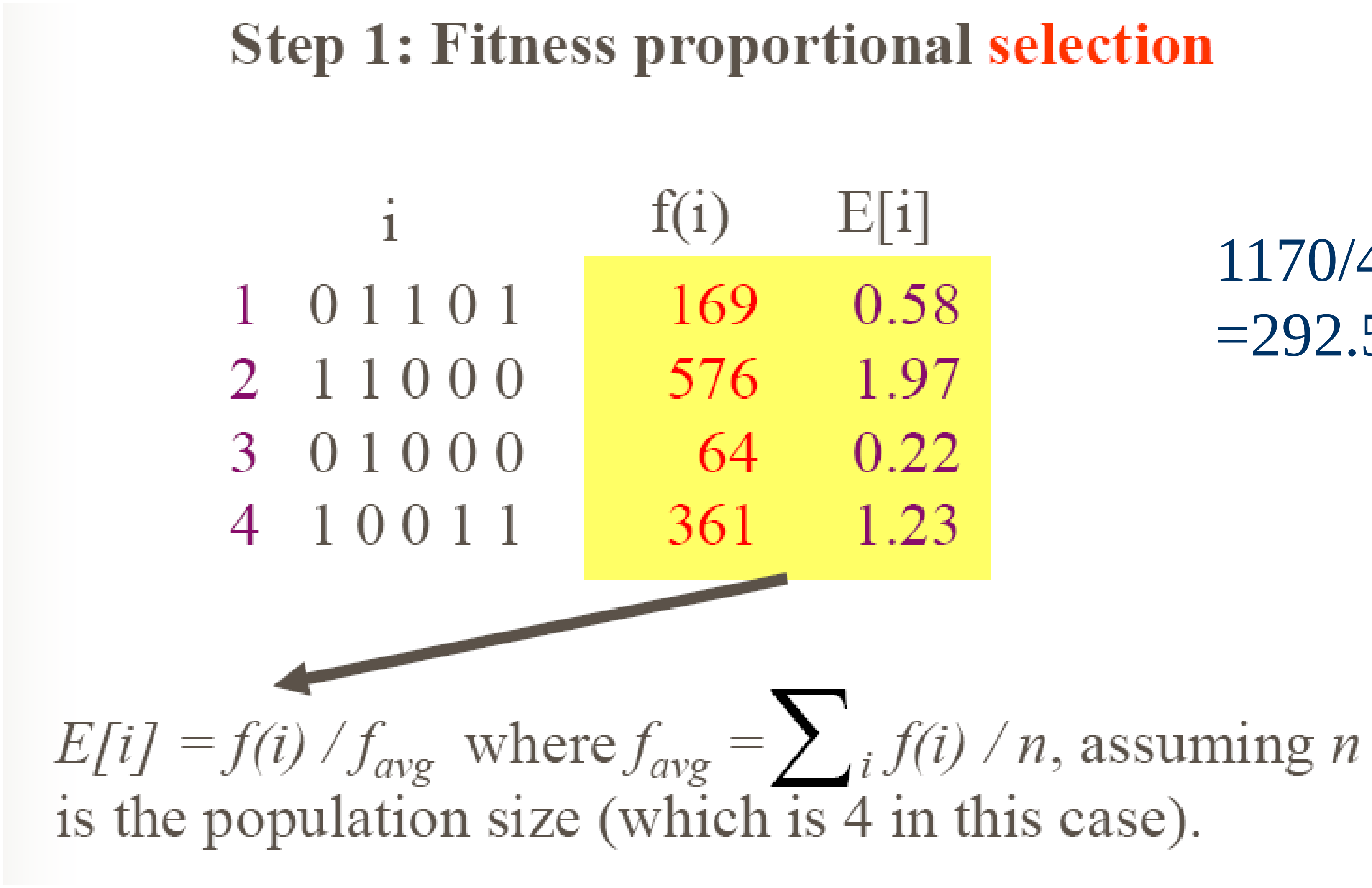
    where c is a constant, usually 2.0

    $\langle f \rangle$: mean

    σ_f : standard deviation

  - **Example**

Stochastic Universal Sampling



Step 1: Fitness proportional selection

	i	f(i)	E[i]
1	0 1 1 0 1	169	0.58
2	1 1 0 0 0	576	1.97
3	0 1 0 0 0	64	0.22
4	1 0 0 1 1	361	1.23

