Genetic Algorithms

What Do the Following 3 Things Have in Common?

Genetic Algorithms (GAs)

- GAs design jet engines.
- GAs draw criminals.
- GAs program computers.

A Potpourri of Applications

1. General Electric's *EnGeneous* (generalized engineering optimization).
2. Face space (criminology).
3. Genetic programming (machine learning).

Gas Turbine Design

Jet engine design at General Electric (Powell, Tong, & Skolkick, 1989)

- Coarse optimization - 100 design variables.
- Hybrid GA + numerical optimization + expert system.
- Found **2%** increase in efficiency.
- Spending $250K to test in laboratory.
- Boeing 777 design based on these results.

Criminal-likeness Reconstruction

No closed form fitness function (Caldwell & Johnston, 1991).

- Human witness chooses faces that match best.
- GA creates new faces from which to choose.

What are GAs?

- GAs are biologically inspired class of algorithms that can be applied to, among other things, the optimization of nonlinear multimodal functions.
- Solves problems in the same way that nature solves the problem of adapting living organisms to the harsh realities of life in a hostile world: evolution.

What is a Genetic Algorithm (GA)?

A GA is an adaptation procedure based on the mechanics of natural selection and genetics.

GAs have 2 essential components:

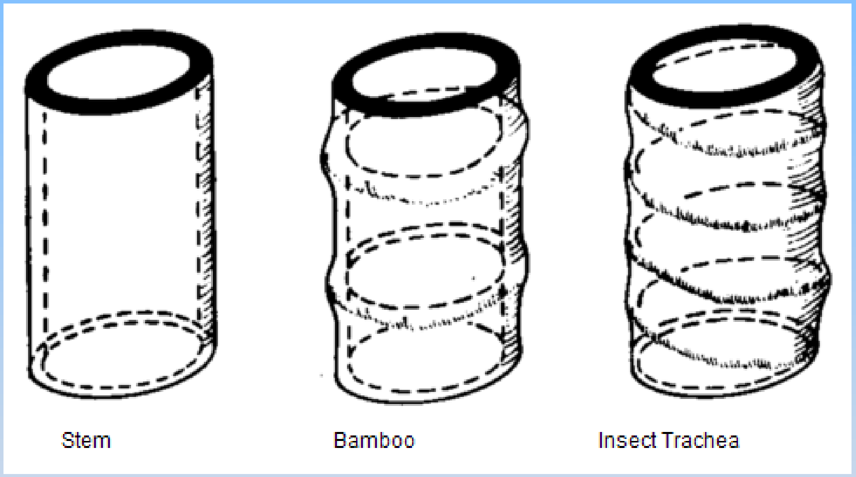
1. Survival of the fittest (selection)
2. Variation

Nature as Problem Solver

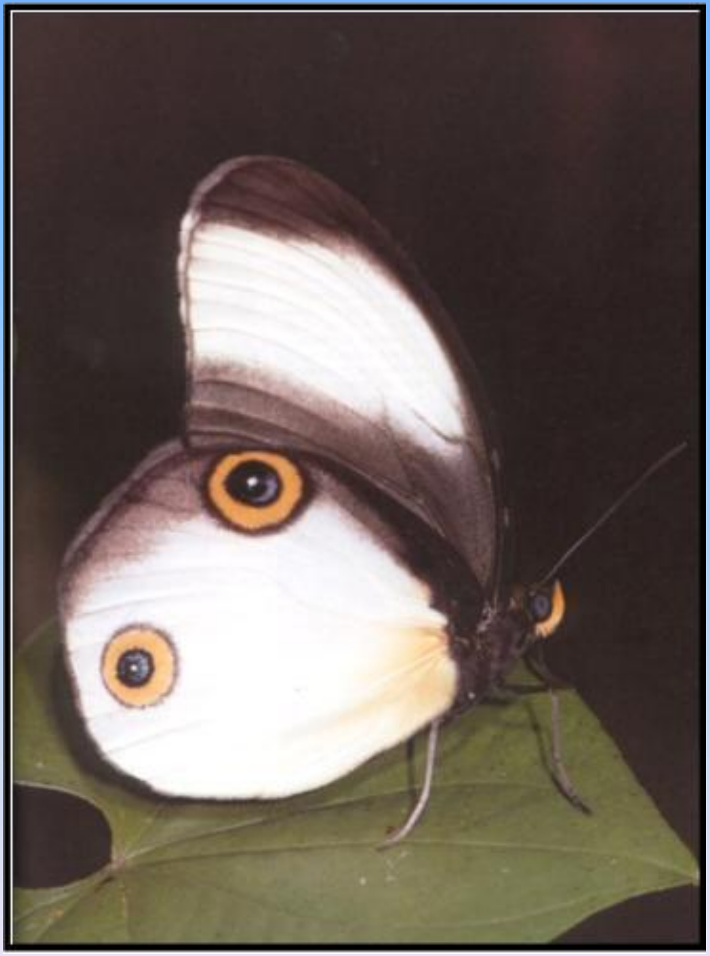
Beauty-of-nature argument

How Life Learned to Live (Tributsch, 1982, MIT Press)

Example: Nature as structural engineer



Owl Butterfly



Evolutionary is Revolutionary!

Street distinction evolutionary vs. revolutionary is false dichotomy.

3.5 Billion years of evolution can’t be wrong.

Complexity achieved in *short* time in nature.

Can we solve complex problems as quickly and reliably on a computer?

Why Bother?

