LAU
الجامعة اللبنانية الأميركية
Lebanese American University

CSC 447: Parallel Programming for Multi-Core and Cluster Systems

Evolutionary Algorithms

Instructor: Haidar M. Harmanani

Spring 2017

Darwinian Evolution Theory: Survival of the fittest

- All environments have finite resources
- Lifeforms have basic instinct/ lifecycles geared towards reproduction
- Therefore some kind of selection is inevitable
- Those individuals that compete for the resources most effectively have increased chance of reproduction

Darwinian Evolution Theory: Diversity drives change

- Phenotypic traits:
  - Behaviour / physical differences that affect response to environment
  - Partly determined by inheritance, partly by factors during development
  - Unique to each individual, partly as a result of random changes
- If phenotypic traits:
  - Lead to higher chances of reproduction
  - Can be inherited
- then they will tend to increase in subsequent generations,
- leading to new combinations of traits …

Darwinian Evolution: Summary

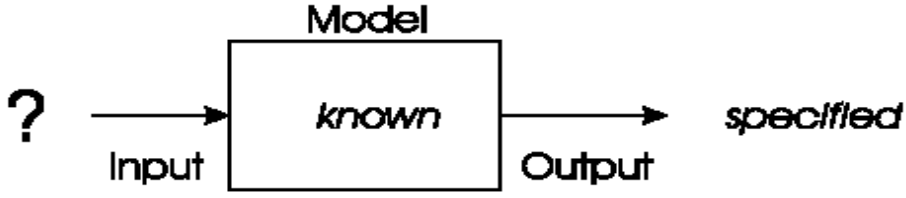
- Population consists of diverse set of individuals
- Combinations of traits that are better adapted tend to increase representation in population
  - Individuals are "units of selection"
- Variations occur through random changes yielding constant source of diversity, coupled with selection means that:
  - Population is the "unit of evolution"
- Note the absence of "guiding force"

Genes and the Genome

- Genes are encoded in strands of DNA called chromosomes
- In most cells, there are two copies of each chromosome (diploidy)
- The complete genetic material in an individual's genotype is called the Genome
- Within a species, most of the genetic material is the same

Problem type 1 : Optimisation

- We have a model of our system and seek inputs that give us a specified goal
- e.g.
  - time tables for university, call center, or hospital
  - design specifications, etc etc

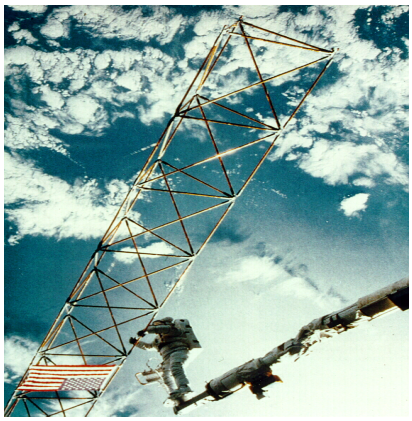


Optimization example 1: University timetabling

- Enormously big search space
- Timetables must be good
- "Good" is defined by a number of competing criteria
- Timetables must be feasible
- Vast majority of search space is infeasible

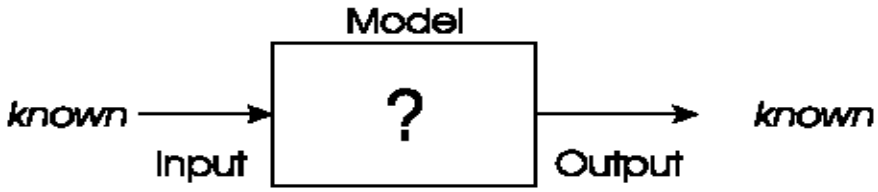
Optimization Example 2: Satellite Structure

- Optimised satellite designs for NASA to maximize vibration isolation
- Evolving: design structures
- Fitness: vibration resistance
- Evolutionary "creativity"



Problem types 2: Modelling

- We have corresponding sets of inputs & outputs and seek model that delivers correct output for every known input



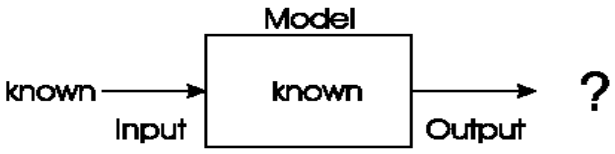
  - Evolutionary machine learning

Modelling example: loan applicant creditibility

- British bank evolved creditability model to predict loan paying behavior of new applicants
- Evolving: prediction models
- Fitness: model accuracy on
- historical data

Problem type 3: Simulation

- We have a given model and wish to know the outputs that arise under different input conditions

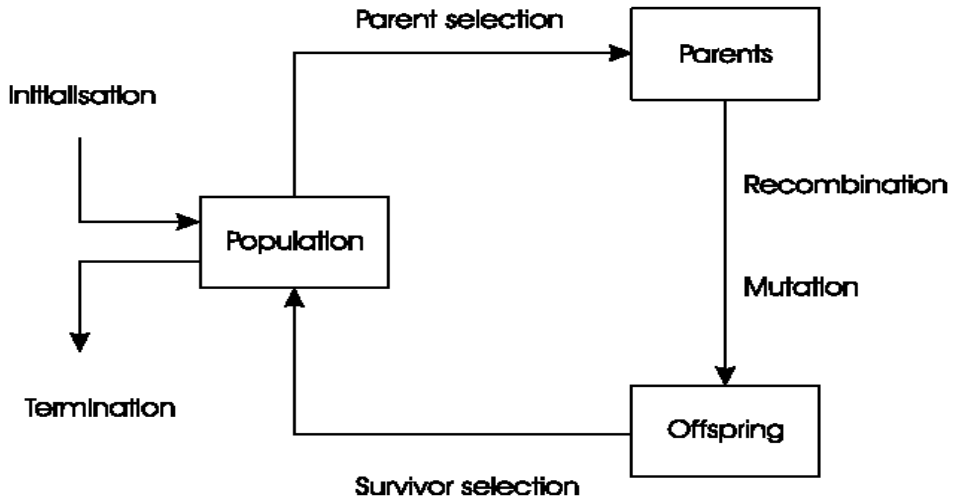


- Often used to answer "what-if" questions in evolving dynamic environments
- e.g. Evolutionary economics, Artificial Life

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

What is an Evolutionary Algorithm?