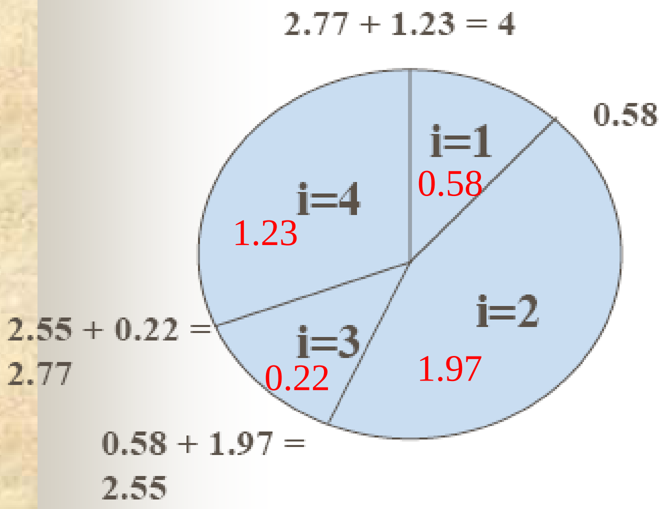
$E[i] = f(i) / f_{avg}$ where $f_{avg} = \sum_i f(i) / n$, assuming n is the population size (which is 4 in this case).

1170/4
=292.5

Generating successive populations

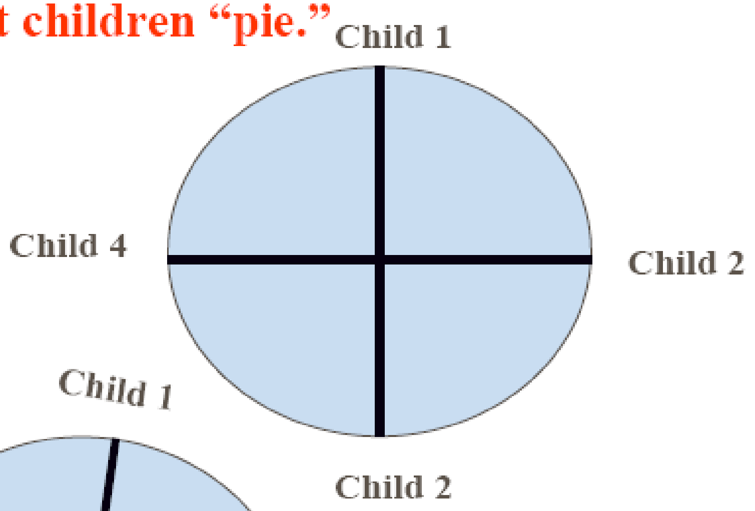
Next step: Stochastic Universal Sampling

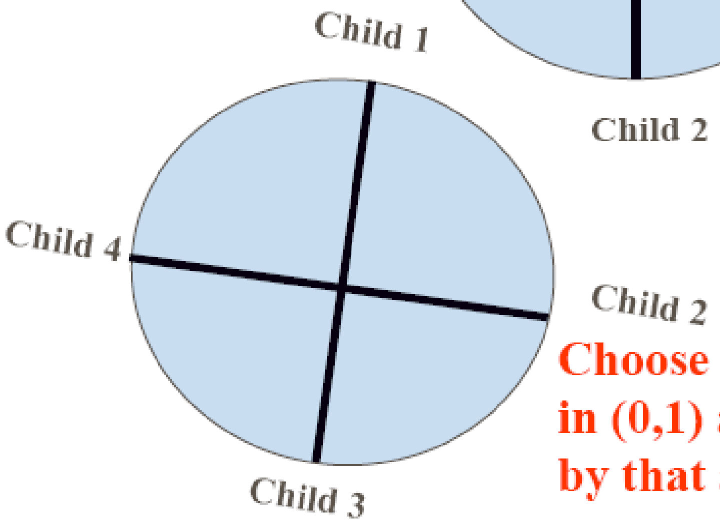
(A) Expectation "pie."