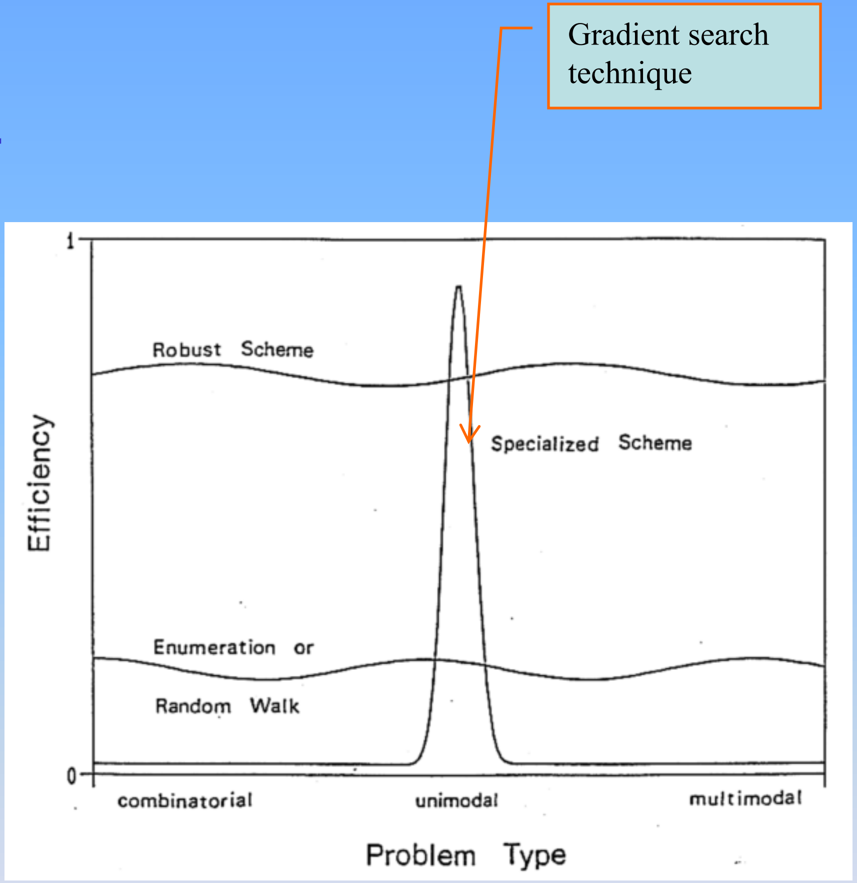
Lots of ways to solve problems:

- Calculus
- Hill-climbing
- Enumeration
- Operations research: linear, quadratic, nonlinear programming

Why bother with biology?

Robustness = Breadth + Efficiency.

A hypothetical problem spectrum:



GAs Not New

John Holland at **University of Michigan** pioneered in the 50s.

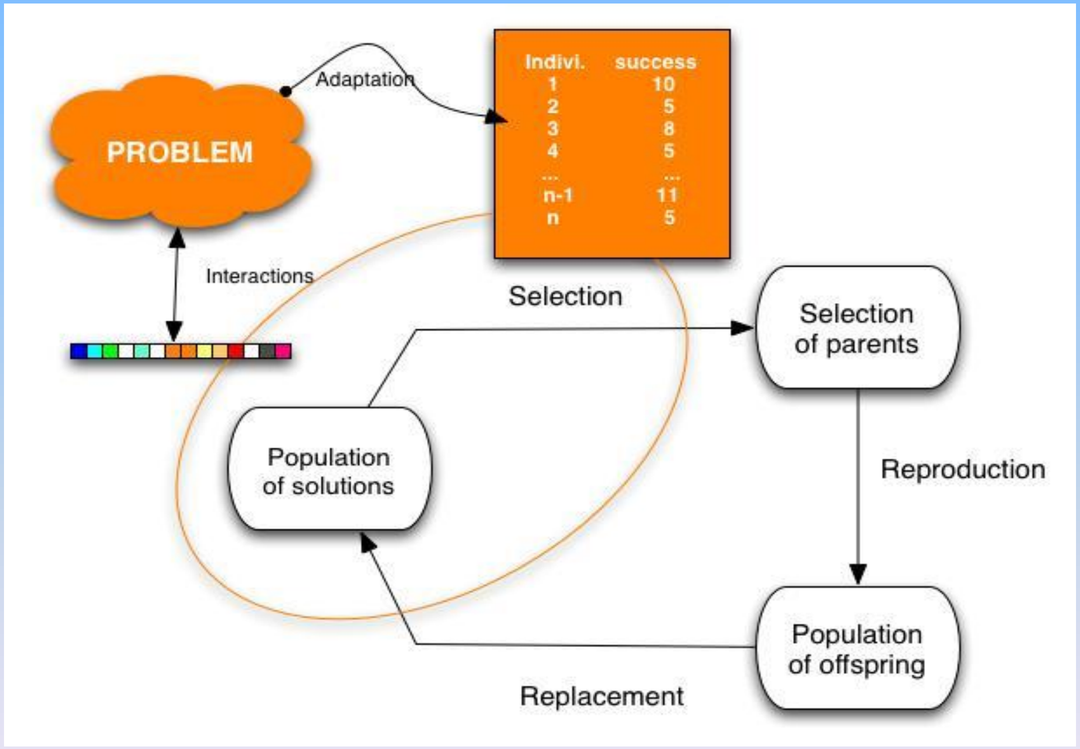
Other evolutionaries: Fogel, Rechenberg, Schwefel.

Back to the cybernetics movement and early computers.

Reborn in the 70s.

Genetic algorithms

Variant of local beam search with *sexual recombination.*



How GAs are different from traditional methods?

1. GAs work with a **coding of the parameter set**, not the parameter themselves.
2. GAs search from a **population of points**, not a single point.
3. GAs use payoff (**objective function**) information, not derivatives or other auxilliary knowledge.
4. GAs use **probabilistic transition rules**, not deterministic rules.

Genetic Algorithm

GAs APPROACH TO THE PROBLEM

Natural parameter set of the optimisation problem is represented as a finite-length string

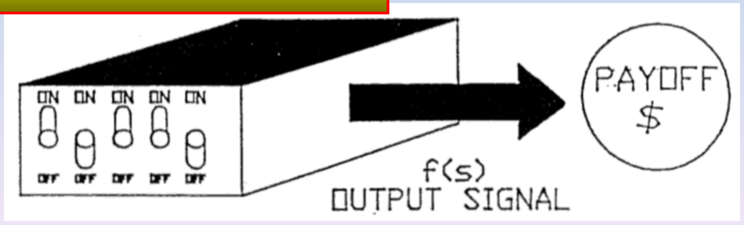
Problem: Maximise the function $f(x) = x^2$ on the integer interval [0, 31]

Traditional approach: twiddle with parameter x

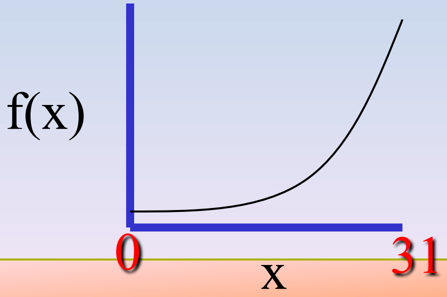
$$f = f(s)$$

Setting of five switches

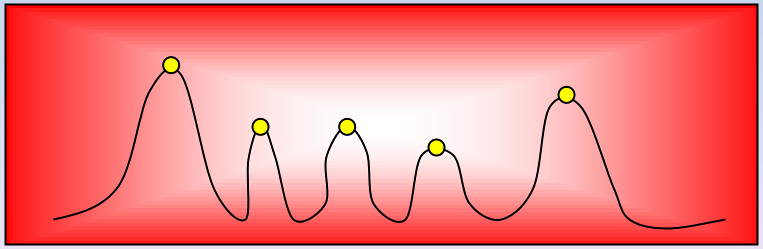
output



GA doesn't need to know the workings of the black box.