General Scheme of EAs



What are the different types of EAs

Historically different flavours of EAs have been associated with different representations

- Binary strings : Genetic Algorithms
- Real-valued vectors : Evolution Strategies
- Finite state Machines: Evolutionary Programming
- LISP trees: Genetic Programming

These differences are largely irrelevant, best strategy

- choose representation to suit problem
- choose variation operators to suit representation

Selection operators only use fitness and so are independent of representation

Representations

Candidate solutions (individuals) exist in *phenotype* space

They are encoded in chromosomes, which exist in *genotype* space

- Encoding : phenotype=> genotype (not necessarily one to one)
- Decoding : genotype=> phenotype (must be one to one)

Chromosomes contain genes, which are in (usually fixed) positions called loci (sing. locus) and have a value (allele)

In order to find the global optimum, every feasible solution must be represented in genotype space

Evaluation (Fitness) Function

Represents the requirements that the population should adapt to

a.k.a. *quality* function or *objective* function

Assigns a single real-valued fitness to each phenotype which forms the basis for selection

- So the more discrimination (different values) the better

Typically we talk about fitness being maximised

- Some problems may be best posed as minimisation problems, but conversion is trivial

Population

Holds (representations of) possible solutions

Usually has a fixed size and is a *multiset* of genotypes

Some sophisticated EAs also assert a spatial structure on the population e.g., a grid.

Selection operators usually take whole population into account i.e., reproductive probabilities are *relative* to *current* generation

Diversity of a population refers to the number of different fitnesses / phenotypes / genotypes present (note not the same thing)

Parent Selection Mechanism

- Assigns variable probabilities of individuals acting as parents depending on their fitnesses
- Usually probabilistic
  - high quality solutions more likely to become parents than low quality
  - but not guaranteed
  - even worst in current population usually has non-zero probability of becoming a parent
- This *stochastic* nature can aid escape from local optima

Variation Operators

Role is to generate new candidate solutions

Usually divided into two types according to their arity (number of inputs):

- Arity 1 : mutation operators
- Arity >1 : Recombination operators
- Arity = 2 typically called crossover

There has been much debate about relative importance of recombination and mutation

- Nowadays most EAs use both
- Choice of particular variation operators is representation dependant

Mutation

- Acts on one genotype and delivers another
- Element of randomness is essential and differentiates it from other unary heuristic operators
- Importance ascribed depends on representation and dialect:
  - Binary GAs – background operator responsible for preserving and introducing diversity
  - EP for FSM's/ continuous variables – only search operator
  - GP – hardly used
- May guarantee connectedness of search space and hence convergence proofs

Recombination

- Merges information from parents into offspring
- Choice of what information to merge is stochastic
- Most offspring may be worse, or the same as the parents
- Hope is that some are better by combining elements of genotypes that lead to good traits
- Principle has been used for millennia by breeders of plants and livestock

Survivor Selection

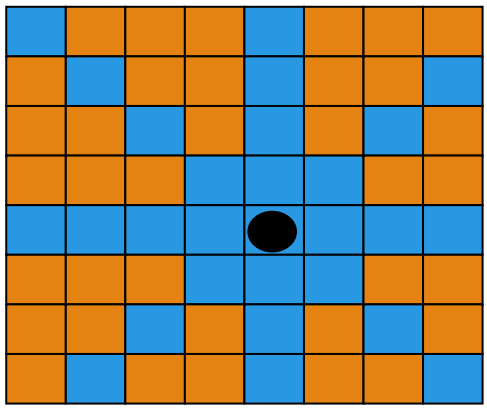
- a.k.a. *replacement*
- Most EAs use fixed population size so need a way of going from (parents + offspring) to next generation
- Often deterministic
  - Fitness based : e.g., rank parents+offspring and take best
  - Age based: make as many offspring as parents and delete all parents
- Sometimes do combination (elitism)

Initialisation / Termination

- Initialisation usually done at random,
  - Need to ensure even spread and mixture of possible allele values
  - Can include existing solutions, or use problem-specific heuristics, to "seed" the population

- Termination condition checked every generation
  - Reaching some (known/hoped for) fitness
  - Reaching some maximum allowed number of generations
  - Reaching some minimum level of diversity
  - Reaching some specified number of generations without fitness improvement

Example: the 8 queens problem

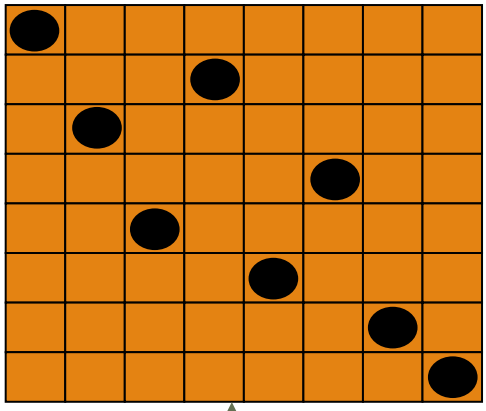


Place 8 queens on an 8x8 chessboard in such a way that they cannot check each other

The 8 queens problem: representation

Phenotype:
a board configuration

Genotype:
a permutation of
the numbers 1 - 8



8 Queens Problem: Fitness evaluation

- Penalty of one queen:
  - The number of queens she can check.
- Penalty of a configuration:
  - The sum of the penalties of all queens.
- Note: penalty is to be minimized
- Fitness of a configuration
  - inverse penalty to be maximized

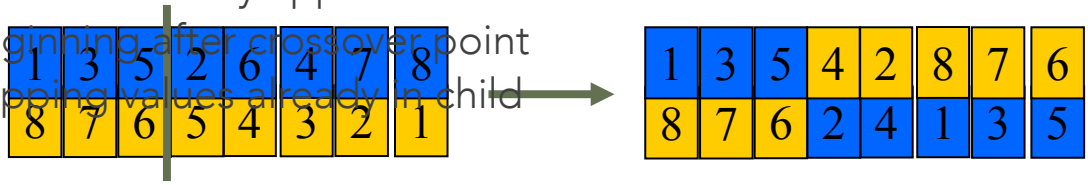
The 8 queens problem: Mutation

Small variation in one permutation, e.g.:

