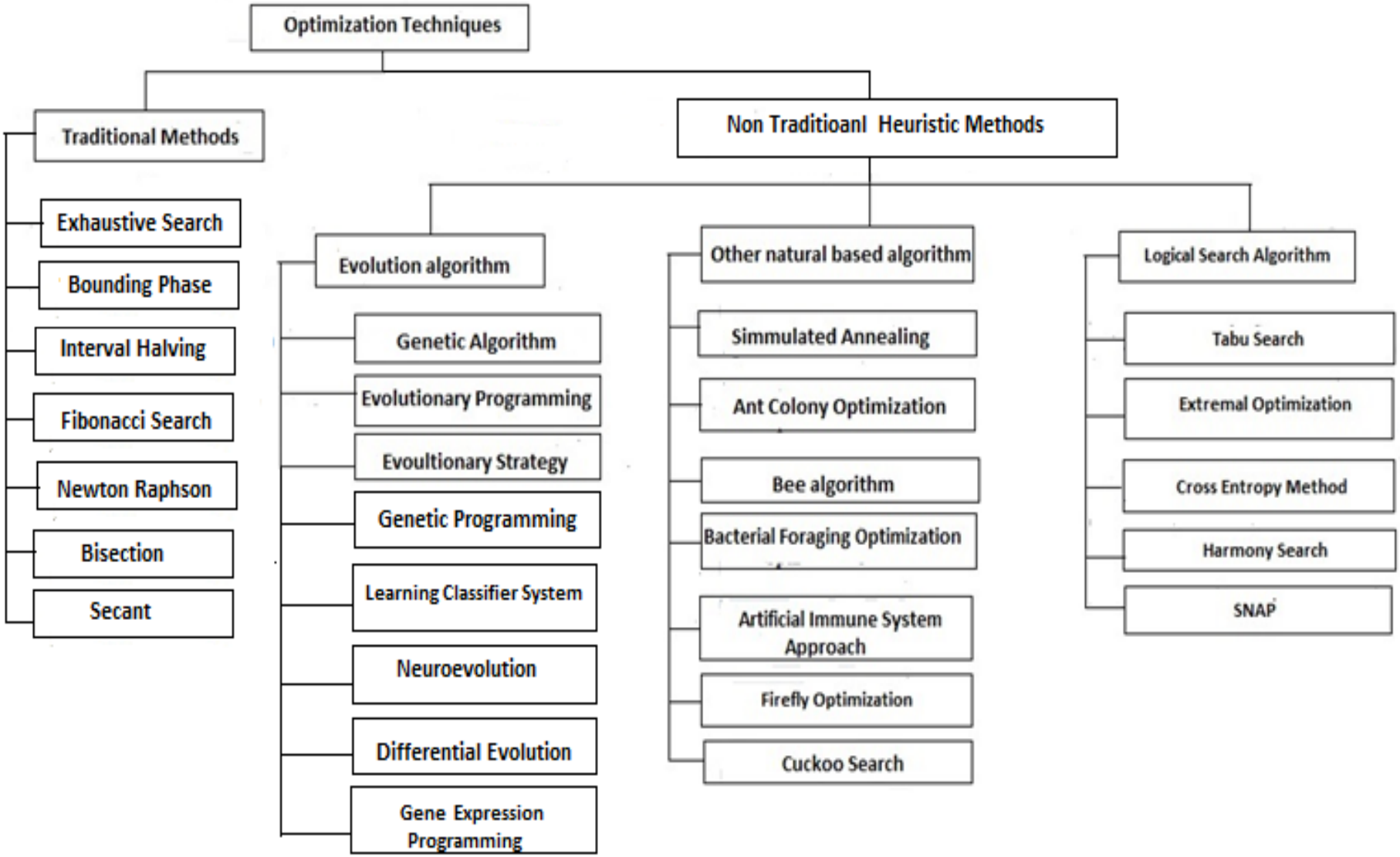
Evolutionary Computation

A C S RAO
Asst.Professor/CSE/ISM Dhanbad
acsrao@iitism.ac.in

References and Resouces for this Lecture

- Books
  - A.E.Eiben and J.E.Smith, Introduction to Evolutionary computing,Natural computiong series,Springer.
  - Dan simon,Evolutionary Optimization Algorithms,wiley .


Optimization Techniques
Traditional Methods
Exhaustive Search
Bounding Phase
Interval Halving
Fibonacci Search
Newton Raphson
Bisection
Secant
Non Traditioanl Heuristic Methods
Evolution algorithm
Genetic Algorithm
Evolutionary Programming
Evoultionary Strategy
Genetic Programming
Learning Classifier System
Neuroevolution
Differential Evolution
Gene Expression Programming
Other natural based algorithm
Simmulated Annealing
Ant Colony Optimization
Bee algorithm
Bacterial Foraging Optimization
Artificial Immune System Approach
Firefly Optimization
Cuckoo Search
Logical Search Algorithm
Tabu Search
Extremal Optimization
Cross Entropy Method
Harmony Search
SNAP

What is Evolutionary Computing

- EC is part of computer science, not part of life sciences/biology.
- Evolutionary computation uses iterative progress, such as growth or development in a population.
- This population is then selected in a guided random search using parallel processing to achieve the desired end.

  (inspired by biological mechanisms of evolution)
- Technically they belong to the family of trial and error problem aiming for global optimization with a metaheuristic or stochastic optimization character, distinguished by the use of a population of candidate solutions.
- They are mostly applied for black box problems (no derivatives known), often in the context of expensive optimization

History

- 1948, Turing: proposes "genetic or evolutionary search"
- 1962, Bremermann: optimization through evolution and recombination
- 1964, Rechenberg : introduces evolution strategies
- 1965, L. Fogel, Owens and Walsh : introduce evolutionary programming
- 1975, Holland : introduces genetic algorithms
- 1992, Koza: introduces genetic programming
- 1985: first international conference (ICGA)
- 1990: first international conference in Europe (PPSN)
- 1993: first scientific EC journal (MIT Press)
- 1997: launch of European EC Research Network EvoNet
- Till today : Many Conferences and Journals are started with great interest to publish articles related to applications of Evolutionary computing approaches to un solved Research problems in almost all scientific areas.

Evolutionary Computing Metaphor

EVOLUTION		PROBLEM SOLVING
Environment	⟷	Problem
Individual	⟷	Candidate Solution
Fitness	⟷	Quality

All environments have finite resources

(i.e., can only support a limited number of individuals)

Fitness → chances for survival and reproduction

Quality → chance for seeding new solutions

Darwin's Principle Of Natural Selection

- IF there are organisms that reproduce, and
- IF offsprings inherit traits from their progenitors, and
- IF there is variability of traits, and
- IF the environment cannot support all members of a growing population,
- THEN those members of the population with less-adaptive traits (determined by the environment) will die out, and
- THEN those members with more-adaptive traits (determined by the environment) will thrive

The result is the evolution of species.

Basic Idea Of Principle Of Natural Selection

"Select The Best, Discard The Rest"

An Example of Natural Selection

- Giraffes have long necks.

Giraffes with slightly longer necks could feed on leaves of higher branches when all lower ones had been eaten off.

→ They had a better chance of survival.

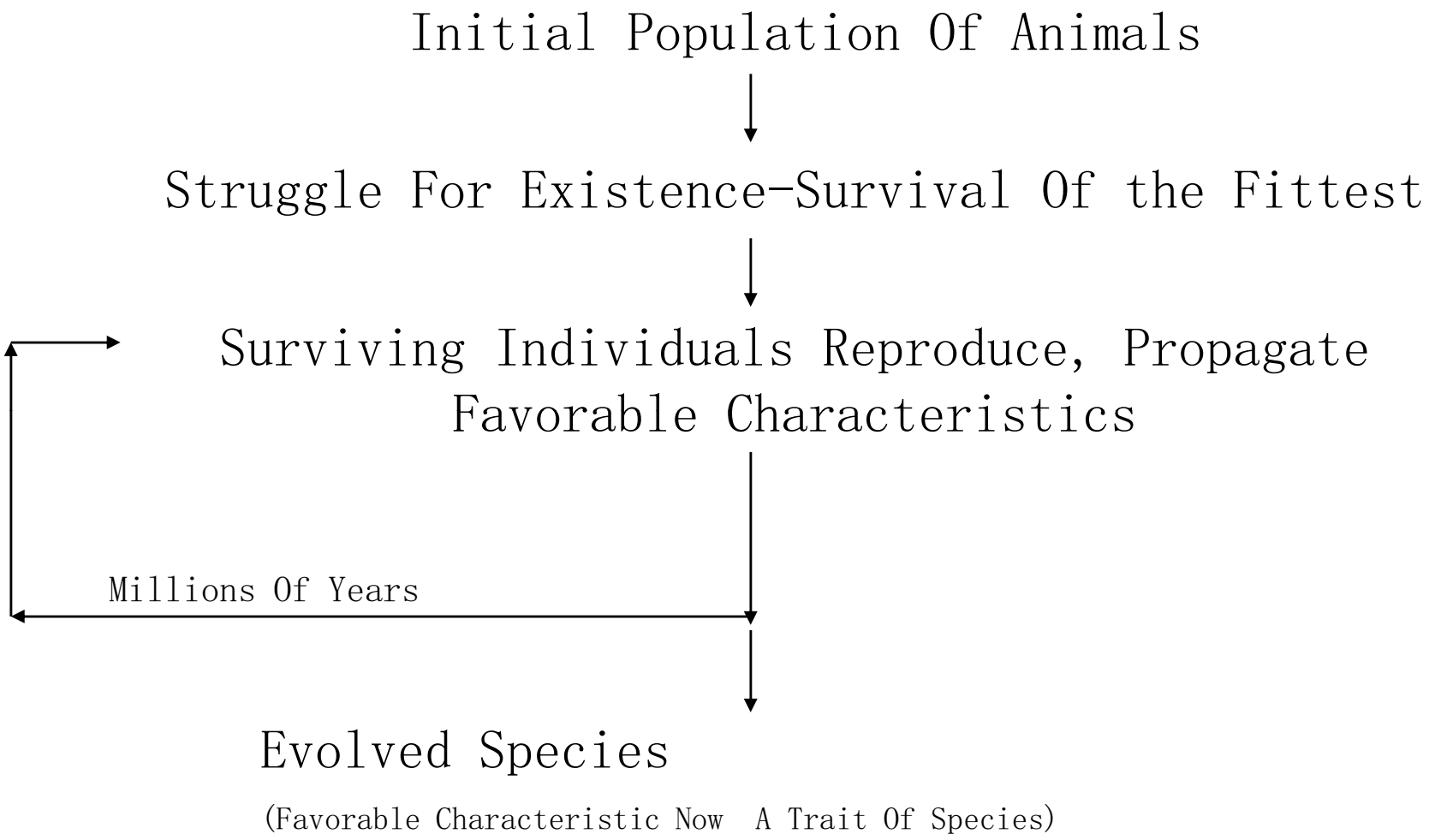
→ Favorable characteristic propagated through generations of giraffes.

→ Now, evolved species has long necks.

NOTE: Longer necks may have been a deviant characteristic (mutation) initially but since it was favorable, was propagated over generations. Now an established trait.

So, some mutations are beneficial.

Evolution Through Natural Selection



Darwinian Evolution 1: Survival of the fittest

- All environments have finite resources
  (i.e., can only support a limited number of individuals)
- 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
- Note: fitness in natural evolution is a derived, secondary measure, i.e., we (humans) assign a high fitness to individuals with many offspring

Darwinian Evolution 2: 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

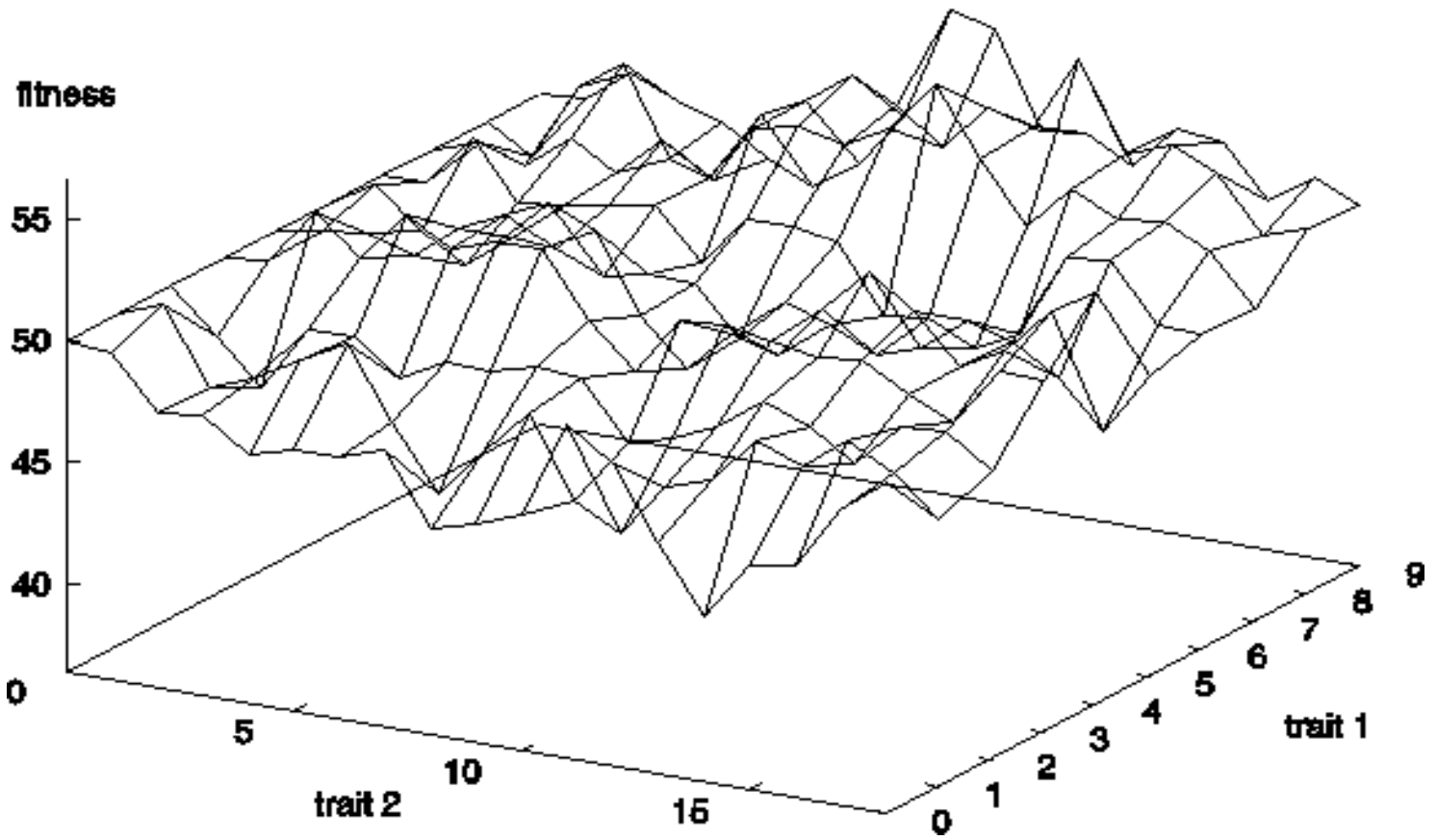
  Individuals are "units of selection"
- Variations occur through random changes yielding constant source of diversity, coupled with selection means that:

  Population is the "unit of evolution"
- Note the absence of "guiding force"

Adaptive landscape metaphor (Wright, 1932)

- Can envisage population with n traits as existing in a $n+1$-dimensional space (landscape) with height corresponding to fitness
- Each different individual (phenotype) represents a single point on the landscape
- Population is therefore a "cloud" of points, moving on the landscape over time as it evolves - adaptation

Example with two traits



Adaptive landscape metaphor (cont'd)

- Selection "pushes" population up the landscape

- Genetic drift:
  - random variations in feature distribution
    (+ or -) arising from sampling error
  - can cause the population "melt down" hills, thus crossing valleys and leaving local optima

Natural Genetics

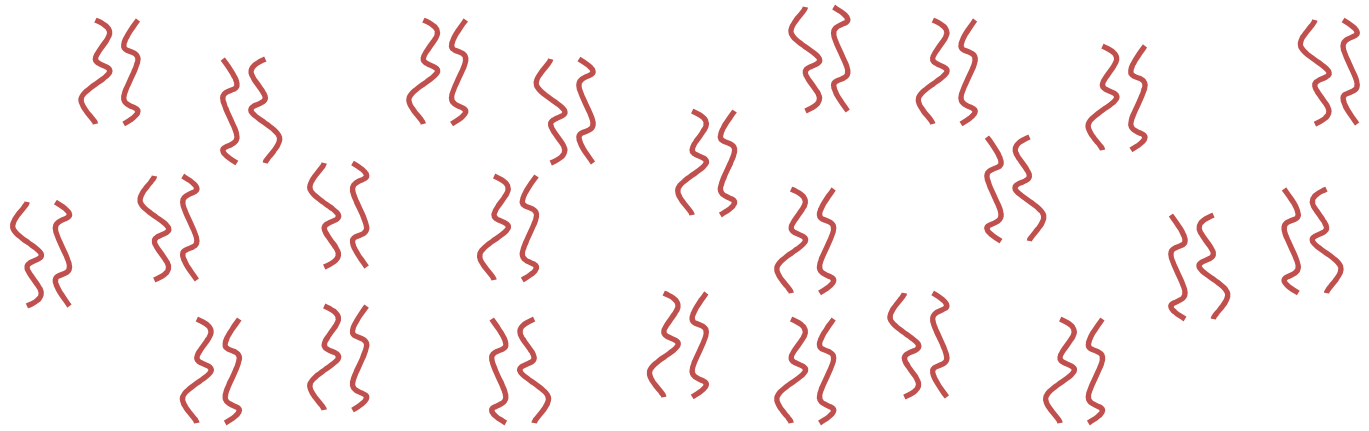
- The information required to build a living organism is coded in the DNA of that organism
- Genotype (DNA inside) determines phenotype
- Genes → phenotypic traits is a complex mapping
  - One gene may affect many traits (pleiotropy)
  - Many genes may affect one trait (polygeny)
- Small changes in the genotype lead to small changes in the organism (e.g., height, hair colour)

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

Example: Homo Sapiens

- Human DNA is organised into chromosomes
- Human body cells contains 23 pairs of chromosomes which together define the physical attributes of the individual:

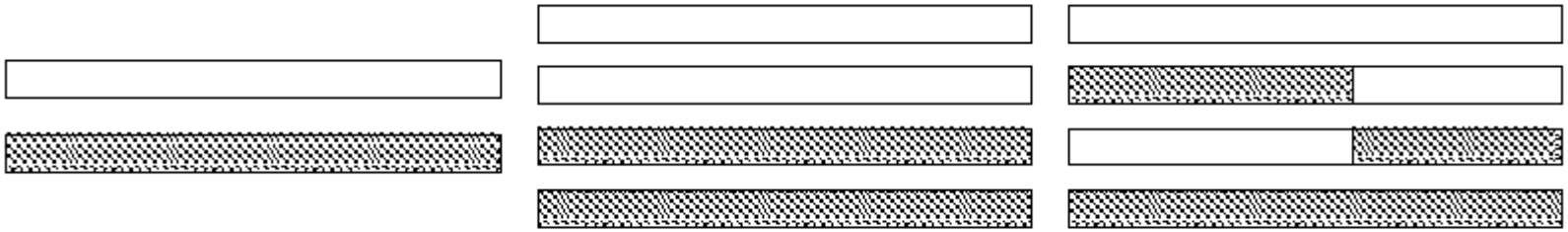


Reproductive Cells

- Gametes (sperm and egg cells) contain 23 individual chromosomes rather than 23 pairs
- Cells with only one copy of each chromosome are called Haploid
- Gametes are formed by a special form of cell splitting called meiosis
- During meiosis the pairs of chromosome undergo an operation called *crossing-over*