Pie slice for each E(i)

(B) Divide another pie by population size to get children "pie."

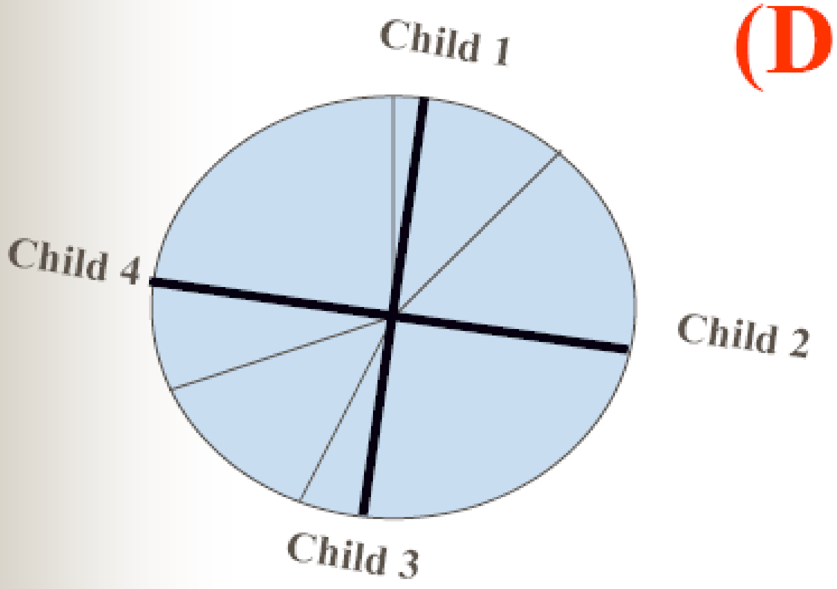




(C) Choose a random number in (0,1) and spin children pie by that amount

Generating successive populations

Next step: Stochastic Universal Sampling



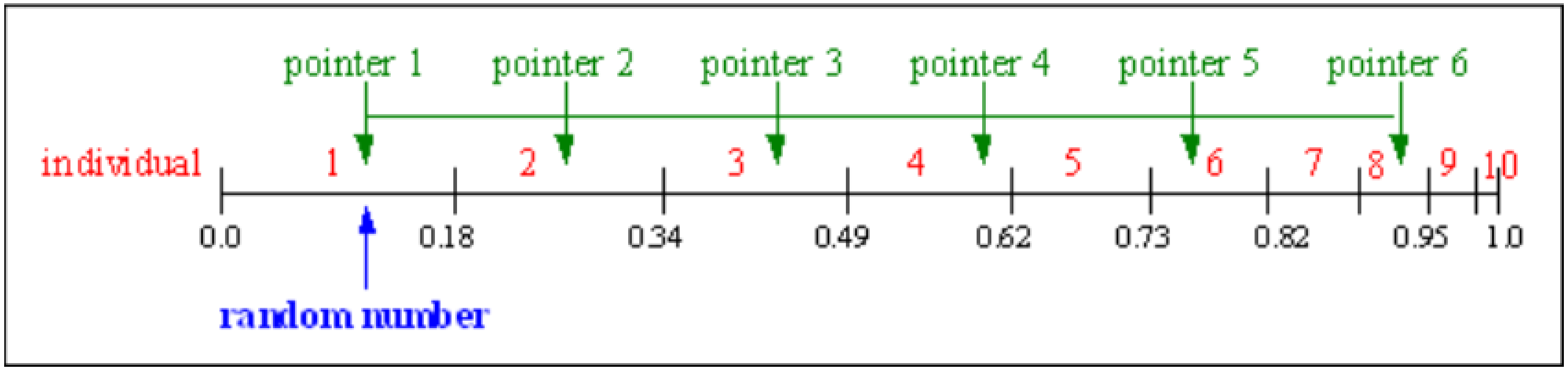
(D) Superimpose children pie on top of expectation pie. This gives the number of children of each individual.