- swapping values of two randomly chosen positions,

The 8 queens problem: Recombination

- Combining two permutations into two new permutations:
  - choose random crossover point
  - copy first parts into children
- create second part by inserting values from other parent:
  - in the order they appear there
  - beginning after crossover point
  - skipping values already in child



The 8 queens problem: Selection

- Parent selection:
  - Pick 5 parents and take best two to undergo crossover
- Survivor selection (replacement)
  - When inserting a new child into the population, choose an existing member to replace by:
  - sorting the whole population by decreasing fitness
  - enumerating this list from high to low
  - replacing the first with a fitness lower than the given child

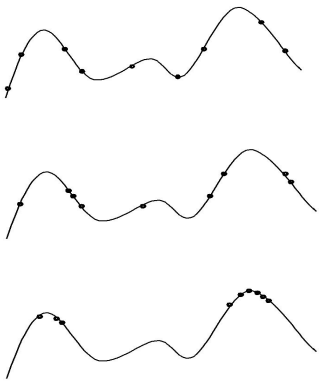
8 Queens Problem: summary

Representation	Permutations
Recombination	"Cut-and-crossfill" crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of Offspring	2
Initialisation	Random
Termination condition	Solution or 10,000 fitness evaluation

*Note that is **only one possible** set of choices of operators and parameters*

Typical behaviour of an EA

- Phases in optimising on a 1-dimensional fitness landscape

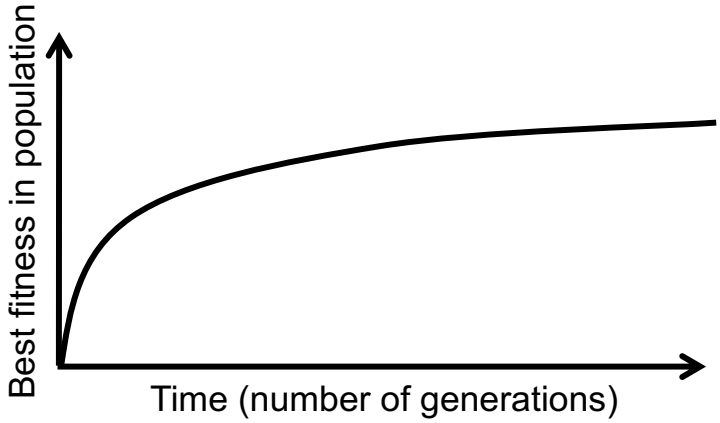


Early phase:
quasi-random population distribution

Mid-phase:
population arranged around/on hills

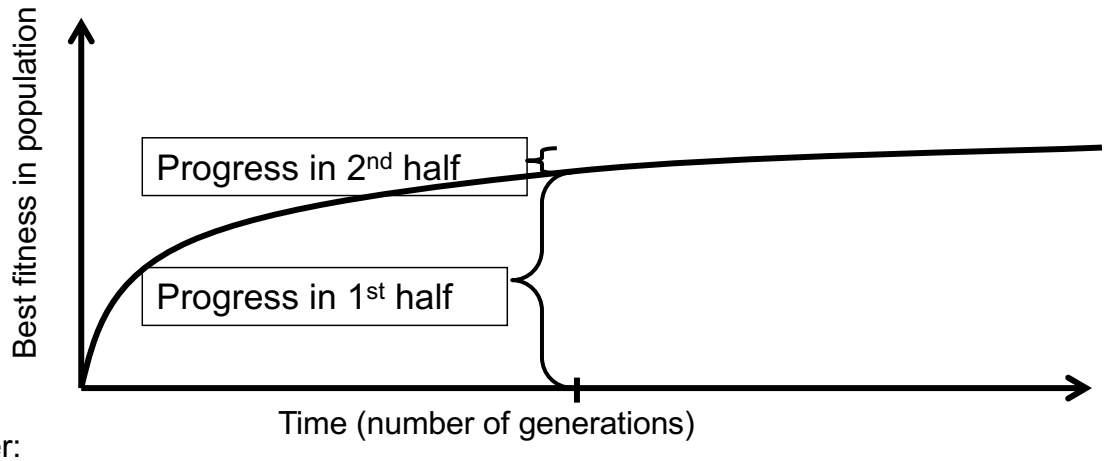
Late phase:
population concentrated on high hills

Typical run: progression of fitness



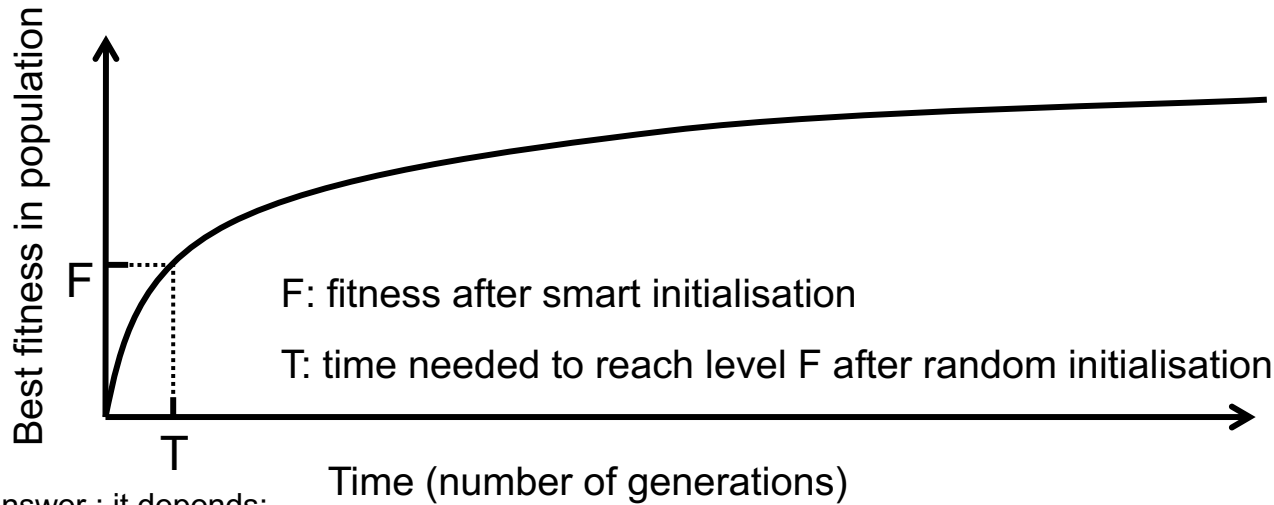
Typical run of an EA shows so-called “anytime behavior”

Are long runs beneficial?