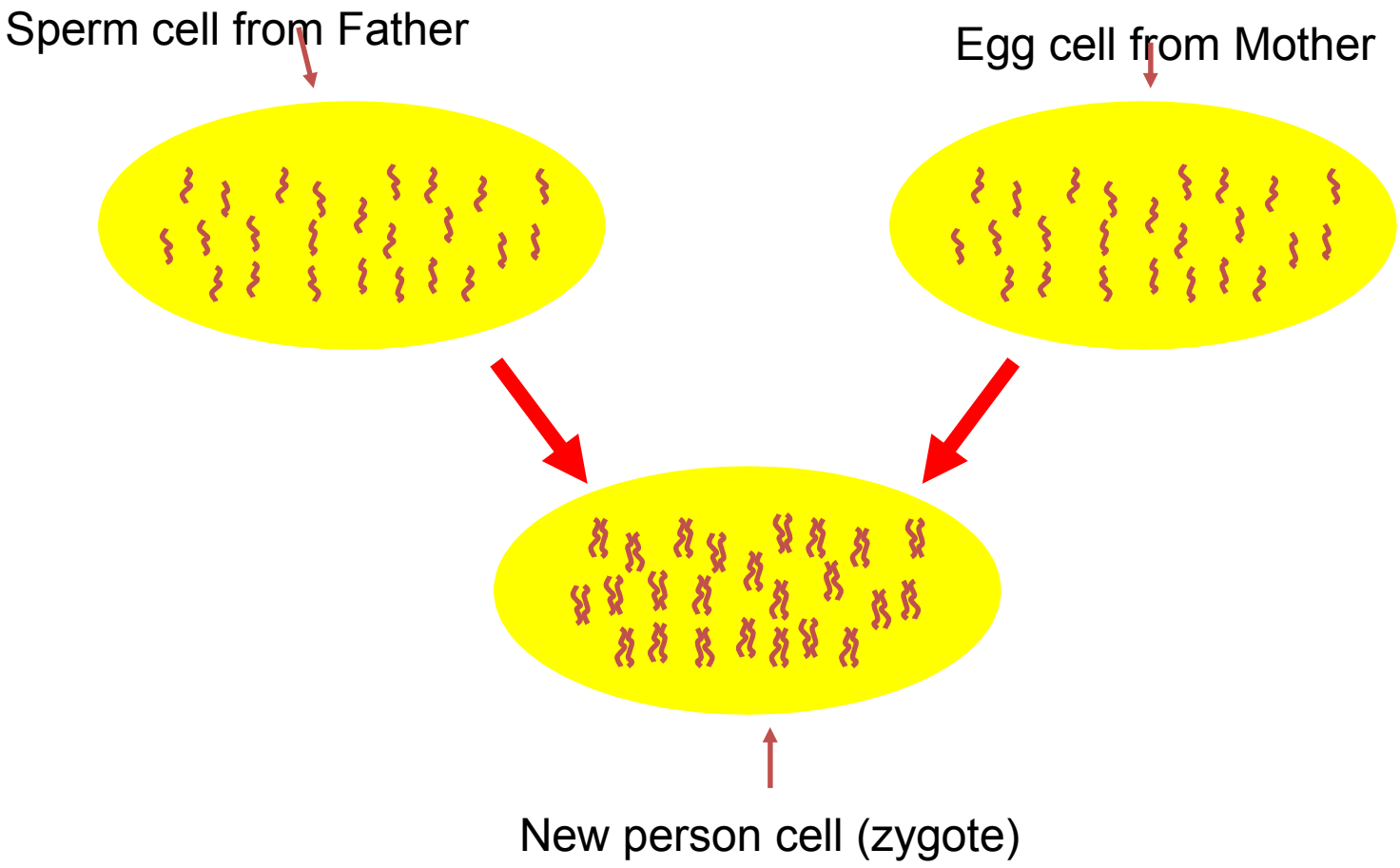
Crossing-over during meiosis

- Chromosome pairs align and duplicate
- Inner pairs link at a *centromere* and swap parts of themselves



- Outcome is one copy of maternal/paternal chromosome plus two entirely new combinations
- After crossing-over one of each pair goes into each gamete

Fertilisation



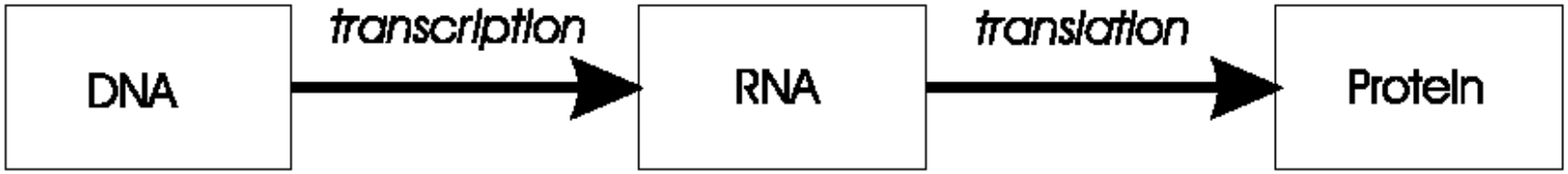
After fertilisation

- New zygote rapidly divides etc creating many cells all with the same genetic contents
- Although all cells contain the same genes, depending on, for example where they are in the organism, they will behave differently
- This process of differential behaviour during development is called ontogenesis
- All of this uses, and is controlled by, the same mechanism for decoding the genes in DNA

Genetic code

- All proteins in life on earth are composed of sequences built from 20 different amino acids
- DNA is built from four nucleotides in a double helix spiral: purines A,G; pyrimidines T,C
- Triplets of these from *codons*, each of which codes for a specific amino acid
- Much redundancy:
  - purines complement pyrimidines
  - the DNA contains much rubbish
  - 4^3=64 codons code for 20 amino acids
  - genetic code = the mapping from codons to amino acids
- For all natural life on earth, the genetic code is the same !

Transcription, translation



Mutation

- Occasionally some of the genetic material changes very slightly during this process (replication error)
- This means that the child might have genetic material information not inherited from either parent
- This can be
  - catastrophic: offspring in not viable (most likely)
  - neutral: new feature not influences fitness
  - advantageous: strong new feature occurs
- Redundancy in the genetic code forms a good way of error checking

Evolution in the real world

- Each cell of a living thing contains *chromosomes* - strings of *DNA*
- Each chromosome contains a set of *genes* - blocks of DNA
- Each gene determines some aspect of the organism (like eye colour)
- A collection of genes is sometimes called a *genotype*
- A collection of aspects (like eye colour) is sometimes called a *phenotype*
- Reproduction involves recombination of genes from parents and then small amounts of *mutation* (errors) in copying
- The *fitness* of an organism is how much it can reproduce before it dies
- Evolution based on "survival of the fittest"

Start with a Dream…

- Suppose you have a problem
- You don't know how to solve it
- What can you do?
- Can you use a computer to somehow find a solution for you?
- This would be nice! Can it be done?

A dumb solution

A "blind generate and test" algorithm:

Repeat

  Generate a random possible solution

  Test the solution and see how good it is

Until solution is good enough

Can we use this dumb idea?

- Sometimes – yes:
  - if there are only a few possible solutions
  - and you have enough time
  - then such a method *could* be used
- For most problems – no:
  - many possible solutions
  - with no time to try them all
  - so this method *can not* be used

A "less-dumb" idea (GA)

Generate a *set* of random solutions

Repeat

  Test each solution in the set (rank them)

  Remove some bad solutions from set

  Duplicate some good solutions

    make small changes to some of them

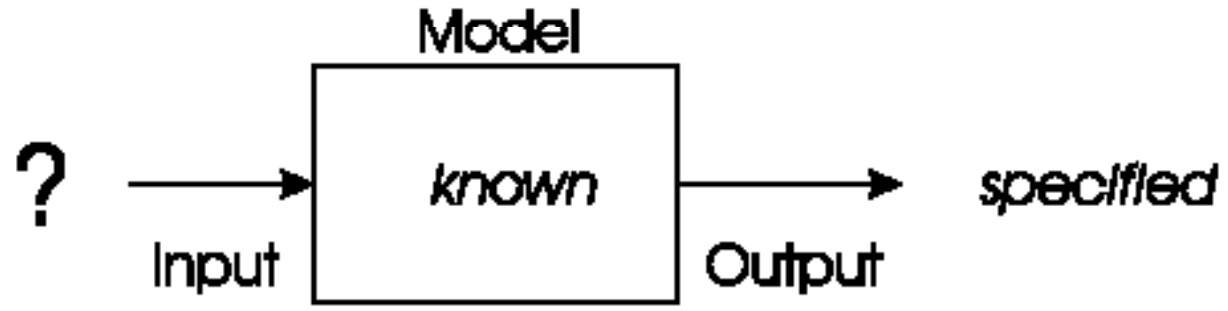
Until best solution is good enough

Motivations for EC: 2

- Developing, analyzing, applying problem solving methods a.k.a. algorithms is a central theme in mathematics and computer science
- Time for thorough problem analysis decreases
- Complexity of problems to be solved increases
- Consequence:
  Robust problem solving technology needed

Problem type 1 : Optimisation

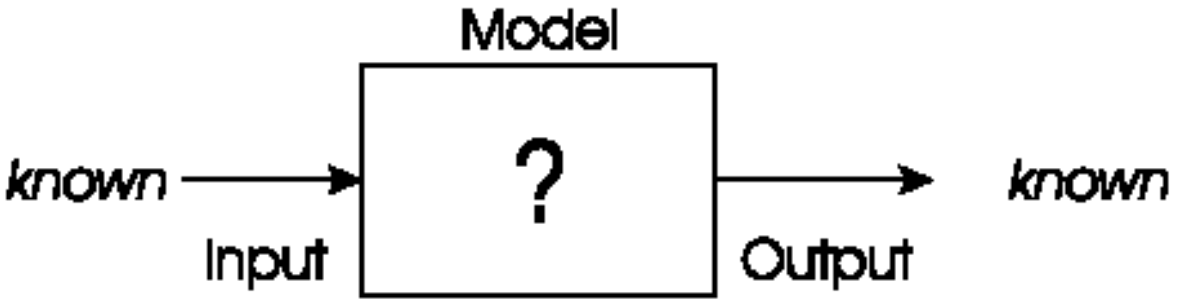
- We have a model of our system and seek inputs that give us a specified goal



- e.g.
  - time tables for university, call center, or hospital
  - design specifications, etc etc

Problem types 2: Modelling

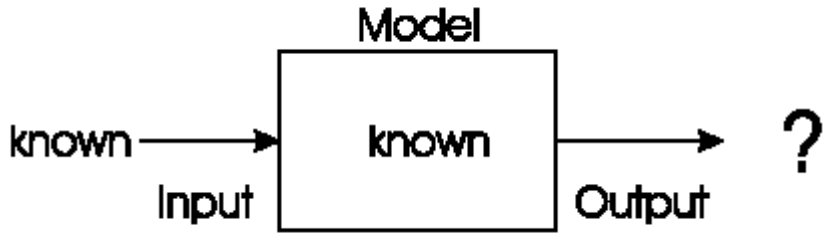
- We have corresponding sets of inputs & outputs and seek model that delivers correct output for every known input



•e.g. Evolutionary machine learning(loan applicant creditibility)
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



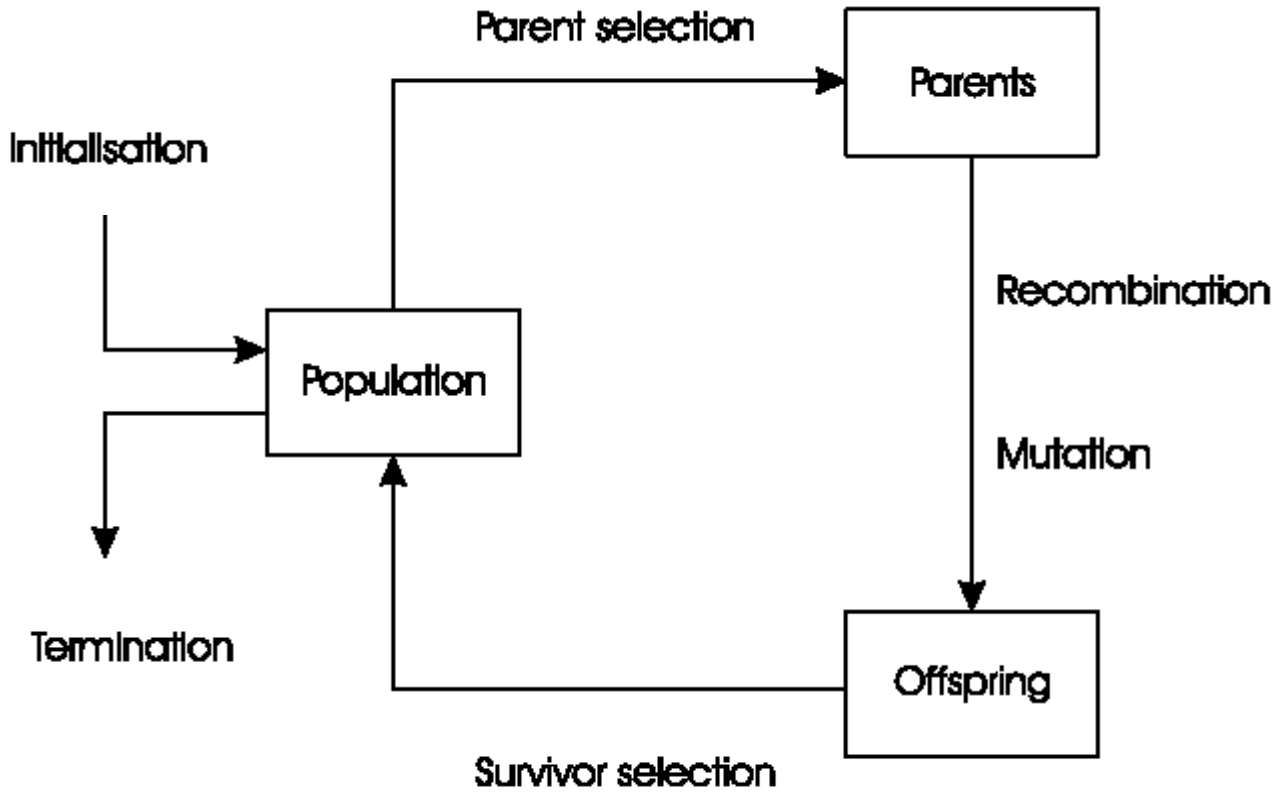
- Often used to answer “what-if” questions in evolving dynamic environments
- e.g. Evolutionary economics, Artificial Life

EC Theory

- EC research is not only for the Optimization , Modelling and Simulation , also there is lot of scope for new theoretical studies that can make new EC techniques.

