Local Search Techniques

Dr. Fayyaz ul Amir Afsar Minhas

PIEAS Biomedical Informatics Research Lab

Department of Computer and Information Sciences

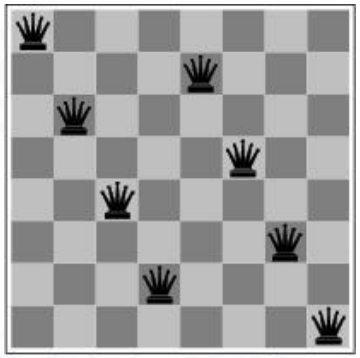
Pakistan Institute of Engineering & Applied Sciences

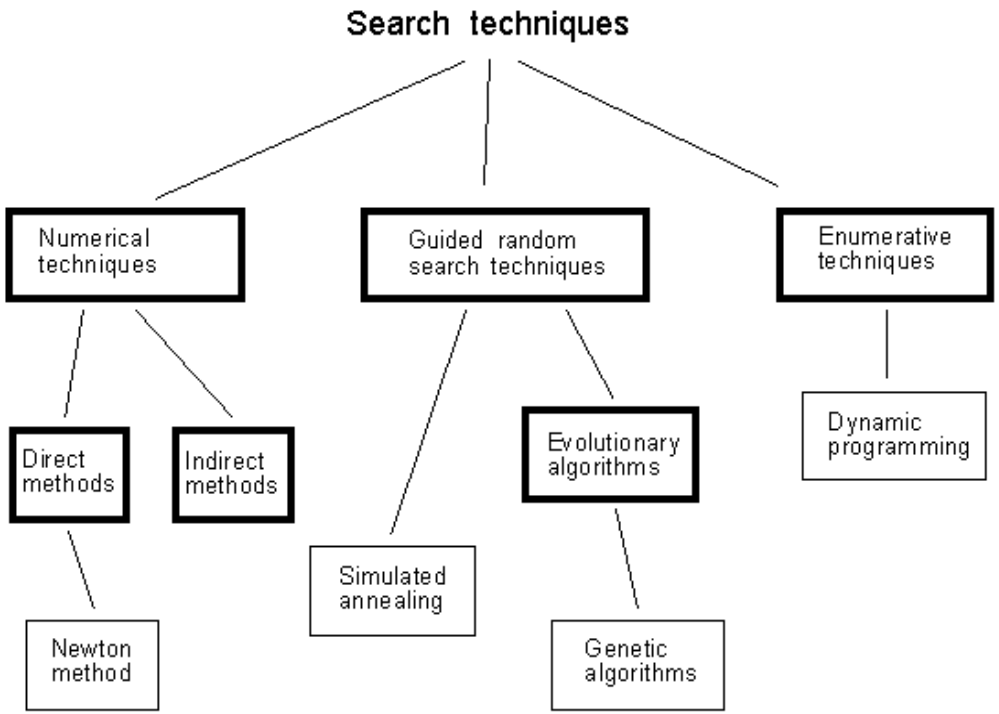
PO Nilore, Islamabad, Pakistan

http://faculty.pieas.edu.pk/fayyaz/

Local search and optimization

- Previously: systematic exploration of search space.
  - Path to goal is solution to problem
- YET, for some problems path is irrelevant.
  - E.g 8-queens
- Different algorithms can be used
  - Local search

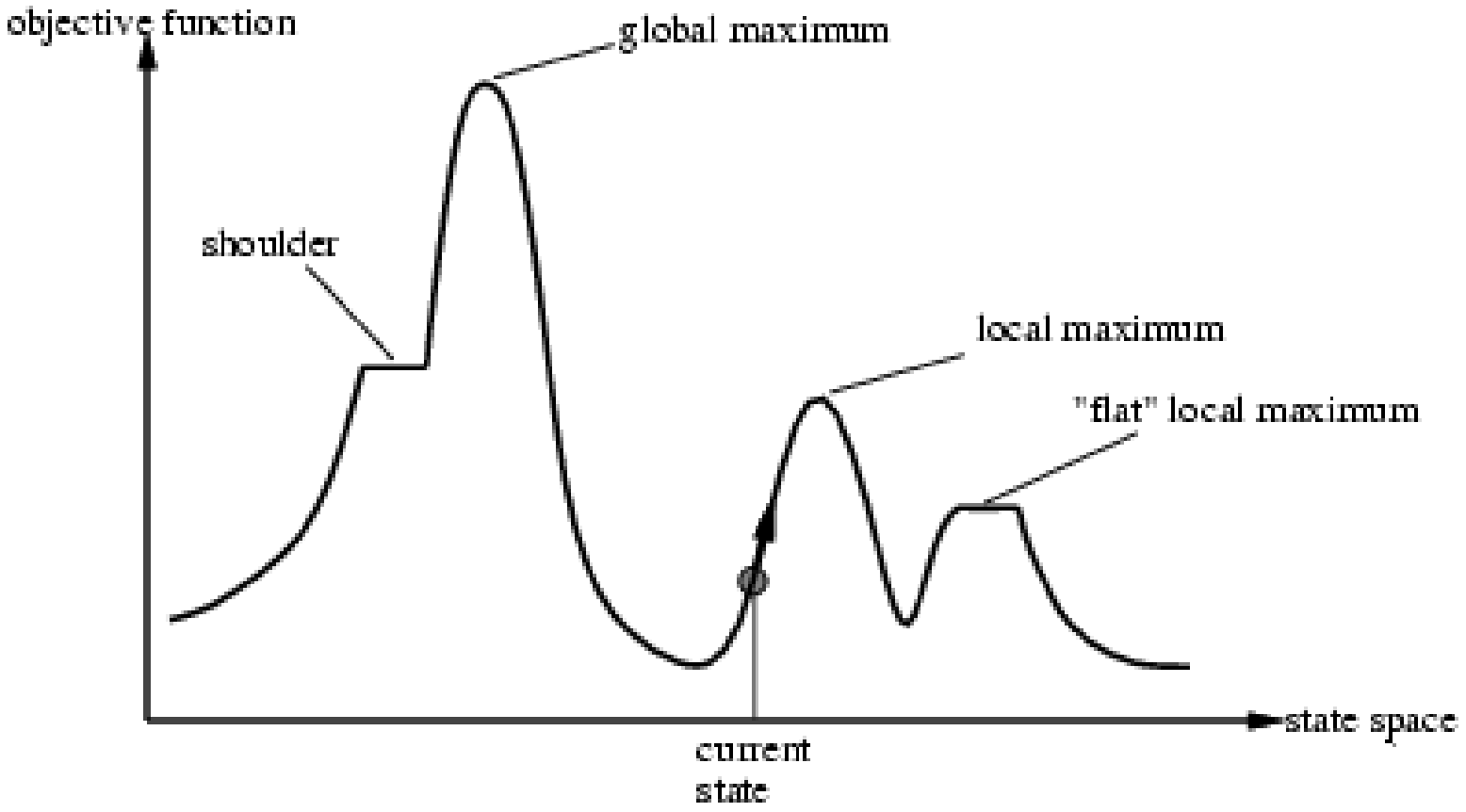




Local search and optimization

- Local search = use single current state and move to neighboring states.
- Advantages:
  - Use very little memory
  - Find often reasonable solutions in large or infinite state spaces.
- Are also useful for pure optimization problems.
  - Find best state according to some *objective function*.
  - e.g. survival of the fittest as a metaphor for optimization.

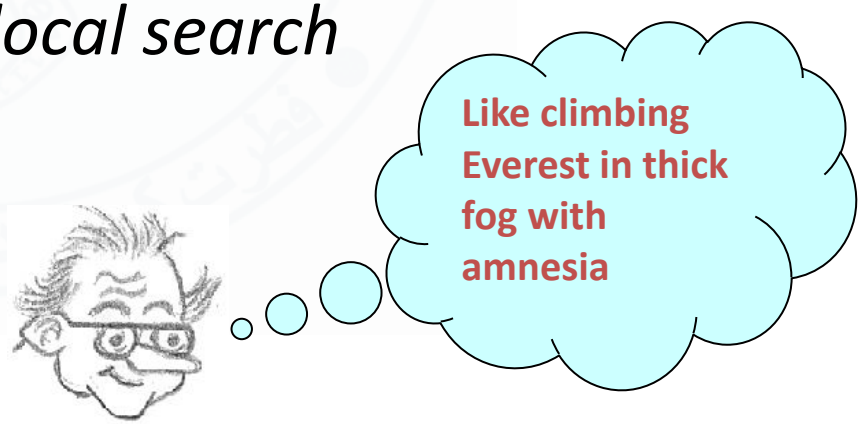
Local search and optimization



Hill-climbing search

- Hill-Climbing Search is a loop that continuously moves in the direction of increasing value
  - It terminates when a peak is reached.

- Hill climbing does not look ahead of the immediate neighbors of the current state.
- Hill-climbing chooses randomly among the set of best successors, if there is more than one.
- Hill-climbing a.k.a. *greedy local search*