- Answer:
  - it depends how much you want the last bit of progress
  - it may be better to do more shorter runs

Is it worth expending effort on smart initialisation?



- Answer : it depends:
  - possibly, if good solutions/methods exist.
  - care is needed, see chapter on hybridisation

Evolutionary Algorithms in Context

- There are many views on the use of EAs as robust problem solving tools
- For most problems a problem-specific tool may:
  - perform better than a generic search algorithm on most instances,
  - have limited utility,
  - not do well on all instances
- Goal is to provide robust tools that provide:
  - evenly good performance
  - over a range of problems and instances

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

Genetic Algorithms

GA Quick Overview

- Developed: USA in the 1970's
- Early names: J. Holland, K. DeJong, D. Goldberg
- Typically applied to:
  - discrete optimization
- Attributed features:
  - not too fast
  - good heuristic for combinatorial problems
- Special Features:
  - Traditionally emphasizes combining information from good parents (crossover)
  - many variants, e.g., reproduction models, operators

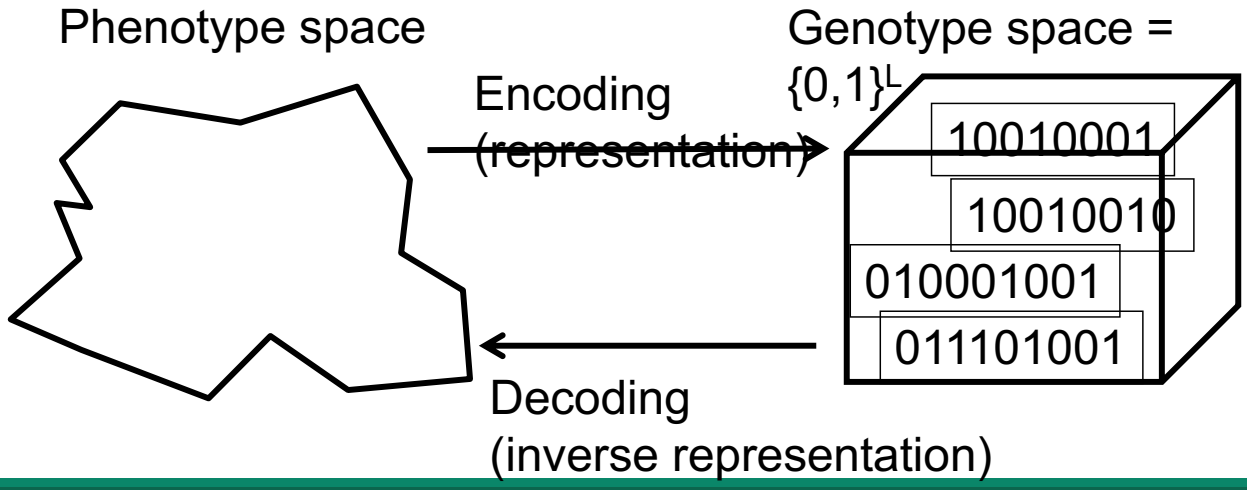
Genetic algorithms

- Holland' s original GA is now known as the simple genetic algorithm (SGA)
- Other GAs use different:
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

SGA technical summary tableau

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

Representation

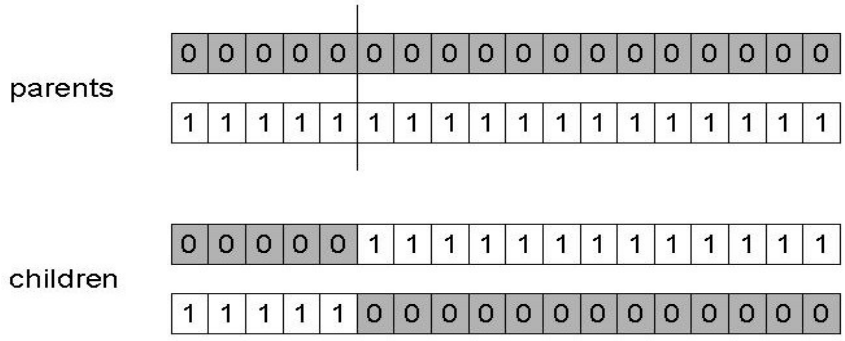


SGA reproduction cycle

1. Select parents for the mating pool
   (size of mating pool = population size)
2. Shuffle the mating pool
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

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)

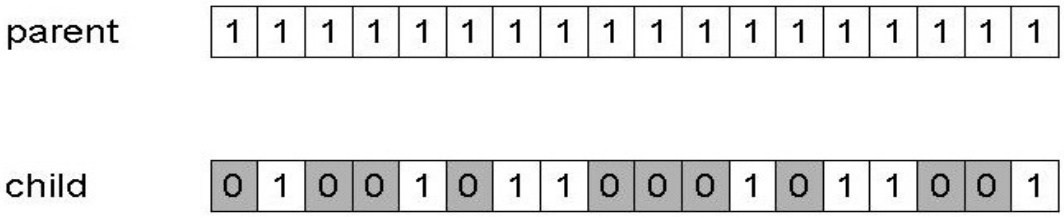


SGA operators: mutation

Alter each gene independently with a probability p_m

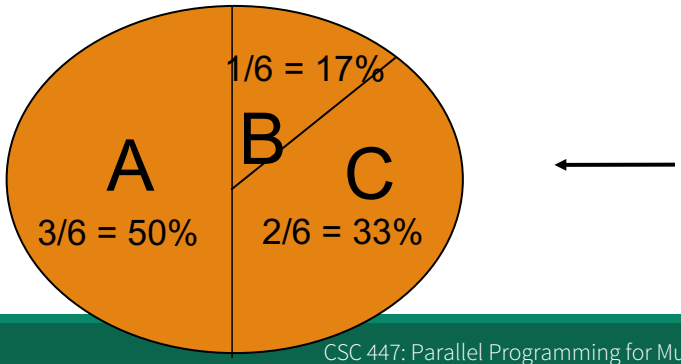
p_m is called the mutation rate

- 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 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,…,31}
- GA approach:
  - Representation: binary code, e.g. 01101 ↔ 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: mutation

