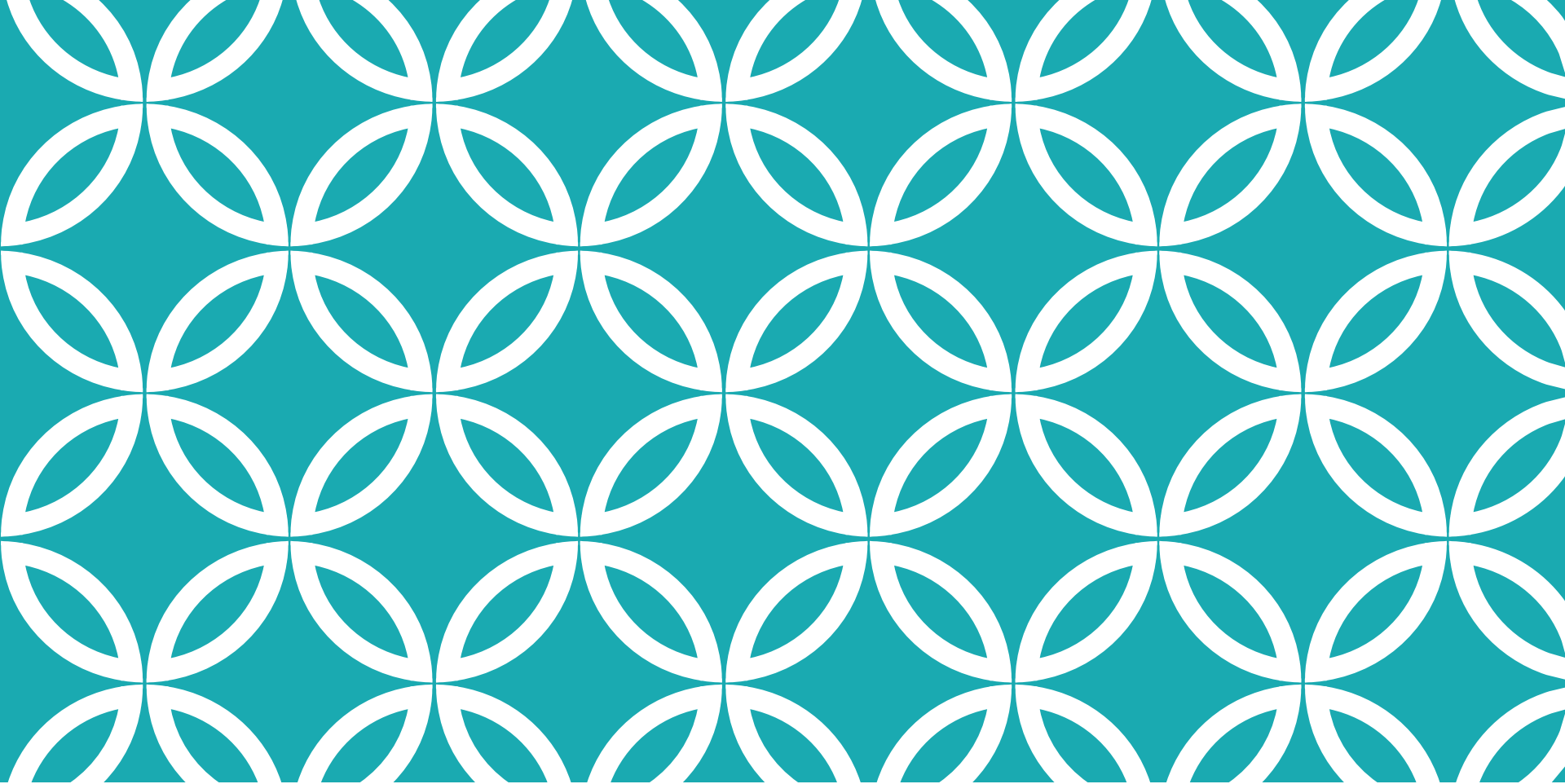



# COOPERATIVE AND ADAPTIVE ALGORITHMS

WEEK 6



# SEARCH AND INFORMATION






Conservation of information theorems indicate that any search algorithm performs on average as well as random search without replacement unless it takes advantage of problem-specific information about the search target or the search-space structure.

Combinatorics shows that even a moderately sized search requires problem-specific information to be successful.

Three measures to characterize the information required for successful search are (1) endogenous information, which measures the difficulty of finding a target using random search;

(2) exogenous information, which measures the difficulty that remains in finding a target once a search takes advantage of problem-specific information; and

(3) active information, which, as the difference between endogenous and exogenous information, measures the contribution of problem-specific information for successfully finding a target.



A methodology is developed based on these information measures to gauge the effectiveness with which problem-specific information facilitates successful search. It then applies this methodology to various search tools widely used in evolutionary search.

# INFORMATION TRANSFORMATION

“The [computing] machine does not create any new information, but it performs a very valuable transformation of known information.”

--Leon Brillouin, *Science and Information Theory* (Academic Press, New York, 1956).

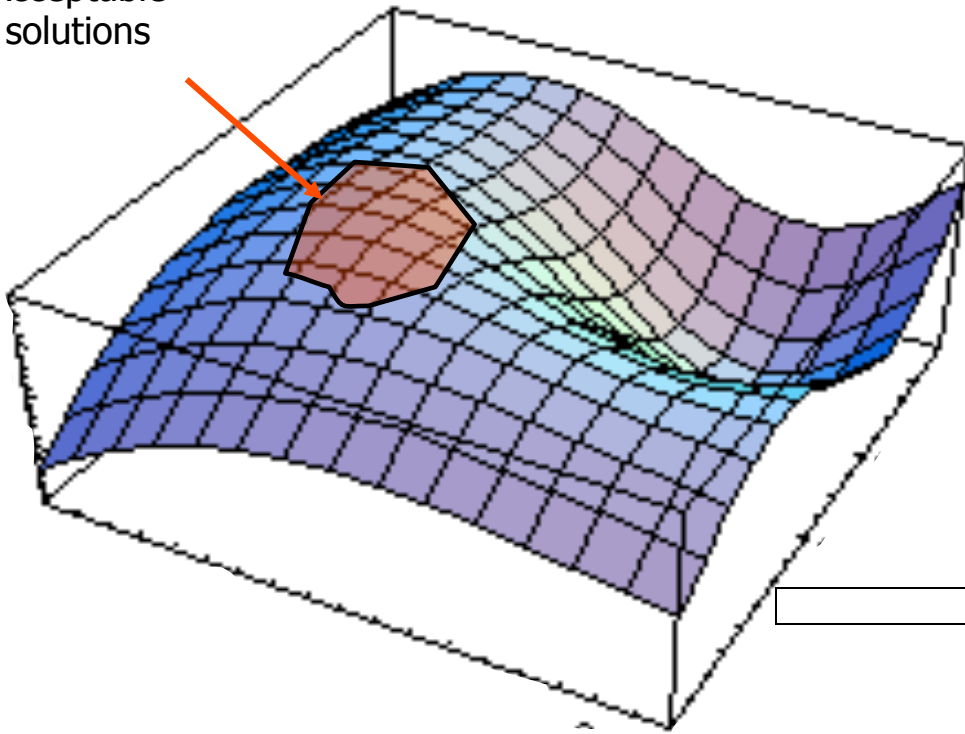
# BERNOULLI'S PRINCIPLE OF INSUFFICIENT REASON

“in the absence of any prior knowledge, we must assume that the events have equal probability

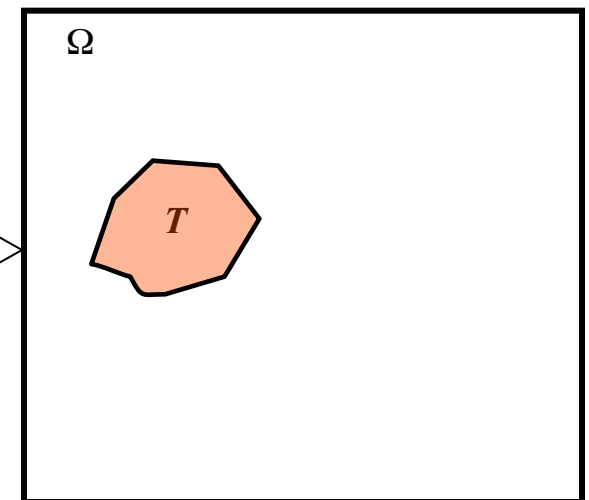
Jakob Bernoulli, “Ars Conjectandi” (“The Art of Conjecturing”), (1713).

# FITNESS

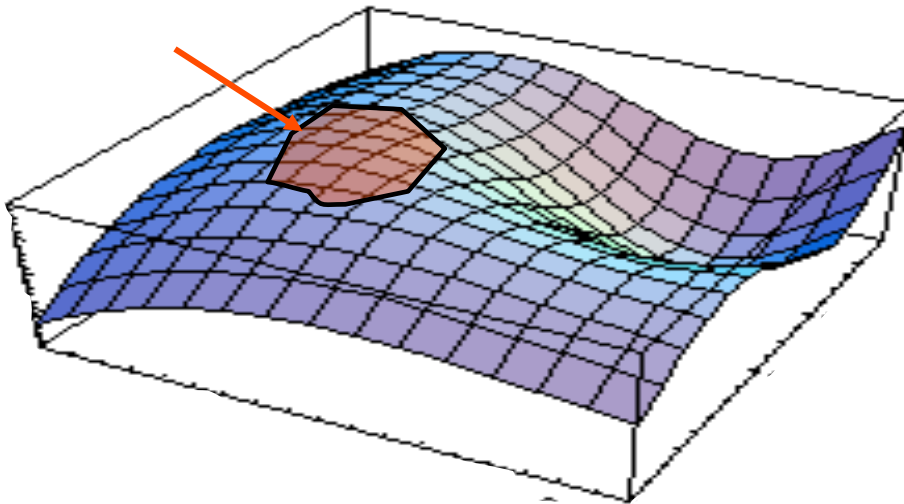
Acceptable  
solutions



Each point in the parameter space  
has a fitness. The problem of the  
search is finding a good enough  
fitness.



# SEARCH ALGORITHMS



Steepest Ascent  
Exhaustive  
Newton-Rapheson  
Levenberg-Marquardt  
Tabu Search  
Simulated Annealing  
Particle Swarm Search  
Evolutionary Approaches

Problem: In order to work better than average, each algorithm implicitly assumes something about the search space and/or location of the target.



# QUOTES ON THE NEED FOR ADDED INFORMATION FOR TARGETED SEARCH ...

1. "...unless you can make prior assumptions about the ... [problems] you are working on, then no search strategy, no matter how sophisticated, can be expected to perform better than any other" Yu-Chi Ho and D.L. Pepyne, (2001).

2. No free lunch theorems "indicate the importance of incorporating problem-specific knowledge into the behavior of the [optimization or search] algorithm." David Wolpert & William G. Macready (1997).