e.g. Hybrid approaches

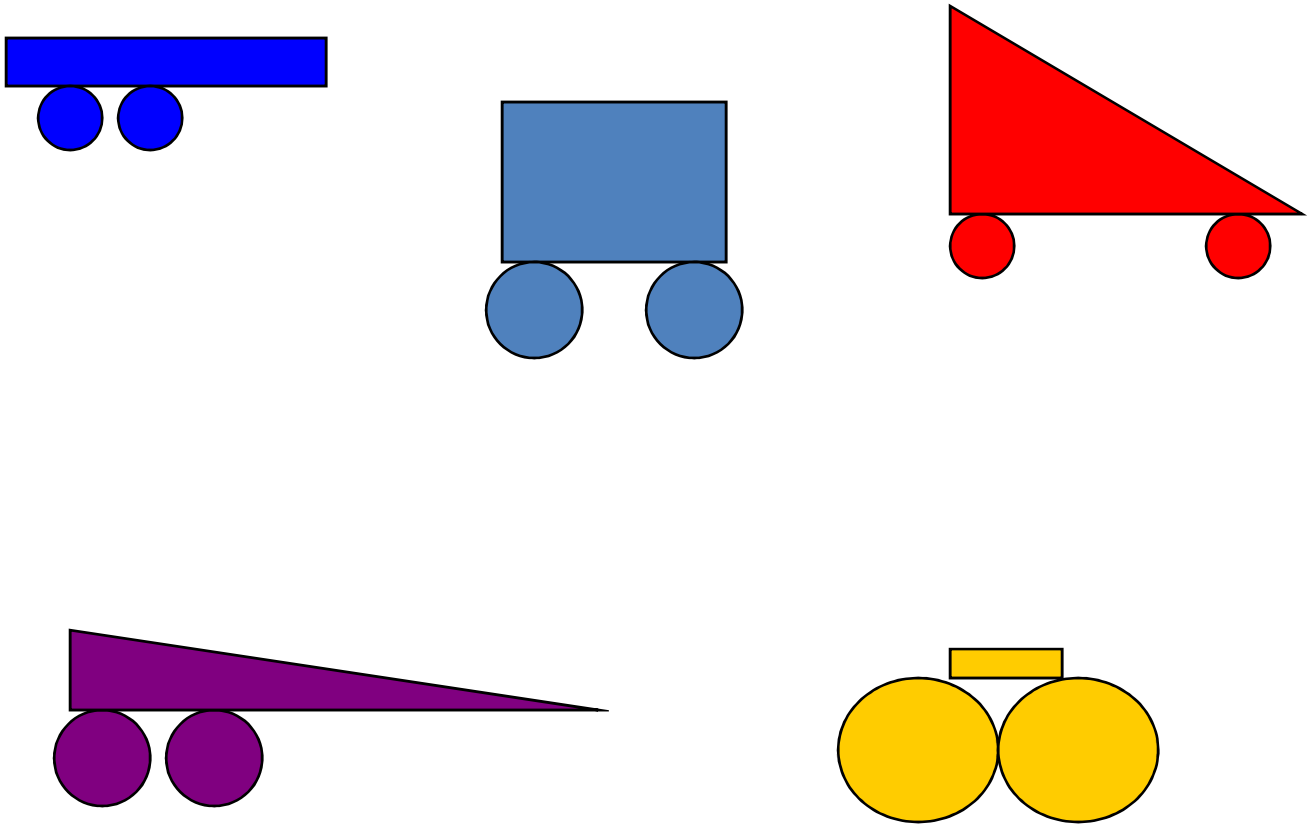
General Scheme of EAs



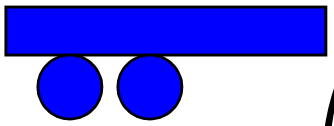
Pseudo-code for typical EA

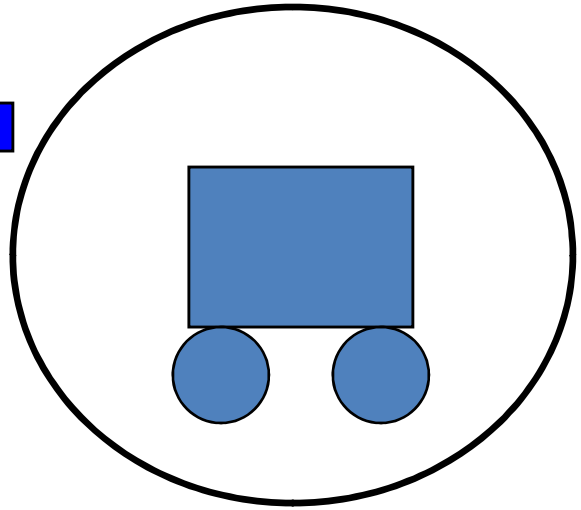
```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

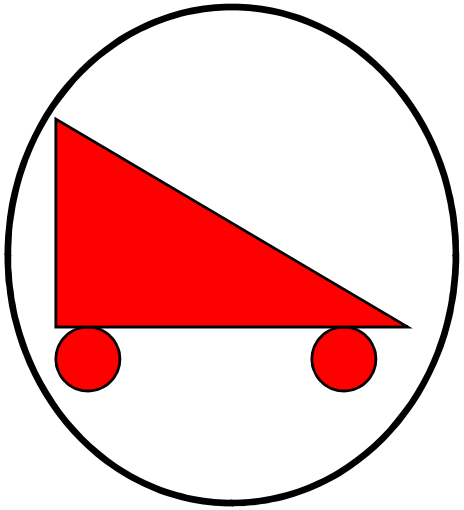
Initial population

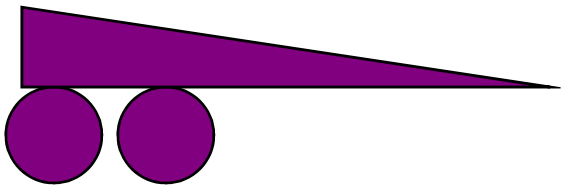


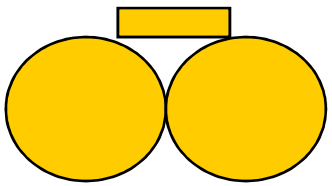
Select

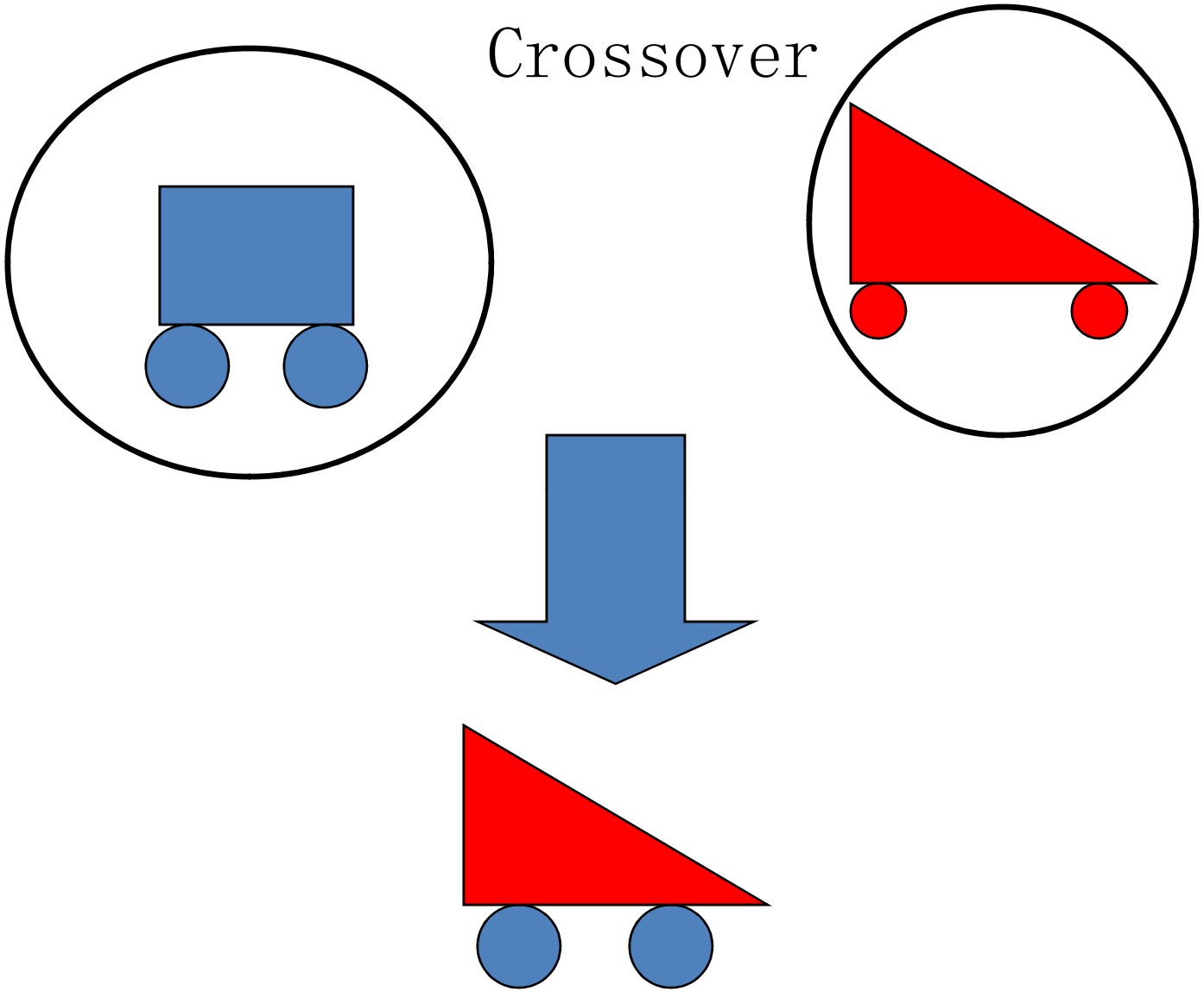




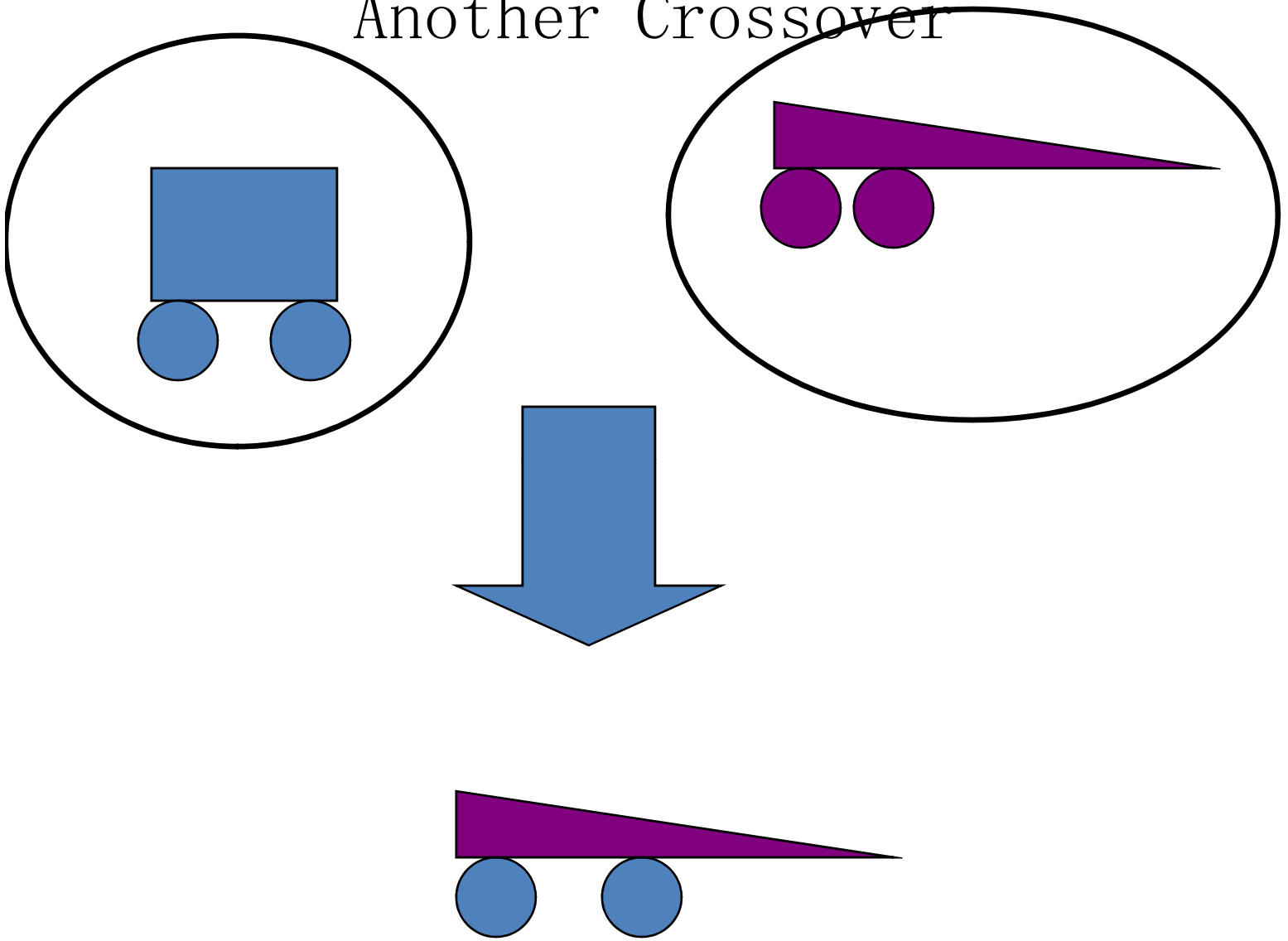




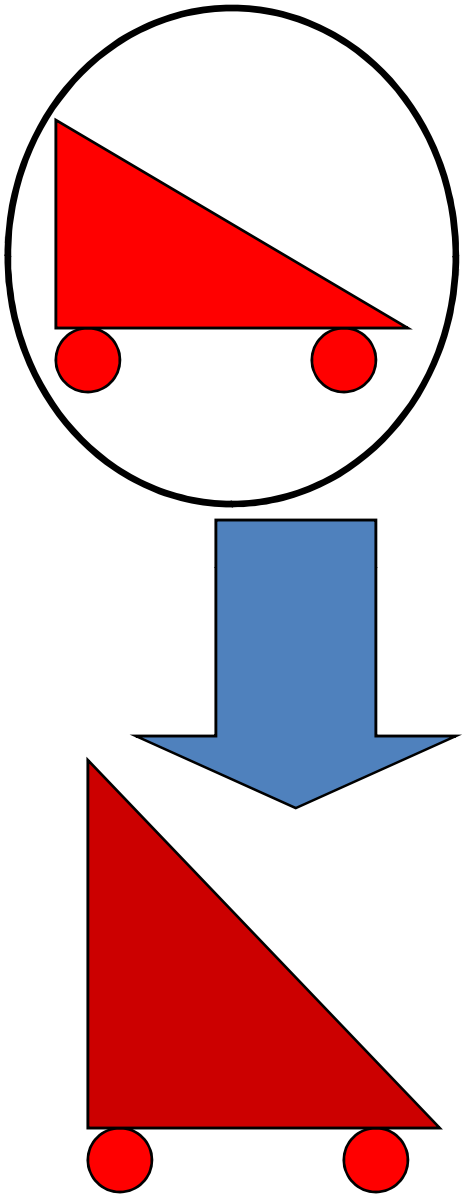



Crossover

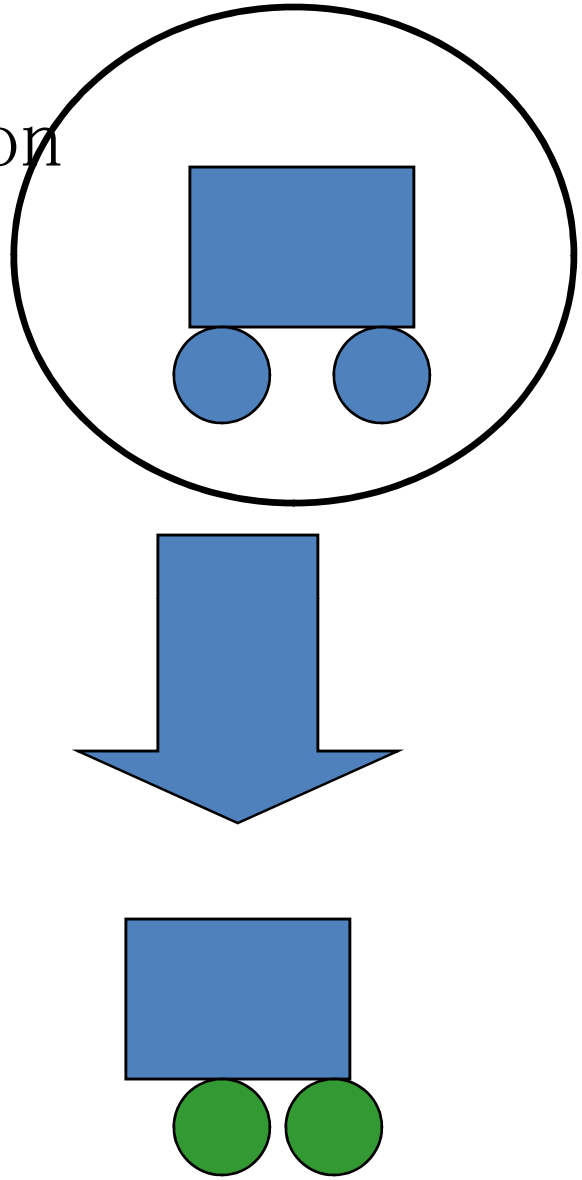
Another Crossover



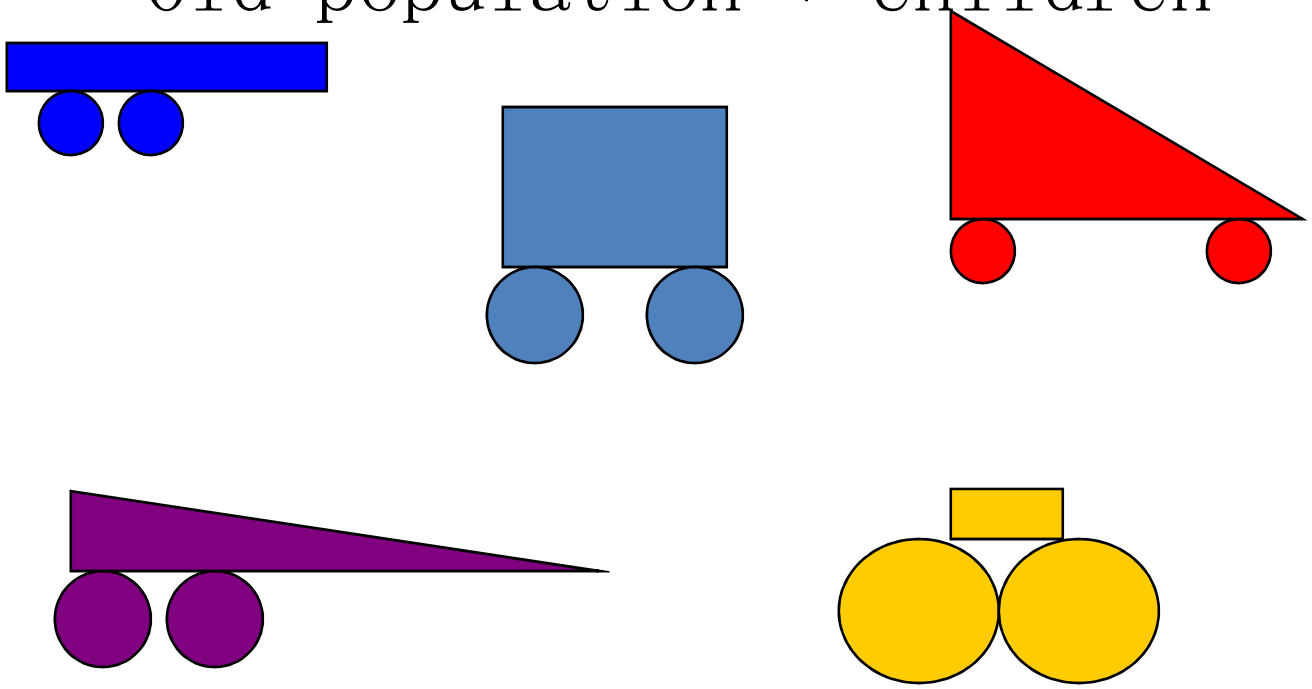
A mutation



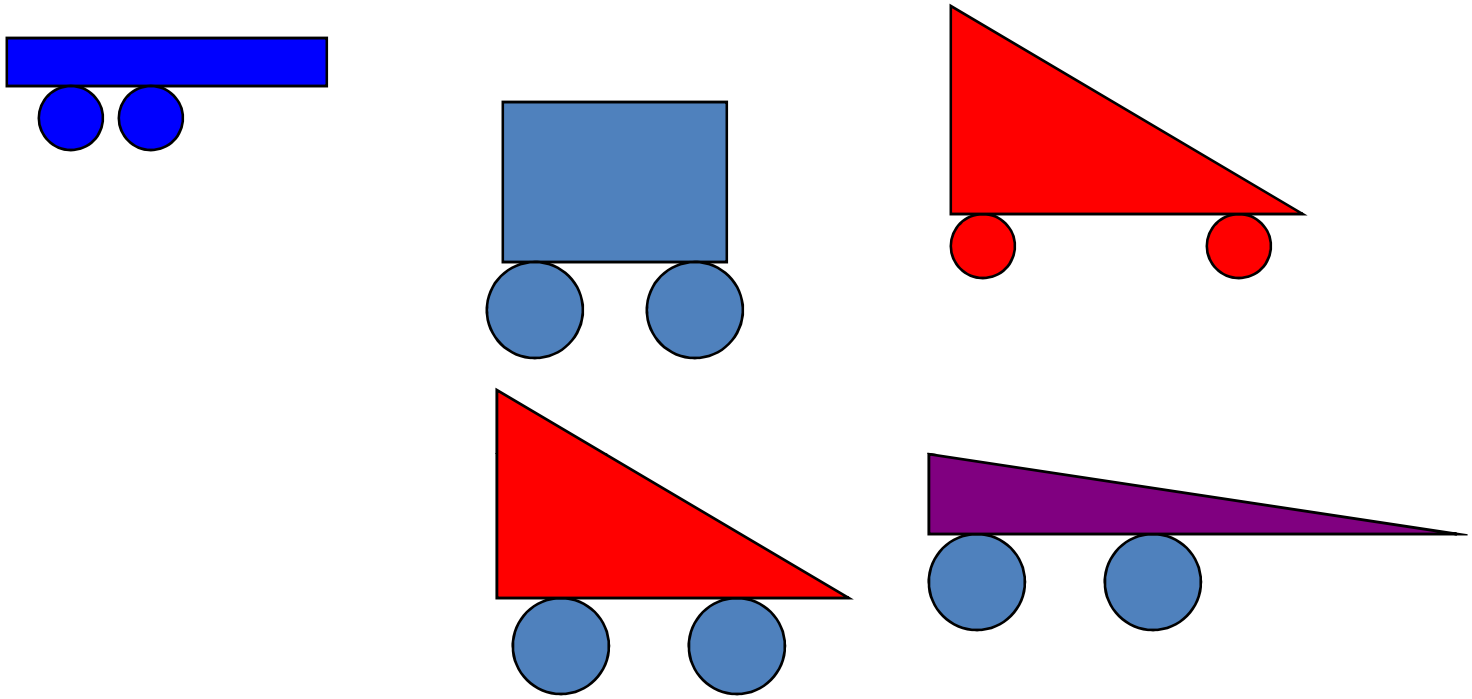
Another Mutation



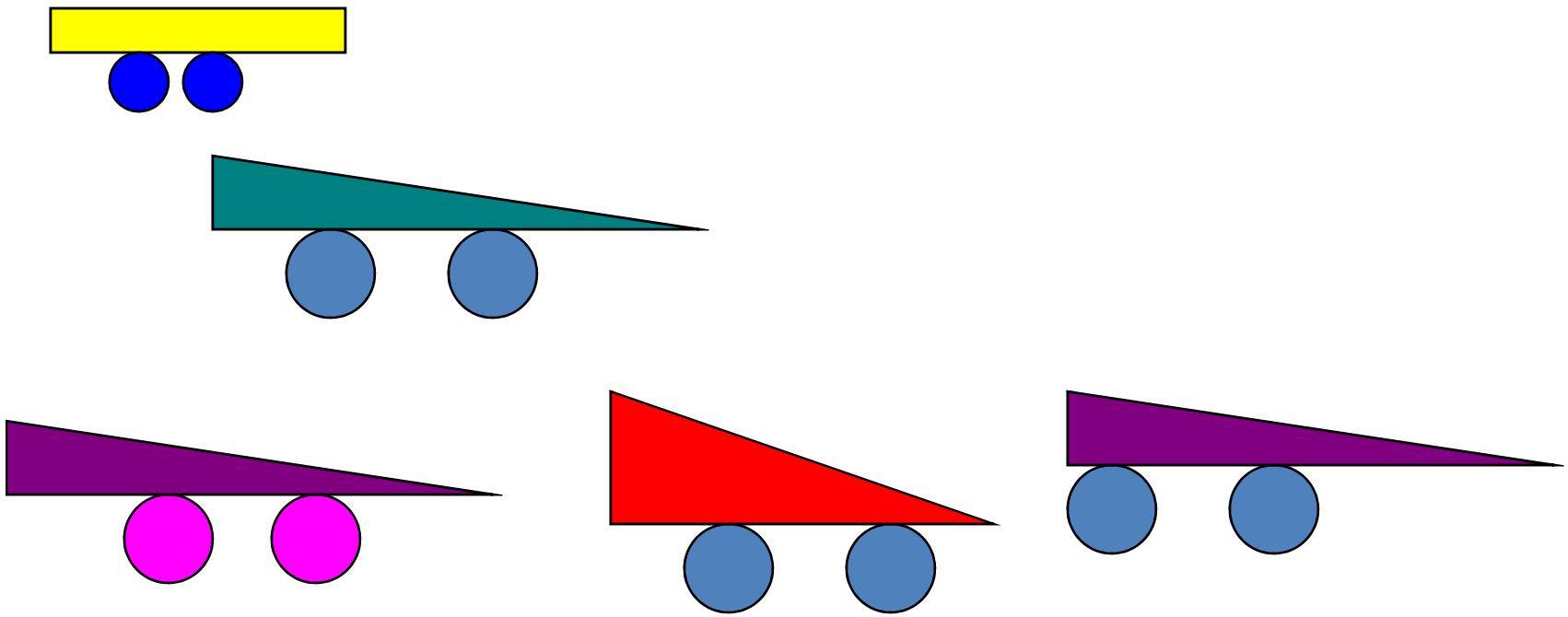
Old population + children



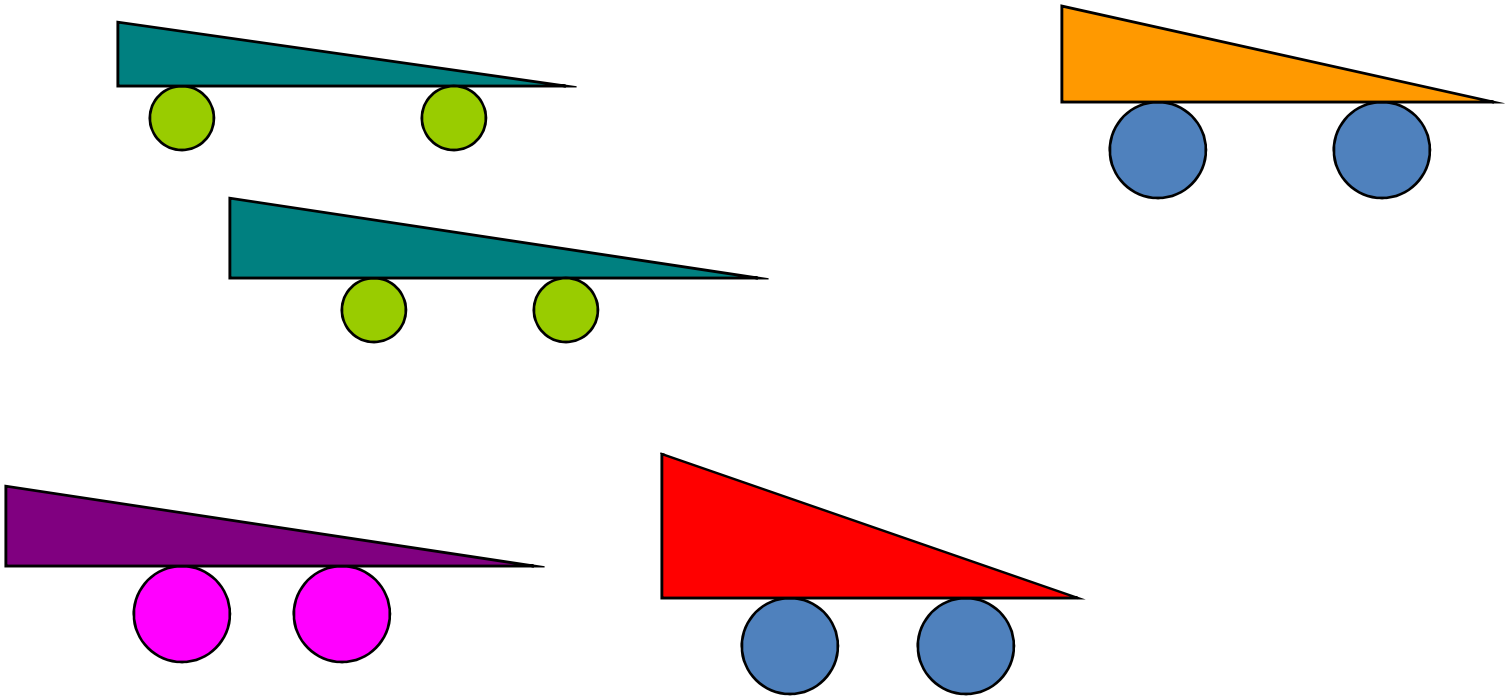
New Population: Generation 2



Generation 3



Generation 4, etc …



Different Flavours 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

Recent Challenge -Distributed EA

- Recent work has concentrated on the implementation of EAs on parallel machines.
- Typically either one processor holds one individual (in SIMD machines), or a subpopulation (in MIMD machines).
- Clearly, such implementations hold promise of execution time decreases.

Components of Evolutionary Algorithms

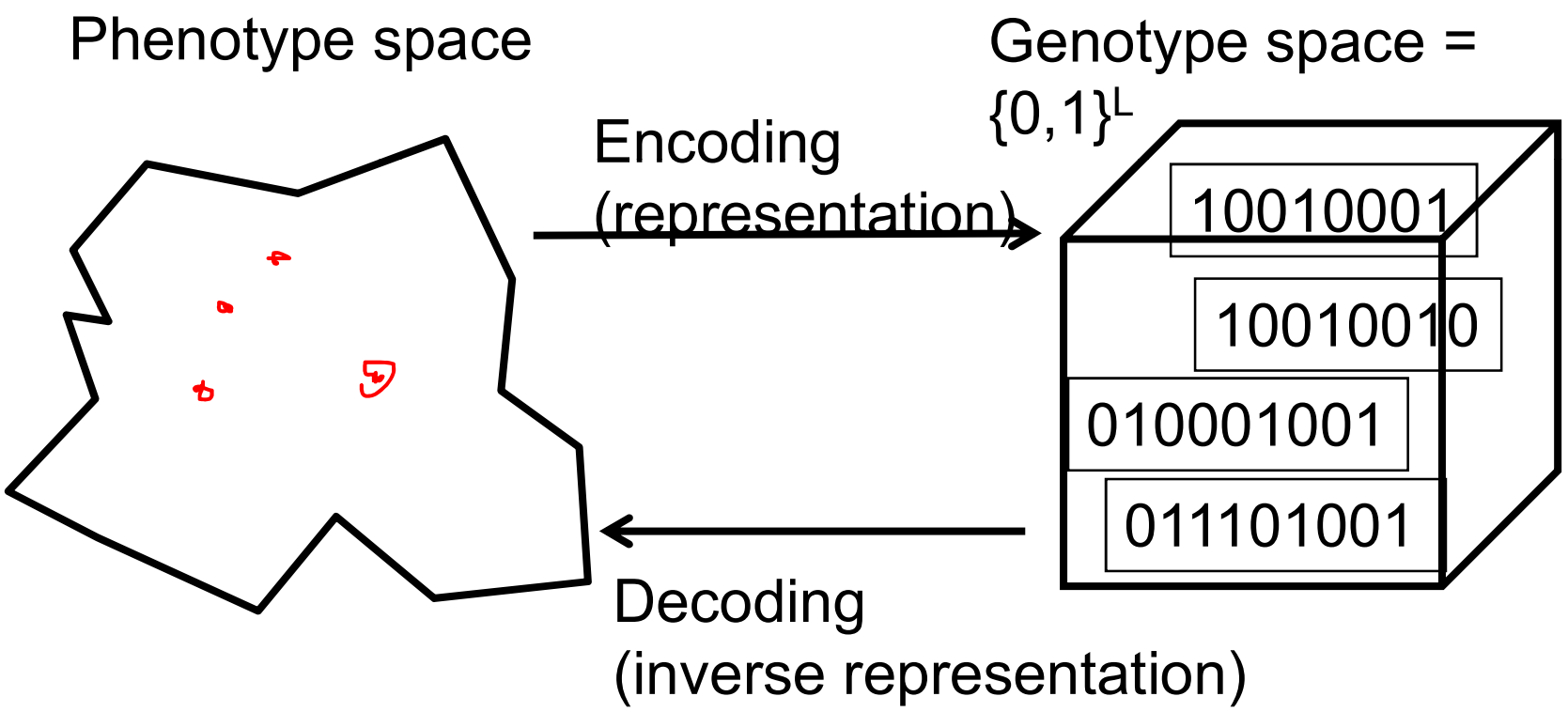
- Representations
- Evaluation (Fitness) Function
- Population
- Selection (Parent Selection , Survivor Selection)
- Variation Operators
- Initialisation / Termination

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

Representation



Binary Representation

- Discrete feature encoding:
  - e.g., 0 and 1 for the presence or absence of the features;
  - chromosomes: 001110101110;
  - the genes do not represent necessarily full features;

Binary Representation (cont..)

- A chromosome should contain information about solution that it represents.
- The commonly used way of encoding is a binary string.

Chromosome 1:	1101100100110110
Chromosome 2:	1101111000011110

- Each bit in the string represents some characteristics of the solution.
- There are many other ways of encoding. The encoding depends mainly on the problem.

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
- Tree based representation

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}

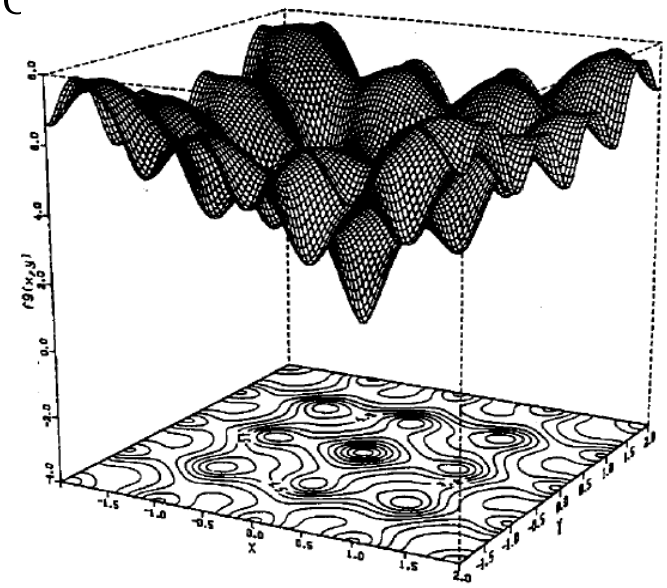
Real valued problems

- Many problems occur as real valued problems, e.g. continuous parameter optimization $f : \mathcal{R}^n \rightarrow \mathcal{R}$
- Illustration: Ackley's function (often used in

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

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

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



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

Representation (EP)

- For continuous parameter optimisation
- Chromosomes consist of two parts:
  - Object variables: $x_1, \ldots, x_n$
  - Mutation step sizes: $\sigma_1, \ldots, \sigma_n$
- Full size: $\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n \rangle$

Modern EP

- No predefined representation in general
- Thus: no predefined mutation (must match representation)
- Often applies self-adaptation of mutation parameters
- In the sequel we present *one EP variant*, not the canonical EP

Representation (evolutionary strategies)

- Chromosomes consist of three parts:
  - Object variables: $x_1, \ldots, x_n$
  - Strategy parameters:
    - Mutation step sizes: $\sigma_1, \ldots, \sigma_{n_\sigma}$
    - Rotation angles: $\alpha_1, \ldots, \alpha_{n_\alpha}$

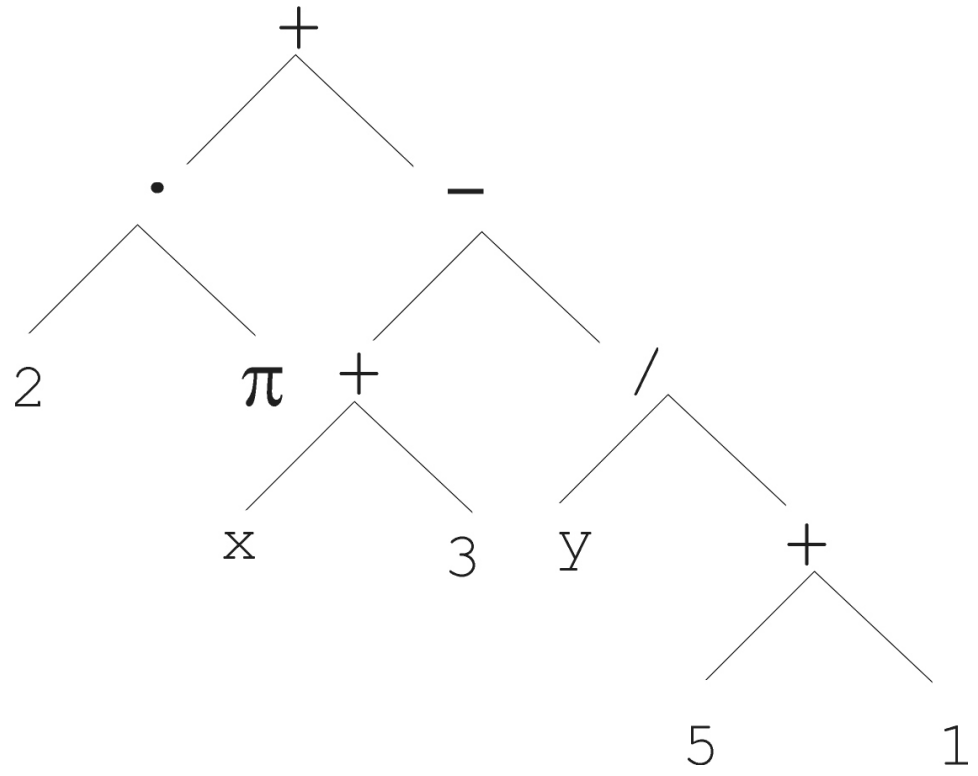
    α is used for the interaction between step sizes of different variables
- Not every component is always present
- Full size: $\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n, \alpha_1, \ldots, \alpha_k \rangle$
- where $k = n(n-1)/2$ (no. of i,j pairs)

Tree based representation (Genetic Programming)

- Trees are a universal form, e.g. consider
- Arithmetic formula $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$
- Logical formula $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program

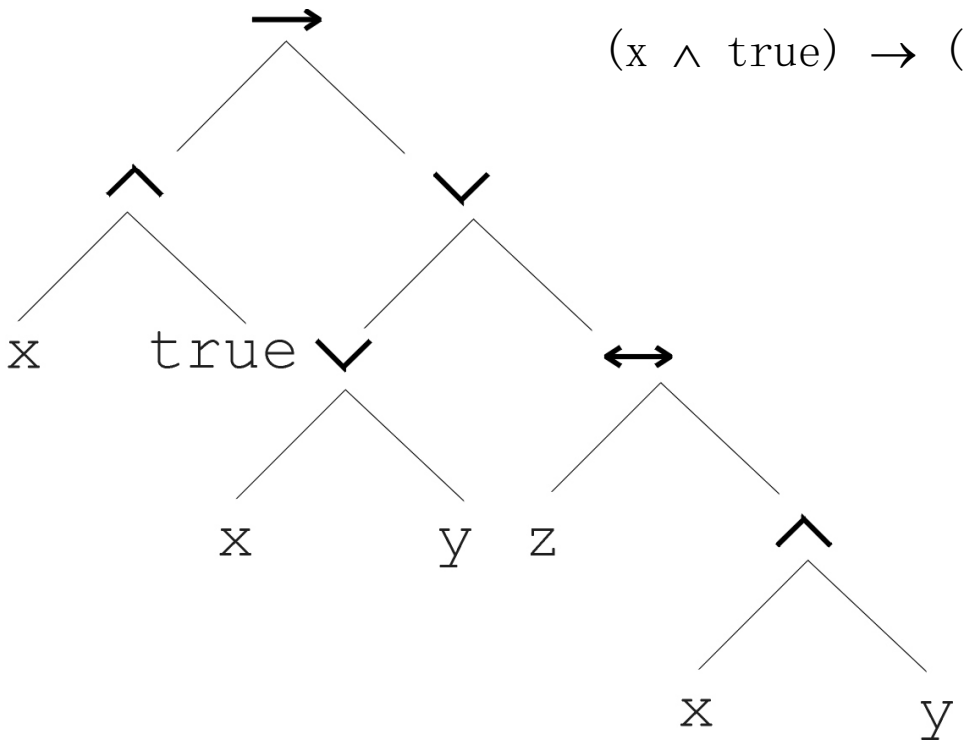
```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation



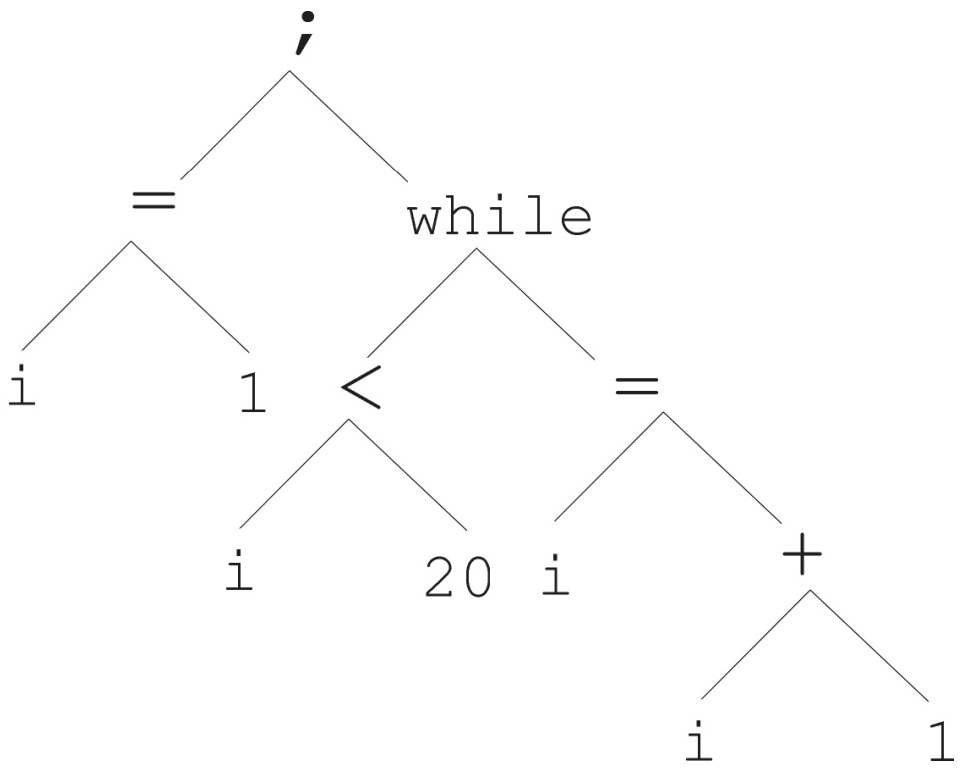
$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

Tree based representation



$(x \wedge \text{true}) \rightarrow ((\ x \vee y\ ) \vee (z \leftrightarrow (x \wedge y)))$