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

X² example: mutation

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

X² example: mutation

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

The simple GA

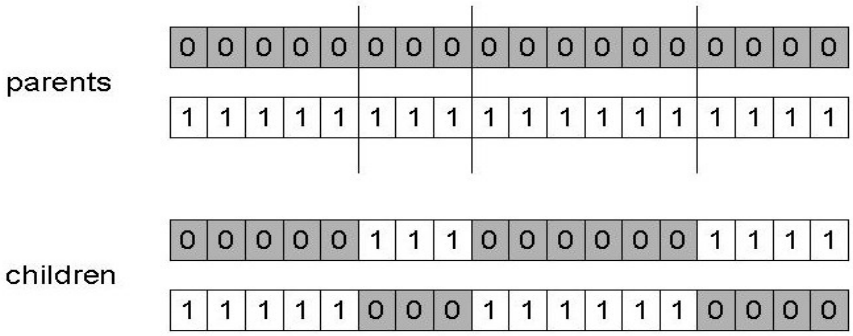
- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
  - Representation is too restrictive
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations with close fitness values
  - Generational population model (step 5 in SGA repr. cycle) can be improved with explicit survivor selection

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
  - 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

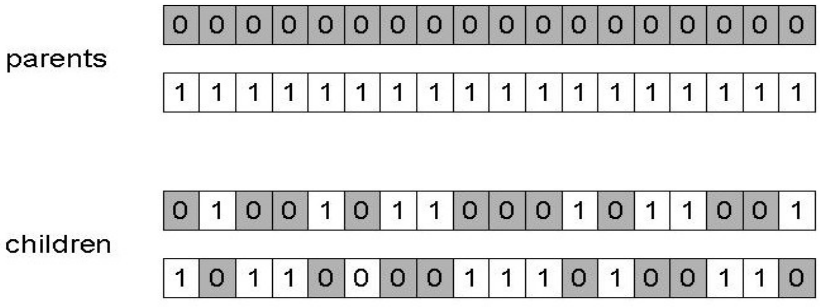
n-point crossover

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



Uniform crossover

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



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)

- 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
- There is co-operation AND competition between them
- 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

Crossover OR mutation? (cont'd)

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0's on first bit in the population, ?% after performing n crossovers)
- To hit the optimum you often need a 'lucky' mutation

Other representations

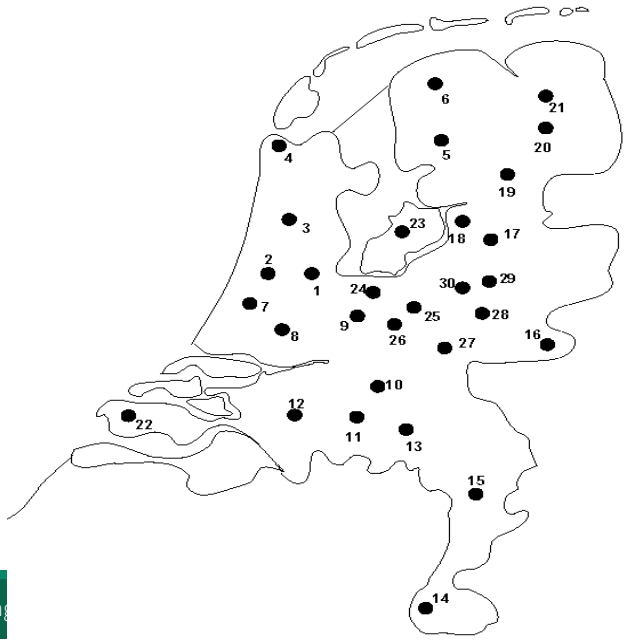
- Gray coding of integers (still binary chromosomes)
  - Gray coding is a mapping that means that small changes in the genotype cause small changes in the phenotype (unlike binary coding). "Smoother" genotype-phenotype mapping makes life easier for the GA
- Nowadays it is generally accepted that it is better to encode numerical variables directly as
- Integers
- Floating point variables

Permutation Representations

- Ordering/sequencing problems form a special type
- Task is (or can be solved by) arranging some objects in a certain order
  - Example: sort algorithm: 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

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 30 cities there are 30! ≈ 1032 possible tours



Permutation representation: TSP example

- Normal mutation operators lead to inadmissible solutions
- Usually special mutation and crossover representations are designed in these cases

Population Models

- SGA uses a Generational model:
  - 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:
  - one offspring is generated per generation,
  - one member of population replaced,
- Generation Gap
  - the proportion of the population replaced
  - 1.0 for GGA, 1/pop_size for SSGA

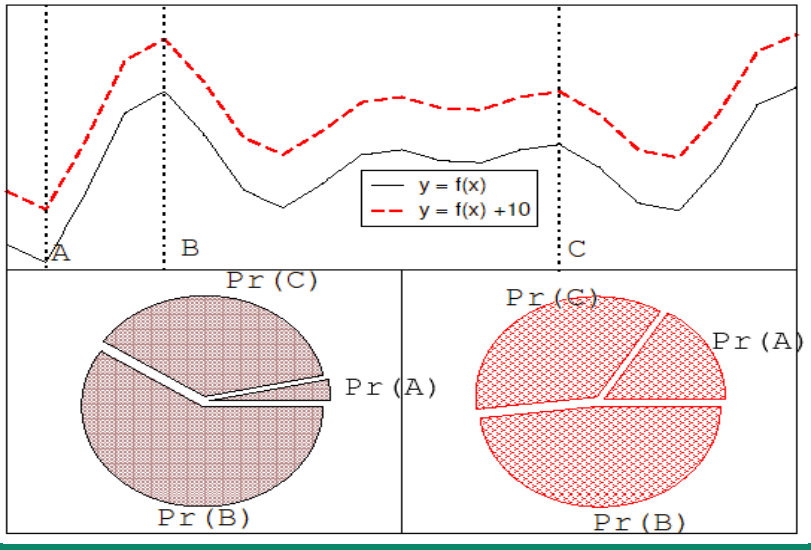
Fitness Based Competition

- 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 individual
  - i.e. they are representation-independent
- Distinction between selection
  - operators: define selection probabilities
  - algorithms: define how probabilities are implemented

Fitness-Proportionate Selection

- 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
- Scaling can fix last two problems
  - Windowing: $f'(i) = f(i) - \beta^t$
    - where β is worst fitness in this (last n) generations
  - Sigma Scaling: $f'(i) = max(f(i) - (\langle f \rangle - c \bullet \sigma_f), 0.0)$
    - where c is a constant, usually 2.0