Hill-climbing search

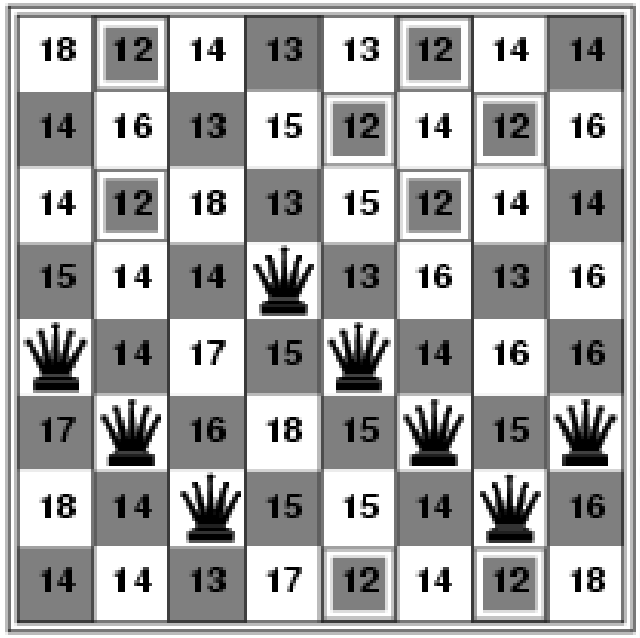
```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Hill-climbing example

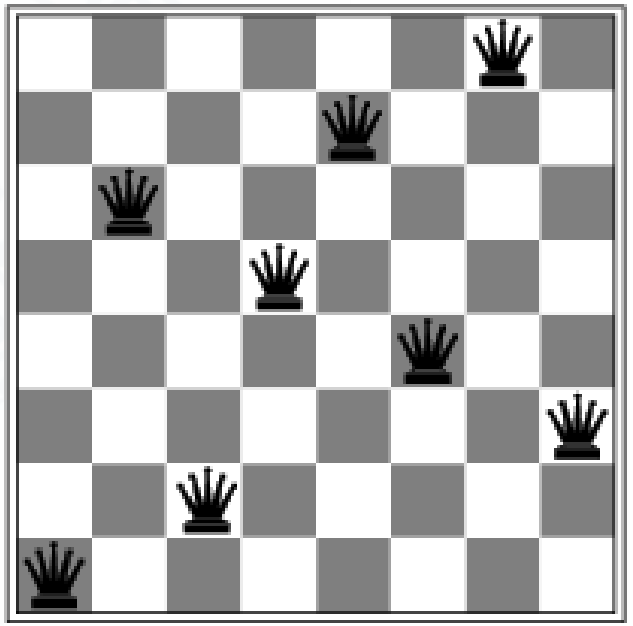
- 8-queens problem (complete-state formulation).
- Successor function: move a single queen to another square in the same column.
- Heuristic function *h(n)*: the number of pairs of queens that are attacking each other (directly or indirectly).

Hill-climbing example

a)



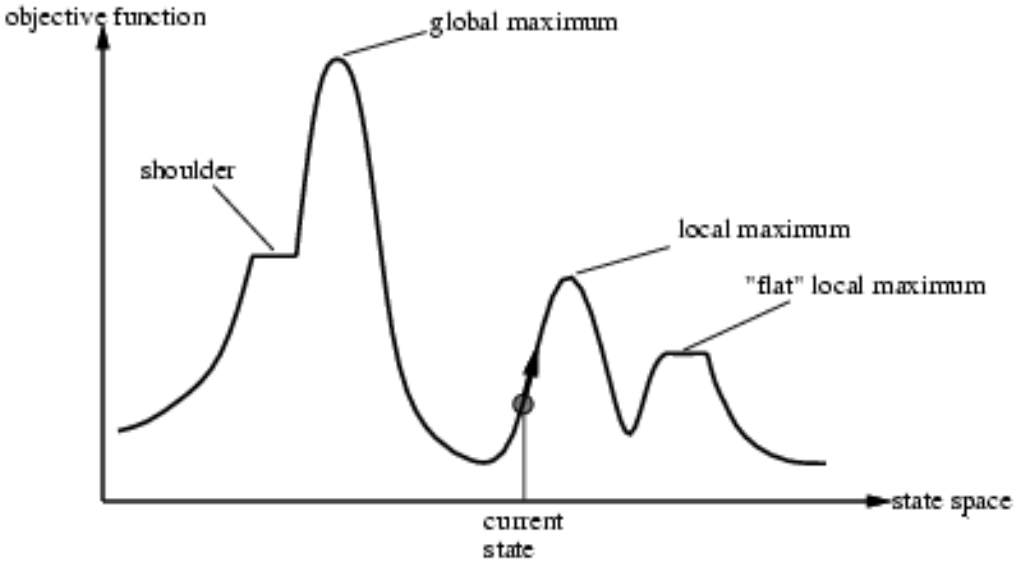
b)

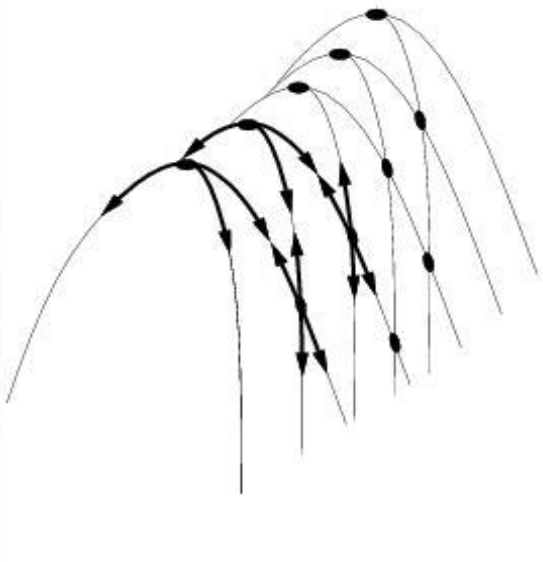


a) shows a state of h=17 and the h-value for each possible successor.

b) A local minimum in the 8-queens state space (h=1).

Drawbacks





- Ridge = sequence of local maxima difficult for greedy algorithms to navigate
- Plateaux = an area of the state space where the evaluation function is flat.
- Gets stuck 86% of the time.

Hill-climbing variations

- Stochastic hill-climbing
  - Random selection among the uphill moves.
  - The selection probability can vary with the steepness of the uphill move.
- First-choice hill-climbing
  - Stochastic hill climbing by generating successors randomly until a better one is found.
- Random-restart hill-climbing
  - Tries to avoid getting stuck in local maxima.

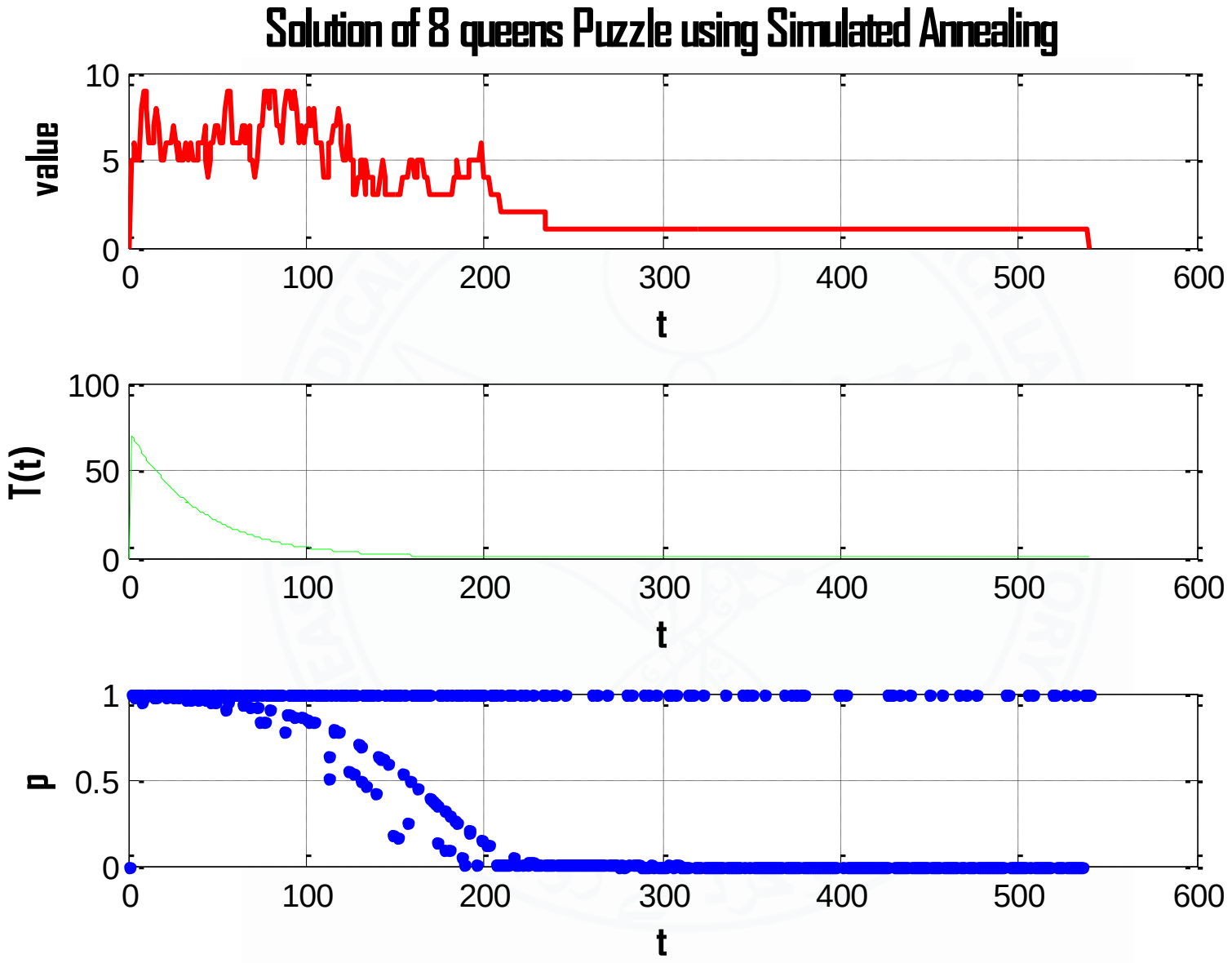
Simulated annealing

- Escape local maxima by allowing *"bad"* moves.
  - Idea: but gradually decrease their size and frequency.
- Origin: Metallurgical Annealing
- Bouncing ball analogy
  - Shaking hard (= high temperature).
  - Shaking less (= lower the temperature).
- If T decreases slowly enough, best state is reached.
- Applied for VLSI layout, airline scheduling, etc.

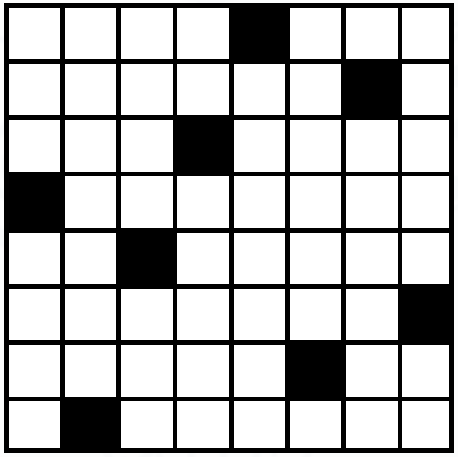
Simulated annealing

function SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

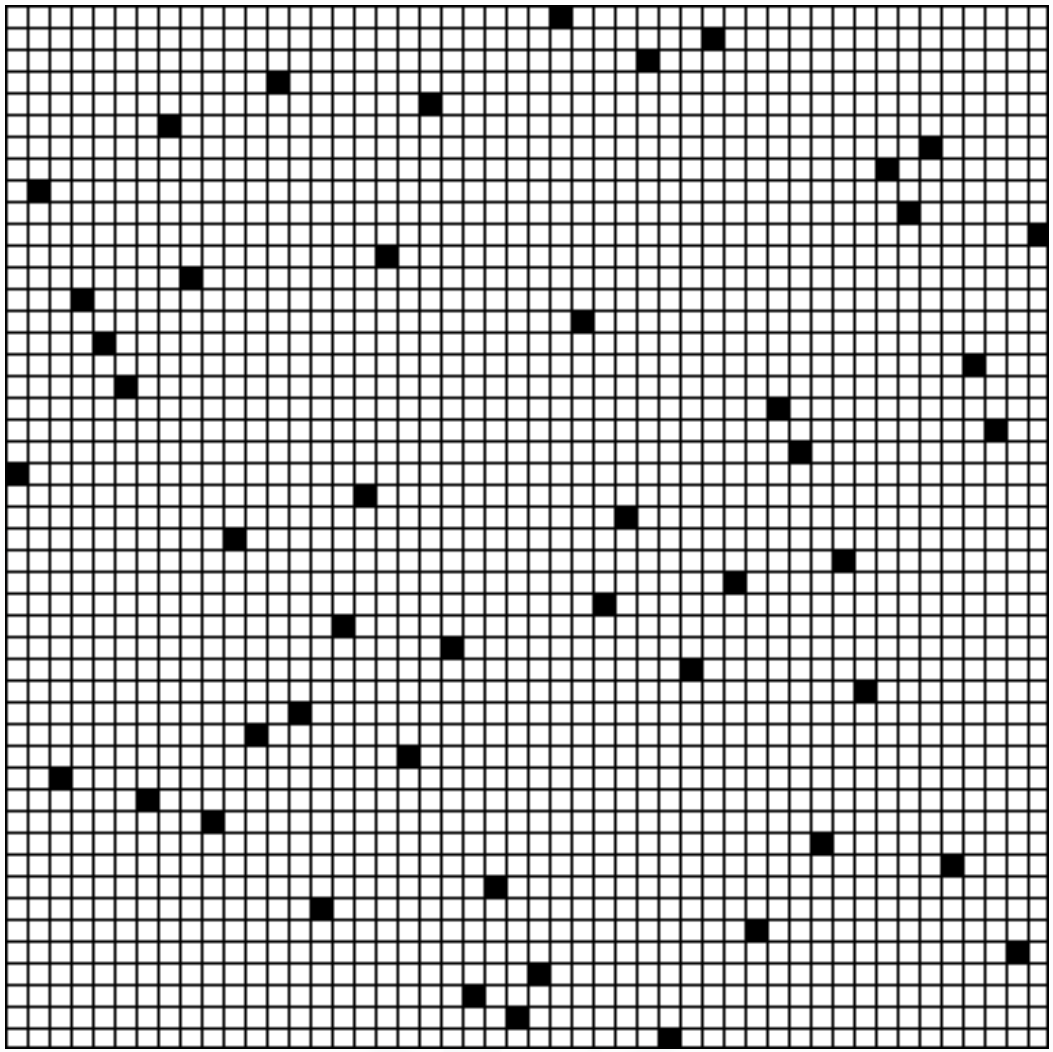
  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow schedule[t]$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta\ E/T}$


Solution of 8 queens Puzzle using Simulated Annealing
value
10
5
0
0
100
200
300
400
500
600
t
T(t)
100
50
0
0
100
200
300
400
500
600
t
p
1
0.5
0
0
100
200
300
400
500
600
t

Solution of the 8 Queens Puzzle using SA



Solution to 48 Queens Problem using Simulated Annealing

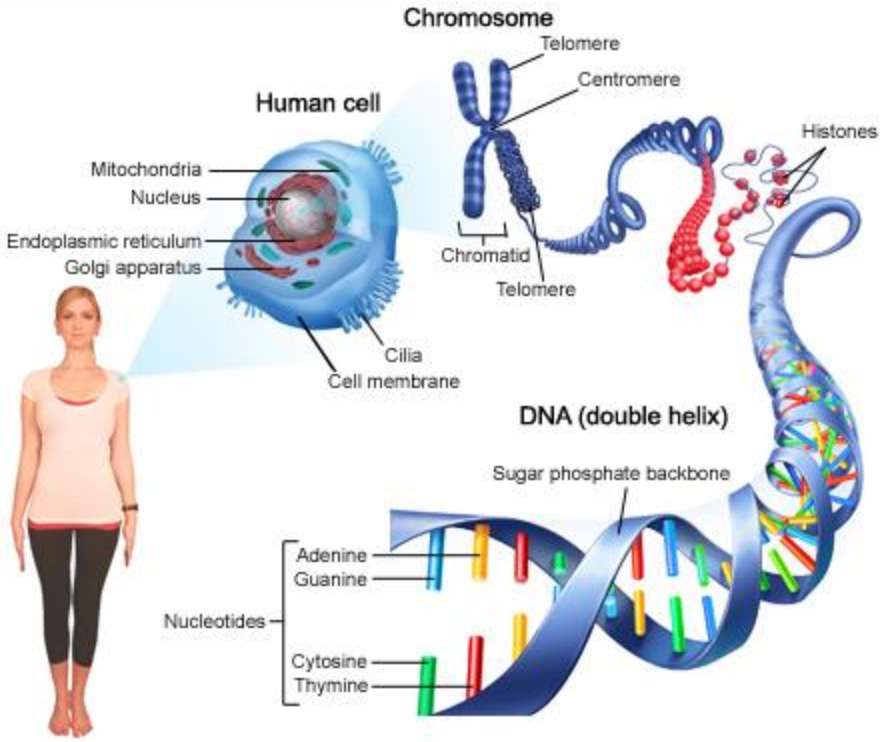


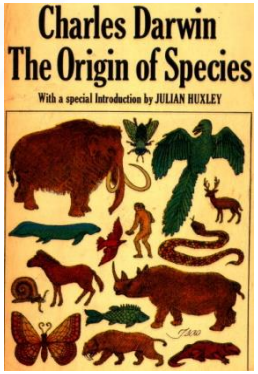
Questions

- What is the difference between Simulated Annealing and a Monte Carlo search?

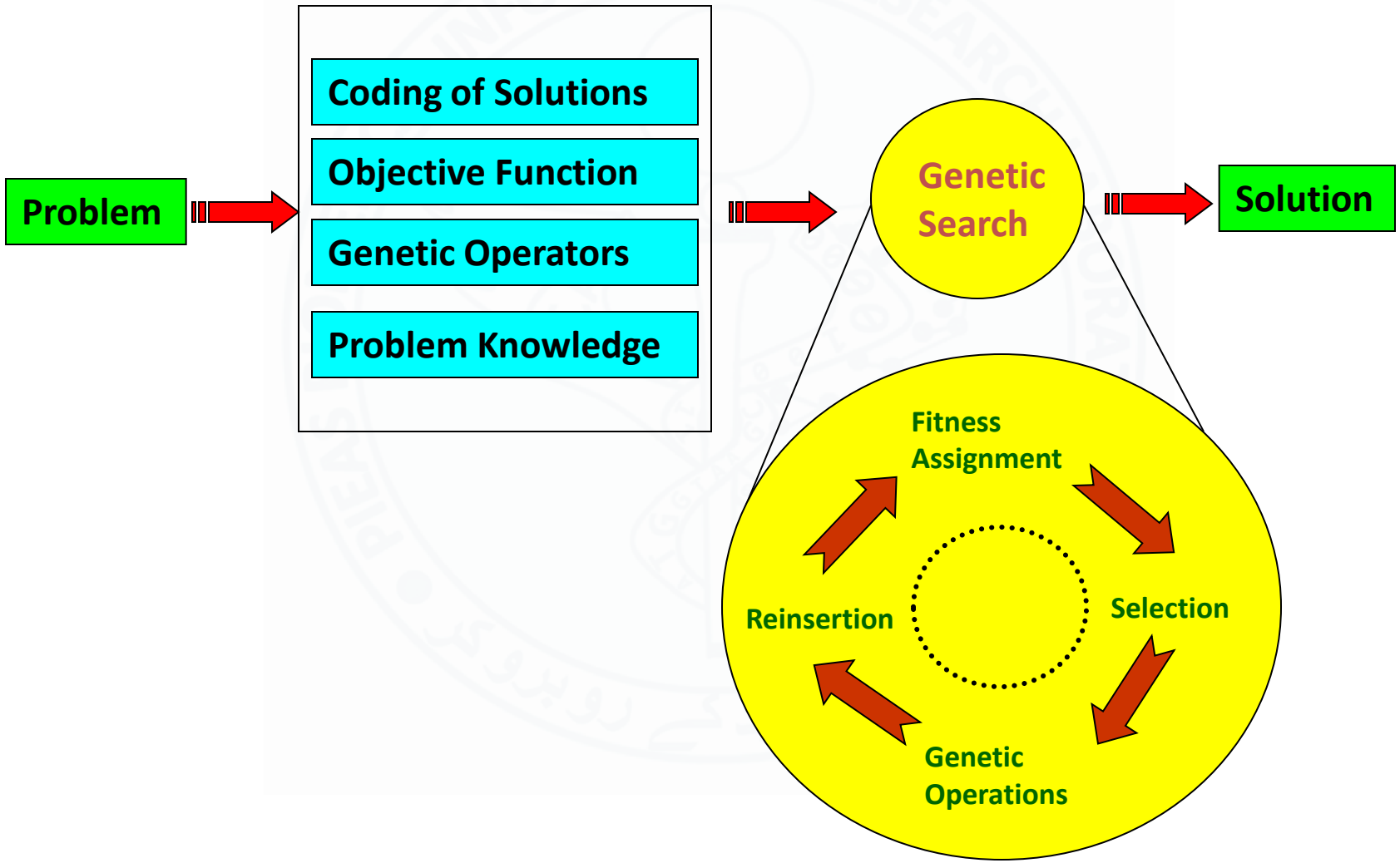
Genetic Algorithms

- Based on ideas of Darwinian Evolution
- Evolutionary computing evolved in the 1960s
- GA's were created by John Holland in mid 1970s





Genetic Algorithms



Genetic algorithm

function GENETIC_ALGORITHM(*population,* FITNESS-FN) return an individual

input: *population*, a set of individuals

FITNESS-FN, a function which determines the quality of the individual

repeat

new_population ← empty set

loop for i from 1 to SIZE(*population*) do

x ← RANDOM_SELECTION(*population*, FITNESS_FN)

y ← RANDOM_SELECTION(*population*, FITNESS_FN)

child ← REPRODUCE(*x*,*y*)

if (small random probability) then *child* ← MUTATE(*child*)

add *child* to *new_population*

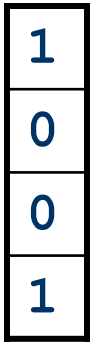
population ← *new_population*

until some individual is fit enough or enough time has elapsed

return the best individual

Genetic Algorithm – Coding

- Chromosomes are encoded by bit strings
- Every bit string therefore is a solution but not necessarily the best solution