Tree based representation




```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation

- In GA, ES, EP chromosomes are linear structures (bit strings, integer string, real-valued vectors, permutations)
- Tree shaped chromosomes are non-linear structures
- In GA, ES, EP the size of the chromosomes is fixed
- Trees in GP may vary in depth and width

Components of Evolutionary Algorithms

- Representations
- Evaluation (Fitness) Function
- Population
- Selection (Parent Selection , Survivor Selection)
- Variation Operators
- Initialisation / Termination

Evaluation (Fitness) Function

- Represents the requirements that the population should adapt to
  - *a.k.a* *quality* function or *objective* function
- The fitness is calculated by first decoding the chromosome and then the evaluating the objective function.
- Fitness function is and indicator of how close the chromosome is to the optimal solution
- Typically we talk about fitness being maximised
  - Some problems may be best posed as minimisation problems, but conversion is trivial

Fitness Function(GP context)

- The most difficult and important concept of GP is the fitness function.
- The fitness function determines how well a program is able to solve the problem.
- It varies greatly from one type of program to the next.
- For example, if one were to create a genetic program to set the time of a clock, the fitness function would simply be the amount of time that the clock is wrong.

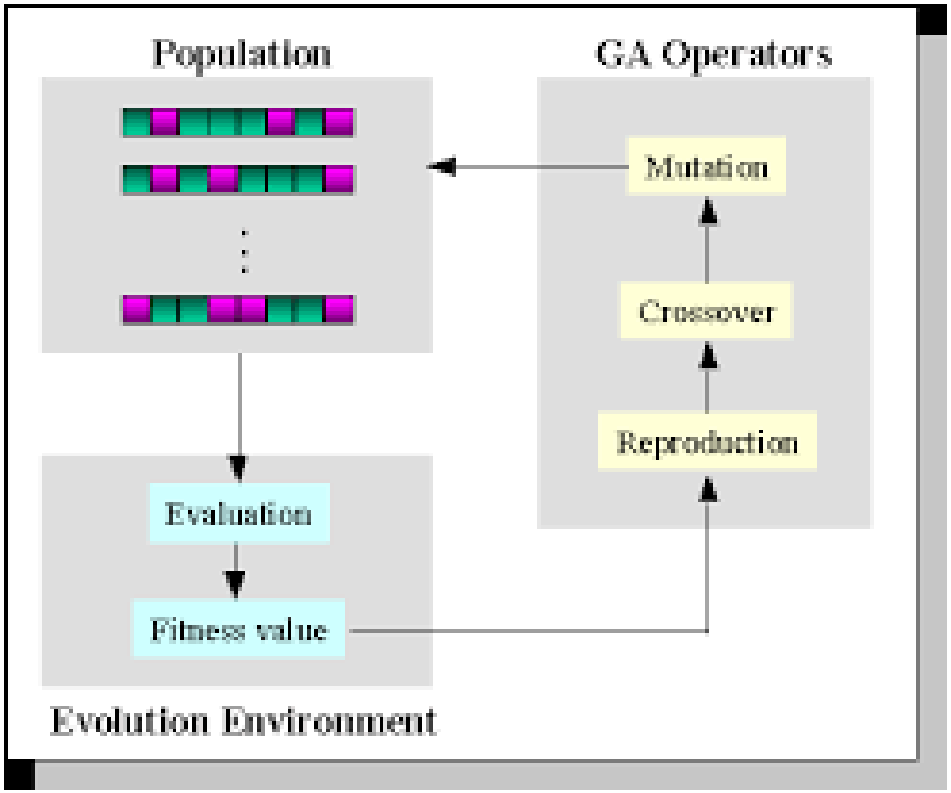
Components of Evolutionary Algorithms

- Representations
- Evaluation (Fitness) Function
- Population
- Selection (Parent Selection , Survivor Selection)
- Variation Operators
- Initialisation / Termination

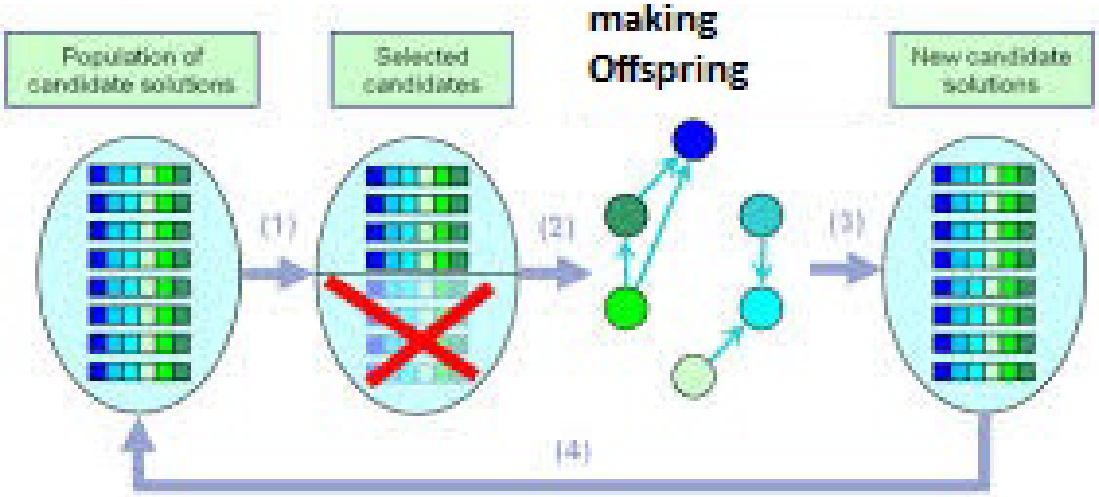
Population

- 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)

Population(cont..)



Population(cont..)



Population Models of GA

- 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

Components of Evolutionary Algorithms

- Representations
- Evaluation (Fitness) Function
- Population
- Selection (Parent Selection , Survivor Selection)
- Variation Operators
- Initialisation / Termination

Selection (Parent Selection , Survivor 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)
- 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

Parent Selection Mechanism

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

Survivor Selection-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 of above two

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

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

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

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

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

Parent selection

- Parents are selected by uniform random distribution whenever an operator needs one/some
- Thus: ES parent selection is unbiased - every individual has the same probability to be selected
- Note that in ES "parent" means a population member (in GA's: a population member selected to undergo variation)

Survivor selection

- Applied after creating λ children from the μ parents by mutation and recombination
- Deterministically chops off the "bad stuff"
- Basis of selection is either:
  - The set of children only: (μ,λ)-selection
  - The set of parents and children: $(\mu+\lambda)$-selection

Survivor selection cont'd

- $(\mu+\lambda)$-selection is an elitist strategy
- (μ,λ)-selection can "forget"
- Often (μ,λ)-selection is preferred for:
  - Better in leaving local optima
  - Better in following moving optima
  - Using the + strategy bad σ values can survive in $\langle x,\sigma\rangle$ too long if their host x is very fit
- Selective pressure in ES is very high ($\lambda \approx 7 \cdot \mu$ is the common setting)

Parent selection (EP)

- Each individual creates one child by mutation
- Thus:
  - Deterministic
  - Not biased by fitness

Survivor selection (EP)

- P(t): μ parents, P′(t): μ offspring
- Pairwise competitions in round-robin format:
  - Each solution x from P(t) $\cup$ P′(t) is evaluated against q other randomly chosen solutions
  - For each comparison, a "win" is assigned if x is better than its opponent
  - The μ solutions with the greatest number of wins are retained to be parents of the next generation
- Parameter q allows tuning selection pressure

Selection (GP)

- Parent selection typically fitness proportionate
- Over-selection in very large populations
  - rank population by fitness and divide it into two groups:
  - group 1: best x% of population, group 2 other (100-x)%
  - 80% of selection operations chooses from group 1, 20% from group 2
  - for pop. size = 1000, 2000, 4000, 8000 x = 32%, 16%, 8%, 4%
  - motivation: to increase efficiency, %'s come from rule of thumb
- Survivor selection:
  - Typical: generational scheme (thus none)
  - Recently steady-state is becoming popular for its elitism

Components of Evolutionary Algorithms

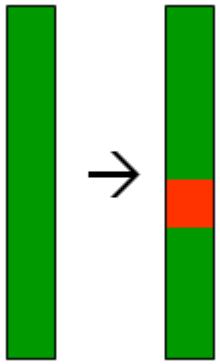
- Representations
- Evaluation (Fitness) Function
- Population
- Selection (Parent Selection , Survivor Selection)
- Variation Operators(mutation, cross over)
- Initialisation / Termination

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 operator:

- randomly change the genes encoding the solution features;
- e.g., changing a 0 into a 1 and inversely;
- e.g., minor modification of a feature encoded by a real number;

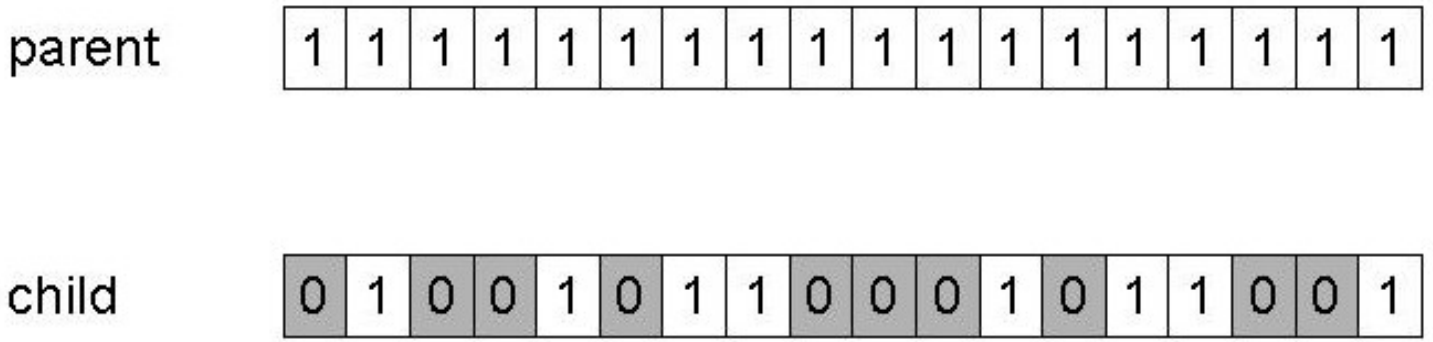


Mutation operator: cont..

- 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

Binary encoding : mutation(SGA)

- 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



Integer representations mutation

- 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}
- Extend bit-flipping mutation to make
  - "creep" i.e. more likely to move to similar value
  - Random choice (esp. categorical variables)
  - For ordinal problems, it is hard to know correct range for creep, so often use two mutation operators in tandem

Floating point mutations 1

General scheme of floating point mutations

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

- Uniform mutation:

$$x_i' \text{ drawn randomly (uniform) from } [LB_i, UB_i]$$

- 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 deviate to each variable separately, taken from N(0, σ) Gaussian distribution and then curtail to range
  - Standard deviation σ controls amount of change (2/3 of deviations will lie in range ($-\sigma$ to $+\sigma$)

Mutation operators for permutations

- Normal mutation operators lead to inadmissible 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 (4 links broken), disrupts order more

Inversion mutation for permutations

- Pick two alleles at random and then invert the substring between them.
- Preserves most adjacency information (only breaks two links) 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)

Mutation (ES)

- Main mechanism: changing value by adding random noise drawn from normal distribution
- $x'_i = x_i + N(0,\sigma)$
- Key idea:
  - σ is part of the chromosome $\langle x_1,\ldots,x_n, \sigma \rangle$
  - σ is also mutated into σ' (see later how)
- Thus: mutation step size σ is coevolving with the solution x

Mutate σ first

- Net mutation effect: ⟨ x, σ ⟩ → ⟨ x′, σ′ ⟩
- Order is important:
  - first σ → σ′ (see later how)
  - then x → x′ = x + N(0,σ′)
- Rationale: new ⟨ x′ ,σ′ ⟩ is evaluated twice
  - Primary: x′ is good if f(x′) is good
  - Secondary: σ′ is good if the x′ it created is good
- Reversing mutation order this would not work

Mutation case 1: Uncorrelated mutation with one σ

- Chromosomes: $\langle x_1, \ldots, x_n, \sigma \rangle$