The number of children generated cannot be less than the floor of E(i) and cannot be greater than the ceiling of E(i).

```
BEGIN
  /*  Given the cumulative probability distribution a */
  /*  and assuming we wish to select $\lambda$ members of the mating pool */
  set $current\_member = i = 1$;
  Pick a random value $r$ uniformly from $[0, 1/\lambda]$;
  WHILE ( $current\_member \leq \lambda$ ) DO
    WHILE (  $r \leq a[i]$ ) DO
      set mating_pool[$current\_member$] = parents[$i$];
      set $r = r + 1/\lambda$;
      set $current\_member = current\_member + 1$;
    OD
    set $i = i + 1$;
  OD
END
```

Fig. 5.2. Pseudocode for the stochastic universal sampling algorithm making λ selections

- For 6 individuals to be selected, the distance between the pointers is 1/6=0.167
- Sample 1 random number in the range [0 0.167]
- random number=0.1, for example



Rank–Based Selection

- Attempt to remove problems of FPS by basing selection probabilities on ***relative*** rather than *absolute* fitness
- Rank population according to fitness and then base selection probabilities on rank where fittest has rank μ (or 1) and worst rank 1 ($\mu \rightarrow \ldots \rightarrow 1$ or $1 \rightarrow \ldots \rightarrow \mu$)
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time
- Fitness $\rightarrow$ rank $\rightarrow$ probability for selection
- Mapping from rank no. to probability is arbitrary, e.g. linear or nonlinear (exponential) decreasing.

Require a user-defined parameter *q* and

a linear function: $prob(rank) = q - (rank - 1)r$

or a nonlinear function: $prob(rank) = q(1-q)^{rank-1}$

rank = 1 ➔ best individual;

rank = pop_size ➔ worst one

Meet the requirement: $\sum_{i=1}^{pop\_size} prob(i) = 1$

Example: (linear $prob(rank) = q - (rank - 1)r$)

q=[1/pop_size, 2/pop_size]

pop_size=100, choose q=0.015,

$$r = q/(pop\_size - 1) = 0.015/(100 - 1) = 0.00015151515$$

$$prob(1) = 0.015$$

$$prob(2) = 0.0148484848$$

$$\vdots$$

$$prob(100) = 0.00000000000000051$$

Example: (nonlinear $prob(rank) = q(1-q)^{rank-1}$)

➔ larger *q* implies stronger selective pressure

$q \in (0..1)$

Choose $pop\_size = 100$

$q = 0.1$

$$prob(1) = 0.100$$
$$prob(2) = 0.1 \cdot 0.9 = 0.09$$
$$prob(3) = 0.1 \cdot 0.9^2 = 0.081$$
$$\vdots$$
$$prob(100) = 0.000003$$

$q \in (0..1)$

Choose $pop\_size = 100$

$q = 0.2$

$$prob(1) = 0.200$$
$$prob(2) = 0.2 \cdot 0.8 = 0.16$$
$$prob(3) = 0.2 \cdot 0.8^2 = 0.128$$
$$\vdots$$
$$prob(100) = 5 \times 10^{-11}$$

$$\sum_{i=1}^{pop\_size} prob(i) = \sum_{i=1}^{pop\_size} q(1-q)^{i-1} \approx 1$$