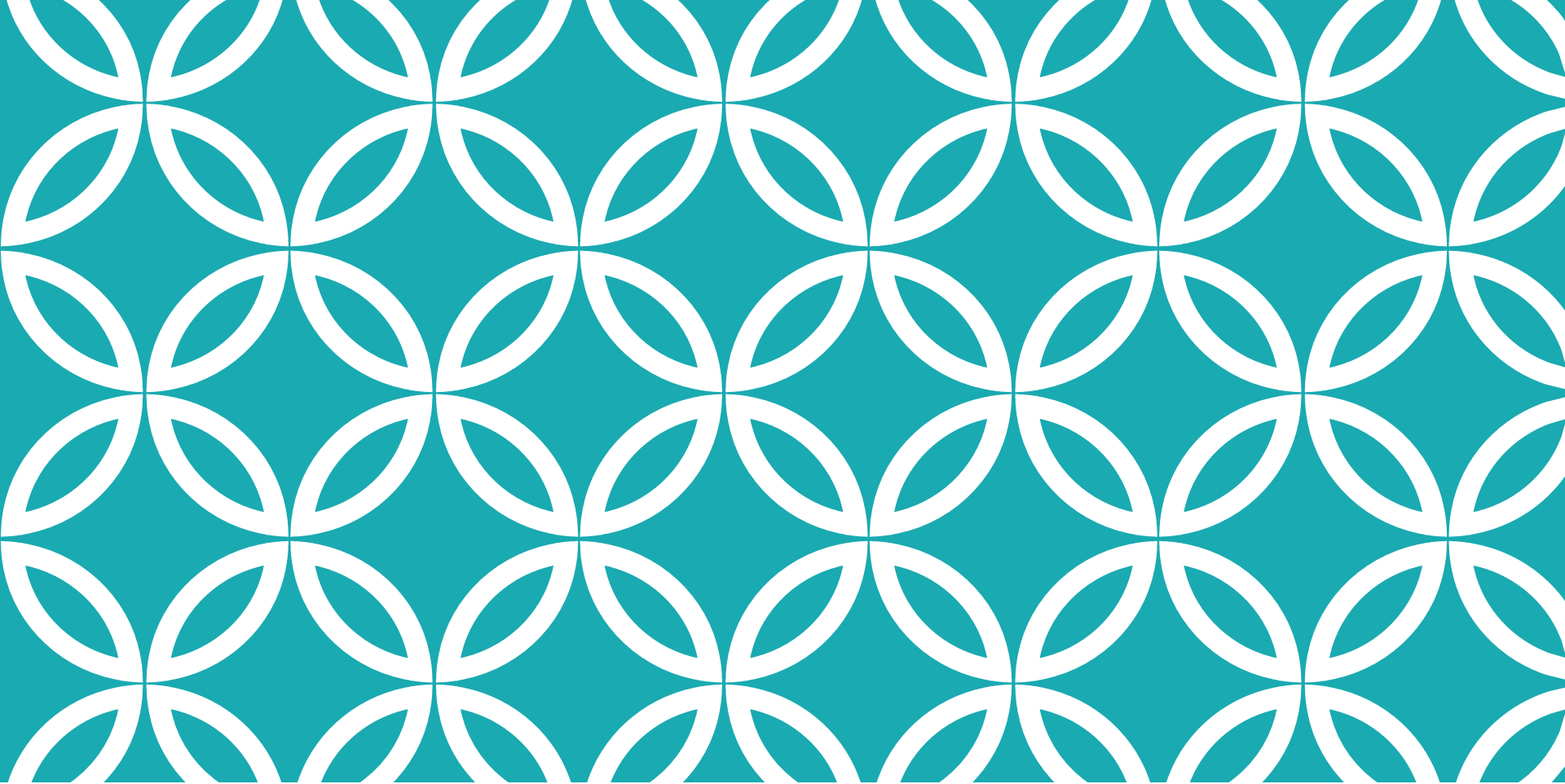
1. ``Simple explanantion of the No Free Lunch Theorem", Proceedings of the 40th IEEE Conference on Decision and Control, Orlando, Florida,
2. "No free lunch theorems for optimization", IEEE Trans. Evolutionary Computation 1(1): 67-82 (1997).

# ACTIVE INFORMATION

Nothing works better, on the average, than random search.

For a search algorithm like evolutionary search to work, we require active information.





# GENETIC ALGORITHMS (GAS)

Based on material and slides from  
A. E. Eiben and J. E. Smith. "Introduction to Evolutionary Computing". Springer, 2003.

# OUTLINE

Introduction,  
The Simple GA,  
Real-valued GAs,  
Permutations GAs, TSP  
Applications,  
GAs, When are they useful?  
Adaptive GAs,  
Parallel GAs,  
References.

# OUTLINE

Introduction,  
The Simple GA,  
Real-valued GAs,  
Permutations GAs, TSP  
Applications,  
GAs, When are they useful?  
Adaptive GAs,  
Parallel GAs.

# INTRODUCTION

Based on the Darwinian theory of evolution,

The theory offers an explanation of the biological diversity and its underlying mechanisms,

*Natural selection* plays an important role.

# INTRODUCTION

Natural selection favours those individuals that compete for the given resource most effectively, those who are adapted to fit the environmental conditions best,

This phenomenon is known as *survival of the fittest*.

# INTRODUCTION

The fitness is determined by the *phenotypic traits* (*properties produced* by the interaction of the genotype and the environment)

- Behavioural features,
- Physical features.

Selected individuals reproduce, passing their properties to the offspring,

Other individuals die without mating, their properties are thus discarded.



# INTRODUCTION

Darwin also recognized that small and random variations, *mutations*, occur in the phenotypic traits during reproduction,

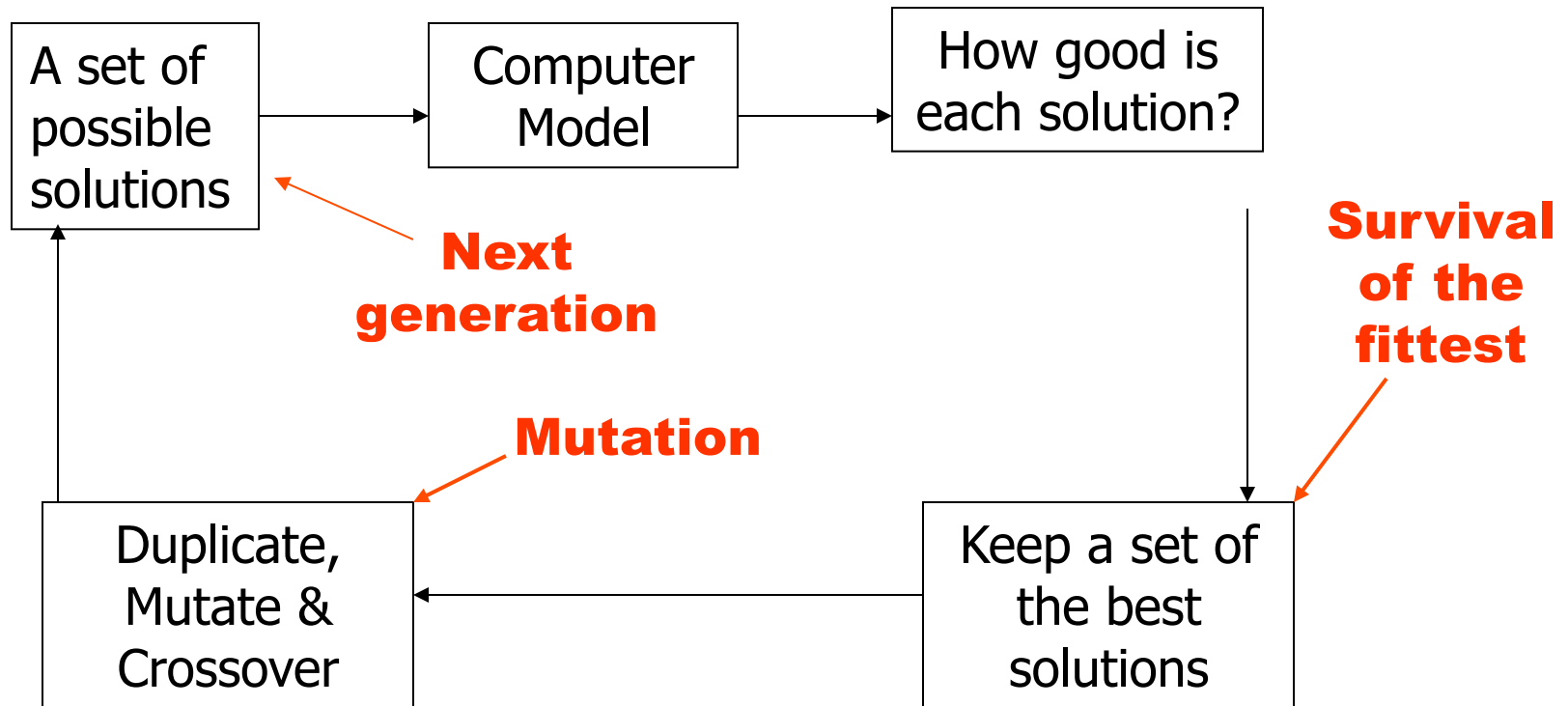
Through these variations, new combinations of traits occur and get evaluated,

The basic operations are hence *selection*, *reproduction* and *mutation*.

# INTRODUCTION

<u>GAs</u>		<u>Problem solving</u>
Environment	↔	Problem
Individual	↔	Candidate solution
Fitness	↔	Quality
Population	↔	Set of possible solutions, constant
Fitness		chances for survival and reproduction
Quality		chances for seeding new solutions

# GA



# OUTLINE

Introduction,  
The Simple GA,  
Real-valued GAs,  
Permutations GAs, TSP  
Applications,  
GAs, When are they useful?  
Adaptive GAs,  
Parallel GAs,  
References.

# THE SIMPLE GA

Introduction,  
The algorithm,  
Representation,  
Selection,  
Crossover,  
Mutation,  
Why crossover and mutation?  
A sample run.

# GAS - INTRODUCTION

First GA was developed by Holland in 1975, referred to as SGA (Simple GA),

Sometimes referred to as the *classical* or *canonical* GA.

# SGA - INTRODUCTION

GAs handle a *population of individuals* (solutions),

The probability of selecting a bad solution is reduced by handling multiple good solutions,

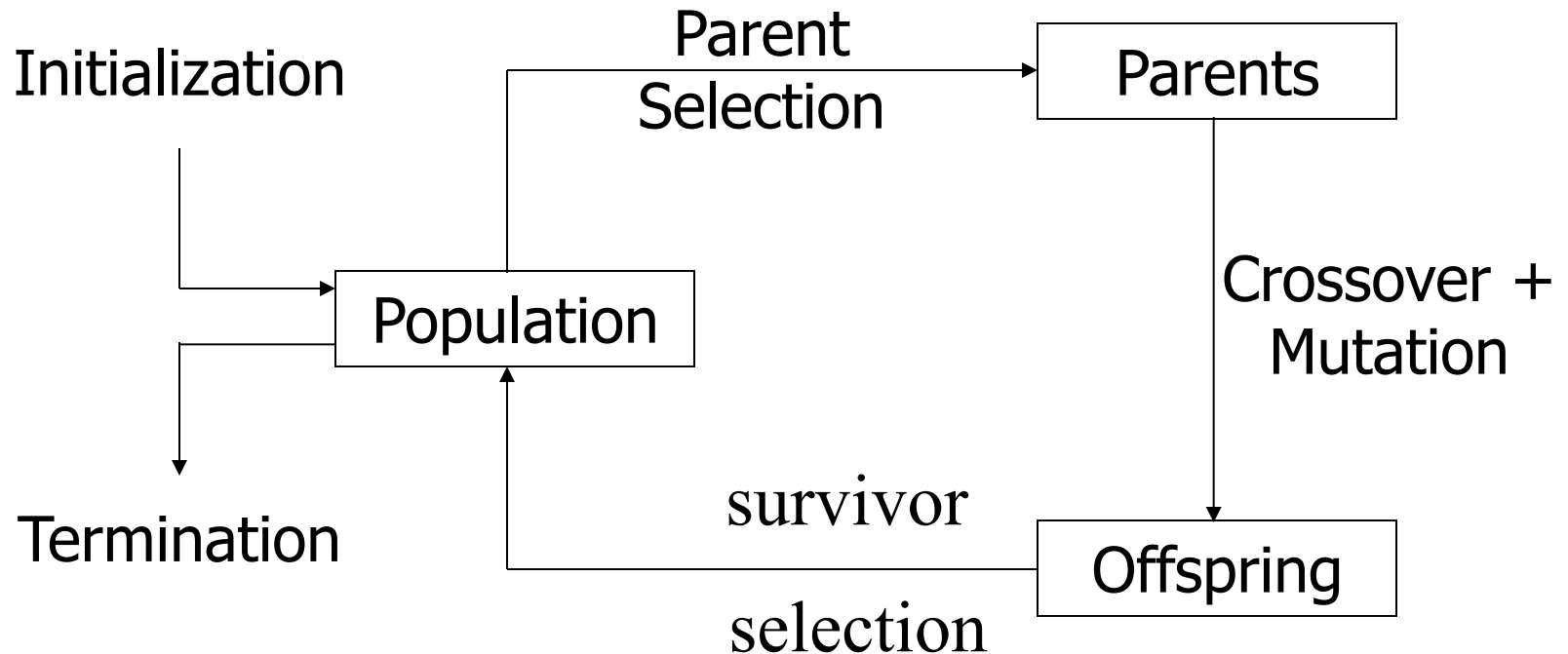
The population evolves from one iteration to the next according to selection and the *search operators (genetic operators)*,

Search operators:

- Recombination (Crossover),
- Mutation.

# SGA - INTRODUCTION

The general scheme [1]:





# SGA INGREDIENTS

Representation	Binary Strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

# SGA – THE ALGORITHM

Initialize population with random candidates,

Evaluate all individuals,

While *termination criteria* is not met

- Select parents,
- Apply crossover,
- Mutate offspring,
- Replace current generation,

end while

# SGA – THE ALGORITHM

The *termination criteria* could be:

- A specified number of generations (or fitness evaluations),
- A minimum threshold reached,
- No improvement in the best individual for a specified number of generations,
- Memory/time constraints,
- Combinations of the above.