- The way bit strings can code differs from problem to problem
  - Either: sequence of on/off or the number 9



Genetic Algorithm – Fitness Assessment

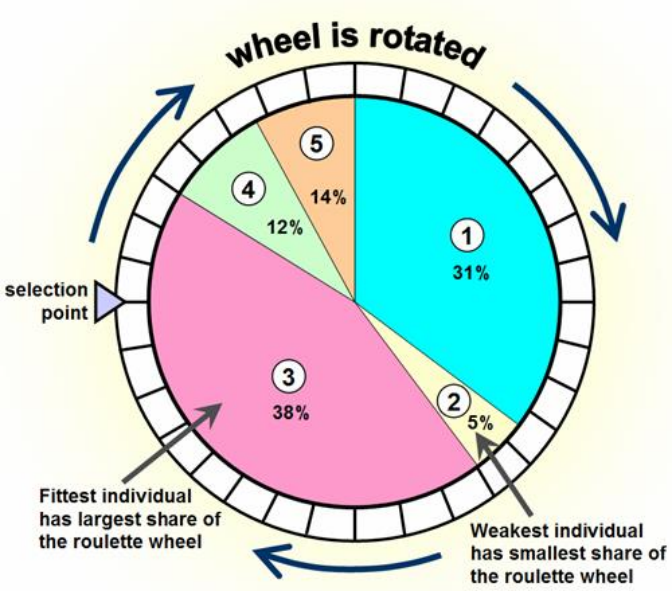
- Decoding of chromosome representation into decision variable domain
- Objective function characterizing an individual's performance
- Fitness values assigned from raw performance measure of an individual

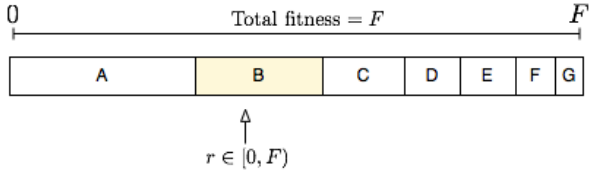
Genetic Algorithms - Selection

- Roulette Wheel Selection (Fitness Proportional Selection)
  - Let the fitness of an individual x_i be f_i
  - Its probability of selection, p_i, is given by

$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i}$$

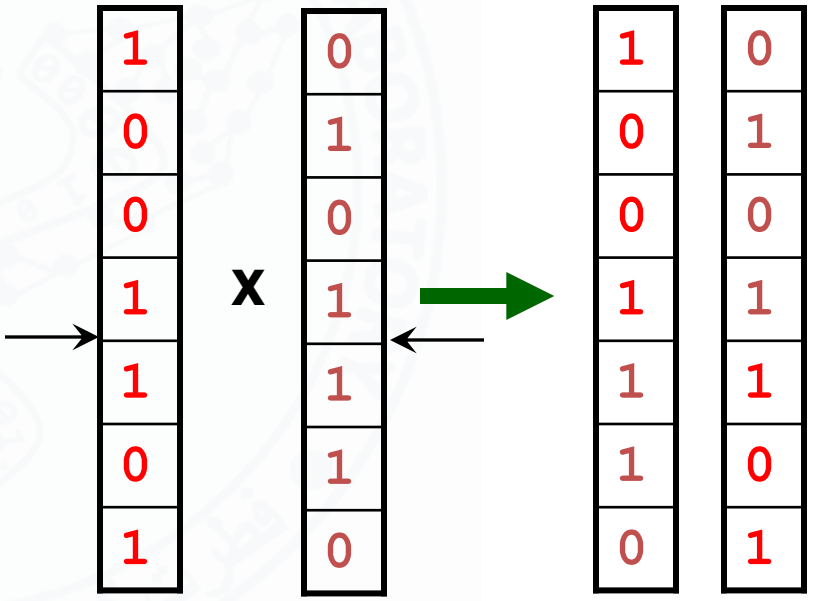
  - Individuals with higher fitness have more chances of being selected
  - Fitness value must always be positive
- Procedure
  - Generate a random number between 0 and 1
  - Select the individual in whose range this number lies
  - Repeat until sufficient individuals are selected





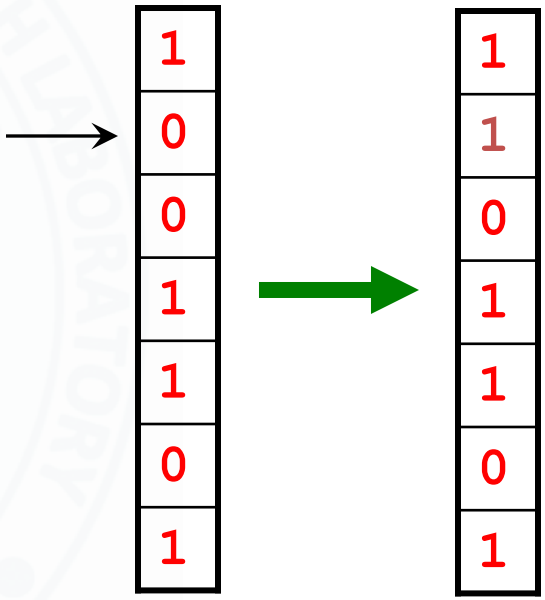
Genetic Algorithm – Cross Over

- Recombination (cross-over) can when using bit strings schematically be represented as shown
- Procedure
  - Form random pairs from the selected individuals
  - Cross these pairs with probability p_c
    - Generate a random number between 0 and 1
    - If this number is less than p_c, perform cross over by choosing a random crossover site
    - Otherwise copy the parents as such



Genetic Algorithm – Mutation

- Mutation prevents the algorithm to be trapped in a local minimum
- In the bit string approach mutation is simply the flipping of one of the bits
- Procedure
  - For each individual after cross over
    - Generate a random number
    - If this number is less than p_m then flip a bit at a random position
    - Otherwise pass-on the individual as such



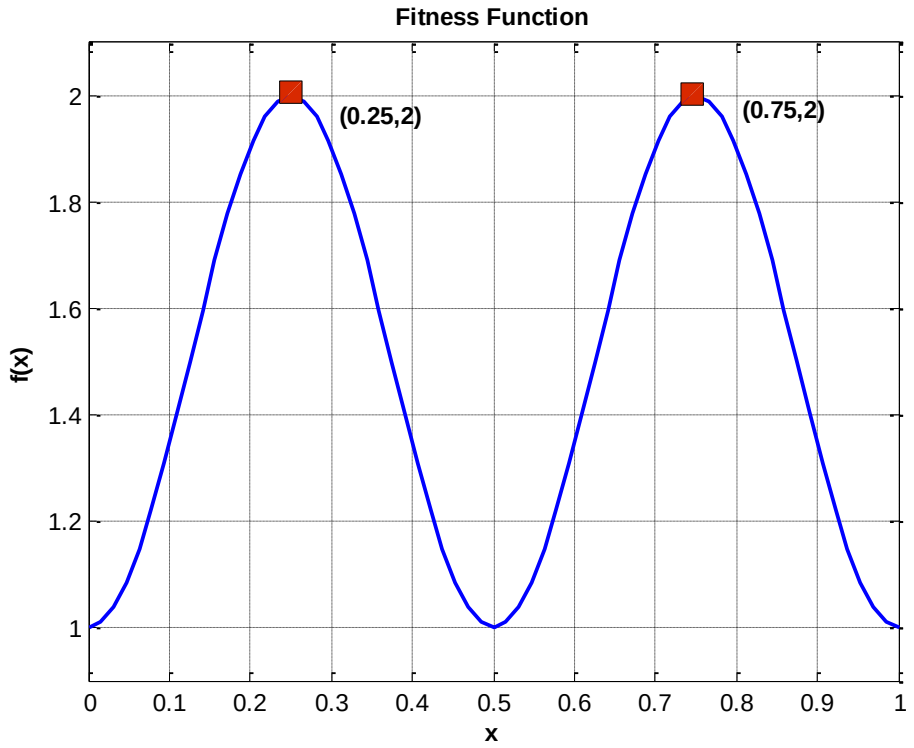
Generation of New Population

- Replace all parents by using the output of the mutation

Simple Example

- Maximize

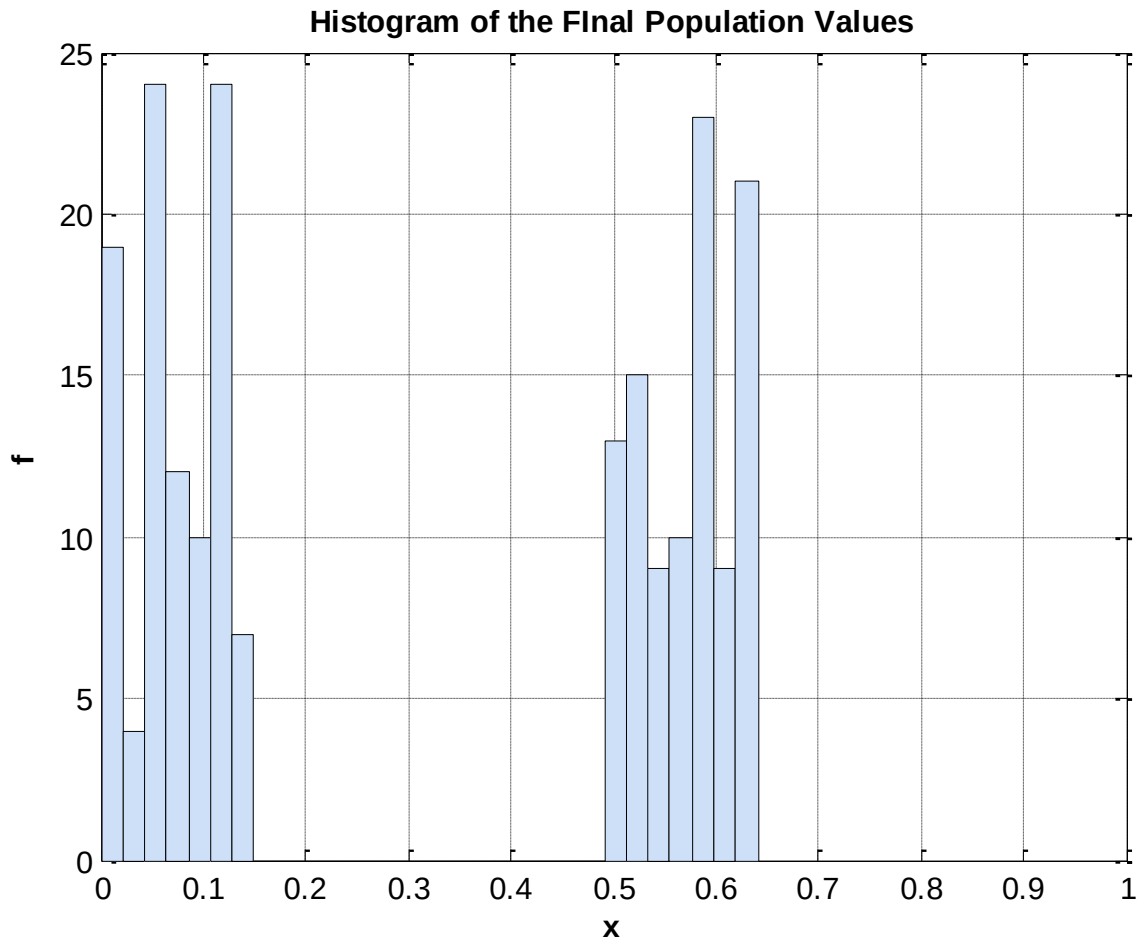
$$f(x) = 1 + \sin^2(x), \quad x \in [0,1)$$

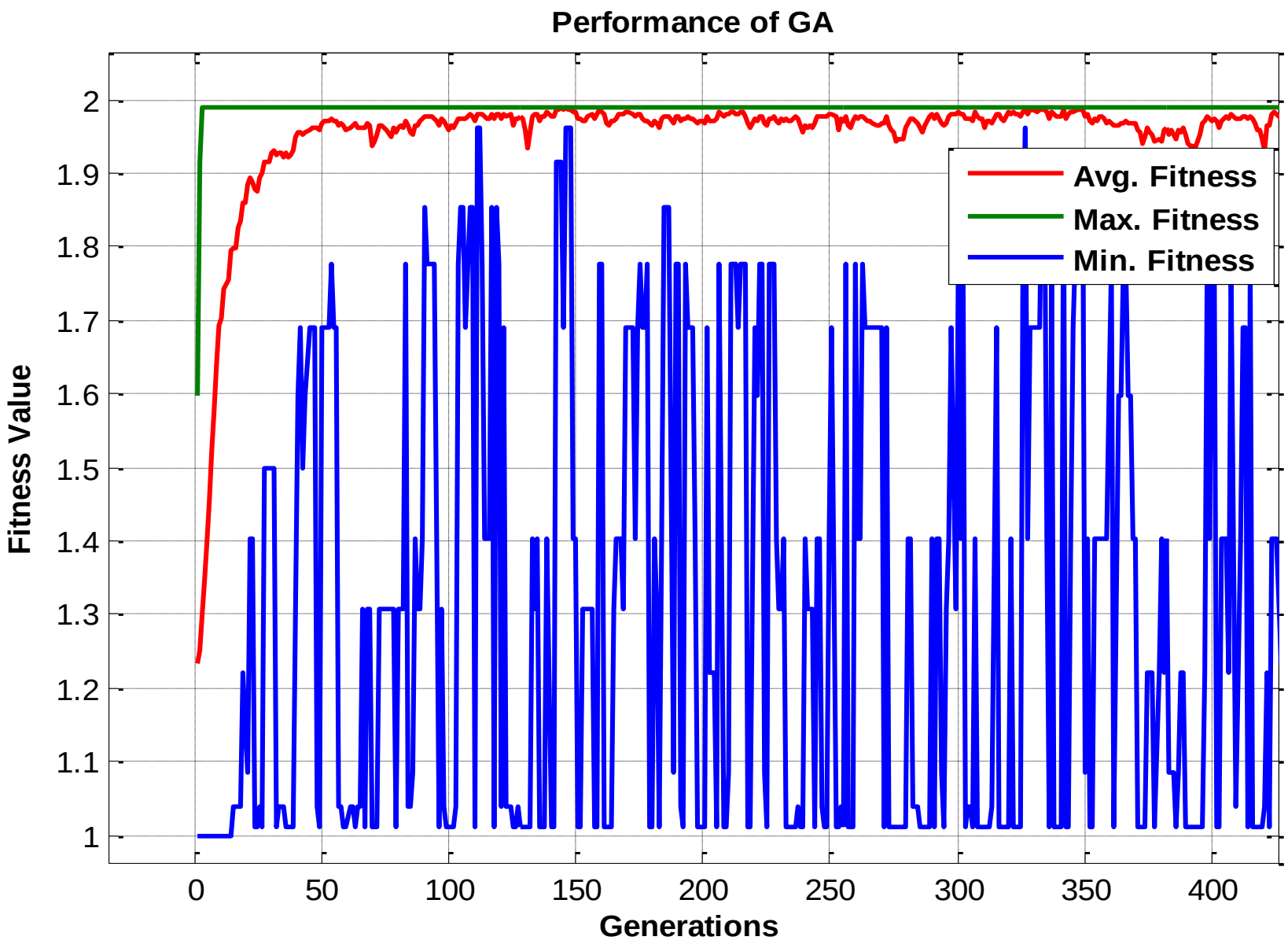


Simple Example: GA Problem Formulation

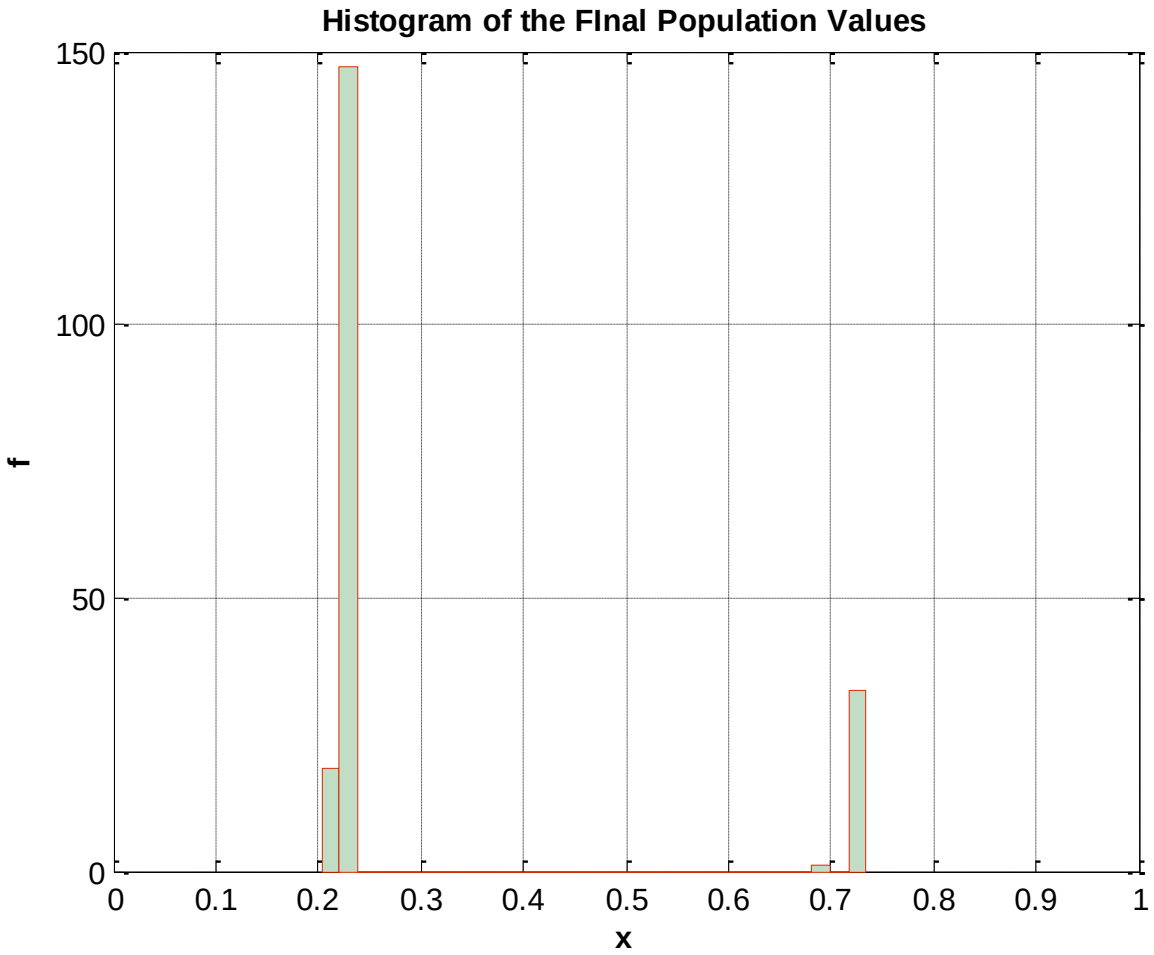
- Coding
  - Six bit unsigned fixed point quantization
    - Word-length = 6
    - Fraction – length = 6
    - $(000000)_2 = 0_{10}$
    - $(111111)_2 = 1_{10}$
    - Optimum points are:
      - $x_1^* = 0.75 = 110000$
      - $x_2^* = 0.25 = 010000$
- Fitness Evaluation
  - Convert number to phenotype
  - Use $f(x) = 1 + \sin^2(x), \quad x \in [0,1)$
- Selection: Roulette Wheel Selection
- Cross-Over: $p_c = 0.8$, Simple Single Point Cross-Over
- Mutation: $p_m = 0.01$, Single bit mutation
- Termination: Based on limit on number of Generations (1000)
- Generation of Initial Population: Bit representation of random numbers between (0,0.15) and (0.5,0.65)
- Population Size: n = 200

Simple Example: Initial Population




Performance of GA
Avg. Fitness
Max. Fitness
Min. Fitness
Fitness Value
Generations
2
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0
50
100
150
200
250
300
350
400

Simple Example: Final Population



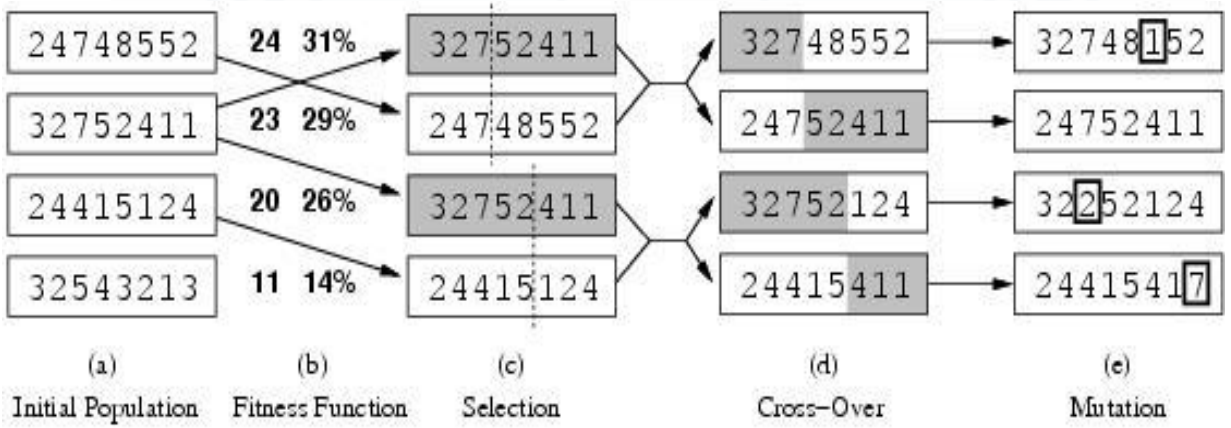
Assignment

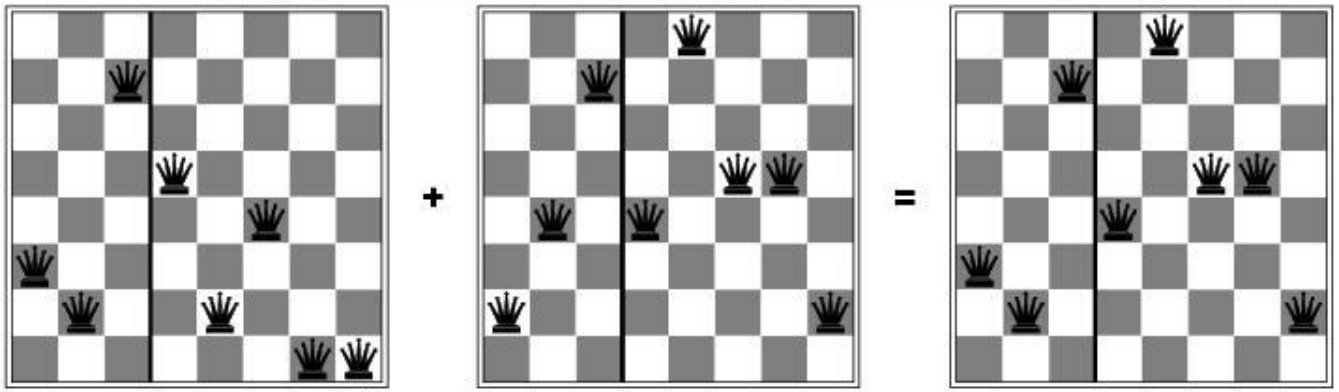
- Install DEAP.
  - https://pypi.python.org/pypi/deap
  - Use N-Queens as a base

- Solve the design problem

Genetic algorithms

- Variant of local beam search with *genetic recombination.*