- $\sigma' = \sigma \cdot \exp(\tau \cdot N(0,1))$
- $x'_i = x_i + \sigma' \cdot N(0,1)$
- Typically the "learning rate" $\tau \propto 1/n^{1/2}$
- And we have a boundary rule $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$

Mutation case 2: Uncorrelated mutation with n σ's

- Chromosomes: $\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n \rangle$
- $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$
- $x'_i = x_i + \sigma'_i \cdot N_i(0,1)$
- Two learning rate parameters:
  - τ' overall learning rate
  - τ coordinate wise learning rate
- $\tau \propto 1/(2n)^{1/2}$ and $\tau \propto 1/(2n^{1/2})^{1/2}$
- And $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$

Mutation case 3: Correlated mutations

- Chromosomes: $\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n, \alpha_1, \ldots, \alpha_k \rangle$
- where $k = n \cdot (n-1)/2$
- and the covariance matrix C is defined as:
  - $c_{ii} = \sigma_i^2$
  - $c_{ij} = 0$ if i and j are not correlated
  - $c_{ij} = \frac{1}{2} \cdot (\sigma_i^2 - \sigma_j^2) \cdot \tan(2 \alpha_{ij})$ if i and j are correlated
- Note the numbering / indices of the α's

Correlated mutations cont′d

The mutation mechanism is then:

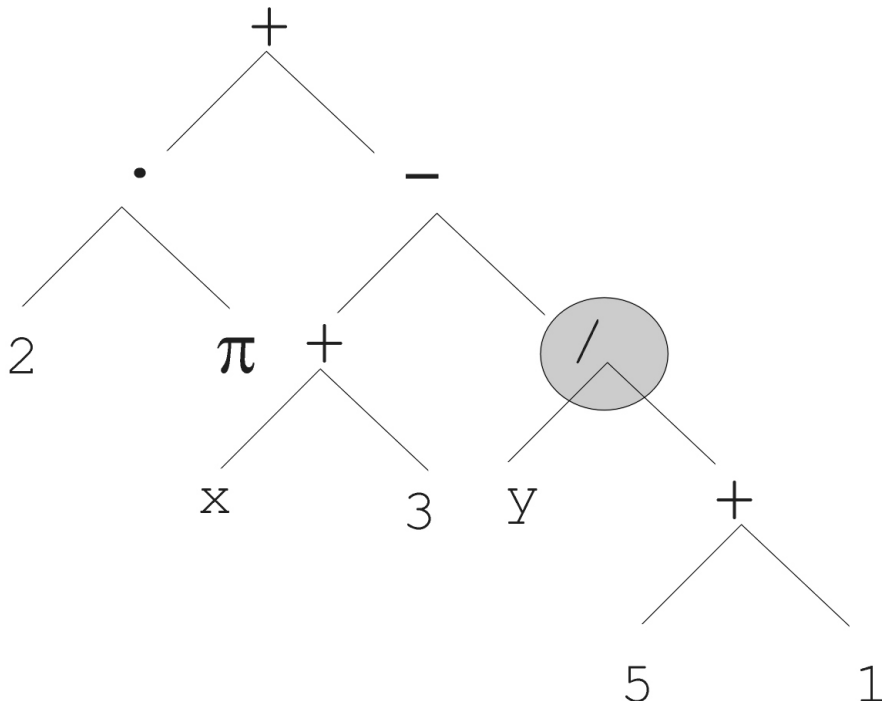
- $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$
- $\alpha'_j = \alpha_j + \beta \cdot N(0,1)$
- $\boldsymbol{x}' = \boldsymbol{x} + N(\boldsymbol{0}, \boldsymbol{C}')$
  - $\boldsymbol{x}$ stands for the vector $\langle x_1, \ldots, x_n \rangle$
  - $\boldsymbol{C}'$ is the covariance matrix $\boldsymbol{C}$ after mutation of the α values
- $\tau \propto 1/(2n)^{1/2}$ and $\tau \propto 1/(2n^{1/2})^{1/2}$ and $\beta \approx 5°$
- $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$ and
- $|\alpha'_j| > \pi \Rightarrow \alpha'_j = \alpha'_j - 2\pi \operatorname{sign}(\alpha'_j)$

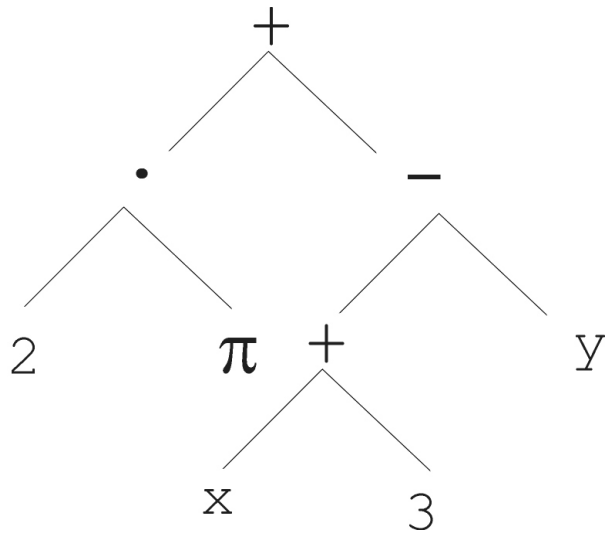
Mutation (EP)

- Chromosomes: $\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n \rangle$
- $\sigma_i' = \sigma_i \cdot (1 + \alpha \cdot N(0,1))$
- $x'_i = x_i + \sigma_i' \cdot N_i(0,1)$
- $\alpha \approx 0.2$
- boundary rule: $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$
- Other variants proposed & tried:
  - Lognormal scheme as in ES
  - Using variance instead of standard deviation
  - Mutate σ-last
  - Other distributions, e.g, Cauchy instead of Gaussian

Mutation (GP)

- Most common mutation: replace randomly chosen subtree by randomly generated tree





Mutation cont'd

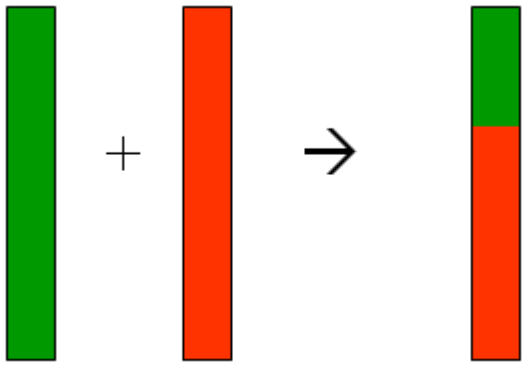
- Mutation has two parameters:
  - Probability p_m to choose mutation vs. recombination
  - Probability to chose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98)
- The size of the child can exceed the size of the parent

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

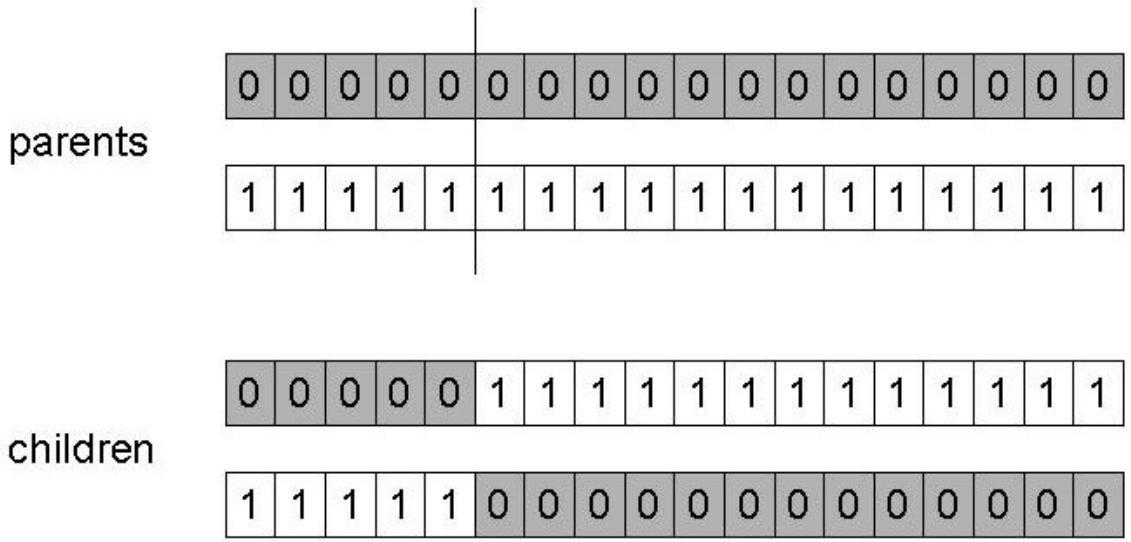
Crossover operator:

- defines how to select exchanged parts of the genetic material;
- e.g., randomly selecting a chromosome splitting position;



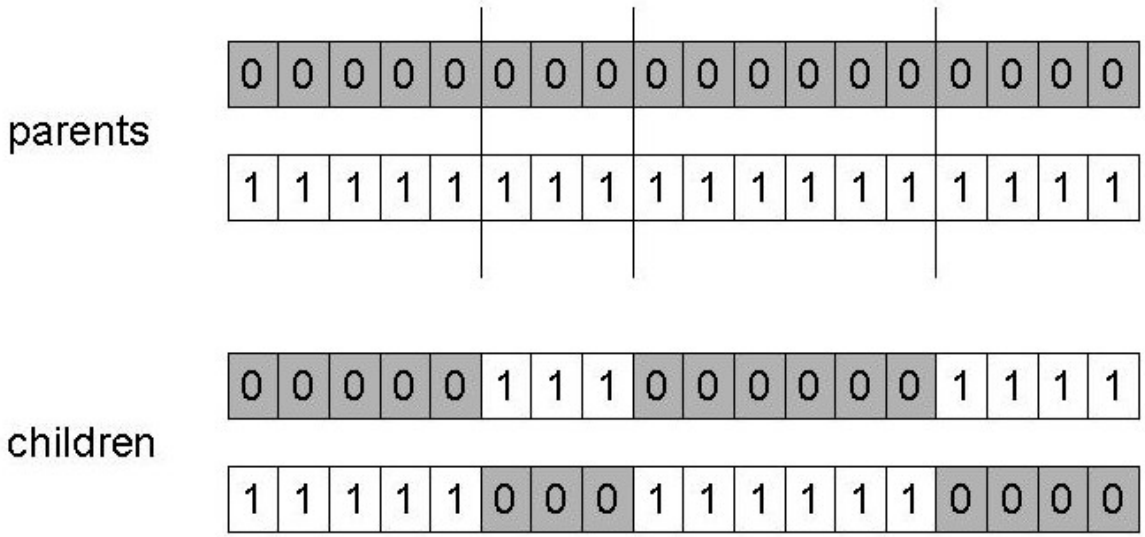
Binary encoding : 1-point crossover (SGA)

- 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)



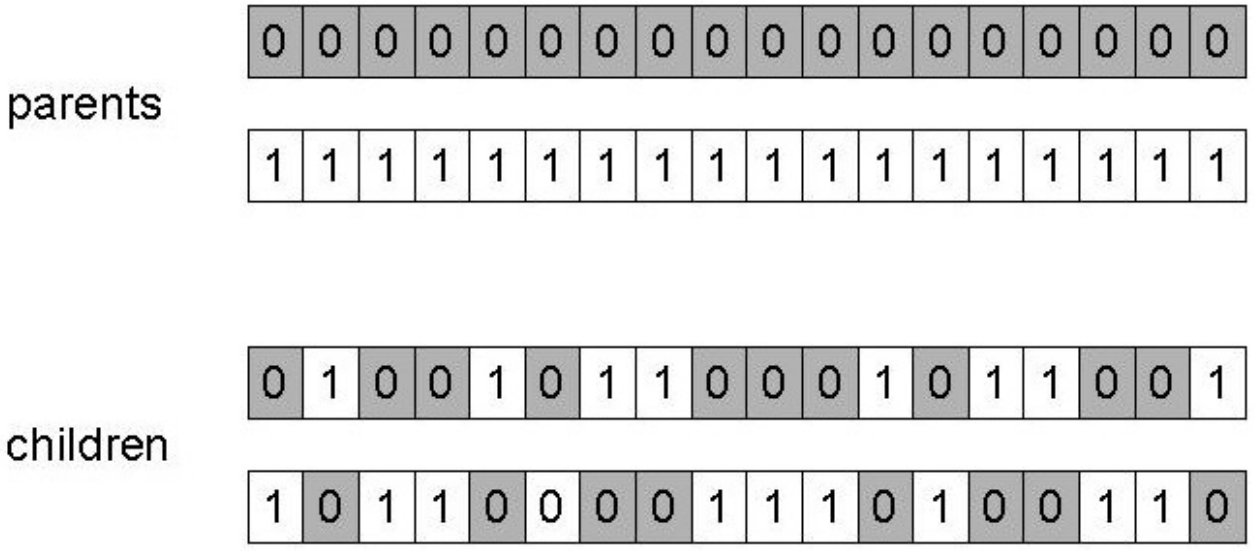
n-point crossover

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



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



Integer representations crossover

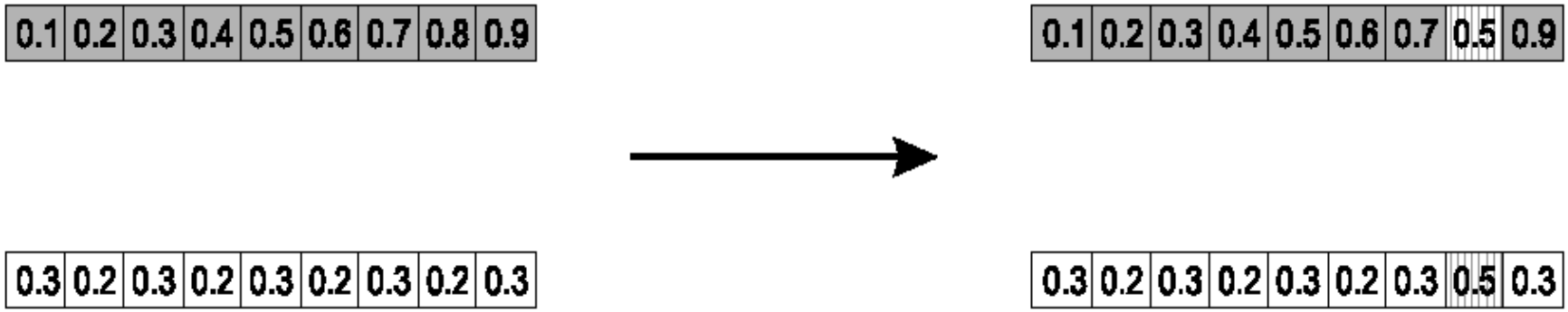
- Some problems naturally have integer variables, e.g. image processing parameters
- Others take *categorical* values from a fixed set e.g. {blue, green, yellow, pink}
- N-point / uniform crossover operators work

Crossover operators for real valued GAs

- Discrete:
  - 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
- Intermediate
  - exploits idea of creating children "between" parents (hence a.k.a. *arithmetic* recombination)
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ where α : $0 \leq \alpha \leq 1$.
  - The parameter α can be:
    - constant: uniform arithmetical crossover
    - 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,
- child$_1$ is: $\left\langle x_1, ..., x_k, \alpha \cdot y_k + (1-\alpha) \cdot x_k, ..., x_n \right\rangle$
- reverse for other child. e.g. with α = 0.5

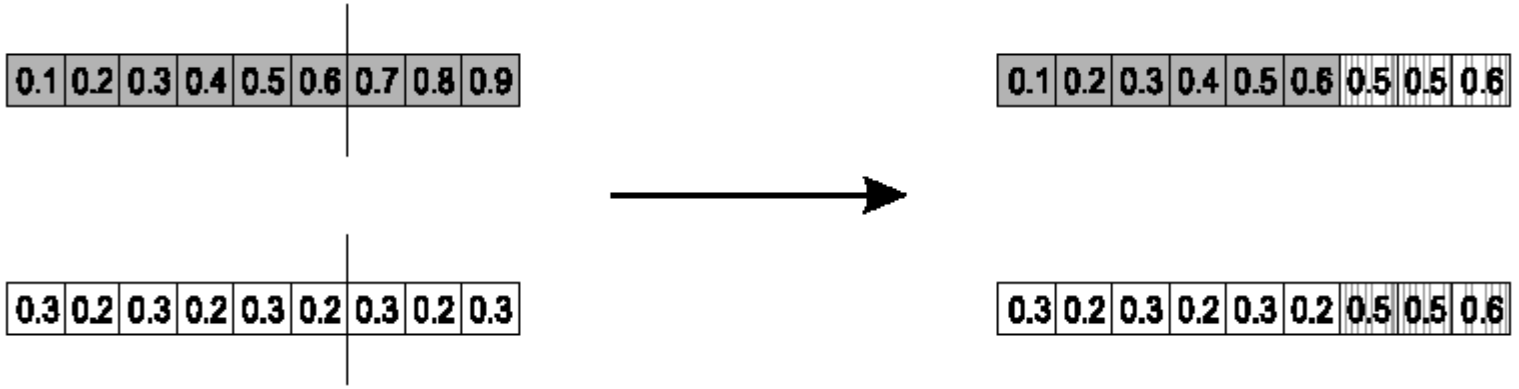


Simple arithmetic crossover

- 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:

$$\left\langle x_1,...,x_k,\alpha\cdot y_{k+1}+(1-\alpha)\cdot x_{k+1},...,\alpha\cdot \mathrm{y}_{\mathrm{n}}+(1-\alpha)\cdot x_n \right\rangle$$

- reverse for other child. e.g. with α = 0.5

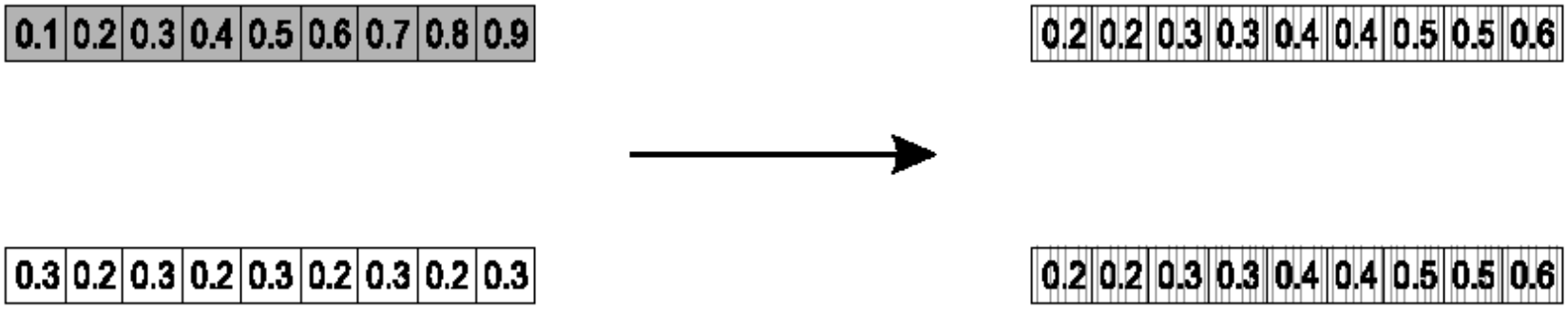


Whole arithmetic crossover

- Most commonly used
- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- $child_1$ is:

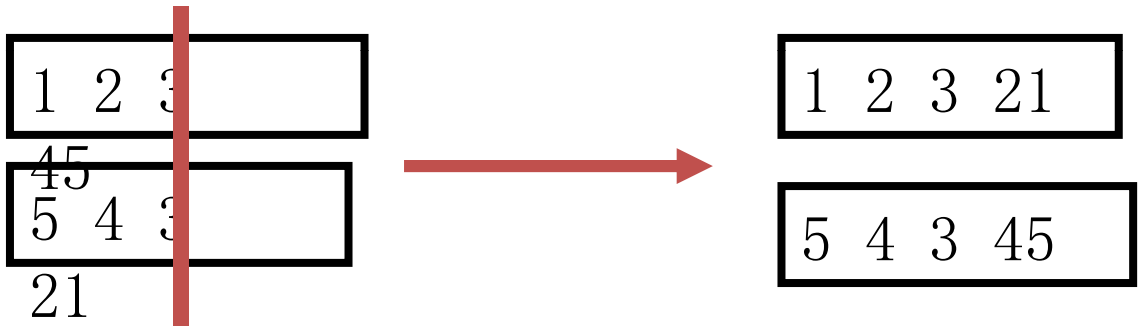
$$a \cdot \bar{x} + (1-a) \cdot \bar{y}$$

- reverse for other child. e.g. with $\alpha = 0.5$



Crossover operators for permutations

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



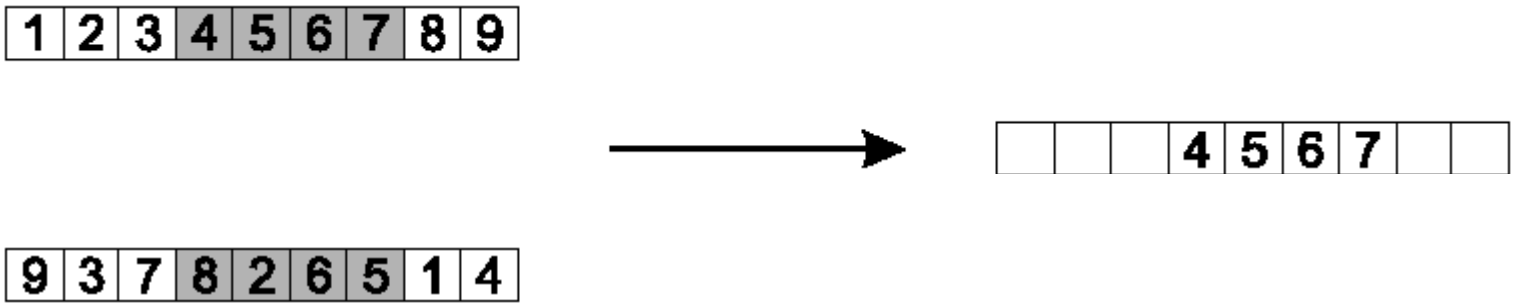
- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Order 1 crossover

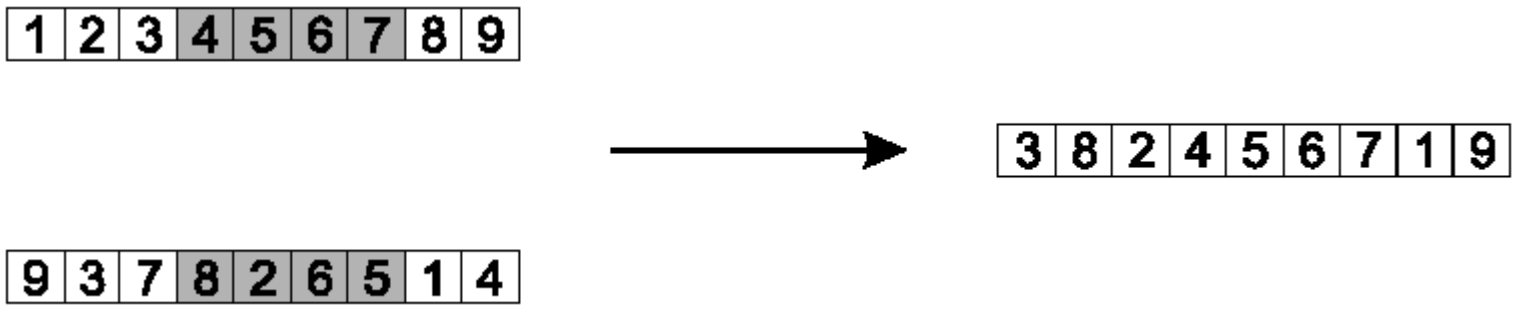
- Idea is to preserve relative order that elements occur
- Informal procedure:
  1. Choose an arbitrary part from the first parent
  2. Copy this part to the first child
  3. Copy the numbers that are not in the first part, to the first child:
     - starting right from cut point of the copied part,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

Order 1 crossover example

- Copy randomly selected set from first parent



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



Partially Mapped Crossover (PMX)

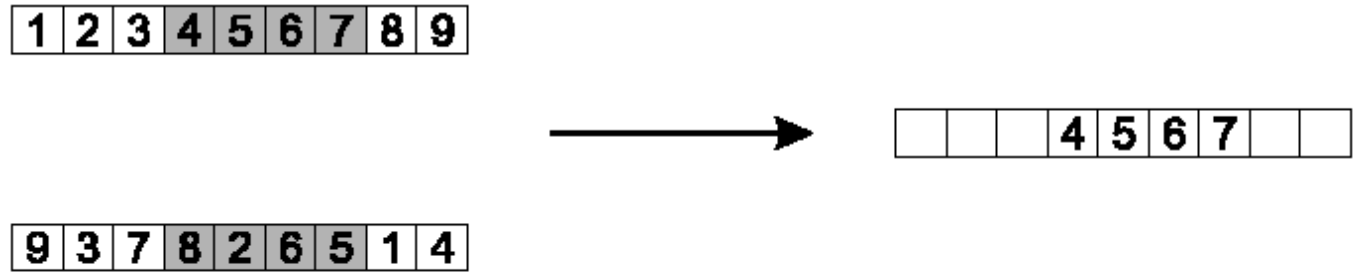
Informal procedure for parents P1 and P2:

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.

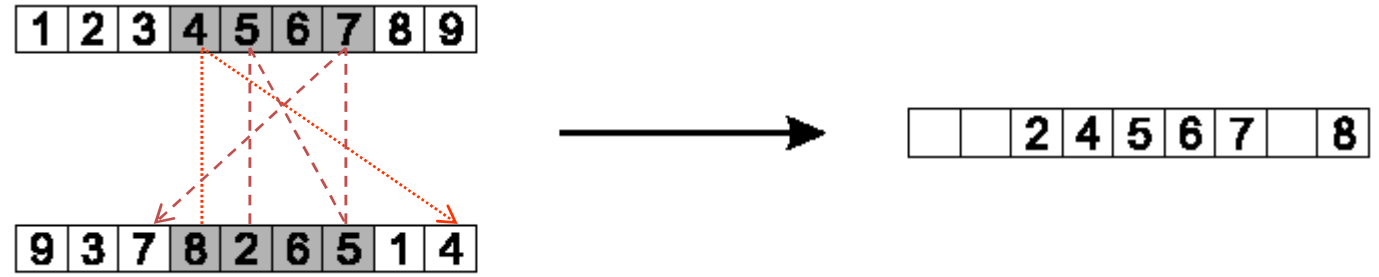
Second child is created analogously

PMX example

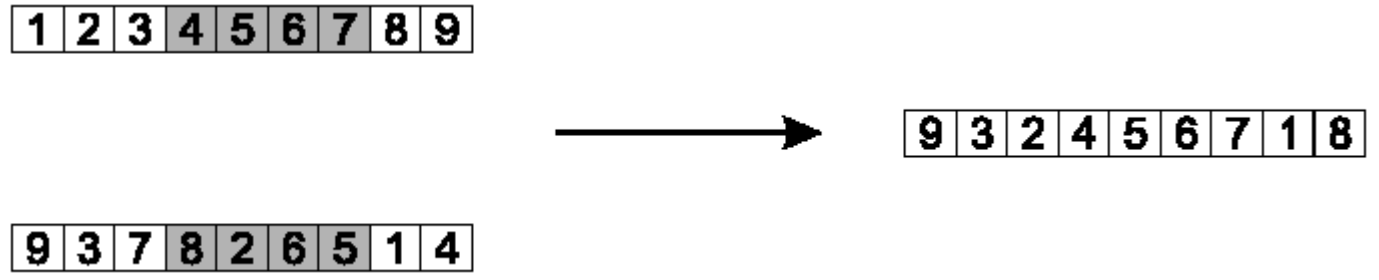
- Step 1



- Step 2



- Step 3



Cycle crossover

Basic idea:

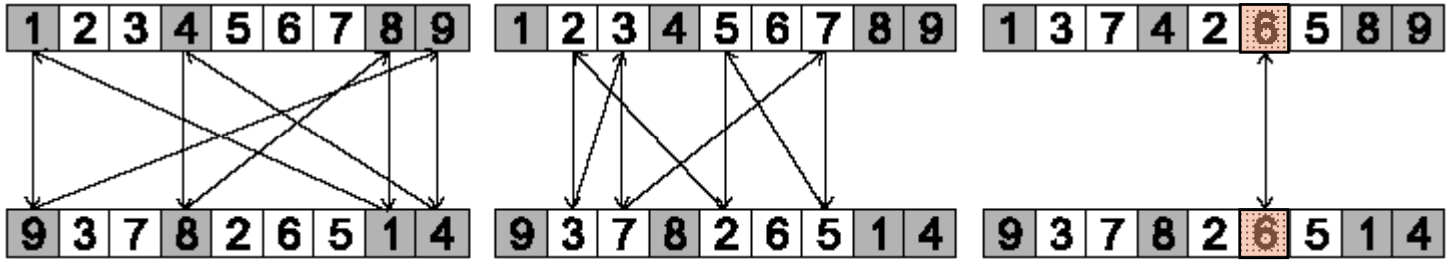
Each allele comes from one parent *together with its position*.

Informal procedure:

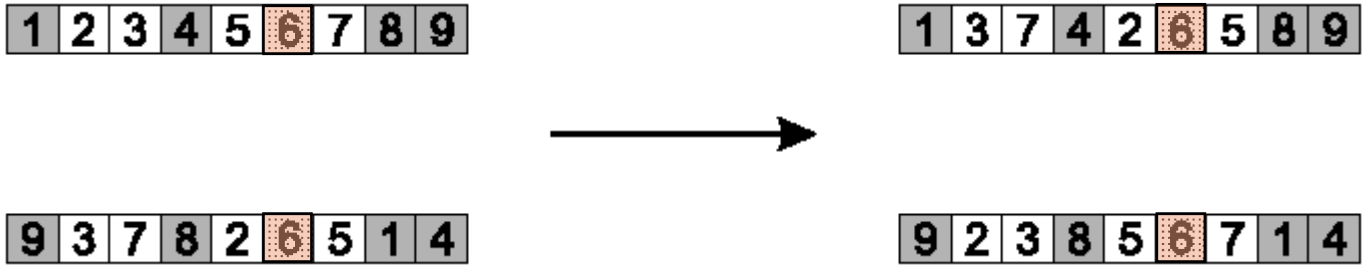
1. Make a cycle of alleles from P1 in the following way
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the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

Cycle crossover example

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



Edge Recombination

- Works by constructing a 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

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*]

Recombination (ES)

- Creates one child
- Acts per variable / position by either
  - Averaging parental values, or
  - Selecting one of the parental values
- From two or more parents by either:
  - Using two selected parents to make a child
  - Selecting two parents for each position anew

Names of recombination's (ES)

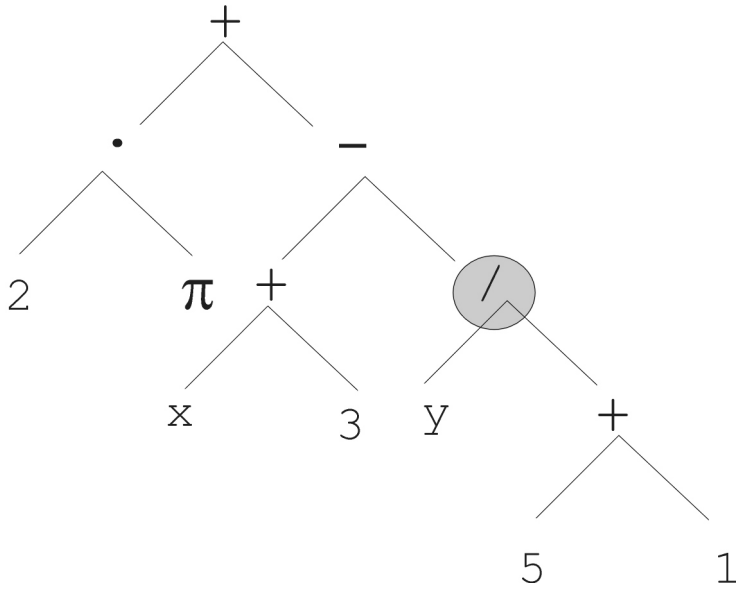
	Two fixed parents	Two parents selected for each i
$z_i = (x_i + y_i)/2$	Local intermediary	Global intermediary
z_i is x_i or y_i chosen randomly	Local discrete	Global discrete

Recombination (EP)

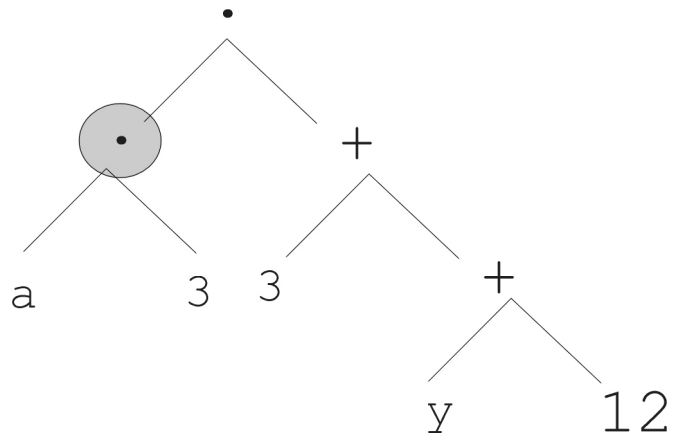
- None
- Rationale: one point in the search space stands for a species, not for an individual and there can be no crossover between species
- Much historical debate “mutation vs. crossover”
- Pragmatic approach seems to prevail today

Recombination(GP)

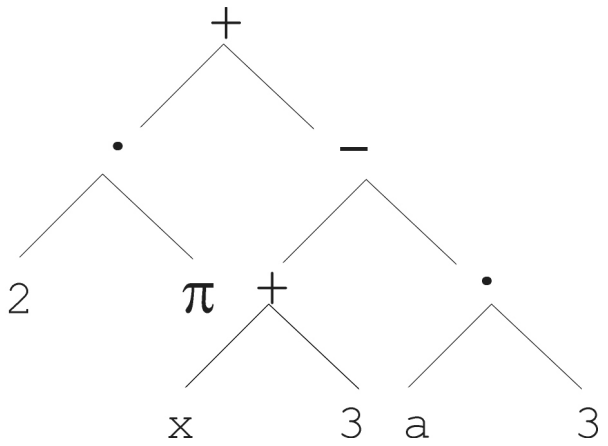
- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
  - Probability p_c to choose recombination vs. mutation
  - Probability to chose an internal point within each parent as crossover point
- The size of offspring can exceed that of the parents