# SGA - REPRESENTATION

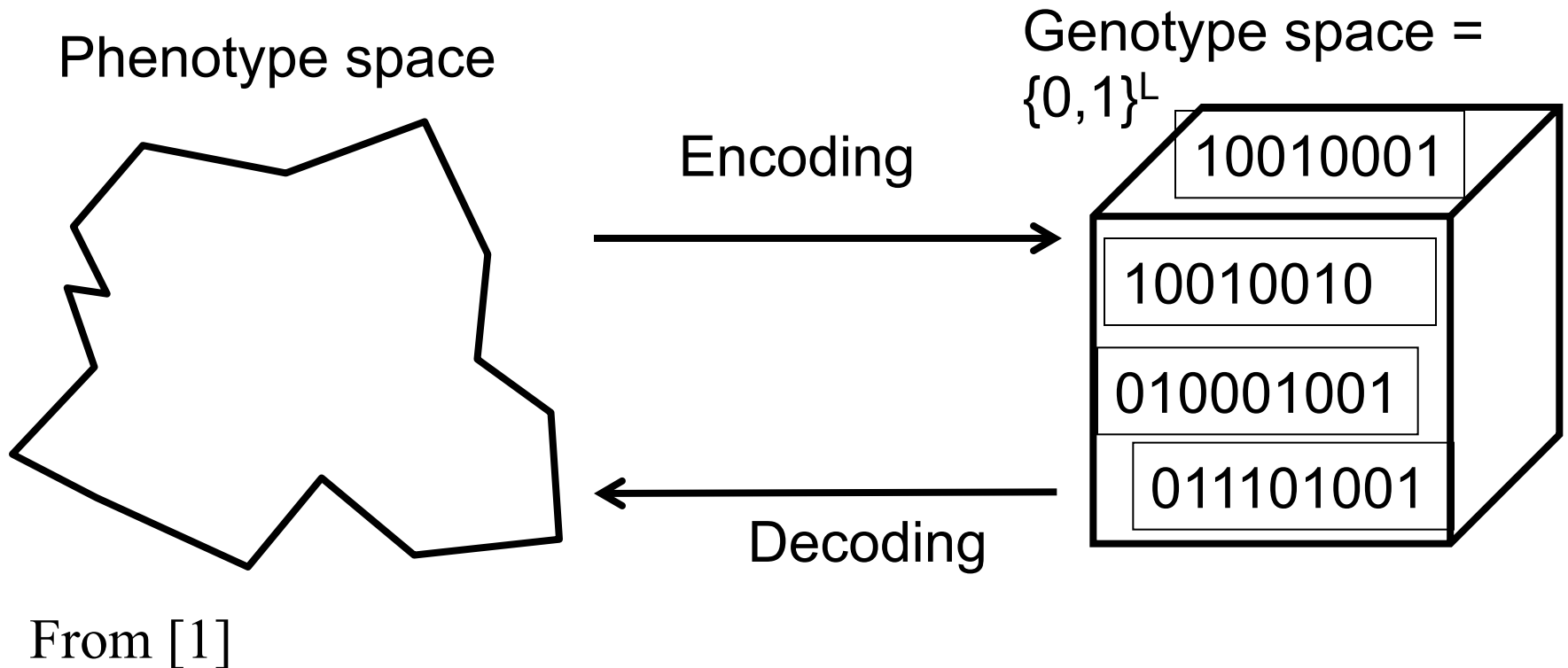
A mapping is applied from the parameter domain to the binary domain,

*Phenotype*, a vector in the parameter domain,

Abstract representation of Phenotype is called *chromosomes* or *genotype* or

In SGA *Genotype*, is a string in the binary domain.

# SGA - REPRESENTATION



# SGA REPRODUCTION CYCLE

**Select parents** for the mating pool

(size of mating pool = population size)

Shuffle the mating pool

**Apply crossover** for each consecutive pair with probability  $p_c$ , otherwise copy parents

**Apply mutation** for each offspring (bit-flip with probability  $p_m$  independently for each bit)

**Replace the whole population** with the resulting offspring

# SGA - SELECTION

Parents are selected with a probability proportional to their fitness, *proportional selection*.

Main idea: better individuals get higher chance:

- Chances proportional to fitness,
- Implementation: roulette wheel technique
  - Assign to each individual a part of the roulette wheel,
  - Spin the wheel  $n$  times to select  $n$  individuals.

# SGA - SELECTION

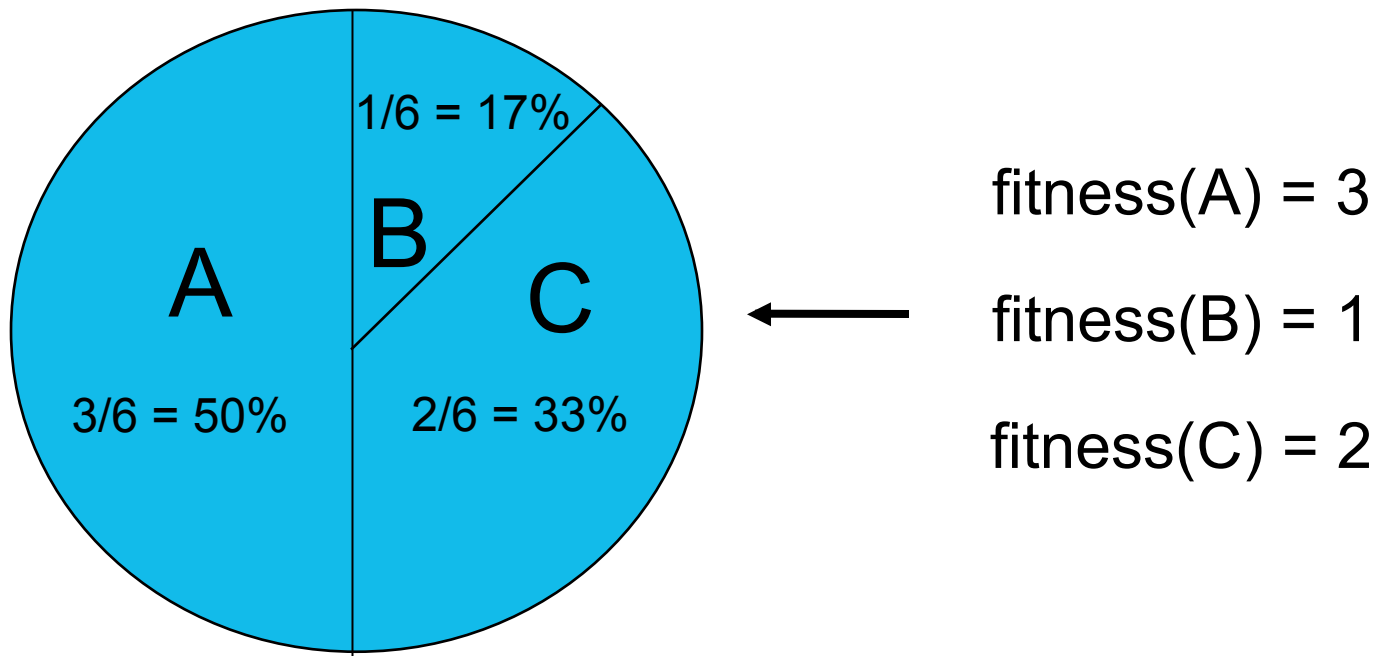


Figure from [1]



# SGA - CROSSOVER

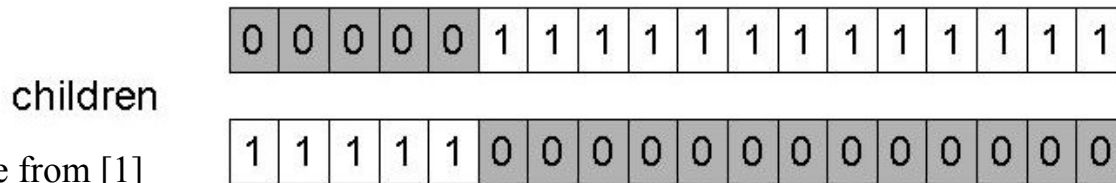
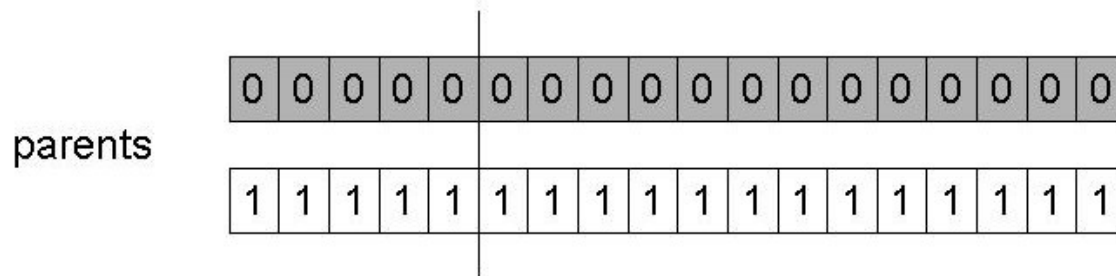
Crossover is applied according to a probability  $P_c$ ,

Otherwise, the two parent are copied to the offspring unmodified,

# SGA - CROSSOVER

1-point crossover (the binary case):

- 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).

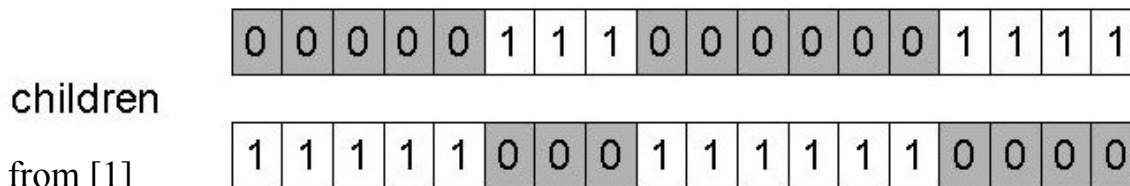
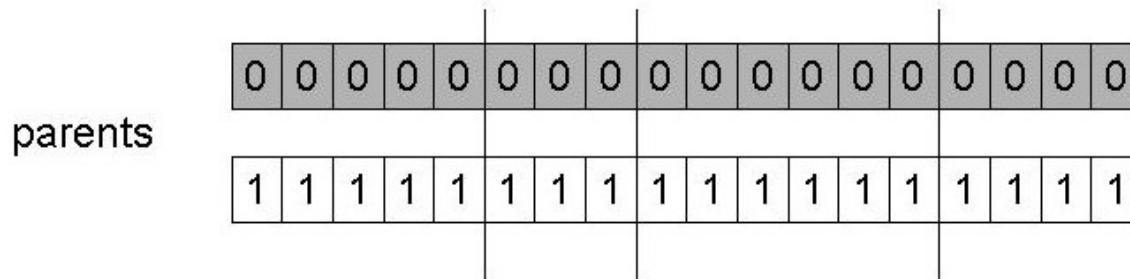


Example from [1]

# SGA - CROSSOVER

n-point crossover (the binary case):

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



Example from [1]

# SGA - CROSSOVER

Uniform crossover (the binary case): **Change**

- Assign 'heads' to one parent, 'tails' to the other,
- 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.

parents

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

children

0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Example from [1]

1	0	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

# SGA - MUTATION

Alter each gene independently with a probability  $p_m$

$p_m$  is called the mutation rate

- Typically between  $1/\text{pop\_size}$  and  $1/\text{chromosome\_length}$

parent

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

child

0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Example from [1]

# WHY CROSSOVER AND MUTATION?

Exploration:

Discovering promising areas in the search space.

Exploitation:

Optimizing within a promising area.

# WHY CROSSOVER AND MUTATION?

Crossover is *explorative*, it makes a big jump to an area somewhere between the two parent areas,

Mutation is *exploitative*, it creates random small diversions, thereby staying near (in the area of) the parent.

# WHY CROSSOVER AND MUTATION?

Only crossover can combine information from two parents,

Only mutation can introduce new information.



# SGA - A SAMPLE RUN

Max  $x^2$  over  $\{0,1,\dots,31\}$ ,

GA approach:

- Binary individuals (5 bits),
- 4 individuals,
- 1-point crossover,
- Roulette wheel selection (fitness proportional).

# SGA – A SAMPLE RUN

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

$$\text{Actual Count} = \frac{f_i}{\bar{f}}$$

$$p_i = \frac{f_i}{\sum f_i}$$

*Actual count = round (Expected count)*

Initial population, evaluation for parent selection

Example from [1]

# SGA – A SAMPLE RUN

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

Example from [1] Crossover and offspring evaluation

# SGA – A SAMPLE RUN

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	<del>26</del> 28	<del>676</del> 784
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	<del>-18</del> 20	<del>324</del> 400
Sum				2538
Average				634.5
Max				784

Mutation and offspring evaluation

Example from [1]

# SGA A SAMPLE RUN

The algorithm has shown progress:

Average fitness grows from 239 to 599.5 and  
best fitness from 576 to 729.

# OTHER EXAMPLES

- How to deal with functions of more than one variable?
- How to formulate a Curve Fitting problem for GA?

# ALTERNATIVE BINARY REPRESENTATION

## Gray coding of integers

- 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

## ■ Gray code

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

Hamming distance  
between any consecutive  
numbers is 1



# GAS – OTHER REPRESENTATIONS

Other representations are also possible:

- Integers,
- Real-valued or floating-point numbers,
- Permutations.

The crossover and mutation actual implementation depends on the representation.