Function transposition for FPS



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 and worst rank 1
- This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

Linear Ranking

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

Parameterised by factor *s:* $1.0 < s \leq 2.0$

- measures advantage of best individual
- in GGA this is the number of children allotted to it

Simple 3 member example

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

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

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
- Informal Procedure:
  - Pick *k* members at random then select the best of these
  - Repeat to select more individuals

Tournament Selection 2

- Probability of selecting *i* will depend on:
  - Rank of *i*
  - Size of sample *k*
    - higher *k* increases selection pressure
  - Whether contestants are picked with replacement
    - Picking without replacement increases selection pressure
  - Whether fittest contestant always wins (deterministic) or this happens with probability *p*
- For $k = 2$, time for fittest individual to take over population is the same as linear ranking with $s = 2 \bullet p$

Survivor Selection

- Most of methods above used for parent selection
- Survivor selection can be divided into two approaches:
  - Age-Based Selection
    - e.g. SGA
    - In SSGA can implement as "delete-random" (not recommended) or as first-in-first-out (a.k.a. delete-oldest)
  - Fitness-Based Selection
    - Using one of the methods above or

Two Special Cases

- Elitism
  - Widely used in both population models (GGA, SSGA)
  - Always keep at least one copy of the fittest solution so far
- GENITOR: a.k.a. "delete-worst"
  - 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

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

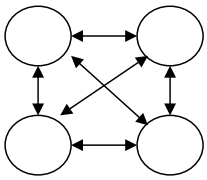
Parallel GA Techniques

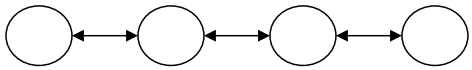
Parallel Genetic Algorithm

- Each processor operates independently on subpopulation
- Periodically (every *k* generations) shares best individuals with other processors through 'migration'

Migration

- *Island model*
  - individuals can be migrated to any other subpopulation
- *Stepping stone model*
  - individuals can be migrated only to neighboring subpopulations
  - less communication

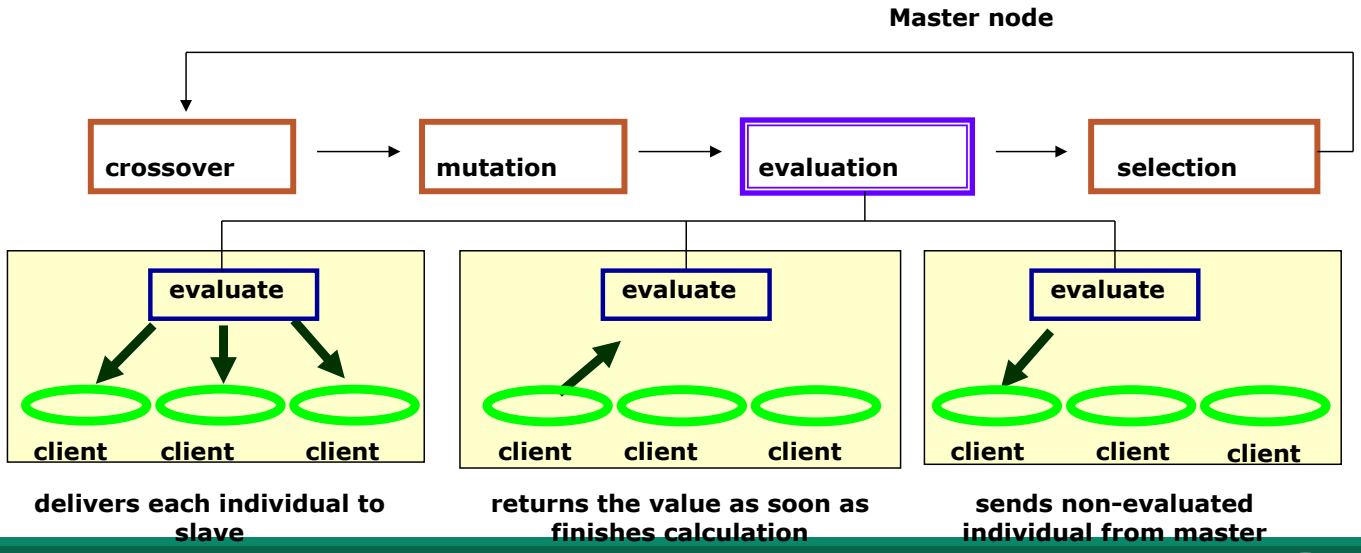




Problems

- Increase the possibility that each subpopulation will converge to its own local optimum and perhaps completely miss the global solution
- Increase the number of generations to converge

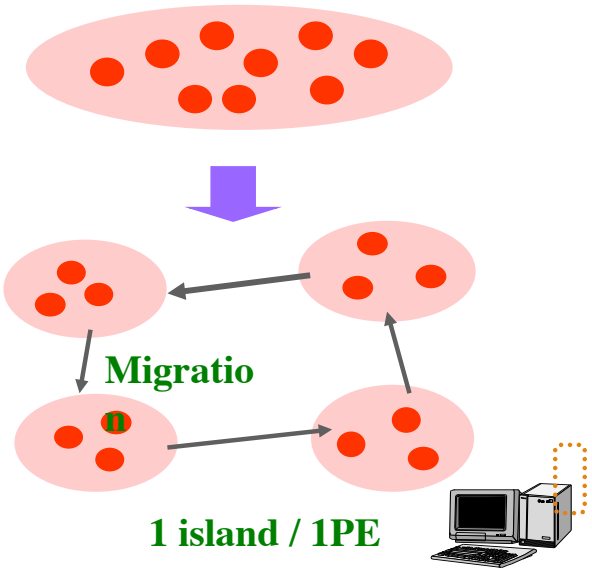
Master slave model



DGA model (island, coarse grained)

Distributed GAs
- A population is divided into subpopulations (islands)
- SGA is performed on each subpopulation