Main Attractions of Genetic Algorithm

GA	Traditional Optimization Approaches
• Simplicity of operation and power of effect	
• **unconstrained**	• Limitations: continuity, derivative existence, unimodality
• work from a rich database of points **simultaneously**, climbing many peaks in parallel	• move gingerly from a single point in the decision space to the next using some transition rule
• population of strings = points • **Population of well-adapted diversity**	

Initial Population

Genetic Algorithm

GA

- **Initial Step:** random start using successive coin flips

GA uses coding

$$\left.\begin{array}{l}01101\\11000\\01000\\10011\end{array}\right\}\quad \textbf{population}$$

- blind to auxiliary information

GAs are blind, **only payoff values** associated with individual strings are required

- Searches from a population

Uses probabilistic transition rules to guide their search towards regions of the search space with likely improvement

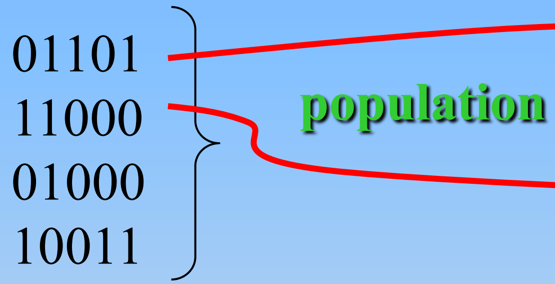
Reproduction

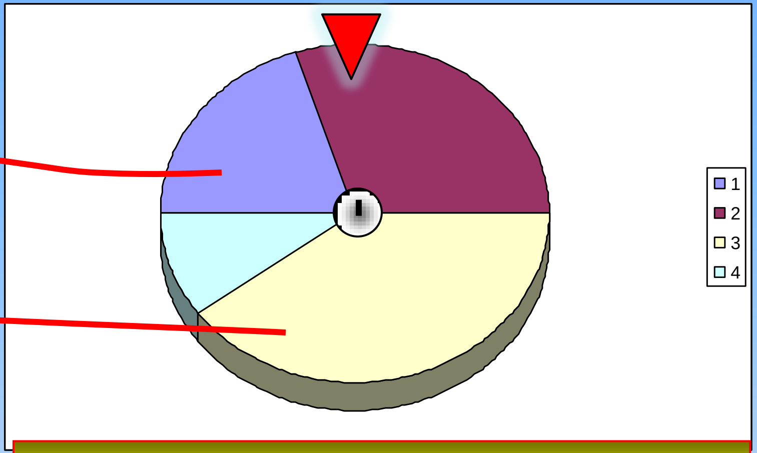
Genetic Algorithm

REPRODUCTION

• **Selection** according to ***fitness***

GA uses coding





Weighted Roulette wheel

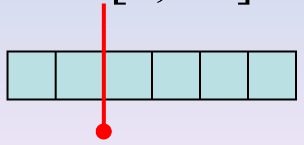
• **Replication**

Mating pool (tentative population)

CROSSOVER

• **Crossover** – randomized information exchange

Crossover point k = [1, ℓ-1]



GA builds solutions from the past partial solutions of the previous trials

Mutation

Genetic Algorithm

MUTATION

- **Reproduction and crossover** may become overzealous and lose some potentially useful genetic material

- **Mutation** protects against irrecoverable loss; it serves as an insurance policy against premature loss of important notions

- **Mutation rates**: in the order of **1 mutation per a thousand bit position transfers**

Sample Problem

Genetic Algorithm

SAMPLE PROBLEM

- **Maximize $f(x) = x^2$; where x is permitted to vary between 0 and 31**

1. **Coding of decision variables as some finite length string**

X as binary unsigned integer of length 5

[0, 31] = [00000, 11111]

2. **Constant settings**

Pmutation=0.0333

Pcross=0.6

Population Size=30

DeJong(1975) suggests high crossover Probability, low mutation probability (inversely proportional to the pop.size), and A moderate population size

Genetic Algorithm

SAMPLE PROBLEM

- Maximize $f(x) = x^2$; where x is permitted to vary between 0 and 31

3. **Select initial population at random** (use even numbered population size)

String number	Initial Population	X value	f(x)	pselect $\frac{f_i}{\sum f}$	Expected count $\frac{f_i}{\bar{f}}$	Actual count(Roulette Wheel)
1	01101	13	169	0.14	0.58	1
2	11000	24	576	0.49	1.97	2
3	01000	8	64	0.06	0.22	0
4	10011	19	361	0.31	1.23	1

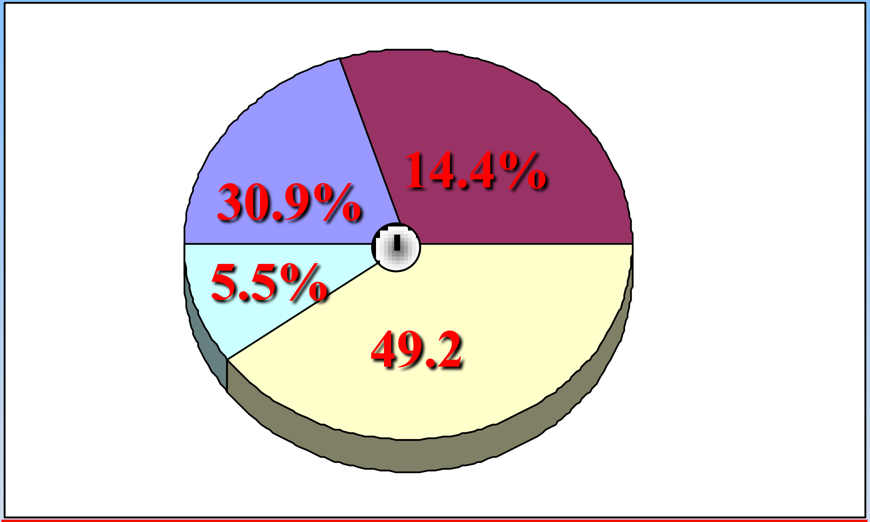
Sum	1170
Ave.	293
Max.	576

Genetic Algorithm

SAMPLE PROBLEM

- Maximize $f(x) = x^2$; where x is permitted to vary between 0 and 31

4. **Reproduction:** select mating pool by spinning roulette wheel 4 times.