# POPULATION MODELS

SGA uses a Generational model (GGA):

- each individual survives for exactly one generation
- the entire set of parents is replaced by the offspring

Steady-State model:

- Only part of the population is replaced by the offspring,
- Usually one member of population is replaced,
- the proportion of the population replaced is called the *Generational Gap*
- 1.0 for GGA,  $1/\text{pop\_size}$  for SSGA

# PARENTS AND OFFSPRING SELECTION

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)

# PARENT SELECTION

Fitness-Proportionate Selection (FPS)

Probability of parent selection

$$p_i = \frac{f_i}{\sum f_j}$$

Expected number of copies of an individual  $i$

$$E(n_i) = f_i / \bar{f}$$

(  $f$  is the fitness and  $\bar{f}$  is the average fitness

# FPS

## Roulette wheel algorithm:

- 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 Stochastic Universal Sampling algorithm:

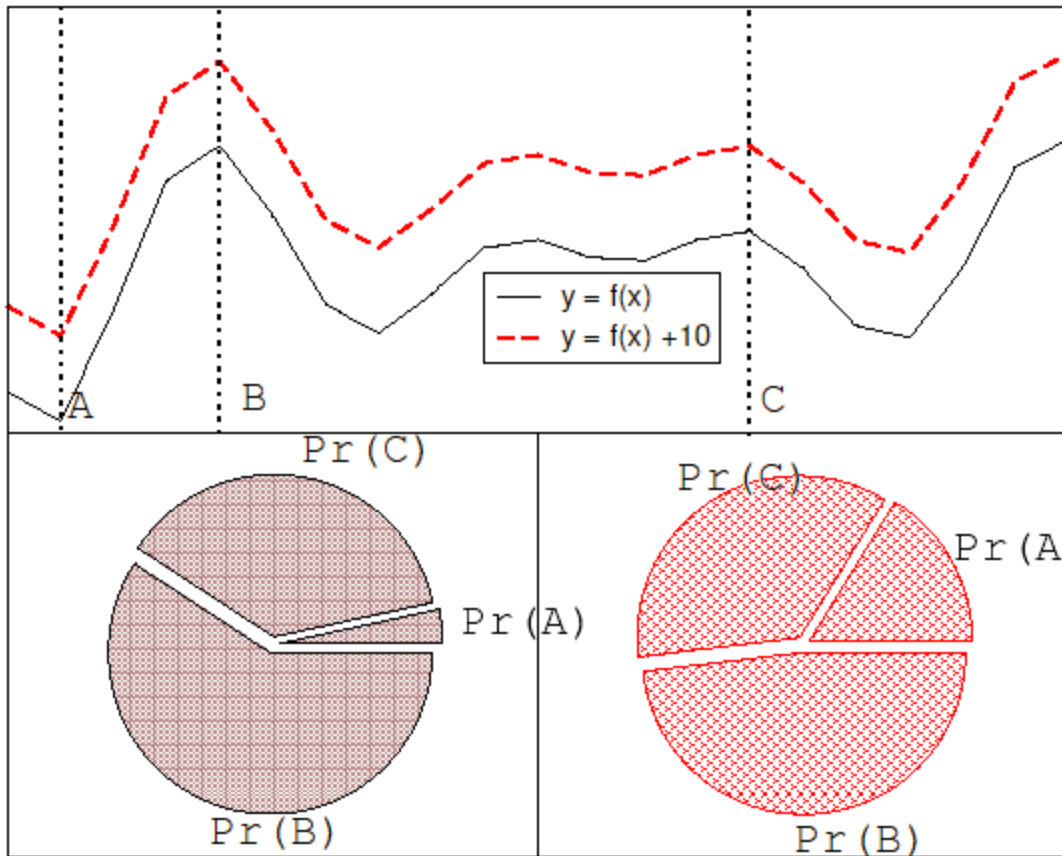
- $n$  evenly spaced arms on wheel and spin once
- Guarantees  $\text{floor}(E(n_i)) \leq n_i \leq \text{ceil}(E(n_i))$

# FPS

## Problems include:

- One highly fit member 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
- Highly susceptible to function transposition

# FUNCTION TRANSPOSITION FOR FPS



Example

$$\begin{aligned} A &= 2 & 12 \\ B &= 20 & 30 \\ C &= 8 & 18 \end{aligned}$$

---


$$\begin{aligned} T &= 30 & 60 \\ A/T &= 1/15 & 1/5 \\ B/T &= 2/3 & 1/2 \\ C/T &= 4/15 & 3/10 \end{aligned}$$

# 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  $\mu$  and worst rank 1

This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time



# TOURNAMENT SELECTION

All methods above rely on global population statistics

- Could be a bottleneck esp. on parallel machines
- Relies on presence of external fitness function which might not exist: e.g. evolving game players

In tournament selection:

- Pick  $k$  members at random then select the best of these
- Repeat to select more individuals

# SURVIVOR SELECTION

Survivor selection can be divided into two approaches:

- Age-Based Selection
  - e.g. GGA
  - In SSGA can implement the selection as “delete-random” or as first-in-first-out (a.k.a. delete-oldest)
- Fitness-Based Selection (FPS)
  - Delete or replace based on inverse of fitness or

# FPS

## Elitism

- Widely used in both population models (GGA, SSGA)
- Always keep best individuals (at least one copy of the fittest solution so far)

## GENITOR: a.k.a. “delete-worst”

- Can lead to rapid improvement and potential premature convergence
- use with large populations or “no duplicates” policy

# OUTLINE

Introduction,  
The Simple GA,  
Real-valued GAs,  
Permutations GAs, TSP  
Applications,  
GAs, When are they useful?  
Adaptive GAs,  
Parallel GAs,  
References.

# REAL-VALUED GAS

Many problems have real-valued parameters,

If mapped to a binary space, high precision solutions would require very long chromosomes,

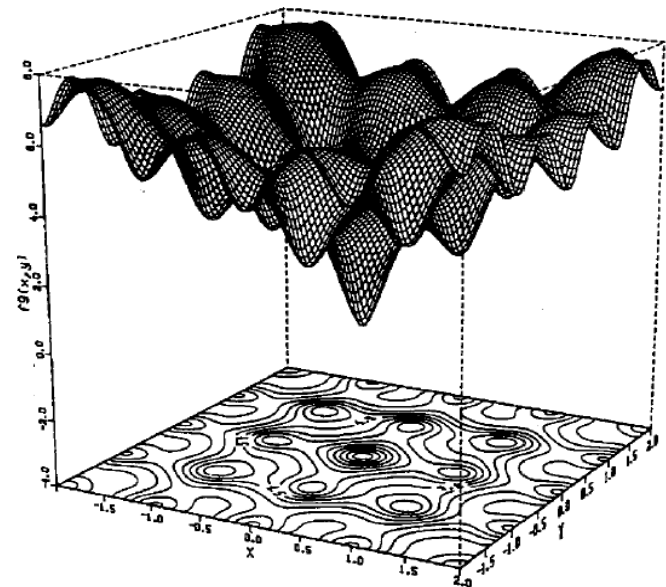
# REAL-VALUED GAS

e.g. continuous parameter optimization  $f: \mathcal{R}^n \rightarrow \mathcal{R}$

Illustration: Ackley's function (often used in EC).

$$f(\bar{x}) = -c_1 \cdot \exp \left( -c_2 \cdot \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right) - \exp \left( \frac{1}{n} \cdot \sum_{i=1}^n \cos(c_3 \cdot x_i) \right) + c_1 + 1$$
$$c_1 = 20, c_2 = 0.2, c_3 = 2\pi$$

Figure from [1]



# REAL-VALUED GAS

Two types of crossover operators:

- Discrete, use any of the crossover operators identified before for binary representations,
- Intermediate (arithmetic):
  - Single arithmetic,
  - Simple arithmetic,
  - Whole arithmetic.

# REAL-VALUED GAS

Single arithmetic crossover:

- Parents:  $\langle x_1, \dots, x_n \rangle$  and  $\langle y_1, \dots, y_n \rangle$ ,
- Pick random gene (k) at random,
- Child 1 is:

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

- Reverse for other child.



# REAL-VALUED GAS

Single arithmetic crossover:

- e.g. with  $\alpha = 0.5$ ,  $k=8$       $\alpha = 0.7$

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
-----	-----	-----	-----	-----	-----	-----	-----	-----

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.5	0.9
-----	-----	-----	-----	-----	-----	-----	-----	-----

0.38



0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3
-----	-----	-----	-----	-----	-----	-----	-----	-----

0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.5	0.3
-----	-----	-----	-----	-----	-----	-----	-----	-----

0.62

Example from [1]

# REAL-VALUED GAS

Simple arithmetic crossover:

- Parents:  $\langle x_1, \dots, x_n \rangle$  and  $\langle y_1, \dots, y_n \rangle$ ,
- Pick random gene (k), after this point mix the values,
- Child 1 is:

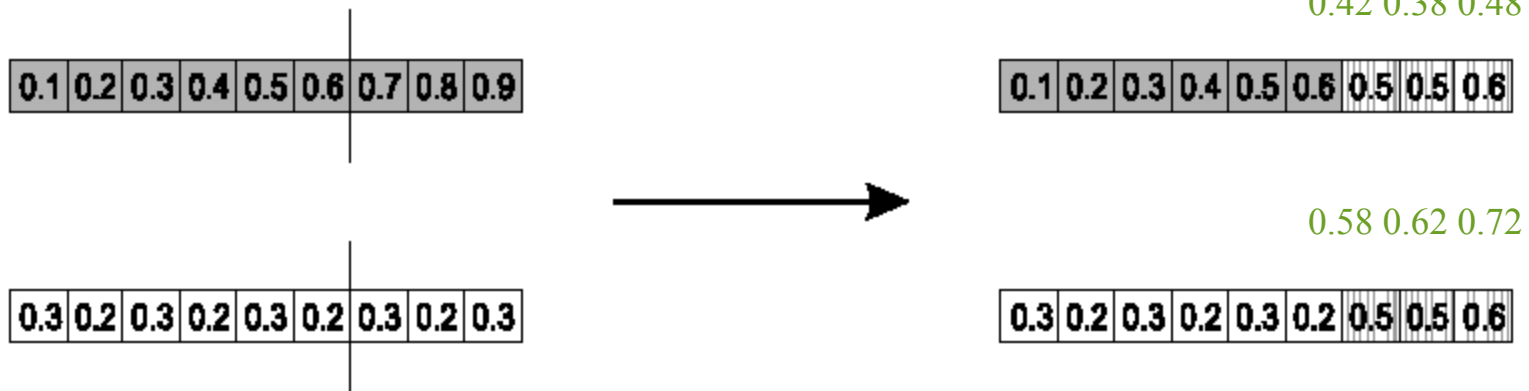
$$\langle x_1, \dots, x_k, \alpha \cdot y_{k+1} + (1 - \alpha) \cdot x_{k+1}, \dots, \alpha \cdot y_n + (1 - \alpha) \cdot x_n \rangle$$

- Reverse for other child.

# REAL-VALUED GAS

Simple arithmetic crossover:

- e.g. with  $\alpha = 0.5$ .       $\alpha = 0.7$



Example from [1]

# REAL-VALUED GAS

Whole arithmetic crossover:

- Most commonly used,
- Parents:  $\langle x_1, \dots, x_n \rangle$  and  $\langle y_1, \dots, y_n \rangle$ ,
- Child 1 is:

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

- Reverse for other child.

# REAL-VALUED GAS

Whole arithmetic crossover:

- e.g. with  $\alpha = 0.5$ .

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
-----	-----	-----	-----	-----	-----	-----	-----	-----

0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.5	0.6
-----	-----	-----	-----	-----	-----	-----	-----	-----



0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3
-----	-----	-----	-----	-----	-----	-----	-----	-----

0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.5	0.6
-----	-----	-----	-----	-----	-----	-----	-----	-----

Example from [1]

# REAL-VALUED GAS

The general scheme for mutation is:

$$\bar{x} = \langle x_1, \dots, x_l \rangle \rightarrow \bar{x}' = \langle x'_1, \dots, x'_l \rangle$$
$$x_i, x'_i \in [LB_i, UB_i]$$

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

Analogous to bit-flipping mutation.

# REAL-VALUED GAS

Another mutation scheme is to add random noise, for each gene:

$$x'_i = x_i + N(0, \sigma)$$

where  $N(0, \sigma)$  is a random Gaussian number with mean zero and standard deviation  $\sigma$ .

# OUTLINE

Introduction,  
The Simple GA,  
Real-valued GAs,  
Permutations GAs, TSP  
Applications,  
GAs, When are they useful?  
Adaptive GAs,  
Parallel GAs,  
References.



# PERMUTATIONS GAS

For problems that take the form of deciding on the order in which a sequence of events should occur,

Two types of problems exist:

- The events use limited resources or time. The *order* of events is important. e.g. Job Shop Scheduling,
- The *adjacency* of elements is important. e.g. Travelling Salesman Problem.