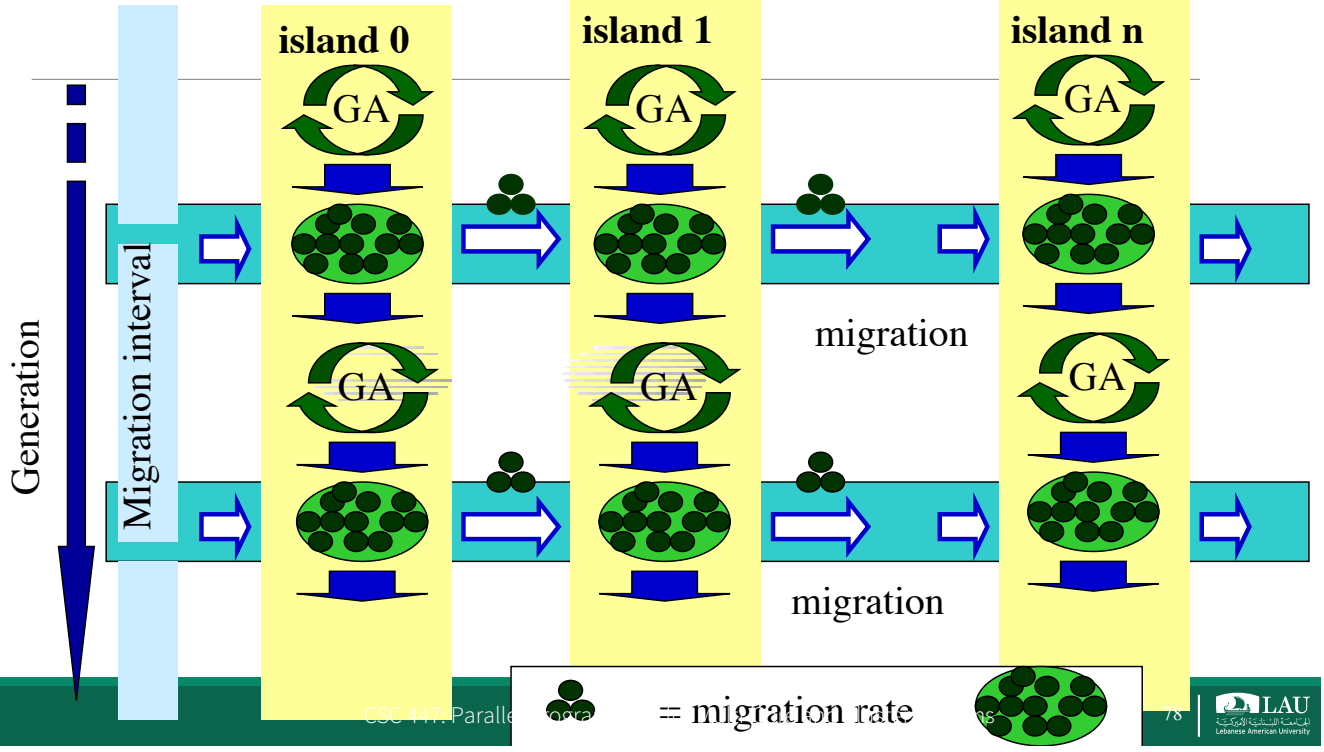
Migration is performed for some generations
- Exchange of individuals



Migration Model

- Island model
  - individuals can be migrated to any other subpopulation
- Stepping stone model
  - individuals can be migrated only to neighboring subpopulations
  - less communication

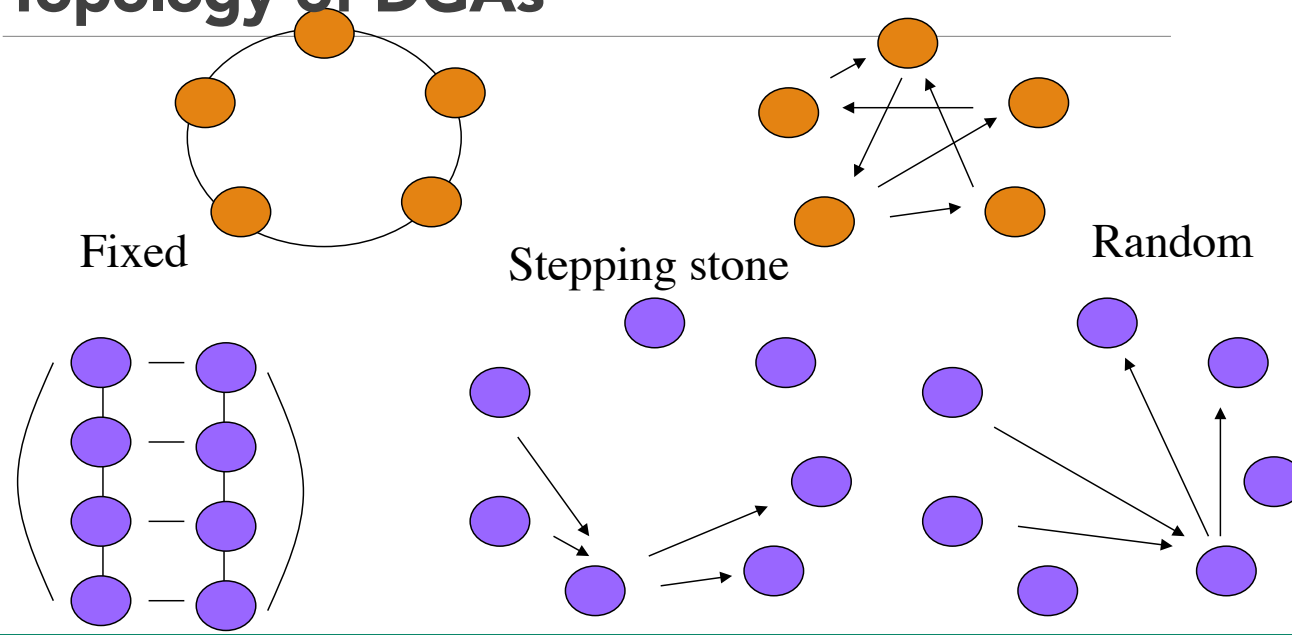
Distributed Genetic Algorithms



Parameters of DGAs

- Topology
- Number of islands
- Migration methods
- Migration interval
- Migration rate
- Migrated individuals