Drawbacks:

- ➢Put the user to decide when to use these mechanisms
- ➢Ignore the info about relative evaluations of different chromosomes
- ➢Treat all cases uniformly, regardless of the magnitude of the problem
- ➢Violate the Schema theorem

Advantages:

- ➢Prevent scaling problem
- ➢Better control of selection pressure

Tournament Selection

- All methods above rely on global population statistics
  - Require global knowledge of the population
  - Could be a bottleneck esp. if the population is distributed in some way on parallel machines
  - Relies on presence of external fitness function which might not exist
- Informal procedure:
  - Pick *k* members at random then select the best one from this set of *k* elements
  - Repeat *pop_size* number of times to select more individuals

Tournament Selection 2

- Probability of selecting *i* will depend on:
  - Rank of *i* (sorting the whole population is not required)
  - Tournament Size *k*
    - higher *k* increases selection pressure
  - Whether contestants are picked with replacement
    - Picking without replacement increases selection pressure
    - Picking with replacement: selected individuals replace the ones in the mating pool, thus preventing premature due to super genes
  - Whether fittest contestant always wins (deterministic, *p*=1.0, higher selection pressure) or this happens with probability *p<1.0* (lower selection pressure)
- Typical value accepted by many applications is *k* = 2, (tournament size)

- Most widely used in GA, due to:
  - its simplicity
  - good control of selection pressure by varying tournament size *k*

```
BEGIN
  /* Assume we wish to select $\lambda$ members of a pool of $\mu$ individuals */
  set $current\_member = 1$;
  WHILE ( $current\_member \leq \lambda$ ) DO
    Pick $k$ individuals randomly, with or without replacement;
    Compare these $k$ individuals and select the best of them;
    Denote this individual as $i$;
    set mating_pool[$current\_member$] = $i$;
    set $current\_member = current\_member + 1$;
  OD
END
```

Fig. 5.3. Pseudocode for the tournament selection algorithm

Survivor Selection

- Most of methods above used for parent selection
- Survivor selection can be divided into two approaches:
  - Age-Based Selection
    - Parents are simply discarded and replaced by the entire set of offspring
    - e.g. SGA (no of offspring λ = no of parent μ)
  - Fitness-Based Selection
    - Choose μ from the (μ+ λ) parents and offspring based on fitness.

Two Special Cases

- Elitism (菁英法)
  - Widely used in both population models
  - Always keep at least one copy of the fittest solution so far
- GENITOR: a.k.a. “delete-worst”
  - Worst λ members of the population are selected for replacement
  - From Whitley’s original Steady-State algorithm (he also used linear ranking for parent selection)
  - Rapid takeover : use with large populations or “no duplicates” policy for preventing premature

Supplementary materials

Cycle crossover

Basic idea:

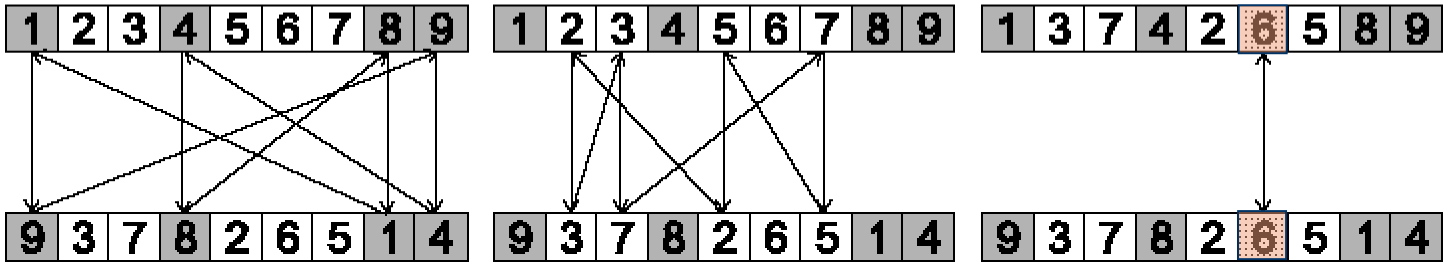
Each allele comes from one parent *together with its position*.