# PERMUTATIONS - GAS

If there are  $n$  variables then the representation is a list of  $n$  integers, each of which occurs exactly once,

5	4	3	2	1	6	10	9	8	7
---	---	---	---	---	---	----	---	---	---

# PERMUTATIONS GAS – TSP EXAMPLE

## Problem:

- Given  $n$  cities,
- Find a complete tour with minimal length.

## Encoding:

- Label the cities  $1, 2, \dots, n$
- One complete tour is one permutation (e.g.  $[1, 2, 3, 4]$ ,  $[3, 4, 2, 1]$ , ...).

Search space is **BIG**: for  $n$  cities there are  $n! \approx$  more than  $10^n$  possible tours.

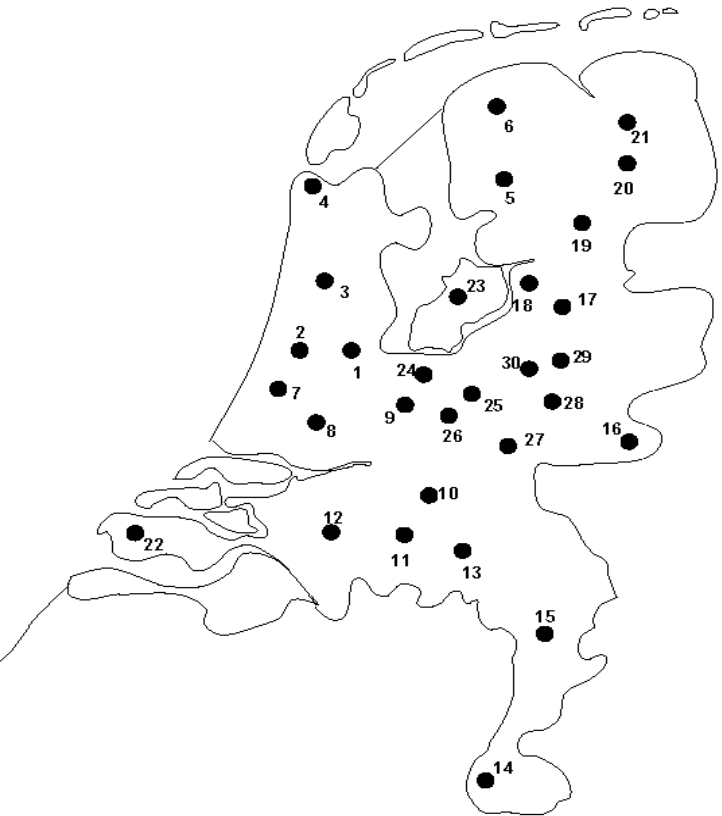
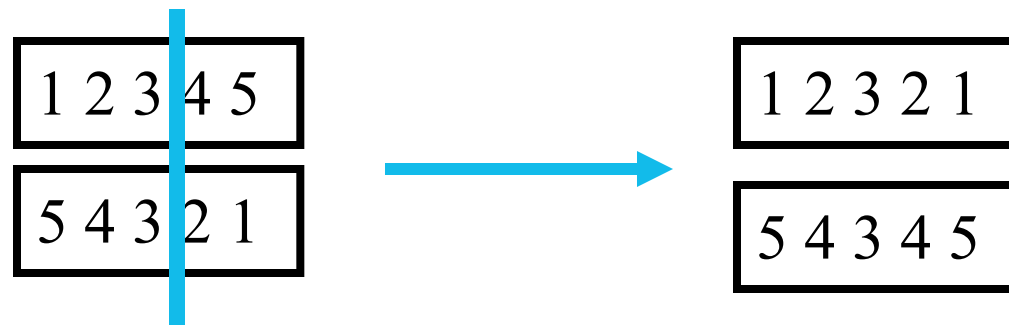


Figure from [1]

# PERMUTATIONS GAS – TSP EXAMPLE

Previously defined crossover operators will often lead to inadmissible solutions,



Specialized crossover operators have been devised.

# PERMUTATIONS GAS – TSP EXAMPLE

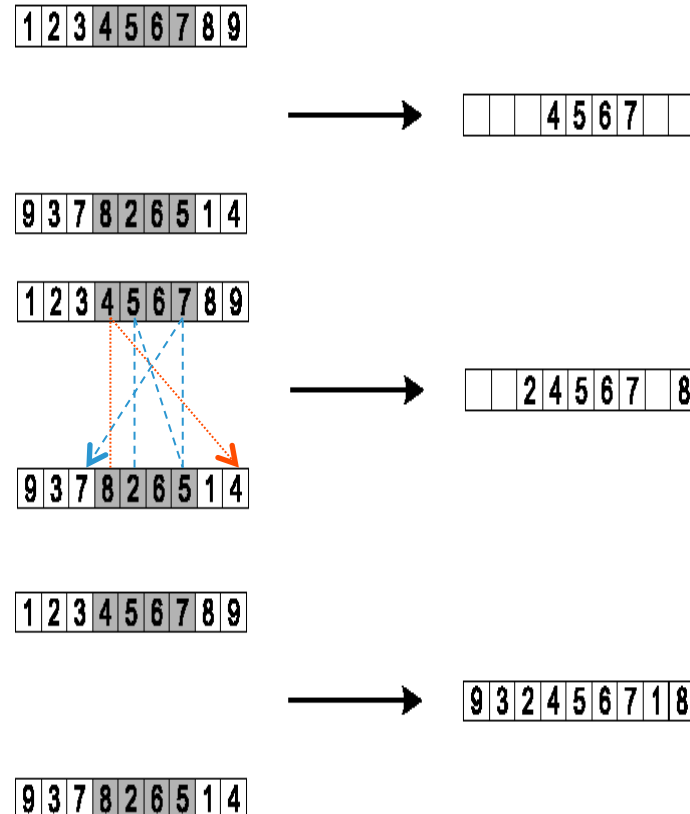
Four types of crossover operators:

- Adjacency-based:
  - **Partially Mapped Crossover (PMX),**
  - Edge crossover.
- Order-based:
  - **Order 1 crossover,**
  - **Cycle crossover.**

# PERMUTATION GA'S

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.



# PERMUTATIONS GAS – TSP EXAMPLE

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).

# PERMUTATIONS GAS – TSP EXAMPLE

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.



# PERMUTATIONS GAS – TSP EXAMPLE

Step 1

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



			4	5	6	7		
--	--	--	---	---	---	---	--	--

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

Step 2

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



		2	4	5	6	7		8
--	--	---	---	---	---	---	--	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

Step 3

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



9	3	2	4	5	6	7	1	8
---	---	---	---	---	---	---	---	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

Example from [1]

# PERMUTATIONS GAS – TSP EXAMPLE

Order 1 crossover: The 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:
  - Start right from cut point of the copied part,
  - Use the **order** of the second parent,
  - Wrap around at the end.
4. Analogous for the second child, with parent roles reversed.

# PERMUTATIONS GAS – TSP EXAMPLE

Copy randomly selected set from first parent

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



			4	5	6	7		
--	--	--	---	---	---	---	--	--

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

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

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



3	8	2	4	5	6	7	1	9
---	---	---	---	---	---	---	---	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

Example from [1]

# PERMUTATION GA'S

Cycle Crossover :Basic idea:

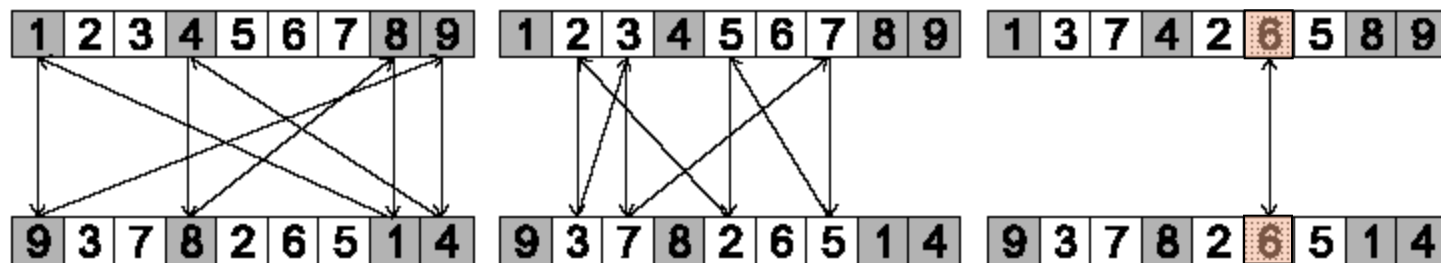
Each allele comes from one parent together with its position.

Informal procedure:

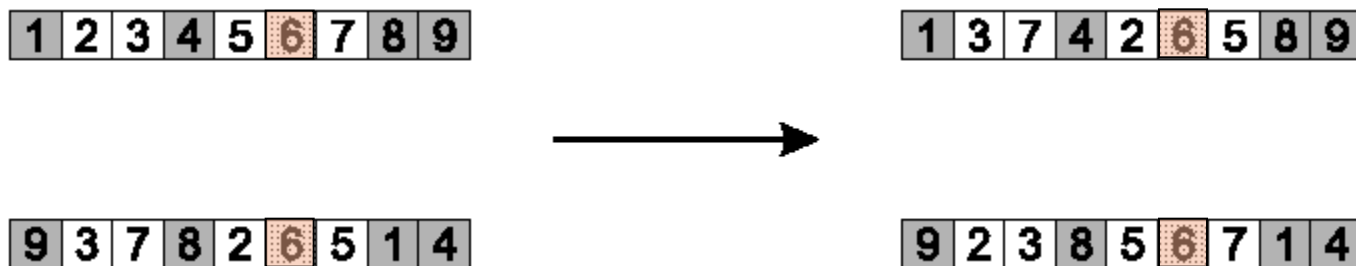
1. Make a cycle of alleles from P1 in the following way.
  - (a) Start with the first allele of P1.
  - (b) Look at the allele at the same position in P2.
  - (c) Go to the position with the same allele in P1.
  - (d) Add this allele to the cycle.
  - (e) Repeat step b through d until you arrive at the 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



# PERMUTATIONS GAS – TSP EXAMPLE

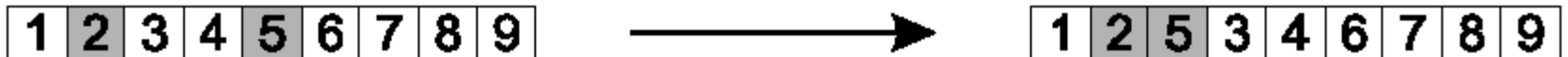
Four types of mutation operators:

- Insert mutation,
- Swap mutation,
- Inversion mutation,
- Scramble mutation.

# PERMUTATIONS GAS – TSP EXAMPLE

Insert mutation:

- Pick two genes values at random,
- Move the second to follow the first, shifting the rest along to accommodate,
- Preserves most of the order and the adjacency information,



Example from [1]

# PERMUTATIONS GAS – TSP EXAMPLE

Swap mutation:

- Pick two genes at random and swap their positions,
- Preserves most of adjacency information (4 links broken), disrupts order more.

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



1	5	3	4	2	6	7	8	9
---	---	---	---	---	---	---	---	---

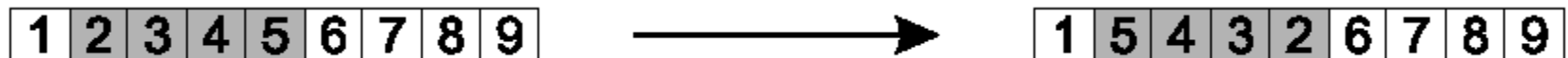
Example from [1]



# PERMUTATIONS GAS – TSP EXAMPLE

Inversion mutation:

- Pick two genes at random and then invert the substring between them,
- Preserves most adjacency information (only breaks two links) but disruptive of order information.