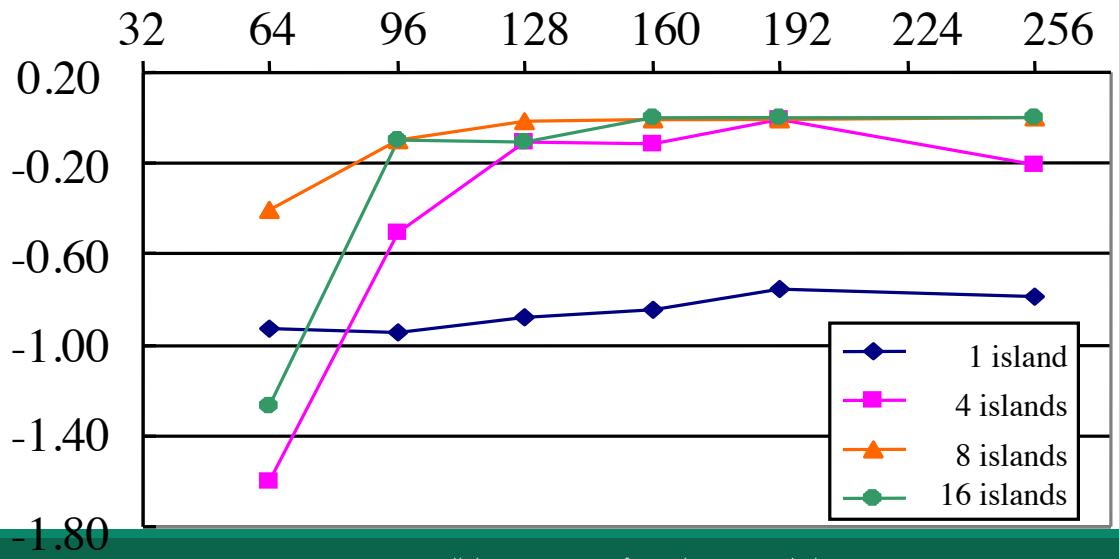
Topology of DGAs



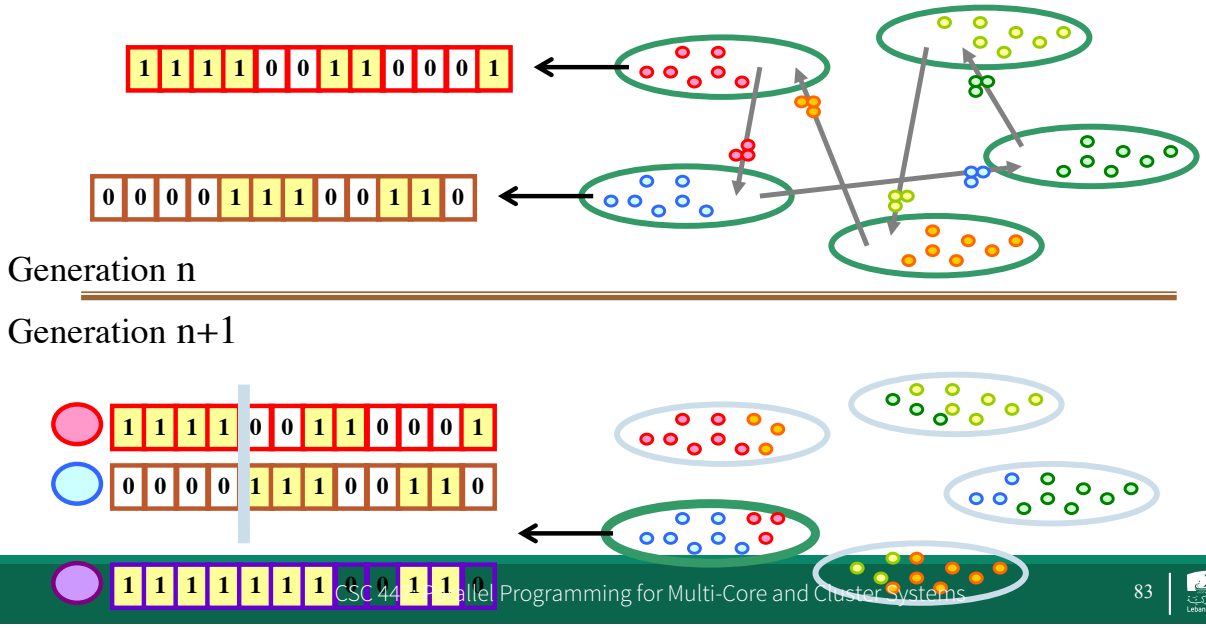
Migrated Individuals

- How to decide migrated individuals?
  - Random choice
  - Elite selection
  - etc.
- How to replace migrated individuals?
  - worst individuals
  - tournament
  - no replace
  - etc.

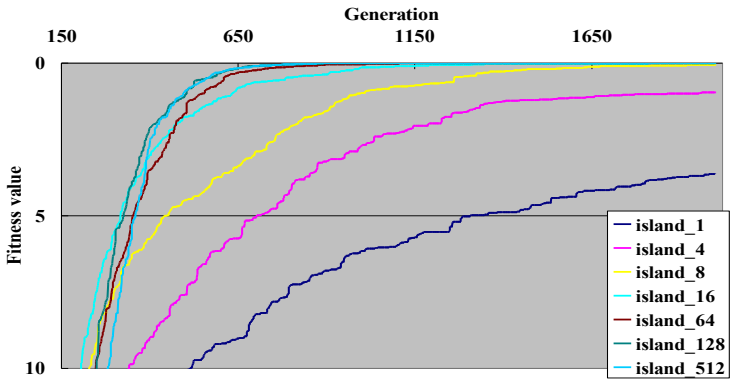
A comparison with DGA of SGA



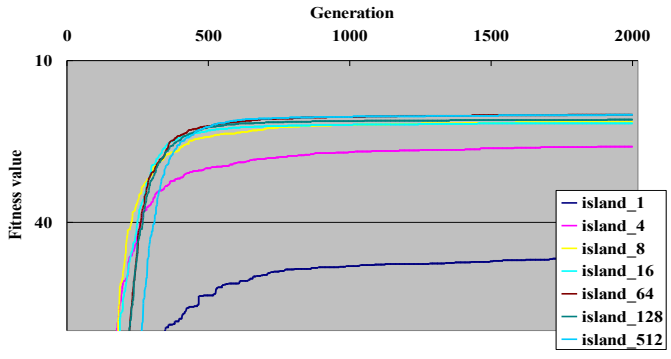
Mechanism of DGAs



Number of islands



Rastrigin



Rosenbrock

Asynchronous DGAs