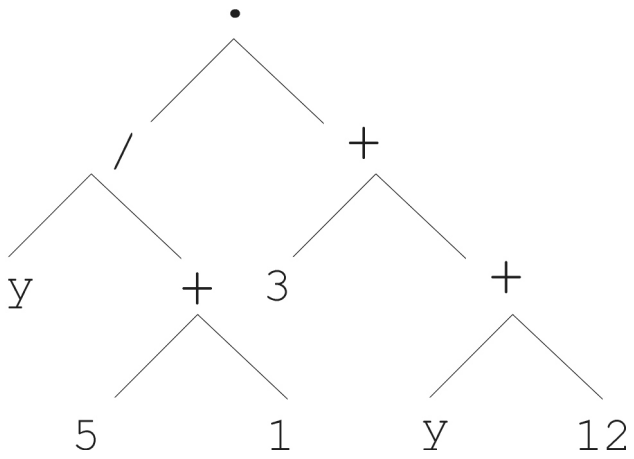
Parent 1



Parent 2



Child 1



Child 2

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)

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

Multi parent recombination (Special case)

- Recall that we are not constricted by the practicalities of nature
- Noting that mutation uses 1 parent, and "traditional" crossover 2, the extension to $a>2$ is natural to examine
- Been around since 1960s, still rare but studies indicate useful
- 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

Components of Evolutionary Algorithms

- Representations
- Evaluation (Fitness) Function
- Population
- Selection (Parent Selection , Survivor Selection)
- Variation Operators(mutation, cross over)
- Initialisation / Termination

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

Genetic Algorithm

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

Working Principles

Before getting into GAs, it is necessary to explain few terms.

- Chromosome : a set of genes; a chromosome contains the solution in form of genes.
- Gene : a part of chromosome; a gene contains a part of solution. It determines the solution. e.g. 16743 is a chromosome and 1, 6, 7, 4 and 3 are its genes.
- Individual : same as chromosome.
- Population: number of individuals present with same length of chromosome.
- Fitness : the value assigned to an individual based on how far or close a individual is from the solution; greater the fitness value better the solution it contains.
- Fitness function : a function that assigns fitness value to the individual. It is problem specific.
- Breeding : taking two fit individuals and then intermingling there chromosome to create new two individuals.
- Mutation : changing a random gene in an individual.
- Selection : selecting individuals for creating the next generation.

Working principles :

Genetic algorithm begins with a set of solutions (represented by chromosomes) called the population.

- Solutions from one population are taken and used to form a new population. This is motivated by the possibility that the new population will be better than the old one.
- Solutions are selected according to their fitness to form new solutions (offspring); more suitable they are, more chances they have to reproduce.
- This is repeated until some condition (e.g. number of populations or improvement of the best solution) is satisfied.

Outline of the Basic Genetic Algorithm

1. [Start] Generate random population of **n** chromosomes (i.e. suitable solutions for the problem).
2. [Fitness] Evaluate the fitness **f(x)** of each chromosome **x** in the population.
3. [New population] Create a new population by repeating following steps until the new population is complete.
   (a) [Selection] Select two parent chromosomes from a population according to their fitness (better the fitness, bigger the chance to be selected)
   (b) [Crossover] With a crossover probability, cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.
   (c) [Mutation] With a mutation probability, mutate new offspring at each locus (position in chromosome).
   (d) [Accepting] Place new offspring in the new population
4. [Replace] Use new generated population for a further run of the algorithm
5. [Test] If the end condition is satisfied, stop, and return the best solution in current population
6. [Loop] Go to step 2

Note : The genetic algorithm's performance is largely influenced by two operators called crossover and mutation. These two operators are the most important parts of GA.

Encoding

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form so that a computer can process.

- One common approach is to encode solutions as binary strings: sequences of **1's** and **0's**, where the digit at each position represents the value of some aspect of the solution.

Example :

A Gene represents some data (eye color, hair color, sight, etc.).

A chromosome is an array of genes. In binary form

a **Gene** looks like : **(11100010)**

a **Chromosome** looks like: **Gene1 Gene2 Gene3 Gene4**

(11000010, 00001110, 001111010, 10100011)

A chromosome should in some way contain information about solution which it represents; it thus requires encoding. The most popular way of encoding is a **binary string** like :

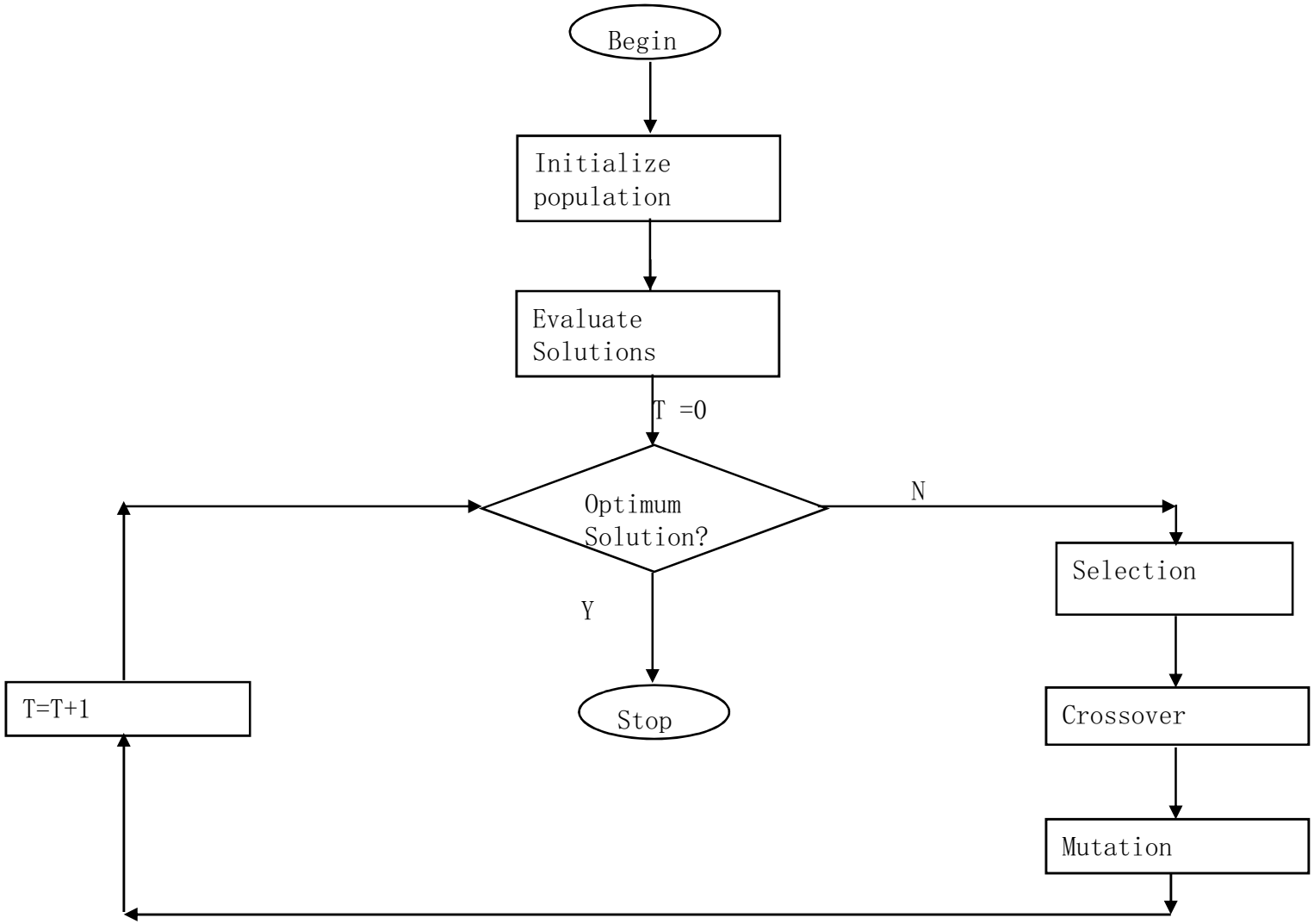
Chromosome 1 : 1101100100110110

Chromosome 2 : 1101111000011110

Each bit in the string represent some characteristics of the solution.

- There are many other ways of encoding, e.g., encoding values as integer or real numbers or some permutations and so on.
- The virtue of these encoding method depends on the problem to work on .

Working Mechanism Of GAs



Some GA Application Types

Domain	Application Types
Control	gas pipeline, pole balancing, missile evasion, pursuit
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, checkers, prisoner's dilemma
Combinatorial Optimization	set covering, travelling salesman, routing, bin packing, graph colouring and partitioning

An example after Goldberg '89 (1)

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

Basic Genetic Algorithm :

Examples to demonstrate and explain : Random population, Fitness, Selection, Crossover, Mutation, and Accepting.

- **Example 1 :**

  Maximize the function **$f(x) = x^2$** over the range of integers from **0 . . . 31**.

  Note : This function could be solved by a variety of traditional methods such as a hill-climbing algorithm which uses the derivative.

  One way is to :

  - Start from any integer **x** in the domain of **f**
  - Evaluate at this point **x** the derivative **f'**
  - Observing that the derivative is **+ve**, pick a new **x** which is at a small distance in the **+ve** direction from current **x**
  - Repeat until **x = 31**

  See, how a genetic algorithm would approach this problem ?

Genetic Algorithm Approach to problem - Maximize the function $f(x) = x^2$

1. Devise a means to represent a solution to the problem :

   Assume, we represent **x** with five-digit unsigned binary integers.

2. Devise a heuristic for evaluating the fitness of any particular solution :

   The function **f(x)** is simple, so it is easy to use the **f(x)** value itself to rate the fitness of a solution; else we might have considered a more simpler heuristic that would more or less serve the same purpose.

3. Coding - Binary and the String length :

   GAs often process binary representations of solutions. This works well, because crossover and mutation can be clearly defined for binary solutions. A Binary string of length **5** can represents 32 numbers (0 to 31).

4. Randomly generate a set of solutions :

   Here, considered a population of four solutions. However, larger populations are used in real applications to explore a larger part of the search. Assume, four randomly generated solutions as : **01101, 11000, 01000, 10011**. These are chromosomes or genotypes.

5. Evaluate the fitness of each member of the population :

The calculated fitness values for each individual are -

(a) Decode the individual into an integer (called phenotypes),

01101 → 13; 11000 → 24; 01000 → 8; 10011 → 19;

(b) Evaluate the fitness according to **$f(x) = x^2$** ,

13 → 169; 24 → 576; 8 → 64; 19 → 361.

(c) Expected count = **N * Prob i** , where **N** is the number of individuals in the population called population size, here **N = 4**.

Thus the evaluation of the initial population summarized in table below .

String No i	Initial Population (chromosome)	X value (Pheno types)	Fitness $f(x) = x^2$	Prob i (fraction of total)	Expected count N * Prob i
1	0 1 1 0 1	13	169	0.14	0.58