Example from [1]

# PERMUTATIONS GAS – TSP EXAMPLE

Scramble mutation:

- Pick two gene values at random,
- Randomly rearrange the genes in those positions (note subset does not have to be contiguous)

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



1	3	5	4	2	6	7	8	9
---	---	---	---	---	---	---	---	---

Example from [1]

# PERMUTATIONS GAS – TSP EXAMPLE

The GA was applied for the berlin52 TSP instance (52 locations in Berlin) by having:

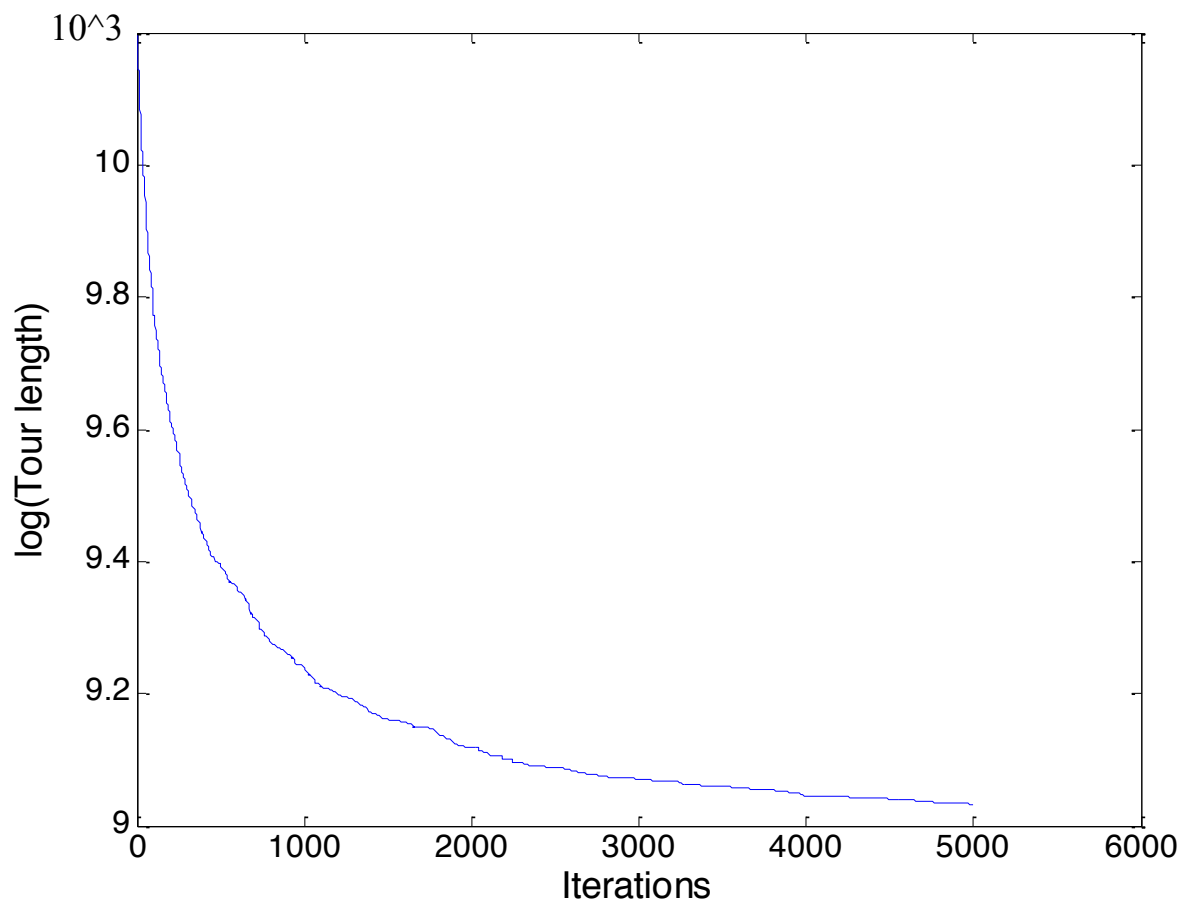
- 20 individuals,
- Using order 1 crossover producing only one child,
- Crossover probability = 0.7,
- Using swap mutation with one swap only,
- Mutation probability = 0.05,
- Using the *elitism* model while keeping only the best individual from one population to the next,
- Using different number of iterations (500, 1000, 2000 and 5000),

# PERMUTATIONS GAS – TSP EXAMPLE

Number of iterations	Tour length
500	12135
1000	10069
2000	8986.2
<b>5000</b>	<b>8730.3</b>

SA obtained a tour of **8861**

# PERMUTATIONS GAS – TSP EXAMPLE



# OUTLINE

Introduction,  
The Simple GA,  
Real-valued GAs,  
Permutations GAs,  
**Applications**,  
GAs, When are they useful?  
Adaptive GAs,  
Parallel GAs,  
References.

# GAS - APPLICATIONS

Antenna design,

Electronic circuits,

Network design,

Protein folding,

VLSI partitioning/routing/placement,

Data mining,

And many more ...

# OUTLINE

Introduction,  
The Simple GA,  
Real-valued GAs,  
Permutations GAs,  
Applications,  
GAs, When are they useful?  
Adaptive GAs,  
Parallel GAs,  
References.



# GAS – WHEN ARE THEY USEFUL?

Highly multimodal functions,

Discrete or discontinuous functions,

High-dimensionality functions, including many combinatorial ones,

Non-linear dependencies between parameters.

# DIFFICULTIES THAT MAY OCCUR WITH GA

Premature convergence

Unable to overcome deception

Need more evaluations than time permits

Bad match of representation/mutation/crossover, making operators destructive

Biased or incomplete representation

Problem too hard

(Problem too easy, makes GA *look* bad)

# OUTLINE

Introduction,

The Simple GA,

Real-valued GAs,

Permutations GAs, TSP

Applications,

GAs, When are they useful?

Adaptive GAs,

Parallel GAs,

References.

# ADAPTIVE GAS

One of the main disadvantages of GAs is the problem of *parameter tuning*,

*Parameter tuning* refers to the problem of finding a suitable values for the different algorithm parameters before the algorithm is run,

# ADAPTIVE GAS

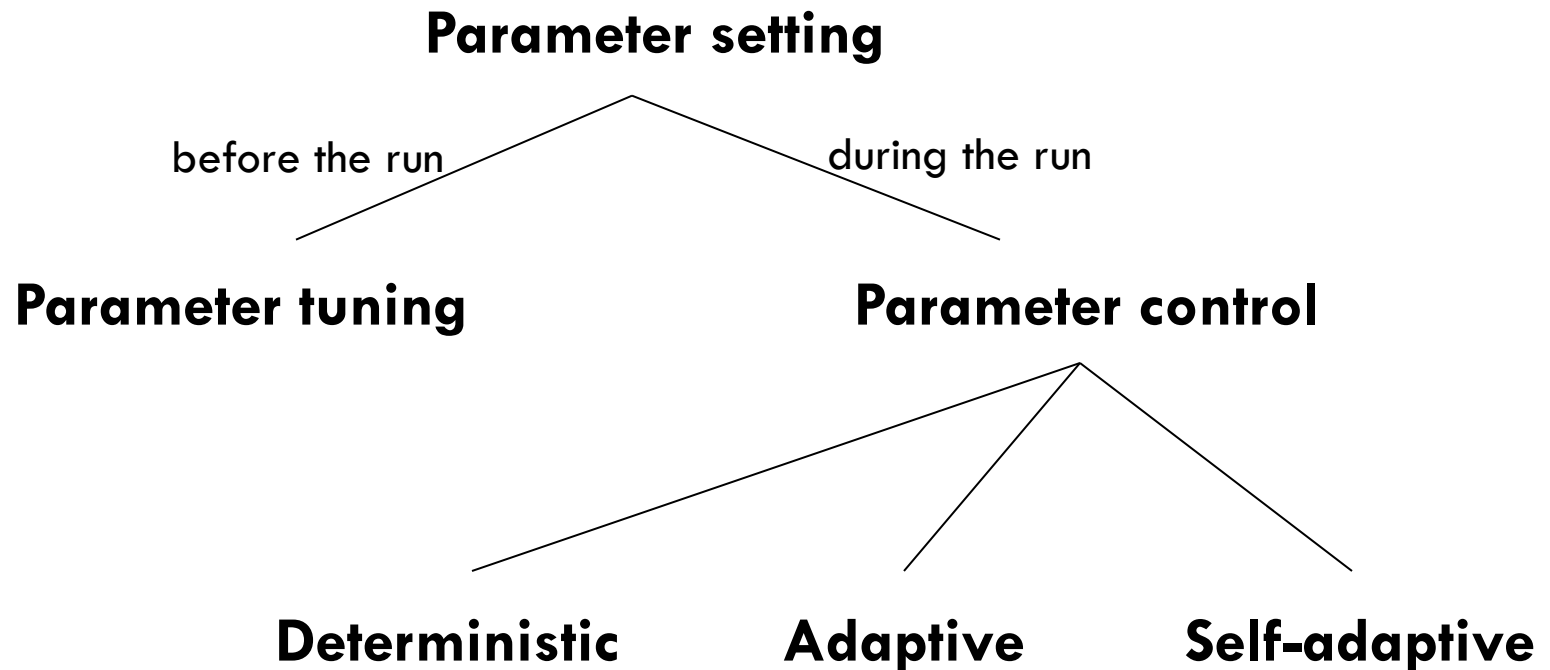
Disadvantages of parameter tuning:

- Time consuming process,
- Parameters were tuned one-by-one, which may lead to sub-optimal choices because the parameters are not independent,
- Simultaneous tuning of parameters may lead to an enormous amount of experiments,
- The best parameters settings are problem specific,
- Different values may be optimal at different stages of the search process.

# ADAPTIVE GAS

- Adaptive techniques have been proposed in GAs to set the values of the different algorithm parameters on-the-fly,
- These algorithms are more efficient and perform better on a wider range of problems.

# ADAPTIVE GAS



Classification from [2]

# ADAPTIVE GAS

**Deterministic** parameter control is characterized by replacing any parameter  $p$  with a function  $p(t)$ , where  $t$  is the generation number,

This approach does not take the progress of the search process into account (or the current state of the search).



# ADAPTIVE GAS

*Adaptive* parameter control takes feedback from the current state of the search and heuristically control the parameter values,

*Self-adaptive* approaches incorporate the parameter values into the chromosomes, in this case the parameter control process will be driven through selection, crossover and mutation.

# ADAPTIVE GAS: A MUTATION EXAMPLE

Recall the Gaussian mutation defined as:

$$x'_i = x_i + N(0, \sigma)$$

We can adaptively set the value of  $\sigma$  during the run.

# ADAPTIVE GAS: A MUTATION EXAMPLE

The *deterministic* approach could set the value of  $\sigma$  as:

$$\sigma(t) = 1 - 0.9 \frac{t}{T}$$

where  $t$  is the generation number ranging from 0 to  $T$  (maximum generation number).

# ADAPTIVE GAS: A MUTATION EXAMPLE

In this case,  $\sigma$  is changed from 1 at the beginning of the run to 0.1 at the end,

Helps in enhancing the fine tuning ability of the search towards the end,

The values of  $\sigma$  are known and determined for every generation.

# ADAPTIVE GAS: A MUTATION EXAMPLE

The *adaptive* approach incorporates feedback from the search process, a well-known method is the *Rechenberg's '1/5 success rule'*,

The rule states that the shown percentage should hold:

$$\frac{\text{\#successful mutations}}{\text{\#mutations}} = \frac{1}{5}$$

# ADAPTIVE GAS: A MUTATION EXAMPLE

A successful mutation is a mutation that produces a better individual,

The values of  $\sigma(t)$  are selected as non-deterministic based on the ratio

If the ratio is greater than  $1/5$ , the mutation step size should be increased, and if the ratio is less than  $1/5$ , the step size should be decreased.

# ADAPTIVE GAS: A MUTATION EXAMPLE

Motivation behind the rule:

- If the moves are not very successful then we should proceed in smaller steps,
- If the moves are very successful then we should try larger steps in order to improve the efficiency of the search.

# ADAPTIVE GAS: A MUTATION EXAMPLE

The update rule is applied every  $n$  generations as follows:

$$\sigma = \begin{cases} \sigma / c, & \text{if } p_s > 1/5 \\ \sigma \cdot c, & \text{if } p_s < 1/5 \\ \sigma, & \text{if } p_s = 1/5 \end{cases}$$

where  $p_s$  is the calculated percentage and  $0.82 < c < 1$ .



# ADAPTIVE GAS: A MUTATION EXAMPLE

The *self-adaptive* approach adds the mutation step size to the individual (used originally in ES):

$$\mathbf{X} = \langle \mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n, \sigma \rangle$$

The value of  $\sigma$  is evolved with the individual,

Each individual will have a different mutation step size.

# ADAPTIVE GAS: A MUTATION EXAMPLE

Mutating the individual is usually on the form:

$$\sigma' = \sigma \cdot e^{N(0, \tau_0)}$$

$$x'_i = x_i + N(0, \sigma'^i)$$

where  $\tau_0$  is a parameter of the method referred to as the *learning rate*.

# ADAPTIVE GAS: A CROSSOVER EXAMPLE

One can also adapt the crossover operator,

This operator could be adapted on three levels [3]:

- Top level, adapting the operator itself,
- Medium level, adapting the probability of crossover
- Bottom level, adapting the crossing position or the swapping probability.

# ADAPTIVE GAS: A CROSSOVER EXAMPLE

One approach in [3], is to adapt the swapping probability,

Instead of uniform crossover, where the probability at each gene is always 0.5, this probability is adapted and different across genes.