Weighted Roulette wheel

	pselect
01101	0.14
11000	0.49
01000	0.06
10011	0.31

The best get more copies.
The average stay even.
The worst die off.

Choosing offspring for the next generation

```
int Select(int Popsize, double Sumfitness, Population Pop){
    partSum = 0                                  [0,1]

    rand=Random * Sumfitness
    j=0
    Repeat
        j++;
        partSum = partSum + Pop[j].fitness
    Until (partSum >= rand) or (j = Popsize)

    Return j
}
```

Genetic Algorithm

SAMPLE PROBLEM

5. Crossover – strings are mated randomly using coin tosses to pair the couples

- mated string couples crossover using coin tosses to select the crossing site

String number	Mating Pool after Reproduction	Mate (randomly selected)	Crossover site (random)	New population	X-value	$f(x)=x^2$
1	0110|1	2	4	01100	12	144
2	1100|0	1	4	11001	25	625
3	11|000	4	2	11011	27	729
4	10|011	3	2	10000	16	256

Sum	1754
Ave.	439
Max.	729

Algorithmic Steps

The Genetic Algorithm

1. **Initialize** the algorithm.

   Randomly initialize each **individual chromosome** in the population of size **N** (N must be **even**), and compute each individual’s **fitness**.

The Genetic Algorithm

1. Initialize the algorithm. Randomly initialize each individual chromosome in the population of size N (N must be even), and compute each individual’s fitness.
2. **Select N/2 pairs** of individuals for **crossover**. The probability that an individual will be selected for crossover is **proportional** to its **fitness**.

The Genetic Algorithm

1. Initialize the algorithm. Randomly initialize each individual chromosome in the population of size N (N must be even), and compute each individual's fitness.
2. Select N/2 pairs of individuals for crossover. The probability that an individual will be selected for crossover is proportional to its fitness.
3. **Perform crossover operation** on **N/2 pairs** selected in Step1.

   Randomly mutate bits with a **small probability** during this operation.

The Genetic Algorithm

1. Initialize the algorithm. Randomly initialize each individual chromosome in the population of size N (N must be even), and compute each individual's fitness.
2. Select N/2 pairs of individuals for crossover. The probability that an individual will be selected for crossover is proportional to its fitness.
3. Perform crossover operation on N/2 pairs selected in Step1. Randomly mutate bits with a small probability during this operation.
4. **Compute fitness** of **all** individuals in new population.

The Genetic Algorithm

5. **(Optional Optimization)**

   Select N fittest individuals from combined population of size **2N** consisting of **old** and **new populations** pooled together.

The Genetic Algorithm

5. (Optional Optimization) Select N fittest individuals from combined population of size 2N consisting of old and new populations pooled together.

6. **(Optional Optimization)**

   Rescale fitness of population.

The Genetic Algorithm

5. (Optional Optimization) Select N fittest individuals from combined population of size 2N consisting of old and new populations pooled together.
6. (Optional Optimization) Rescale fitness of population.
7. **Determine maximum fitness of individuals** in the population.

```
If |max fitness – optimum fitness| < tolerance Then
    Stop
Else
    Go to Step1.
```

A Simple GA Example

Let's see a demonstration for a **GA** that maximizes the function

$$f(x) = \left(\frac{x}{c}\right)^n$$

n =10

c = 2^{30} -1 = 1,073,741,823

Simple GA Example

Function to evaluate:

$$f(x) = \left(\frac{x}{coeff} \right)^{10}$$

Fitness Function or Objective Function

coeff – chosen to normalize the x parameter when a bit string of length lchrom =30 is chosen.

$$coeff = 2^{30} - 1$$

When the x value is normalized, the max. value of the function will be:

$$f(x) = 1.0$$

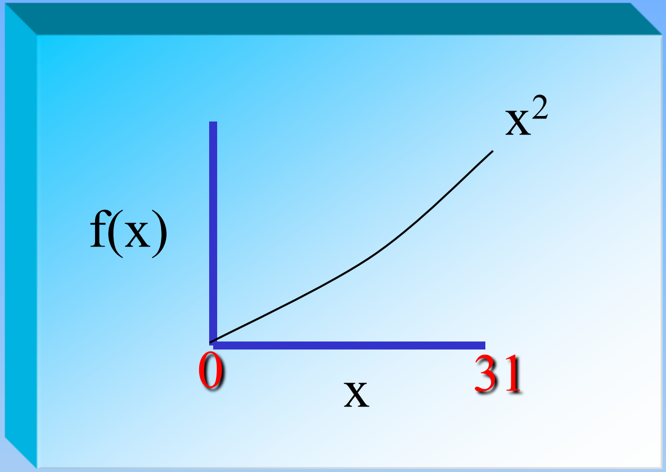
This happens when $x = 2^{30} - 1$ for the case when lchrom=30

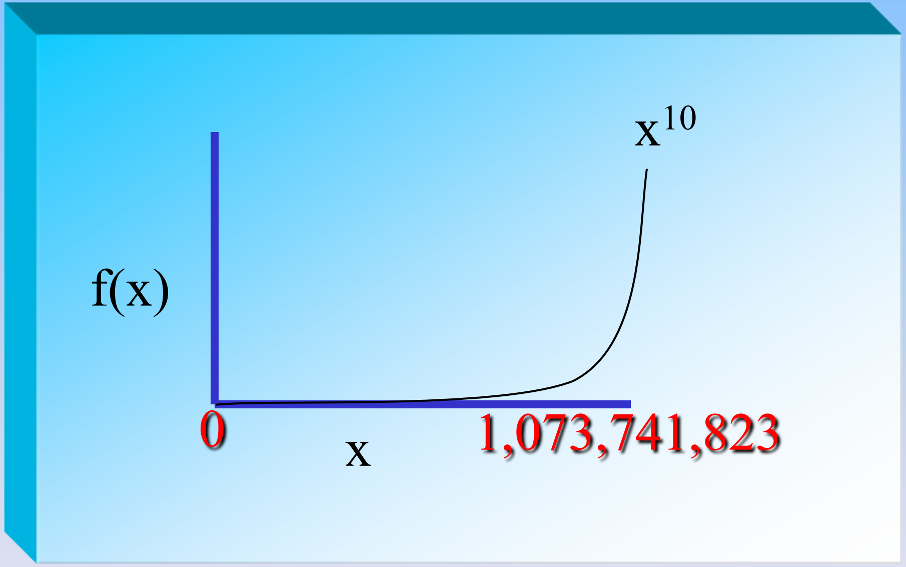
Test Problem Characteristics

With a string length=**30**, the search space is much larger, and random walk or enumeration should not be so profitable.