Informal procedure:

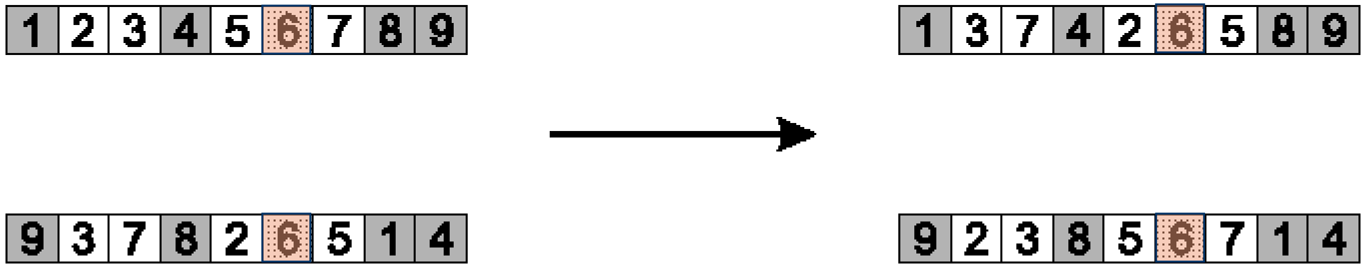
1. Make a cycle of alleles from P1 in the following way.
    (a) Start with the first allele of P1.
    (b) Look at the allele at the *same position* in P2.
    (c) Go to the position with the *same allele* in P1.
    (d) Add this allele to the cycle.
    (e) Repeat step b through d until you arrive at the first allele of P1.
2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
3. Take next cycle from second parent

Cycle crossover example

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



Linear Ranking

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

- Parameterised by factor *s:* 1.0 < *s* ≤ 2.0
  - measures advantage of best individual
  - in GGA this is the number of children allotted to it
- Simple 3 member example

	Fitness	Rank	P_{selFP}	P_{selLR} $(s=2)$	P_{selLR} $(s=1.5)$
A	1	1	0.1	0	0.167
B	5	23	0.5	0.67	0.5
C	4	2	0.4	0.33	0.33
Sum	10		1.0	1.0	1.0

Exponential Ranking

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

- Linear Ranking is limited to selection pressure
- Exponential Ranking can allocate more than 2 copies to fittest individual
- Normalise constant factor *c* according to population size

Example application of order based GAs: JSSP

Precedence constrained job shop scheduling problem

- J is a set of jobs.
- O is a set of operations
- M is a set of machines
- *Able* $\subseteq$ O $\times$ M defines which machines can perform which operations
- *Pre* $\subseteq$ O $\times$ O defines which operation should precede which
- *Dur* : $\subseteq$ O $\times$ M $\rightarrow$ IR defines the duration of o $\in$ O on m $\in$ M