Advantages of GA

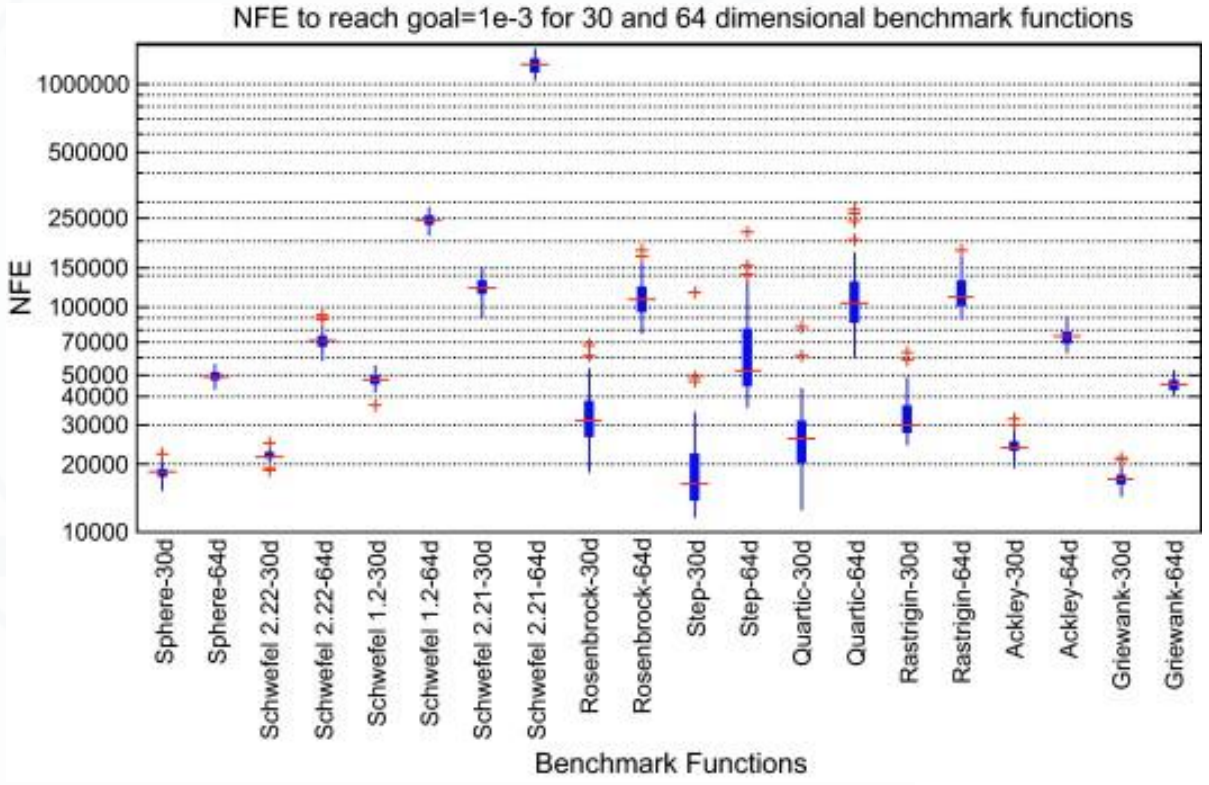
- It search the population of points in parallel, not a single point.
- It does not require any derivative information or other knowledge except an objective function
- It uses probabilistic transition rules.
- It can provide more than one potential solutions to a problem.

Application Areas of GA

- Optimization
- Automatic Programming
- Machine and Robot Learning
- Economic Models
- Image Processing and Pattern Recognition
- Control Engineering

Other Algorithms

- Directed Evolution
- Evolutionary Strategies
- CMA-ES
- PSO
- ACO
- MOX
- GP



http://www.sciencedirect.com/science/article/pii/S156849461100281X

Variations & Enhancements in GA

Variations in Simple Genetic Algorithm

- The SGA can be changed by changing
  - Encoding of the chromosomes
  - Selection Strategy
  - Reinsertion Strategy
  - Genetic Operators
- Enhancements to SGA
  - Introducing Diversity
  - Handling constraints

Variations in Chromosome Encoding

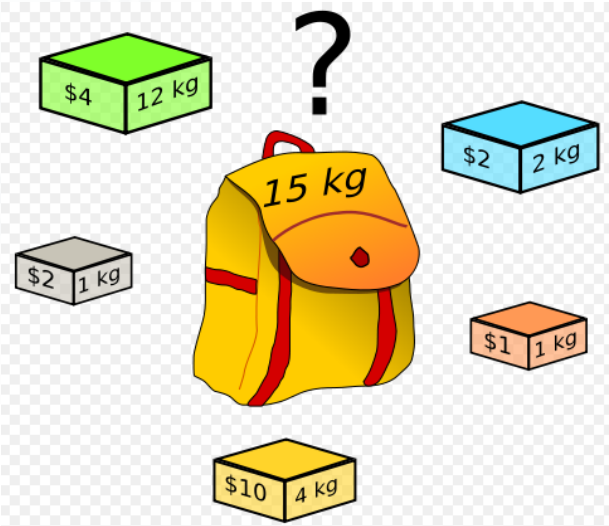
- Binary Encoding
  - Most common and most frequently used encoding scheme
  - Every chromosome is a string of ones (1s) and zeros (0s)

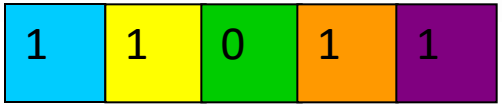
Chromosome A	10110010110010101110010 1
Chromosome B	11111110000011000001111 1

  - Gives many possible chromosomes even with a small number of alleles
  - It may not be natural for many problems and corrections must be made after crossover and mutation

Variations in Chromosome Encoding…

- Binary Encoding…
  - Example
  - The Knapsack Problem
    - There are things with given value and size. The knapsack has given capacity. The problem is to select things to maximize the value of things in the knapsack while keeping the capacity of the knapsack constant
  - GA Solution
    - Each bit says if the corresponding thing is in the knapsack





Variations in Chromosome Encoding…

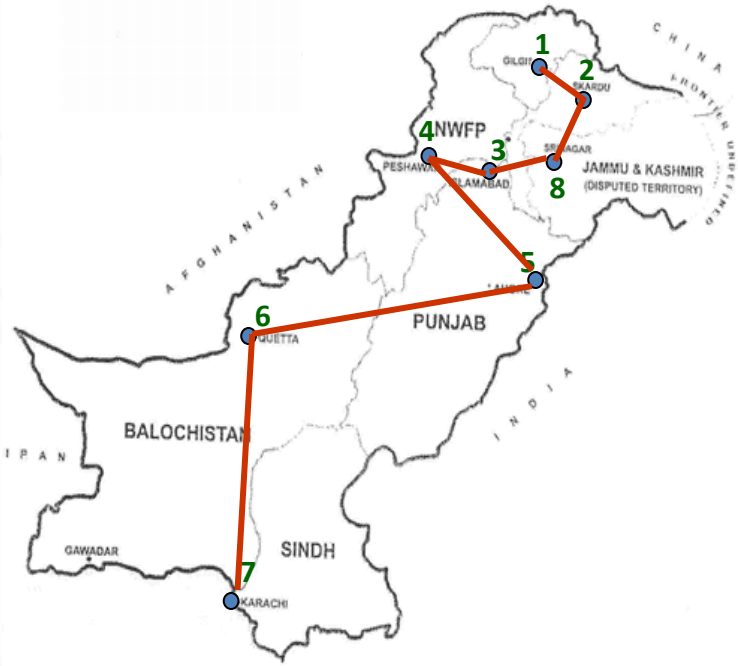
- Permutation Encoding
  - Can be used in ordering problems such as the Traveling Salesman Problem (TSP) or task ordering problem
  - Every chromosome is a string of numbers, which represents number in a sequence

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

  - Special types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e. have real sequence)

Variations in Chromosome Encoding…

- Permutation Encoding…
- Example
  - Problem: The Traveling Salesman Problem (TSP)
    - There are cities and given distances between them. A traveling salesman is to visit them all but has to minimize the distance of overall travel
  - GA Solution Encoding: Chromosome says order of the cities in which the salesman will visit them



1	2	8	3	4	5	6	7

Variations in Chromosome Encoding…

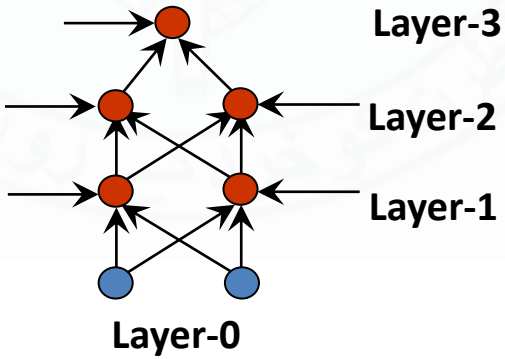
- Value Encoding
  - Direct value encoding (e.g. real numbers) can be used in problems
  - Every chromosome is a string of some values which represent some connection to the problem being studied

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

  - New mutation and crossover operators needed

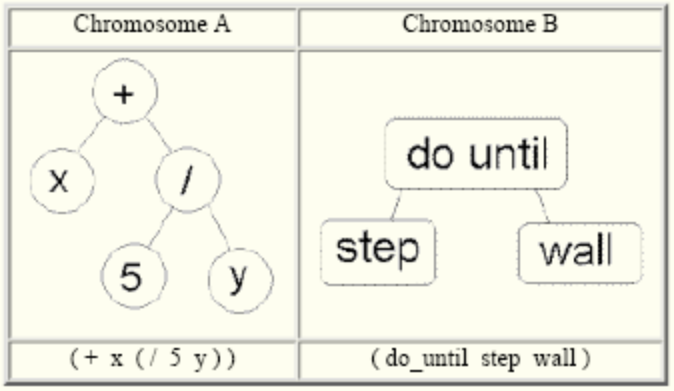
Variations in Chromosome Encoding…

- Value Encoding…
  - Example
    - Finding Weights of a Neural Network so that it gives required output given some training data
    - Real value in chromosomes represents weights of the neural network



Variations in Chromosome Encoding…

- Tree Encoding