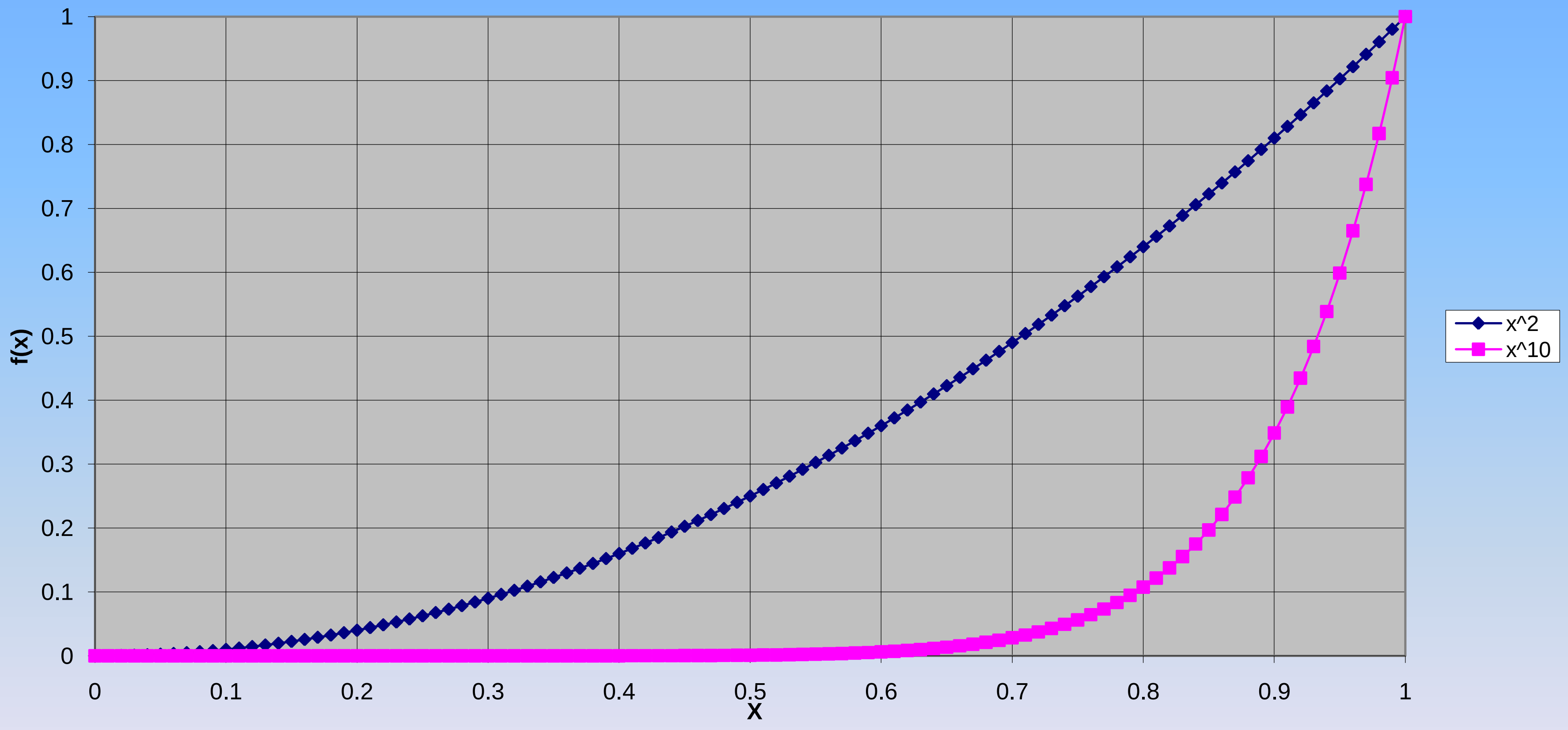
There are **2^{30}=1.07(10^{10}) points**. With over 1.07 billion points in the space, one-at-a-time methods are unlikely to do very much very quickly. Also, only **1.05** percent of the points have a value greater than **0.9**.

Comparison of the functions on the unit interval



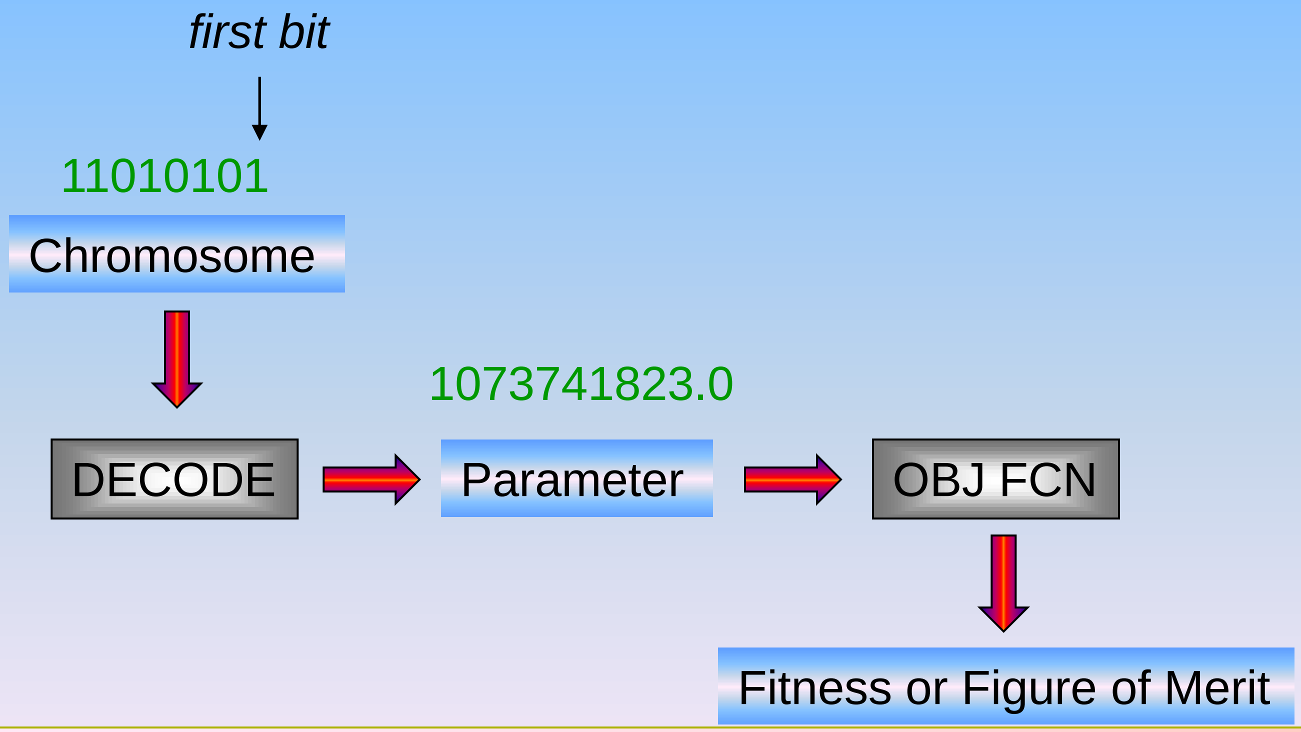


Actual Plot



Decoding a String

For every problem, we must create a procedure that decodes a string to create a parameter (or set of parameters) appropriate for that problem.