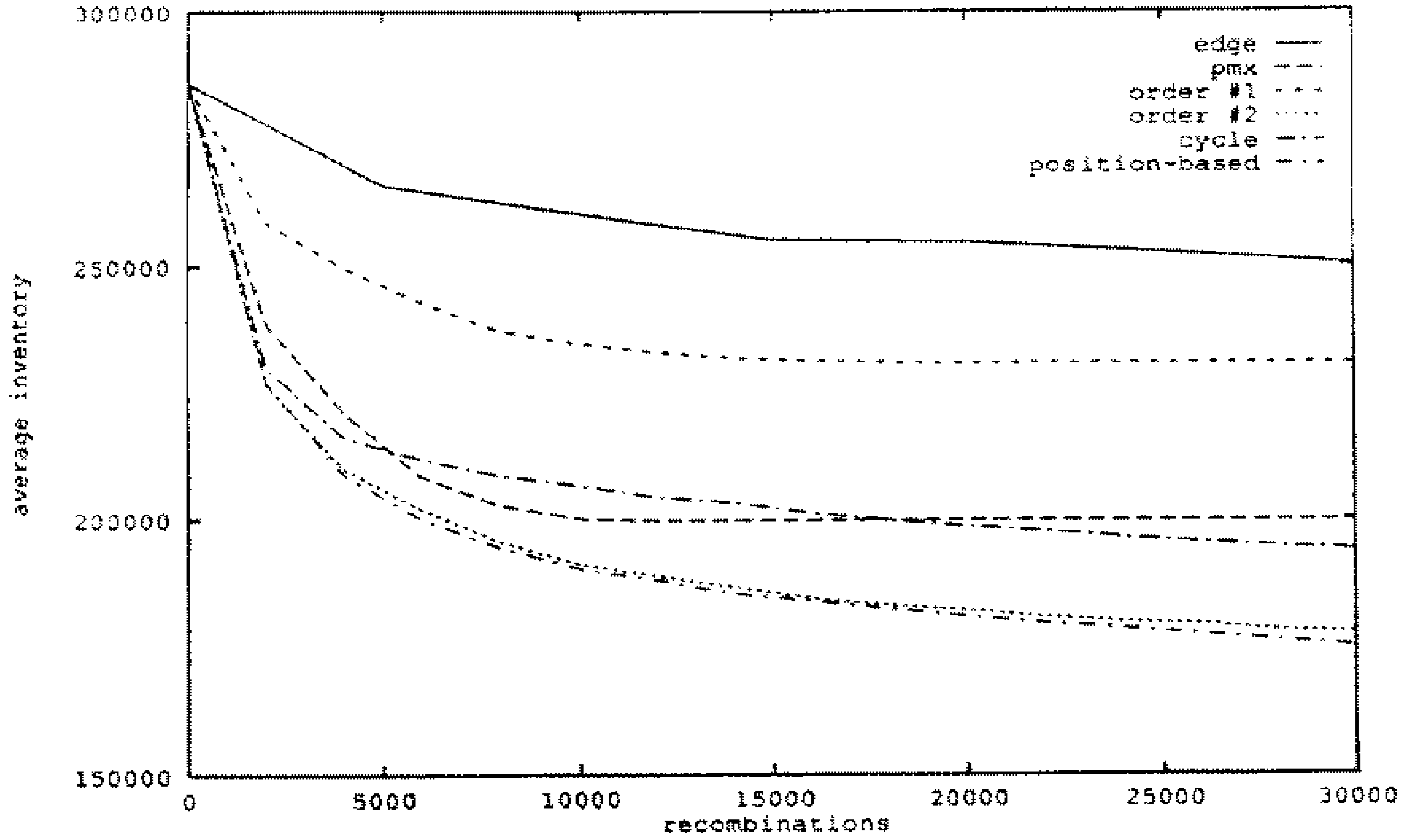
The goal is now to find a schedule that is:

- Complete: all jobs are scheduled
- Correct: all conditions defined by *Able* and *Pre* are satisfied
- Optimal: the total duration of the schedule is minimal

Precedence constrained job shop scheduling GA

- Representation: individuals are permutations of operations
- Permutations are decoded to schedules by a decoding procedure
  - take the first (next) operation from the individual
  - look up its machine (here we assume there is only one)
  - assign the earliest possible starting time on this machine, subject to
    - machine occupation
    - precedence relations holding for this operation in the schedule created so far
- fitness of a permutation is the duration of the corresponding schedule (to be minimized)
- use any suitable mutation and crossover
- use roulette wheel parent selection on inverse fitness
- Generational GA model for survivor selection
- use random initialisation

JSSP example: operator comparison


edge
pmx
order #1
order #2
cycle
position-based
average inventory
recombinations