# ADAPTIVE GAS: A CROSSOVER EXAMPLE

A real-valued  $f_1(t)$  vector holding the frequencies of the 1-bits at each gene is calculated after every generation  $t$ ,

Then, a vector holding the probabilities of swapping the different genes is calculated  $p_s(t)$ ,

# ADAPTIVE GAS: A CROSSOVER EXAMPLE

The swapping probability vector is calculated as follows (referred to as the triangular rule):

$$p_s(t) = P_{\max} - 2|f(t) - 0.5|(P_{\max} - P_{\min})$$

where  $|\cdot|$  is the absolute function and  $P_{\max}$  and  $P_{\min}$  are the minimum and maximum allowable probabilities.

# ADAPTIVE GAS: A CROSSOVER EXAMPLE

Motivation behind the rule:

- As the GA progresses, the genes that are 1-intrinsic (or 0-intrinsic) will converge towards these values,
- Hence, their frequencies will approach 1 (or 0),
- Consequently, their swapping probabilities will be very low.

# OUTLINE

Introduction,  
The Simple GA,  
Real-valued GAs,  
Permutations GAs, TSP  
Applications,  
GAs, When are they useful?  
Adaptive GAs,  
Parallel GAs,  
References.



# PARALLEL GAS

Different approaches have been proposed to study parallelization in GAs [4]:

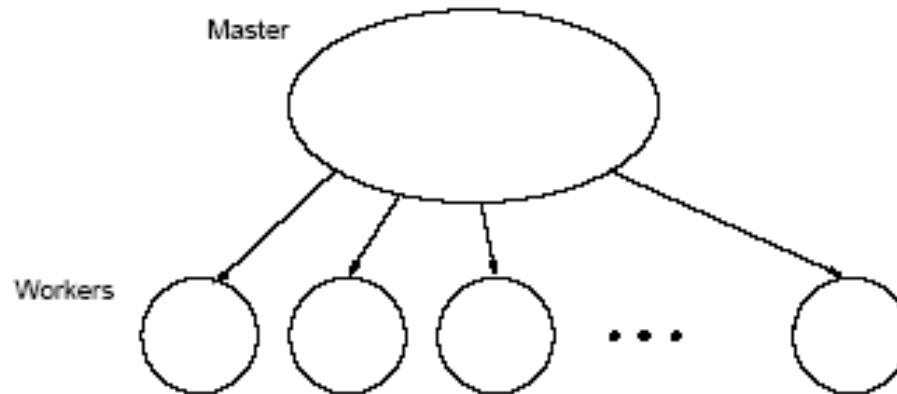
- Master-slave approaches,
- Fine-grained GAs,
- Coarse-grained GAs,
- Hierarchical parallel GAs.

# PARALLEL GAS

In the **master-slave** approach:

- Selection and mating are handled by the master processor and applied on the entire population,
- The fitness evaluation process is distributed among different slave processors
- Also known as global parallel GAs.

# MASTER-SLAVE



**Figure 1.** *A schematic of a master-slave parallel GA. The master stores the population, executes GA operations, and distributes individuals to the slaves. The slaves only evaluate the fitness of the individuals.*

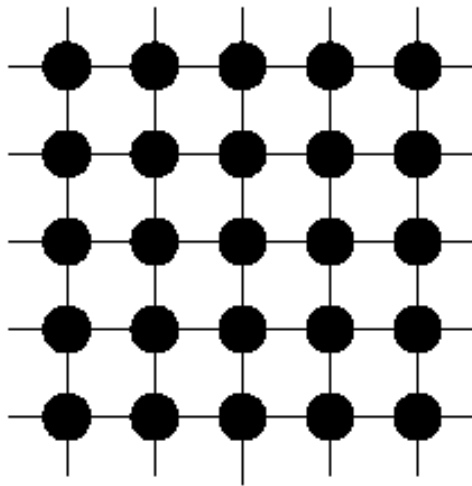
From [4]

# PARALLEL GAS

In the **fine-grained** approach:

- Selection and mating is restricted to local neighbourhoods (may overlap),
- Suitable for massively parallel computers,
- The ideal case is to have one individual per processor.

# FINE-GRAINED



**Figure 2.** *A schematic of a fine-grained parallel GA. This class of parallel GAs has one spatially-distributed population, and it can be implemented very efficiently on massively parallel computers.*

From [4]

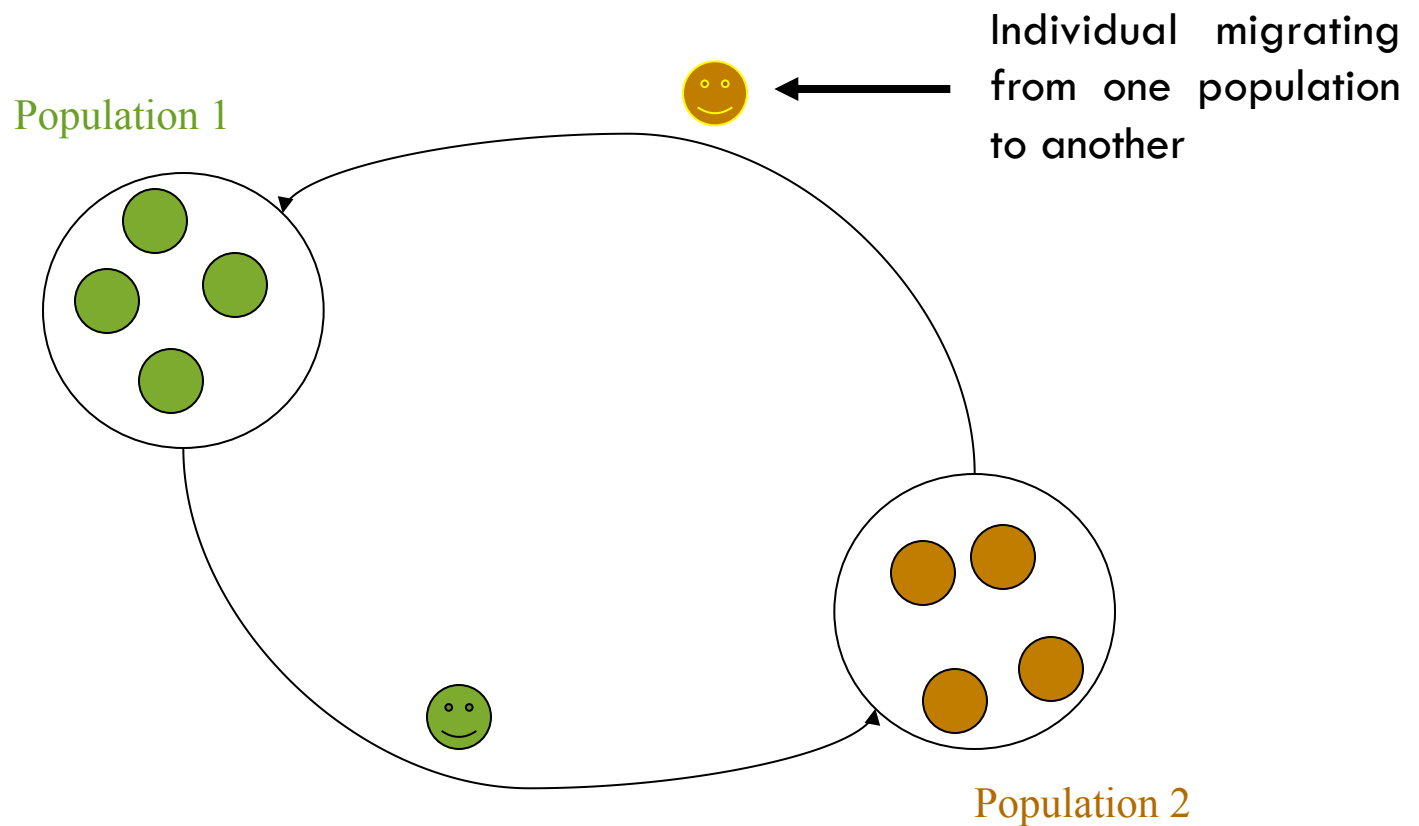
# PARALLEL GAS

In the *coarse-grained* approach:

- Multiple populations evolving in parallel,
- Selection and mating is restricted within a population,
- The populations exchange individuals every now and then,
- Also referred to as multiple-population GAs, multiple-deme GAs, distributed GAs or “island” parallel GAs.

deme: local population

# PARALLEL GAS – MULTIPLE-DEME



# MULTIPLE-DEME - FACTORS

There are a lot of different issues to consider in multiple-deme GAs:

- Number and sizes of populations,
- Different topologies,
- Different migration policies,
- Different migration frequencies,
- Different migration rates.



# MULTIPLE-DEME - TOPOLOGY

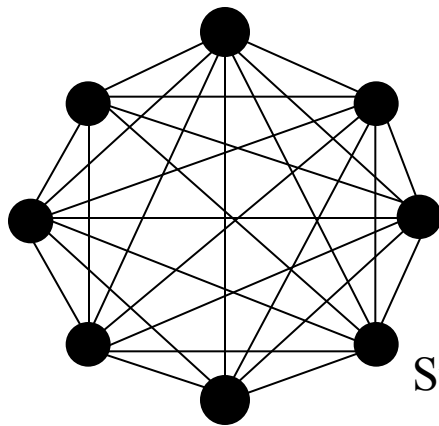
The *topology* factor is important when evolving more than two populations,

It states which populations are connected to each other,

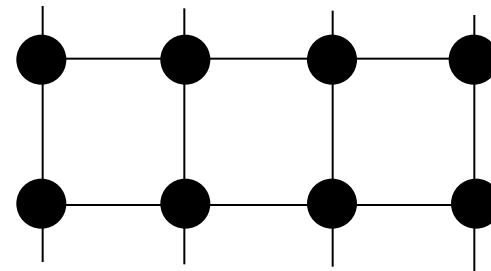
Individuals migrate between connected populations only,

The communication topology also affects the cost of migration.

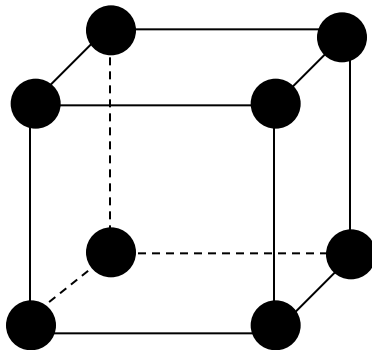
# MULTIPLE-DEME - TOPOLOGY



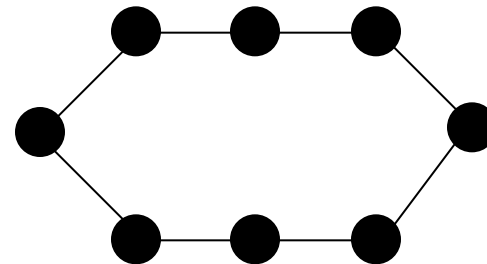
Star topology



Grid topology



Hyper-cube topology



Ring topology

# MULTIPLE-DEME - TOPOLOGY

In dense topologies, good solutions spread faster and quickly take over the populations,

In sparse topologies:

- Solutions spread slower causing the demes to be more isolated,
- Allow different solutions to appear,
- These solutions may come together towards the end and recombine to produce even better individuals.

*Static vs. Dynamic* topologies.

# MULTIPLE-DEME – MIGRATION POLICY

The *migration policy* dictates how the migrating individuals are being selected and how are they being handled at the receiving population,

Four different exchange approaches could be adopted:

- Best-Worst,
- Best-Random,
- Random-Random,
- Random-Worst.

# MULTIPLE-DEME – MIGRATION FREQUENCY

The migration frequency controls when the communication is being carried out,

Two types of communication occur:

- Synchronous communication, migration happens every predetermined number of generations,
- Asynchronous communication, migration happens when a certain event occur (e.g. populations have converged).

# MULTIPLE-DEME – MIGRATION FREQUENCY

Assuming synchronous communication, an interesting question is “When is the right time to migrate?”,

Communication in the early stages may lead the populations to sub-optimal solutions as well as causing high communication costs.

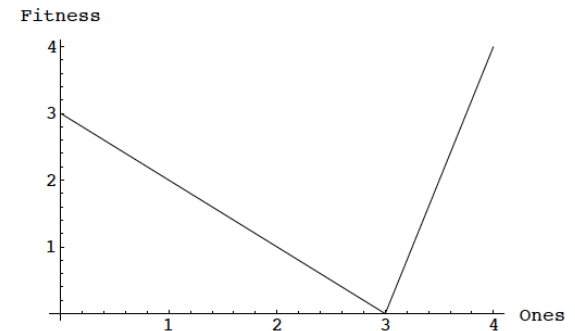
# MULTIPLE-DEME – MIGRATION RATE

The *migration rate* is the number of individuals that are being migrated from one population to another at every communication step,

Low migration rates may cause the demes to behave independently, the migrants don't have a significant effect,

High migration rates may cause a fast convergence to sub-optimal solutions.