GA Parameters

A series of parametric studies [De Jong, 1975] across a five function suite of optimization problems suggested that good GA performance requires the choice of:

- High crossover probability
- Low mutation probability (inversely proportional to the population size)
- Moderate Population Size

(e.g. pmutation=0.0333**, pcross=**0.6**, popsize=**30**)**

Limits of GA

- GAs are characterized by a **voracious appetite** for **processing power** and **storage capacity**.

- GAs have **no convergence guarantees** in arbitrary problems.

Limits of GAs

- **GA**s sort out interesting areas of a space **quickly**, but they are a **weak method**, <u>without</u> the guarantees of more convergent procedures.

- This does not reduce their utility however. More convergent methods sacrifice **globality** and **flexibility** for their convergence, and are limited to a narrow class of problem.

- GAs can be used where more convergent methods dare not tread.

Advantages of GAs

- Well-suited to a **wide-class of problems**
- Do <u>not</u> rely on the **analytical properties** of the function to be optimized (such as the existence of a derivative)

Advanced GA Architectures

GA + Any Local Convergent Method

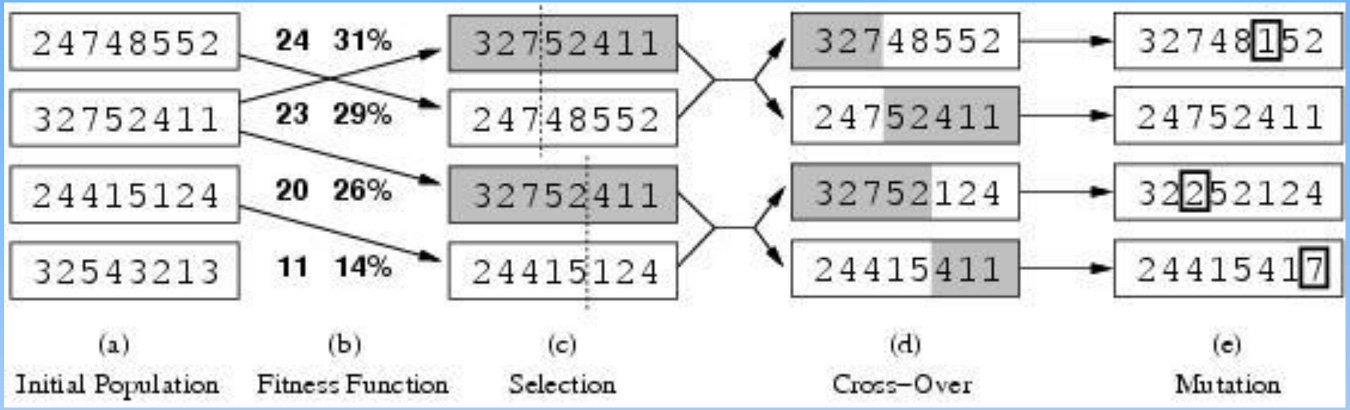
- Start search using GA to sort out the interesting hills in your problem. Once GA ferrets out the best regions, apply locally convergent scheme to climb the local peaks.