  - Used mainly for evolving programs and expressions in Genetic Programming
  - Every chromosome is a tree of some objects such as functions or commands in a programming language



Variations in Chromosome Encoding…

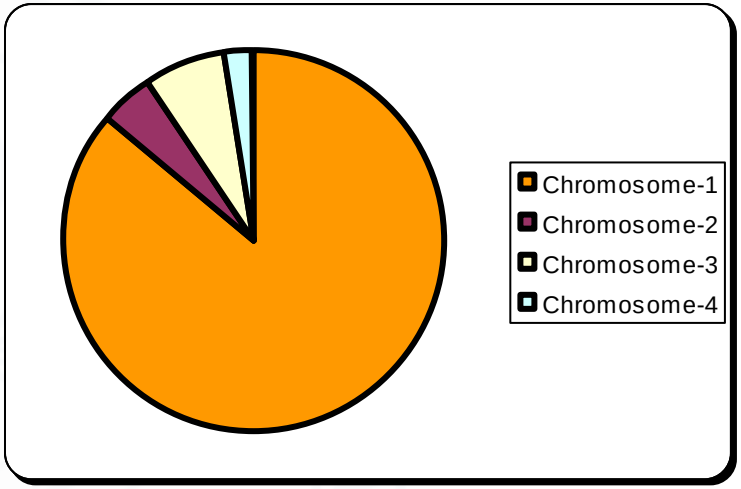
- Tree Encoding…
  - Example: Finding a function that matches given input and output data
  - GA solution encoding: Chromosomes are Functions represented by a tree

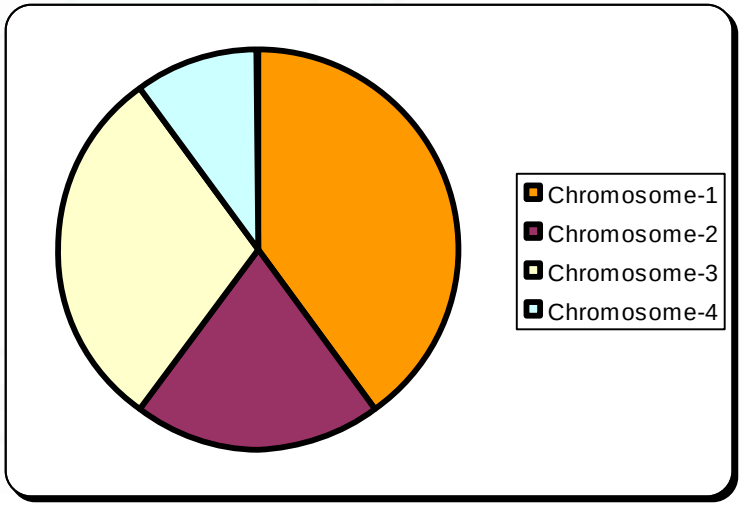
Variations in Selection

- Steady State Selection
  - No particular method of selecting parents
  - Main idea is that big part of chromosomes should survive to the next generations
  - A few good chromosomes are selected for creating new offspring
  - No probabilistic selection
- Roulette Wheel Selection
  - Already Discussed
  - Faces problems when fitness values of chromosomes differ very much
  - Fitness scaling required to enable selection of some of the ‘non-best’ chromosomes to maintain diversity

Variations in Selection…

- Rank Based Selection
  - Rank selection ranks first ranks the chromosomes and then assigns fitness to each of the N chromosomes based on its rank
    - The worst chromosome will have fitness = 1
    - The best chromosome will have fitness = N
    - Apply roulette wheel selection on these values
  - May lead to more diverse solutions
  - May lead to slower convergence





Variations in Selection…

- Tournament Selection
  - choose k (the tournament size) individuals from the population at random
  - choose the best individual from pool/tournament with probability p
  - choose the second best individual with probability p(1-p)
  - choose the third best individual with probability p $(1-p)^2$ and so on...
- More Efficient
- Can be run on parallel architectures

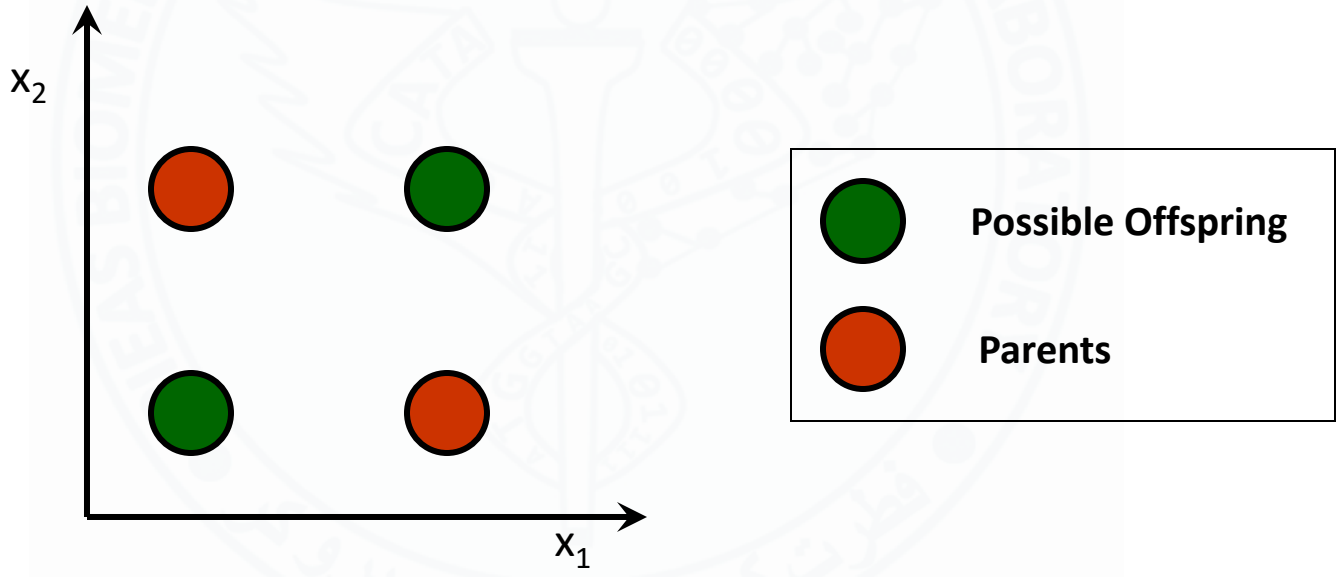
Variations in Operators: Recombination

- Real valued Recombination
  - Discrete Recombination
  - Intermediate Recombination
  - Line Recombination
- Binary valued Recombination
  - Single point Crossover
  - Multi-point Crossover
  - Uniform Crossover

Discrete Recombination

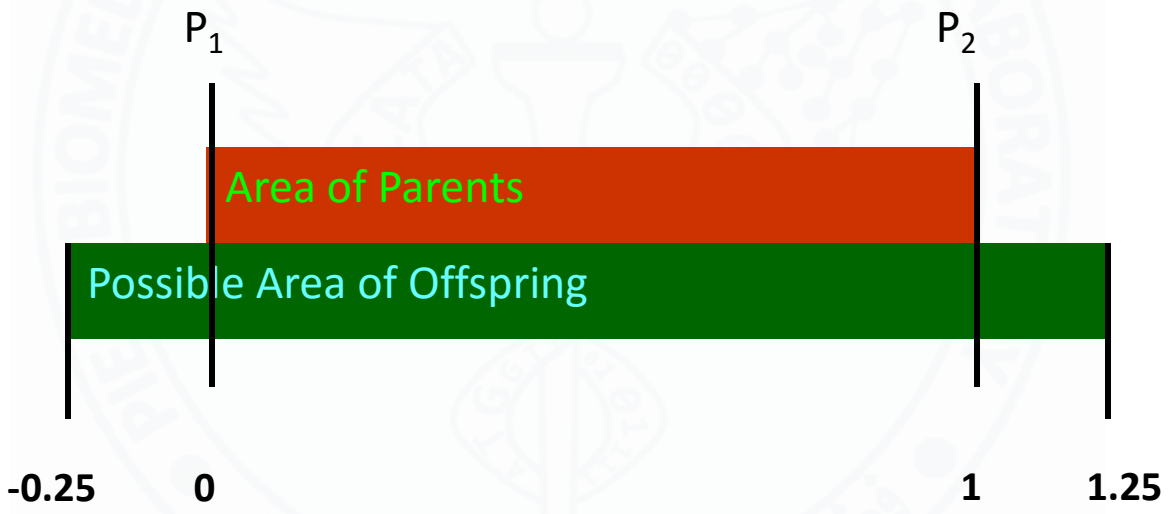
Individual 1	9	12
Individual 2	33	5
Offspring 1	33	12
Offspring 2	9	5

Effect of Discrete Recombination



Intermediate Recombination

$$O_1 = P_1 + \alpha (P_2 - P_1) \quad \alpha \in [-0.25, 1.25]$$

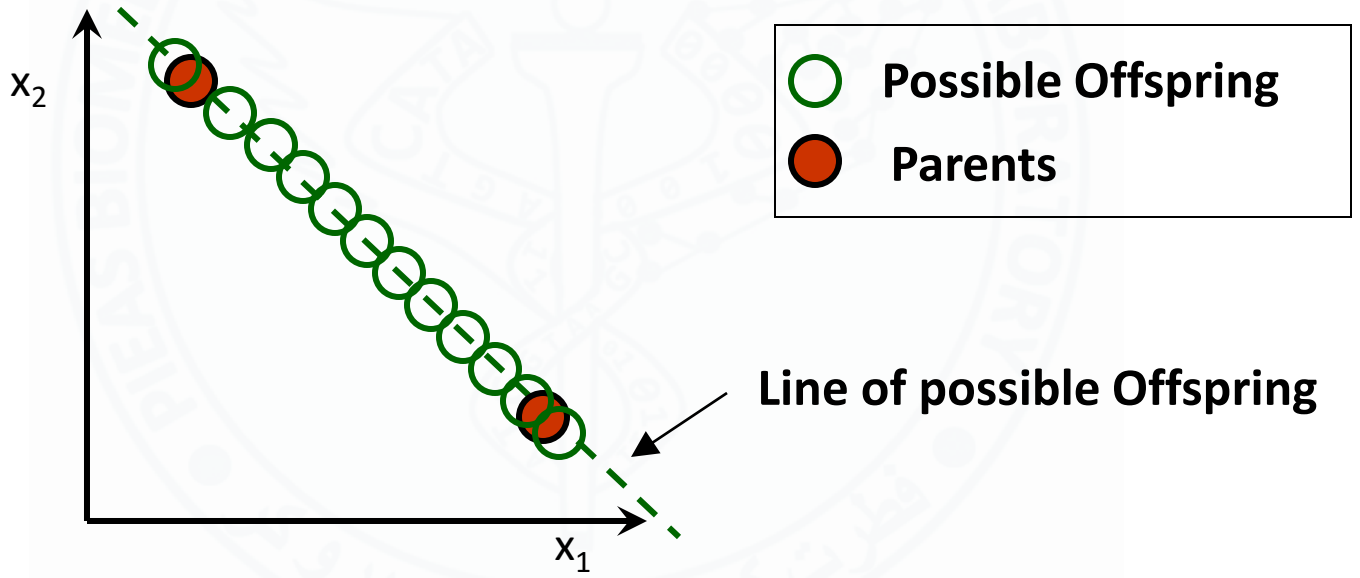


Variations in Operators: Line Recombination

- Only one α for all variables

Individual 1	12	25
Individual 2	123	4
α_1	0.5	
α_2	0.1	
Offspring 1	67.5	14.5
Offspring 2	23.1	22.9

Effects of Line Recombination



Single Point Crossover

- Choose randomly crossover pt. $c \in [1, N_b]$

$P_1 = 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0$

$P_2 = 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1$

$O_1 = 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1$

$O_2 = 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0$

Multi-point Crossover

- *m* crossover points $c_i \in [1, N_b]$, $i=1,\ldots,m$

$P_1 = 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0$

$P_2 = 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1$

$O_1 = 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0$

$O_2 = 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1$

Uniform Crossover

- Every locus a potential crossover point

  P_1 = 0 0 0 1 1 1 1 0

  P_2 = 1 0 1 1 0 0 1 1

  M_1 = 0 1 1 0 0 1 0 1

  M_2 = 1 0 0 1 1 0 1 0

  O_1 = 1 0 0 1 0 1 1 0

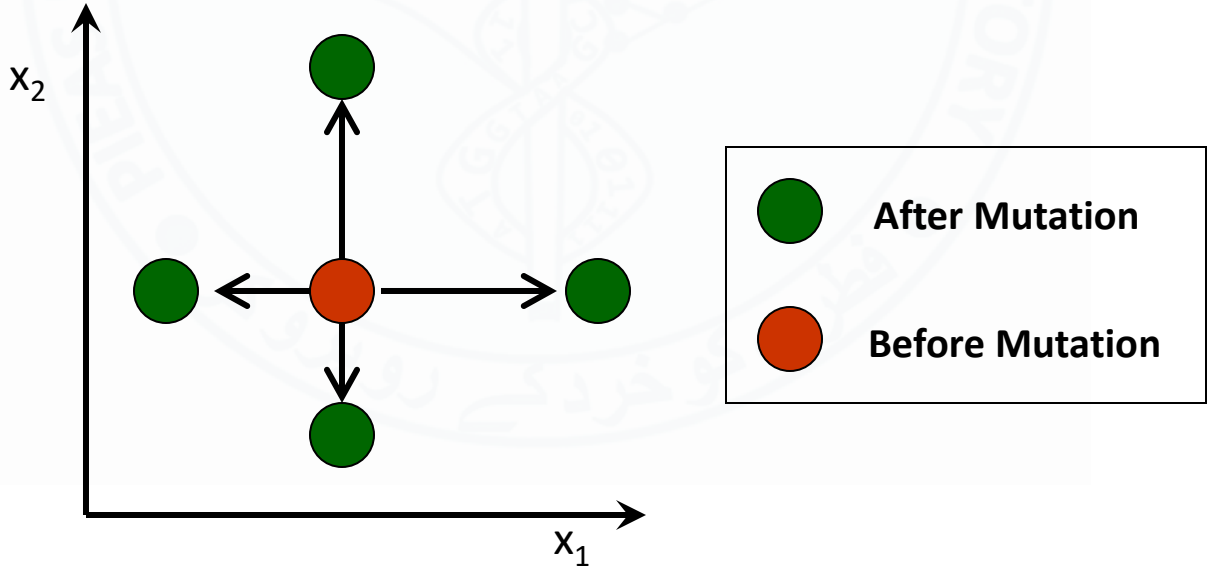