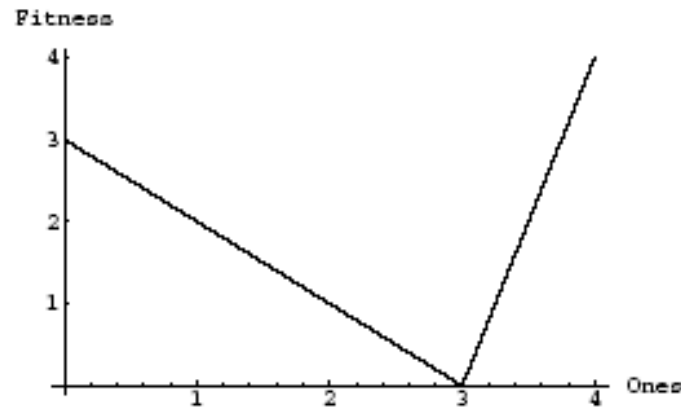
# UNITY TRAP FUNCTIONS



The first deceptive test function is based on the 4-bit trap function depicted in Figure 8, which was also used by Goldberg et al. (1992) in their study of population sizing. As in the one-max, the value of this function depends on the number of bits set to one, but in this case the fitness increases with more bits set to zero until it reaches a local (deceptive) optimum. The global maximum of the function occurs precisely at the opposite extreme where all four bits are set to one, so an algorithm cannot use any partial information to find it. The signal difference  $d$  (the difference between the global and the deceptive maxima) is 1 and the fitness variance ( $\sigma_{bb}^2$ ) is 1.215. The test function is formed by concatenating  $m = 20$  copies of the trap function for a total string length of 80 bits. The probability of making the right decision between two individuals with the best and the second best BBs is  $p = 0.5585$ . The experiments with this function use two-point crossover to avoid the excessive disruption that uniform crossover would cause on the longer BBs. The crossover probability is set to 1.0 and, as before, there is no mutation.

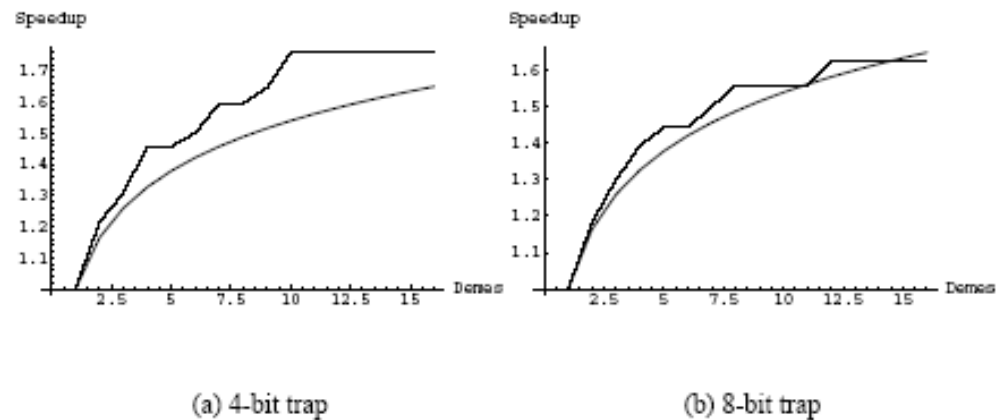


# TEST FUNCTION



**Figure 7.** *A deceptive 4-bit trap function of unity. The horizontal axis is the number of bits set to 1 and the vertical axis is the fitness value. The difficulty of this function can be varied easily by using more bits in the basin of the deceptive optimum, or by reducing the fitness difference between the global and deceptive maxima. The first test problem was constructed by concatenating 20 copies of this function and the second test problem is formed with 20 copies of a similar deceptive function of 8 bits.*

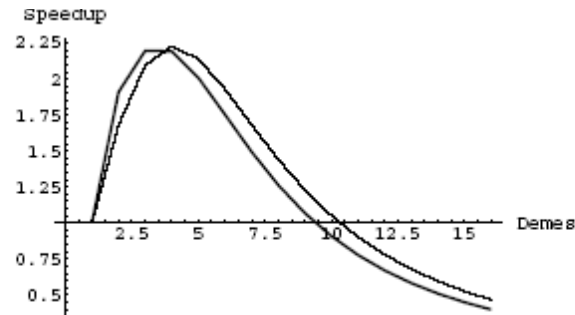
# ISOLATED DEMES



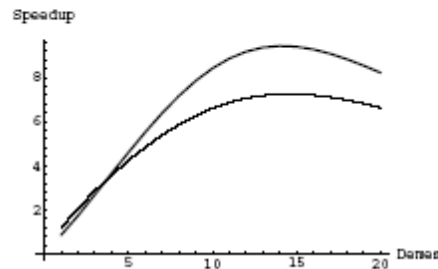
**Figure 8.** Projected and experimental speed-ups for test functions with 20 copies of 4-bit and 8-bit trap functions using from 1 to 16 isolated demes. The thin lines show the experimental results and the thick lines are the theoretical predictions. The quality demanded was to find at least 16 copies correct.

From [4]

# FULLY CONNECTED DEMES



(a) 4-bit trap



(b) 8-bit trap

From [4]

**Figure 9.** Projected and experimental speed-ups for test functions with 20 copies of 4-bit and 8-bit trap functions using from 1 to 16 fully connected demes. The quality requirement was to find at least 16 correct copies.

# PARALLEL GAS

Another form of parallelization is what is known as *Cooperative GAs*,

Having multiple populations where the fitness of any individual in any population depends on the individuals of the other populations,

Proposed by Potter et. al. in 1994 [5].

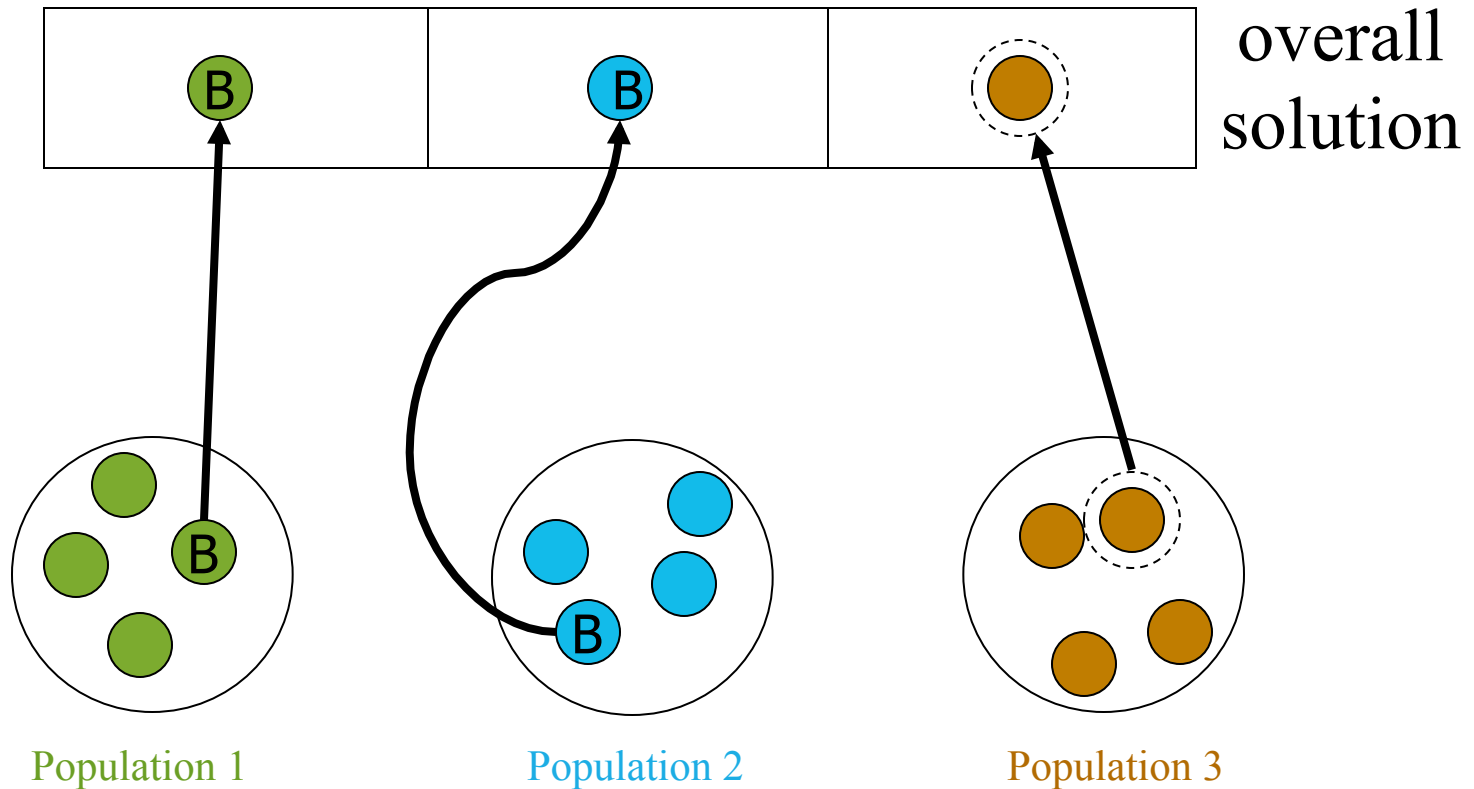
# PARALLEL GAS

The general idea is to have different populations optimizing different variables of the problem (different dimensions of the solution),

The fitness of any individual is determined by its value and the value of the best individuals in all the other populations,

Performs best If the problem variables are independent.

# PARALLEL GAS



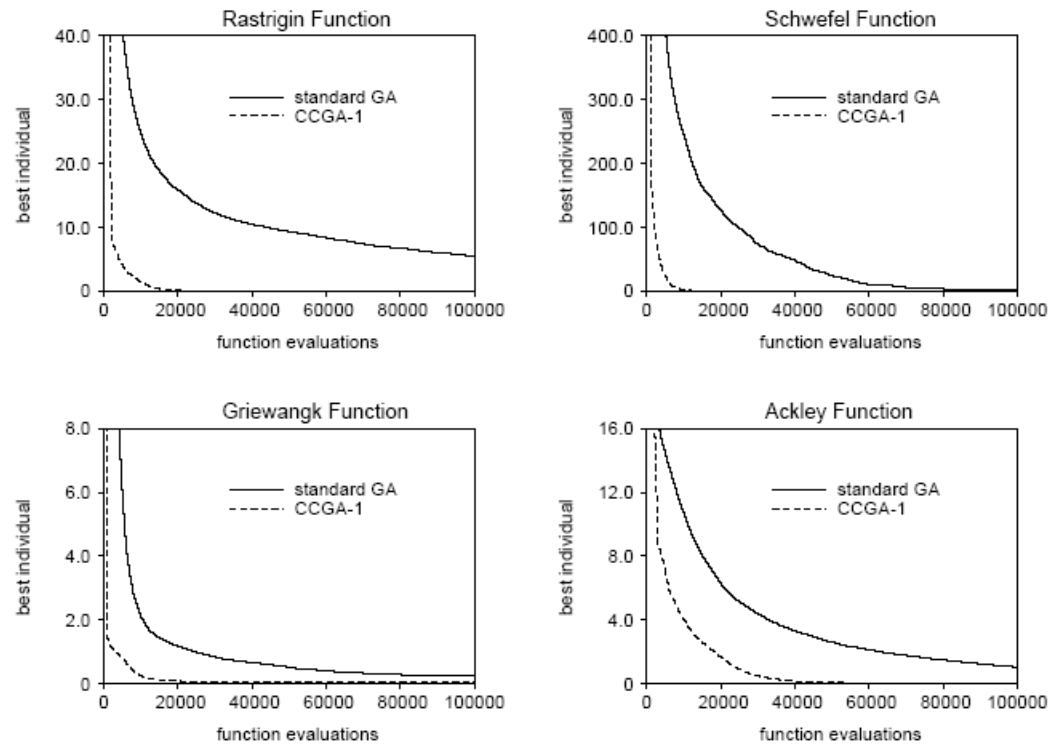


Figure 3: Comparisons of standard GA and CCGA-1 performance

From [5]

# REFERENCES

1. A. E. Eiben and J. E. Smith. "Introduction to Evolutionary Computing". Springer, Natural Computing Series, 1st edition, 2003, ISBN: 3-540-40184-9.
2. A. E. Eiben, R. Hinterding and Z. Michalewicz. "Parameter Control in Evolutionary Algorithms". IEEE Transactions on Evolutionary Computing, vol. 3, issue 2, pp. 124-141, 1999.
3. S. Yang. "Adaptive Non-uniform Crossover Based on Statistics for Genetic Algorithms". Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, pp. 650-657, 2002.
4. E. Cantu-Paz. "A Survey of Parallel Genetic Algorithms". Calculateurs Paralleles, Reseaux et Systems Repartis, Vol. 10, No. 2, pp. 141-171, 1998.
5. M. Potter and K. de Jong. "A Cooperative Coevolutionary Approach to Function Optimization". Proceedings of the third Parallel Problem Solving from Nature, Springer, pp. 249-257, 1994.