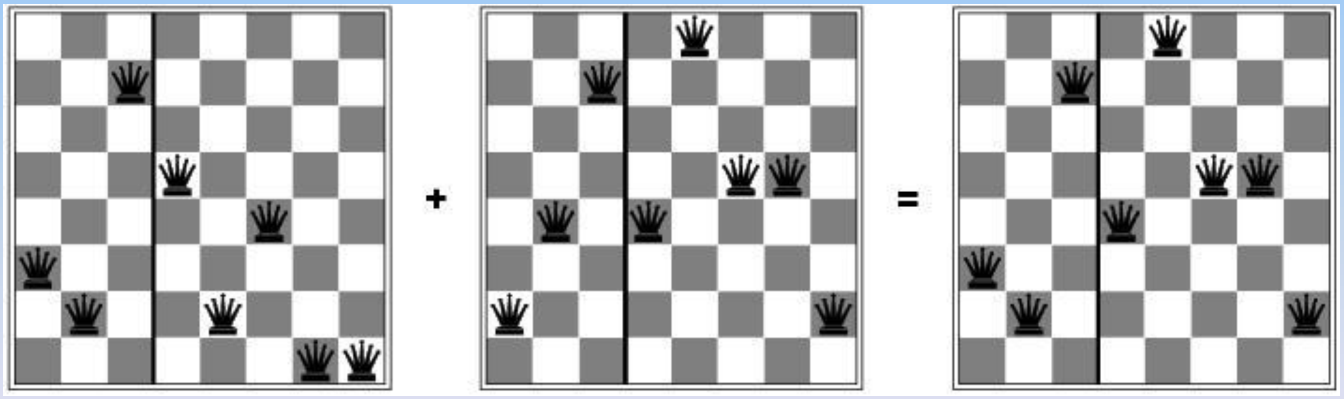
Other Applications

Optimization of a choice of Fuzzy Logic parameters

Genetic algorithms

Variant of local beam search with *sexual recombination.*



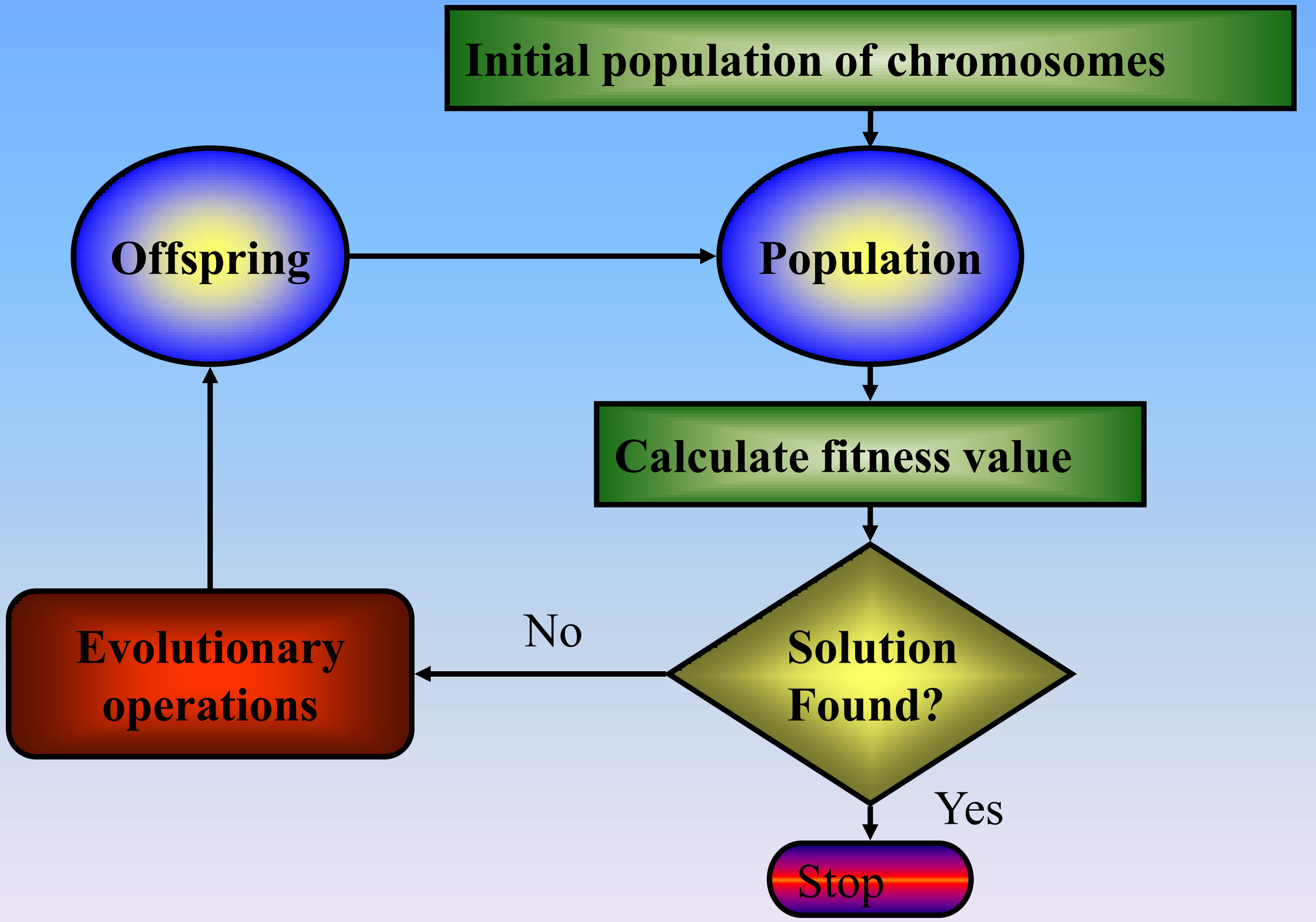


<u>Fitness function</u>: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28);