  O_2 = 0 0 1 1 1 0 1 1

Variations in Operators: Mutation

- Real valued Mutation
- Binary Mutation

Real Valued Mutation

- Mutated Variable = Variable $\pm R\,\delta$
  - R = Range of variable
- The radius of mutation can be decreased as the generations proceed to avoid loosing the good solutions



Binary Valued Mutation

- Changing the variable value at mutation site selected randomly

$O = 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0$

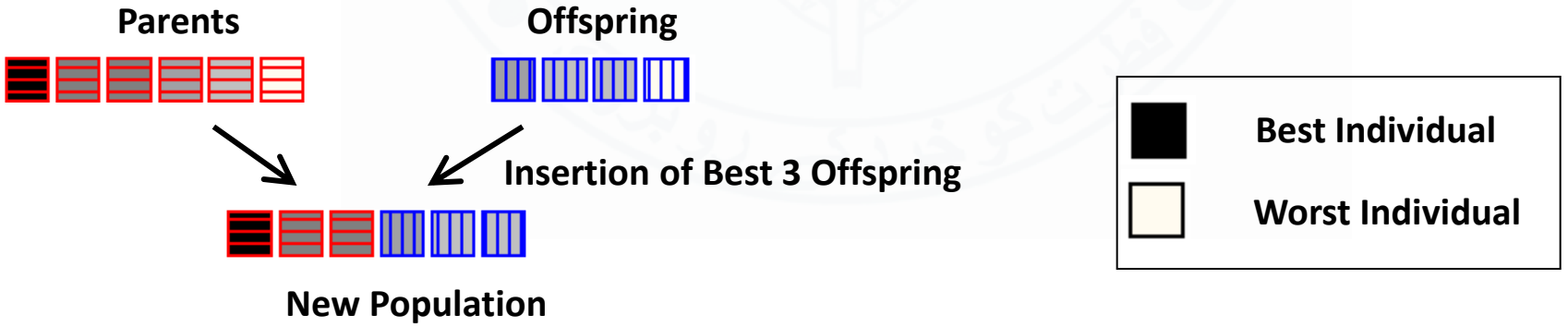
$O_m = 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0$

Variations in Reinsertion Strategies

- Global Reinsertion
- Local Reinsertion

Global Reinsertion

- Different Schemes
  - produce as many offspring as parents and replace all parents
  - produce less offspring than parents and replace parents uniformly at random
  - produce less offspring than parents and replace only worst parents
  - produce more offspring than parents and reinsert only best offspring

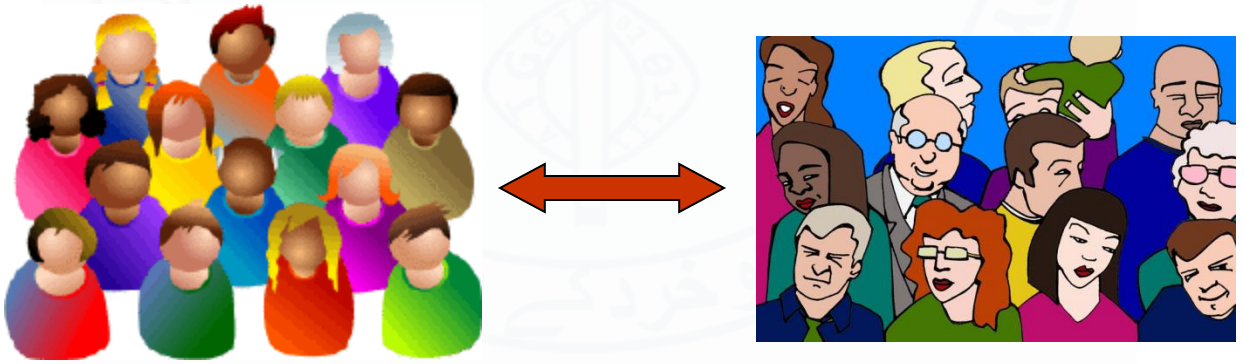


Local reinsertion

- Select a neighborhood and reinsertion take place only in that specified neighborhood

Migration

- It divides the population into many sub-populations. These population evolves independently and after certain number of generations, some individuals may migrate from one sub-population to other sub-population.
- Maintains Diversity in the population
- Allows for parallelism in GA



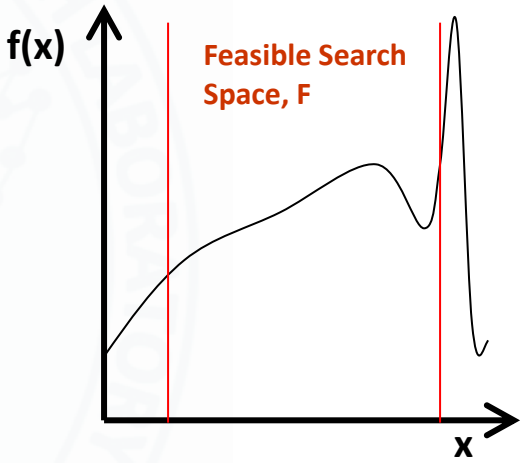
Maintaining Diversity: Niche Penalty

- Diversity must be maintained across the population in order to prevent trapping in a local optima
- A common technique to maintain diversity is to impose a "niche penalty", wherein, any group of individuals of sufficient similarity (niche radius) have a penalty added, which will reduce the representation of that group in subsequent generations, permitting other (less similar) individuals to be maintained in the population.

Dealing with Constraints

- At times we wish to optimize a given fitness function under some constraints
- Mathematically,

$$\begin{aligned} \text{minimize} \quad & f(\underline{X}),\ \underline{X} = (x_1, \ldots, x_n) \in R^n \\ \text{subject to} \quad & g_j(\underline{X}) \le 0 \qquad j = 1 \ldots q \\ & h_j(\underline{X}) = 0 \qquad j = q+1 \ldots m \end{aligned}$$



- For handling these constraints in Genetic Algorithms, there are no general methods but guidelines do exist

Dealing with Constraints…

- Handling constraints through coding
  - The chromosome encoding can be used to handle some simple types of constraints
    - For example, when using binary encoding for real number representation, the range of the binary string can be selected in a way so that the constraints are satisfied
  - Not Generally possible
- Rejecting Infeasible Individuals in the population (Death Penalty)
  - Simple
  - Wastage of computation
  - Burden on the population

Dealing with Constraints…

- Using Penalty Functions
  - Penalize unfeasible solutions during fitness evaluation
  - In general: (F is the feasible search space)

$$eval(\underline{X}) = \begin{cases} f(\underline{X}) & \text{if } \underline{X} \in F \\ f(\underline{X}) + penalty(\underline{X}) & \text{else} \end{cases}$$

  - The penalty term is zero if no violations occur and positive otherwise
  - It is expressed in terms of 'violation measuring functions'

$$f_j(\underline{X}) = \begin{cases} \max\{0, g_j(\underline{X})\}, & 1 \le j \le q \\ |h_j(\underline{X})|, & q+1 \le j \le m \end{cases}$$