2	1 1 0 0 0	24	576	0.49	1.97
3	0 1 0 0 0	8	64	0.06	0.22
4	1 0 0 1 1	19	361	0.31	1.23
Total (sum)			1170	1.00	4.00
Average			293	0.25	1.00
Max			576	0.49	1.97

Thus, the string no 2 has maximum chance of selection.

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	1 1 1 0 0	28	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	20	400
Sum				2538
Average				634.5
Max				784

The Problem

The **Traveling Salesman Problem** is defined as:

'We are given a set of cities and a symmetric distance matrix that indicates the cost of travel from each city to every other city.

*The goal is to find **the shortest circular tour**, visiting every city exactly once, so as to **minimize the total travel cost**, which includes the cost of traveling from the last city back to the first city'.*

Encoding

- represent every city with an integer .
- Consider 6 Indian cities –

  Mumbai, Nagpur , Calcutta, Delhi , Bangalore and Chennai and assign a number to each.

Mumbai → 1
Nagpur → 2
Calcutta → 3
Delhi → 4
Bangalore → 5
Chennai → 6

Encoding (contd.)

- Thus a path would be represented as a sequence of integers from 1 to 6.

- The path [1 2 3 4 5 6] represents a path from Mumbai to Nagpur, Nagpur to Calcutta, Calcutta to Delhi, Delhi to Bangalore, Bangalore to Chennai, and finally from Chennai to Mumbai.

- This is an example of Permutation Encoding as the position of the elements determines the fitness of the solution.

Fitness Function

- The fitness function will be the total cost of the tour represented by each chromosome.

- This can be calculated as the sum of the distances traversed in each travel segment.

The Lesser The Sum, The Fitter The Solution Represented By That Chromosome.

Distance/Cost Matrix For TSP

	1	2	3	4	5	6
1	0	863	1987	1407	998	1369
2	863	0	1124	1012	1049	1083
3	1987	1124	0	1461	1881	1676
4	1407	1012	1461	0	2061	2095
5	998	1049	1881	2061	0	331
6	1369	1083	1676	2095	331	0

Cost matrix for six city example.

Distances in Kilometers

Fitness Function (contd.)

- So, for a chromosome [4 1 3 2 5 6], the total cost of travel or fitness will be calculated as shown below

- Fitness $= 1407 + 1987 + 1124 + 1049 + 331 + 2095$
  $= 7993$ *kms.*

- Since our objective is to Minimize the distance, the lesser the total distance, the fitter the solution.

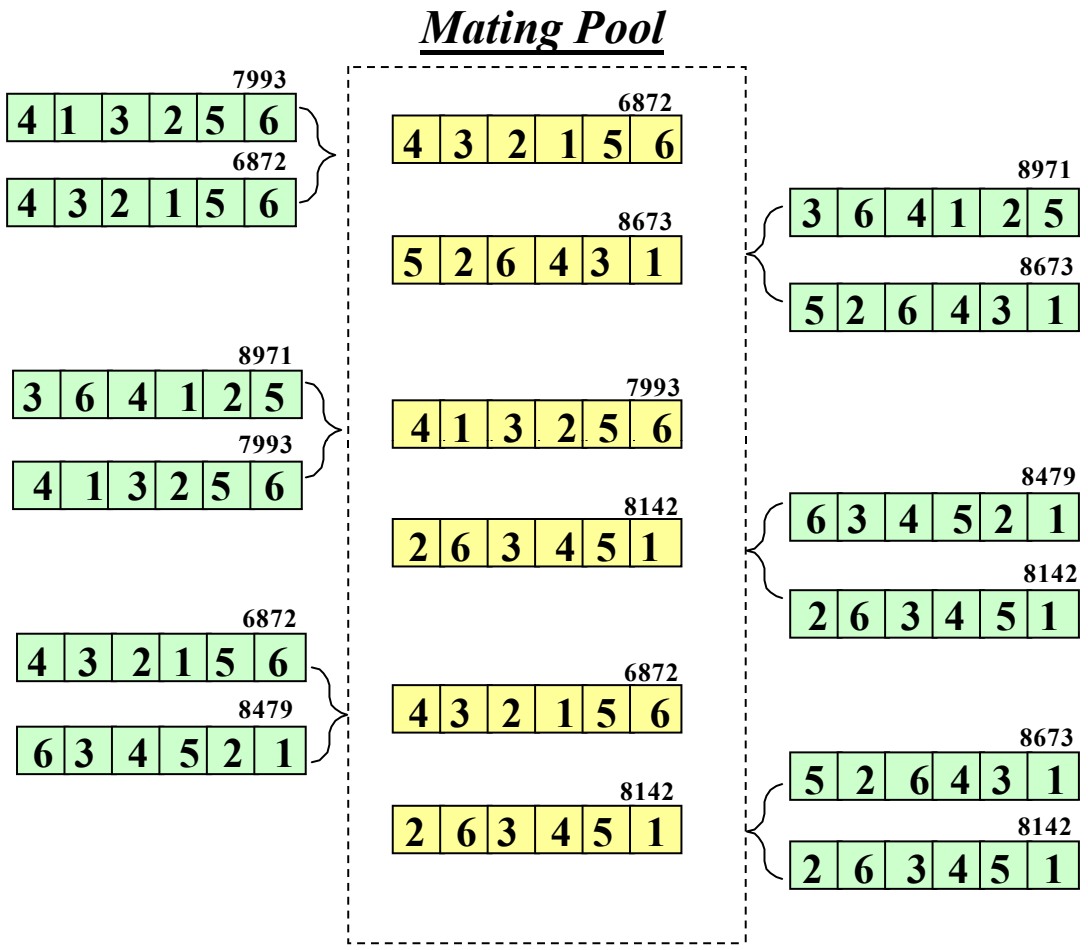
Selection Operator

I used *Tournament Selection*.

As the name suggests *tournaments* are played between two solutions and the better solution is chosen and placed in the *mating pool*.

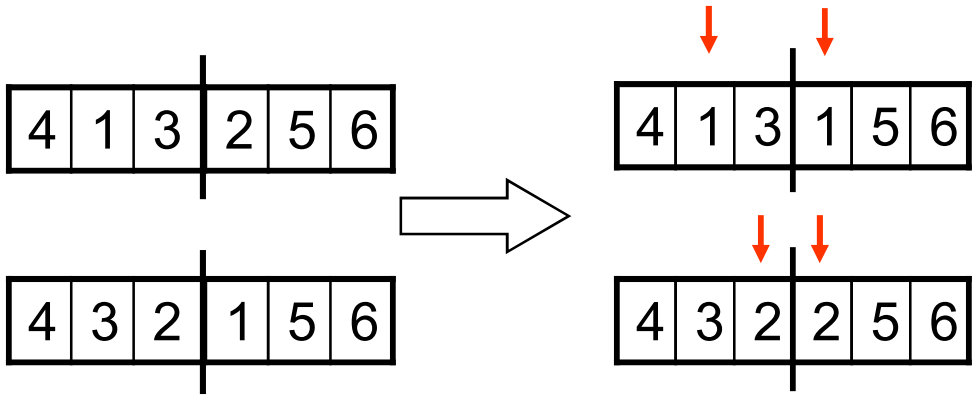
Two other solutions are picked again and another slot in the *mating pool* is filled up with the better solution.

Tournament Selection (contd.)



Why we cannot use single-point crossover:

- Single point crossover method randomly selects a crossover point in the string and swaps the substrings.
- This may produce some invalid offsprings as shown below.



Crossover Operator

- I used the *Enhanced Edge Recombination* operator (T.Starkweather, et al, *'A Comparison of Genetic Sequencing Operators,* International Conference of GAs, 1991) .

- This operator is different from other genetic sequencing operators in that it emphasizes *adjacency information* instead of the order or position of items in the sequence.

- The algorithm for the Edge-Recombination Operator involves constructing an Edge Table first.

Edge Table

The *Edge Table* is an *adjacency table* that lists links *into* and *out of* a city found in the two parent sequences.

If an item is already in the edge table and we are trying to insert it again, that element of a sequence must be a *common edge* and is represented by inverting it's sign.

Finding The Edge Table

Parent 1	**4**	**1**	**3**	**2**	**5**	**6**
Parent 2	**4**	**3**	**2**	**1**	**5**	**6**

1	4	3	2	5
2	-3	5	1	
3	1	-2	4	
4	-6	1	3	
5	1	2	-6	
6	-5	-4		

Enhanced Edge Recombination Algorithm

1. Choose the initial city from one of the two parent tours. (It can be chosen randomly as according to criteria outlined in *step 4*). This is the *current city.*

2. Remove all occurrences of the *current city* from the left hand side of the edge table.(These can be found by referring to the edge-list for the *current city*).

3. If the *current city* has entries in it's edge-list, go to s*tep 4* otherwise go to *step 5.*

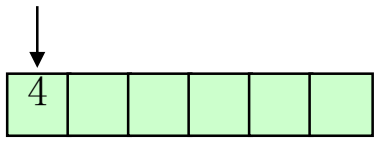
4. Determine which of the cities in the edge-list of the *current city* has the fewest entries in it's own edge-list. The city with fewest entries becomes the *current city.* In case a negative integer is present, it is given preference. Ties are broken randomly. Go to *step 2.*

5. If there are no remaining *unvisited* cities, then *stop.* Otherwise, randomly choose an *unvisited* city and go to *step 2.*

Example Of Enhanced Edge Recombination Operator

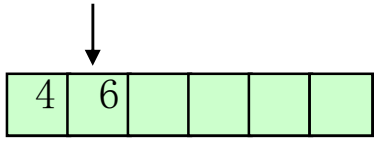
Step 1

1	4	3	2	5
2	-3	5	1	
3	1	-2	4	
4	-6	1	3	
5	3	2	-6	
6	-5	-4		



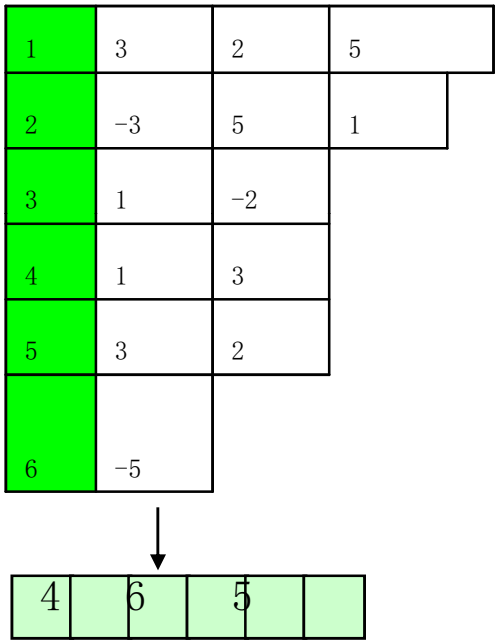
Step 2

1	3	2	5
2	-3	5	1
3	1	-2	
4	-6	1	3
5	3	2	-6
6	-5		

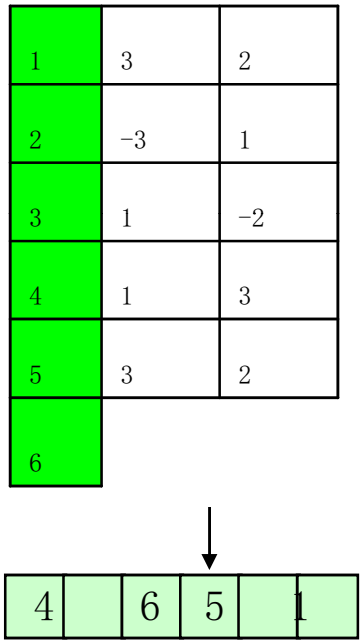


Example Of Enhanced Edge Recombination Operator (contd.)

Step 3

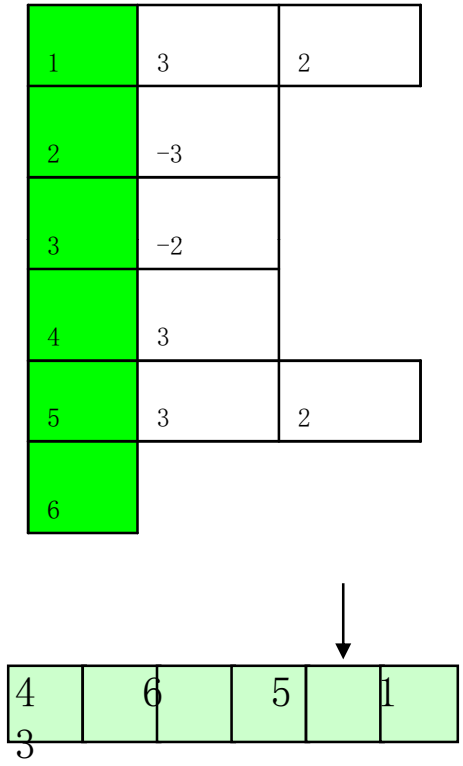


Step 4

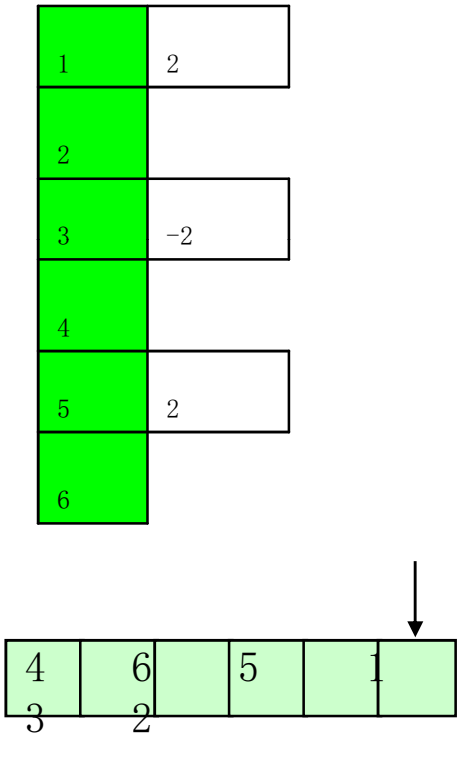


Example Of Enhanced Edge Recombination Operator (contd.)

Step 5

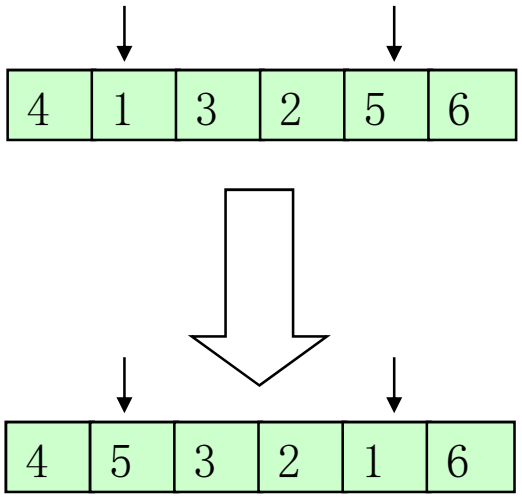


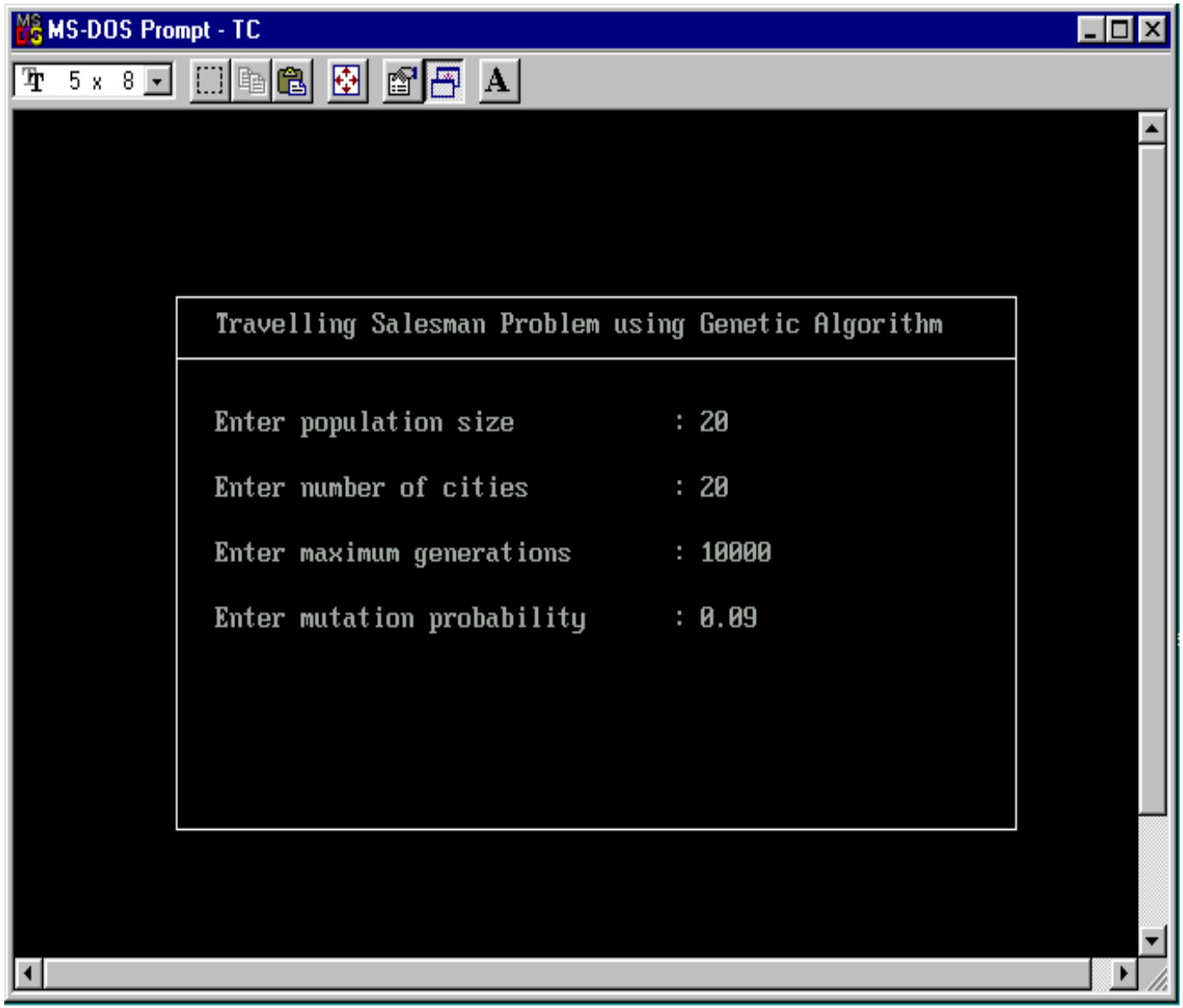
Step 6



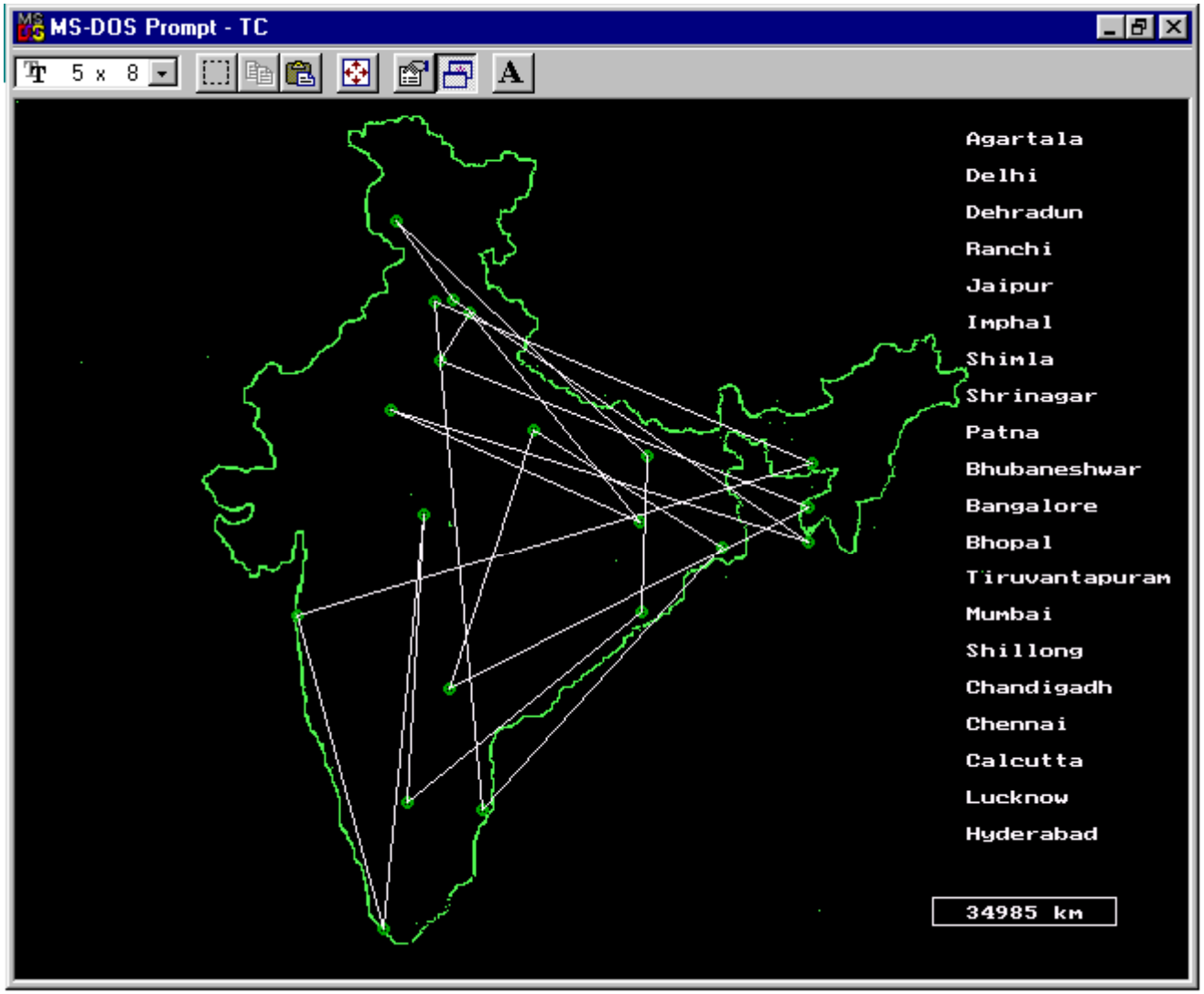
Mutation Operator

- The mutation operator induces a change in the solution, so as to maintain diversity in the population and prevent *Premature Convergence*.
- In our project, we mutate the string by randomly selecting any two cities and interchanging their positions in the solution, thus giving rise to a new tour.

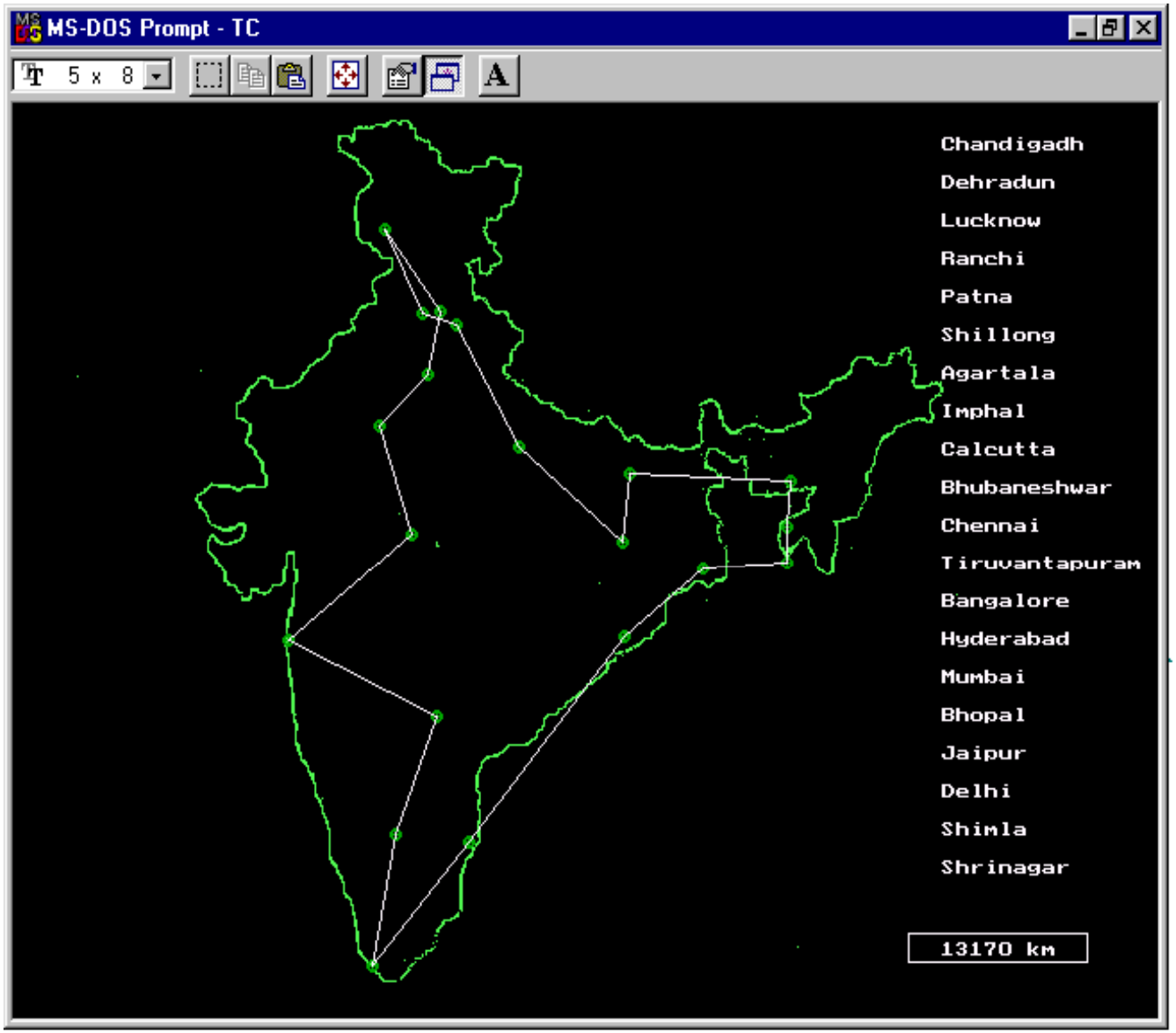




Input To Program



Initial Output For 20 cities : Distance=34985 km
Initial Population



Final Output For 20 cities : Distance=13170 km
Generation 4786

ES technical summary tableau

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	(μ,λ) or $(\mu+\lambda)$
Specialty	Self-adaptation of mutation step sizes

Evolutionary Programming

EP technical summary tableau

Representation	Real-valued vectors
Recombination	None
Mutation	Gaussian perturbation
Parent selection	Deterministic
Survivor selection	Probabilistic ($\mu+\mu$)
Specialty	Self-adaptation of mutation step sizes (in meta-EP)

GP technical summary tableau

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

Introductory example: credit scoring

- Bank wants to distinguish good from bad loan applicants
- Model needed that matches historical data

ID	No of children	Salary	Marital status	OK?
ID-1	2	45000	Married	0
ID-2	0	30000	Single	1
ID-3	1	40000	Divorced	1
…				

Introductory example: credit scoring

- A possible model:

  IF (NOC = 2) AND (S > 80000) THEN good ELSE bad

- In general:

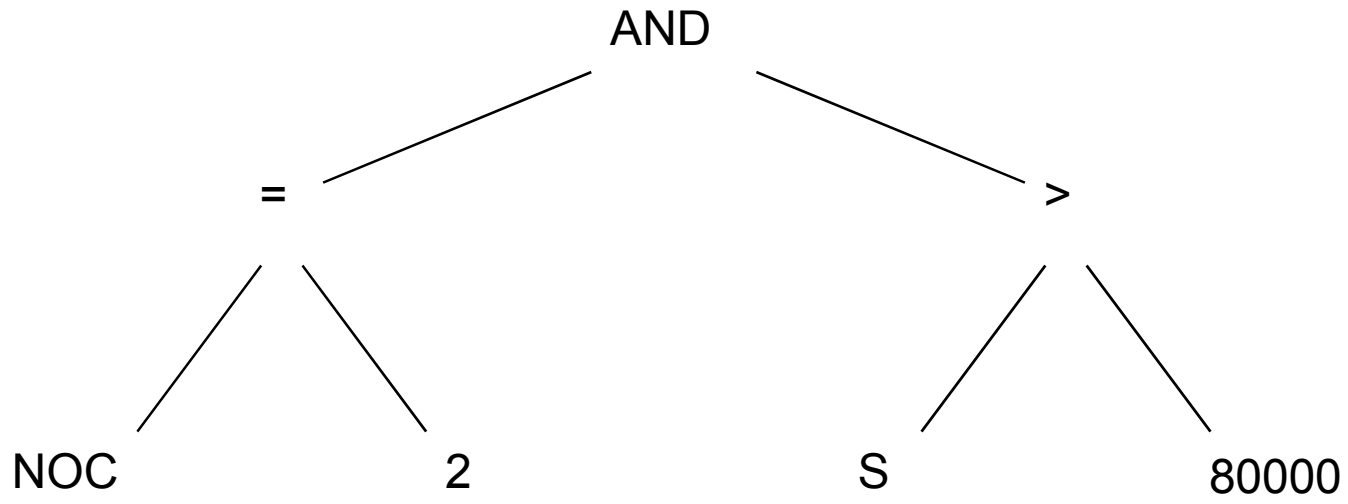
  IF formula THEN good ELSE bad

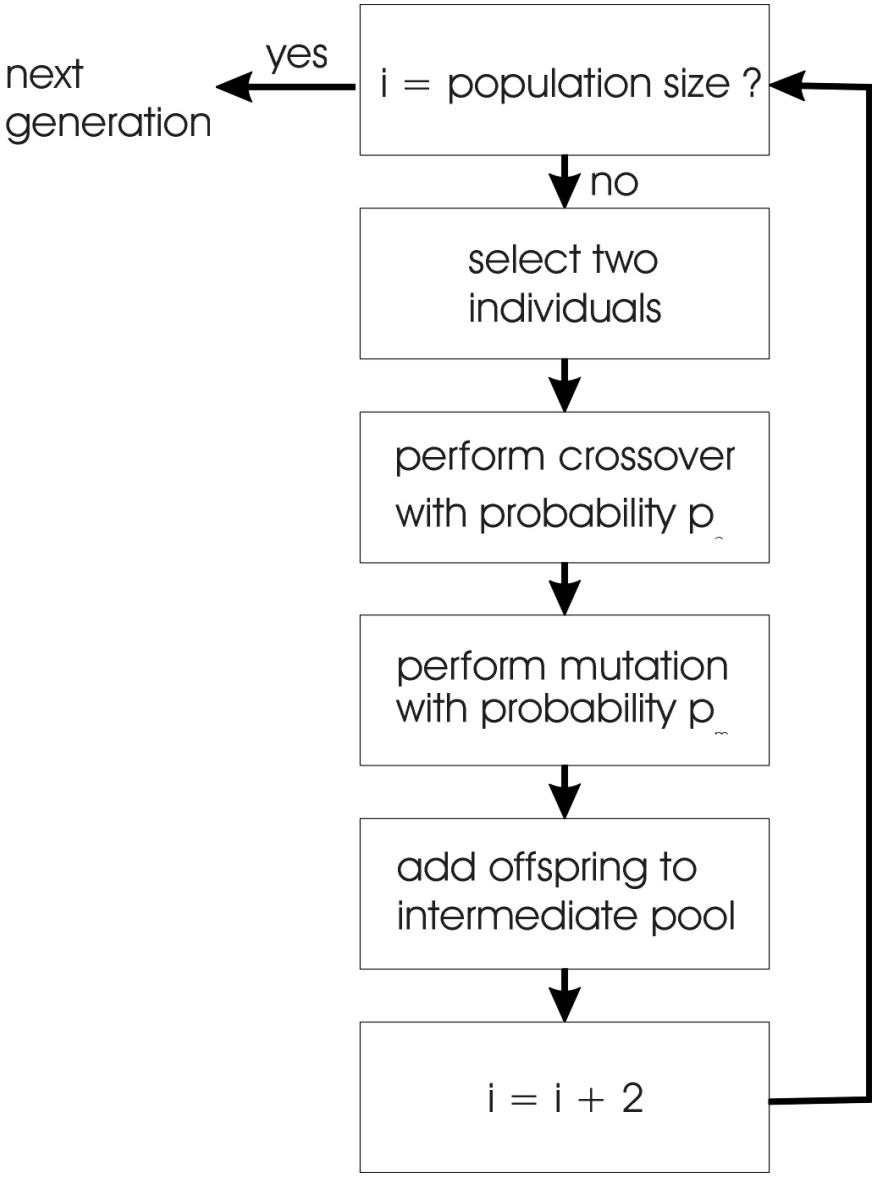
- Only unknown is the right formula, hence
- Our search space (phenotypes) is the set of formulas
- Natural fitness of a formula: percentage of well classified cases of the model it stands for
- Natural representation of formulas (genotypes) is: parse trees

Introductory example: credit scoring

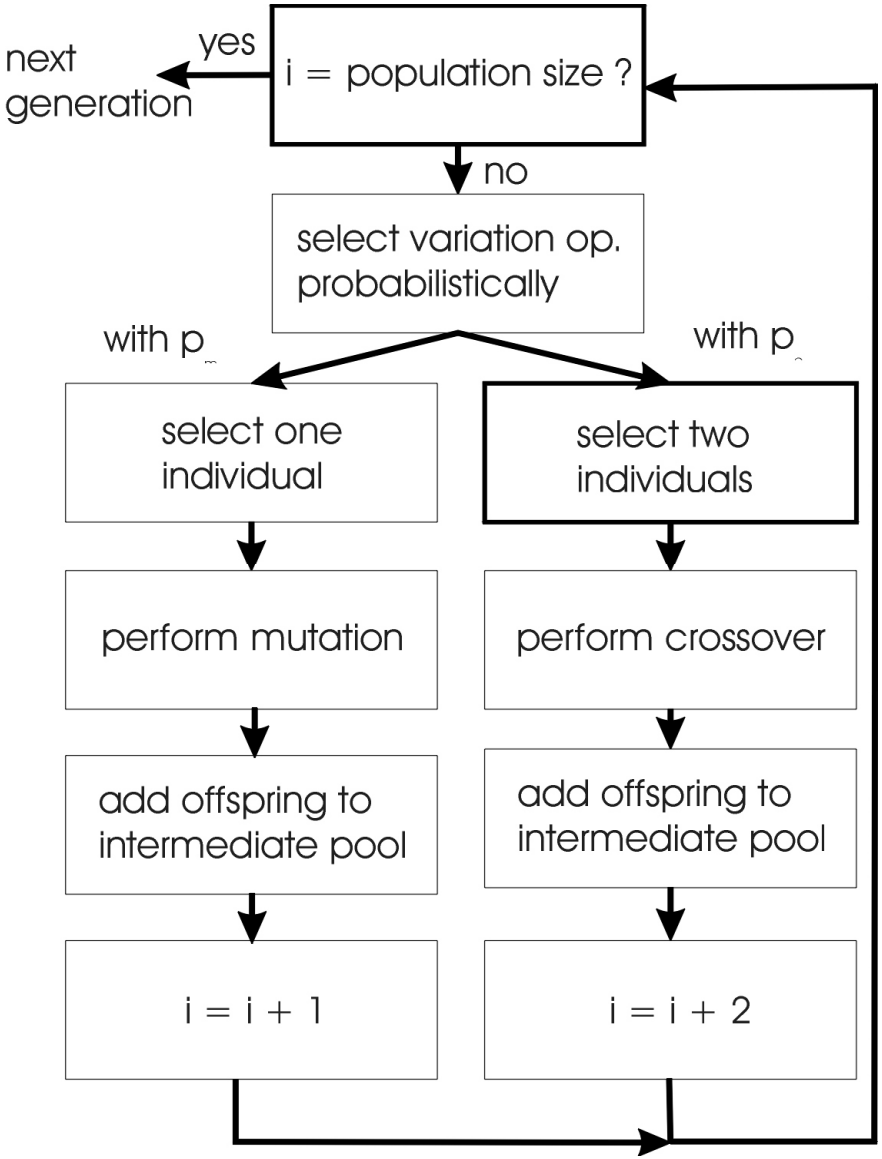
IF (NOC = 2) AND (S > 80000) THEN good ELSE bad

can be represented by the following tree





GA flowchart



GP flowchart

References and Resouces for this Lecture

- Books
  - A.E.Eiben and J.E.Smith, Introduction to Evolutionary computing,Natural computiong series,Springer.
  - Dan simon,Evolutionary Optimization Algorithms,wiley .