Dealing with Constraints…

- Using Penalty Functions…
  - Method-1: We can use a weighted combination of the 'violation measuring functions' and the original optimization function

$$eval(\underline{X}) = f(\underline{X}) + \sum_{j=1}^{m} \beta_j f_j$$

    - Problems: Finding optimal weights

  - Method-2: Another similar approach uses dynamic penalty dependent upon the number of generations, t

$$eval(\underline{X}) = f(\underline{X}) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}, \quad C = 0.5, \alpha = \beta = 2$$

    - Pressure on infeasible solutions increased over generations
    - Problems: May cause problems if a feasible solution is not found quickly

Dealing with Constraints…

- Using Penalty Functions…
  - Method-3: Make the values of the infeasible individuals worse than the worst feasible individual
    - Map the evaluations of feasible solutions into an interval [a,b) and unfeasible solutions into the interval [b,c]
  - Method-4: Schoenauer et al. (1993)
    - Start with a random population
    - Set j = 1
    - While (j<m)
      - Evolve the population using $eval(\underline{X}) = f_j(\underline{X})$, During this phase individuals violating any of the 1…(j-1)th constraints are eliminated
      - Evolution is stopped when a threshold percentage (called flip threshold) of the population satisfies the jth constraint
      - The current population is the starting point for the next phase
      - Set j = j+1
    - End
    - Optimize the objective function $eval(\underline{X}) = f(\underline{X})$, rejecting infeasible individuals

Dealing with Constraints…

- Genetic Repair
  - Force an infeasible individual to the feasible region
    - Example: Crossover and mutation in TSP can generate sequences of cities in which cities can be duplicated
    - Solution: Replace the duplicated cities with the missing cities (Genetic Repair Operator)
- Other Methods
  - Use of MOGA: Use the values of the objective functions and penalties as optimization functions in MOGA

Some Observations on Genetic Algorithms

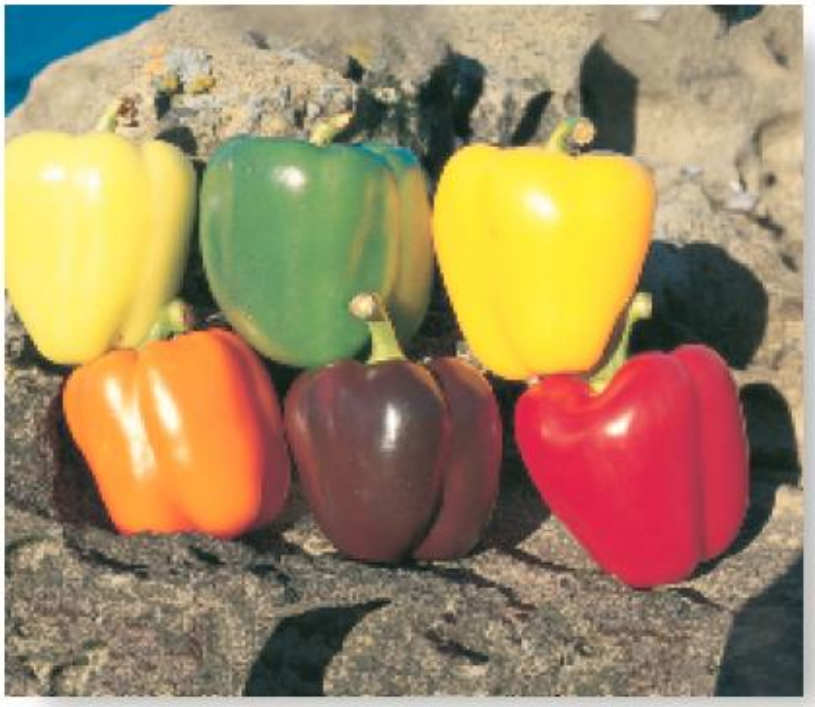
- Local vs. Global Optima
  - No-Free-Lunch Theorem holds here
  - Chances of convergence to Global Optima can be increased through maintaining population diversity (by niche-penalty or increased mutation rate)
- Dynamic Data
  - Operating on dynamic data sets is difficult, as genomes begin to converge early on towards solutions which may no longer be valid for later data.
  - An effective approach is not to maintain any original parents in the population parents, i.e., new parents are selected only from offspring
- GAs cannot effectively solve problems in which the only fitness measure is right/wrong

Some Observations on Genetic Algorithms…

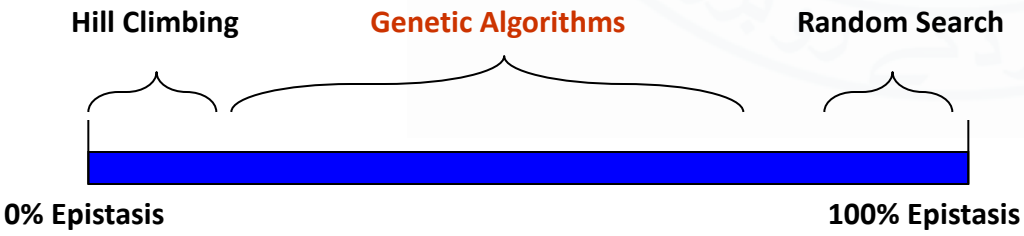
- For specific optimization problems and problem instantiations, simpler optimization algorithms may find better solutions than genetic algorithms (given the same amount of computation time).
- As with all current machine learning problems it is worth tuning the parameters such as mutation probability, recombination probability and population size to find reasonable settings for the problem class being worked on.
- A very small mutation rate may lead to genetic drift (which is non-ergodic in nature).
- A mutation rate that is too high may lead to loss of good solutions unless there is elitist selection.
- A recombination rate that is too high may lead to premature convergence of the genetic algorithm.
- There are theoretical but not yet practical upper and lower bounds for these parameters that can help guide selection.

Some Observations on Genetic Algorithms…

- The Epistasis Dogma
  - Epistasis, in genetics, is defined as the interaction amongst genes
  - In GA, it means that the alleles interact to determine the fitness of the individual
  - GA is most effective in problems that involve medium epistasis



The colors of peppers are determined by the interaction of several genes. An allele *Y* promotes early elimination of chlorophyll (a green pigment), whereas *y* does not. *R* determines red and *r* yellow carotenoid pigments. Alleles *c1* and *c2* of two different genes down-regulate the amounts of carotenoids, causing the lighter shades. Orange is down-regulated red. Brown is green plus red. Pale yellow is down-regulated yellow. [Anthony Griffiths.]



Future Enhancements in GAs

- Use of MOGA
- Parallel GA
- Use of Grid Computing

Some Applications of Genetic Algorithms

- Artificial Creativity
- Automated design, including research on composite material design and multi-objective design of automotive components
- Automated design of mechatronic systems
- Automated design of industrial equipment
- Automated design of sophisticated trading systems in the financial sector.
- Calculation of Bound states and Local-density approximations.
- Chemical kinetics
- Configuration applications,
- Container loading optimization.
- Code-breaking
- Design of water distribution systems.
- Distributed computer network topologies.
- Electronic circuit design, known as Evolvable hardware.
- File allocation for a distributed system.
- Game Theory Equilibrium Resolution.
- Learning Robot behavior using Genetic Algorithms.
- Learning fuzzy rule base using genetic algorithms.
- Linguistic analysis, including Grammar Induction and other aspects of Natural Language Processing (NLP) such as word sense disambiguation.

Some Applications of Genetic Algorithms…

- Marketing Mix Analysis
- Mobile communications infrastructure optimization.
- Molecular Structure Optimization (Chemistry).
- Multiple population topologies and interchange methodologies.
- Optimization of data compression systems, for example using wavelets.
- Protein folding and protein/ligand docking.
- Plant floor layout.
- Representing rational agents in economic models such as the cobweb model.
- Scheduling applications, including job-shop scheduling. The objective being to schedule jobs in a sequence dependent or non-sequence dependent setup environment in order to maximize the volume of production while minimizing penalties such as tardiness.
- Selection of optimal mathematical model to describe biological systems.
- Software engineering
- Solving the machine-component grouping problem required for cellular manufacturing systems.
- Hardware bug finding during the Validation process.

Some Applications of Genetic Algorithms…

- Tactical asset allocation and international equity strategies.
- Timetabling problems, such as designing a non-conflicting class timetable for a large university.
- Training artificial neural networks when pre-classified training examples are not readily obtainable
- Traveling Salesman Problem.
- Control Applications
- Robotic Path Planning

End of Lecture

The law of heredity is that all undesirable traits come from the other parent.

Anon.