24/(24+23+20+11) = 31%

23/(24+23+20+11) = 29% etc

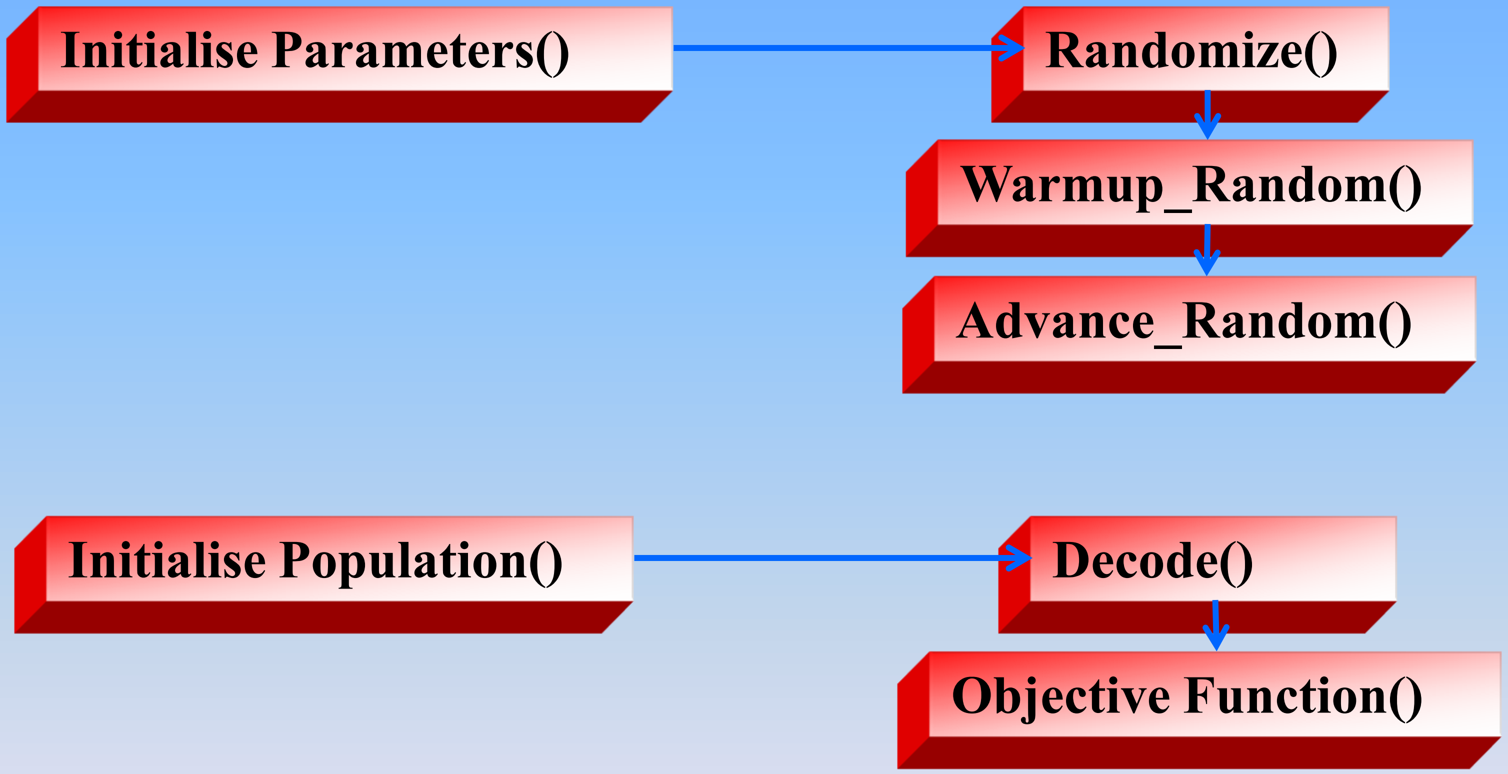
Simple GA Implementation



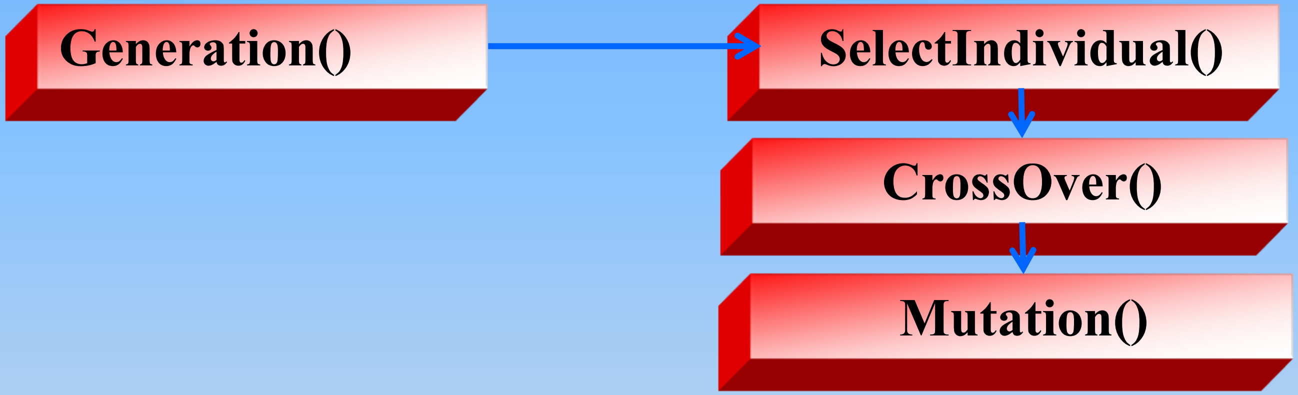
Implementation

Based on **SGA-C,** A C-language Implementation of a Simple Genetic Algorithm

Phase 1 – General Initialisation



Phase 2 – Generation of Chromosomes



Running the GA System

```
Gen = 0
Initialize( OldPop )

Do
  Gen = Gen + 1
  Generation( OldPop, NewPop )

  For ii = 1 To PopSize
      OldPop(ii) = NewPop(ii)  'advance the generation
  Next ii

Loop Until ( (Gen > MaxGen) or (MaxFitness > DesiredFitness) )
```

Initialisation

Initialise Parameters

```
PopSize = 30      'population size
lchrom = 30       'chromosome length
MaxGen = 10
PCross = 0.6
PMutation = 0.0333

ReDim GenStat(1 To (MaxGen + 1))

'Initialize random number generator
Randomize

'Initialize counters
NMutation = 0
NCross = 0
```

Randomization

Randomize

[0,1]

```
Sub Randomize()
    Randomize Timer
    Warmup_Random (Rnd * 1)
End Sub
```

Randomization

Warmup_Random()

[0, 1]

```
Sub Warmup_Random(RandomSeed As Single)
  Dim j1  As Integer
  Dim ii  As Integer

  Dim NewRandom As Single
  Dim PrevRandom As Single

  OldRand(55) = RandomSeed
  NewRandom = 0.000000001
  PrevRandom = RandomSeed

  For  j1 = 1 To 54
          ii = (21 * j1) Mod 55 'multiply first, before modulus
          OldRand(ii) = NewRandom
          NewRandom = PrevRandom - NewRandom
          If (NewRandom < 0) Then NewRandom = NewRandom + 1
          PrevRandom = OldRand(ii)
  Next j1
  Advance_Random
  Advance_Random
  Advance_Random
  jrand = 0
End Sub
```

[0, 1]

Randomization

Advance_Random()

```
Sub Advance_Random()
  Dim j1 As Integer
  Dim New_Random As Single

  For  j1 = 1 To 24
     New_Random = OldRand(j1) - OldRand(j1 + 31)
     If (New_Random < 0) Then New_Random = New_Random + 1
     OldRand(j1) = New_Random
  Next j1

  For  j1 = 25 To 55
     New_Random = OldRand(j1) - OldRand(j1 - 24)
     If (New_Random < 0) Then New_Random = New_Random + 1
     OldRand(j1) = New_Random
  Next j1
End Sub
```

Max: 24+31=55

Max: 55-24=31

Random

```
'Fetch a single random number between 0.0 and 1.0 -
Subtractive Method
'See Knuth, D. (1969), v. 2 for details

Function Random() As Single
  jrand = jrand + 1
  If jrand > 55 Then
     jrand = 1
     Advance_Random
  End If
  Random = OldRand(jrand)
End Function
```

Knuth, 1969. D.E. Knuth *The Art of Computer Programming* **2**, Addison-Wesley, Reading, MA (1969).

Initialise Population

InitPop()

```
Sub InitPop()
  Dim j As Integer
  Dim j1 As Integer

  For j = 1 To PopSize
        With OldPop(j)
              For j1 = 1 To lchrom
                    .Chromosome(j1) = Flip(0.5)
              Next j1
              .x = Decode(.Chromosome, lchrom) 'decode the string
              .Fitness = ObjFunc(.x) 'evaluate initial fitness
              .Parent1 = 0
              .Parent2 = 0
              .XSite = 0
        End With
  Next j
End Sub
```

Max: 24+31=55

Initialise Population

Decode()

```
'decodes the string to create a parameter or set of parameters
'appropriate for that problem
'Decode string as unsigned binary integer: true=1, false=0

Function Decode(Chrom() As Boolean, lbits As Integer) As Single
  Dim j As Integer
  Dim Accum As Single
  Dim PowerOf2 As Single

  Accum = 0
  PowerOf2 = 1
  For j = 1 To lbits
      If Chrom(j) Then Accum = Accum + PowerOf2
      PowerOf2 = PowerOf2 * 2
  Next j
  Decode = Accum
End Function
```

Initialise Population

Objective Function()

```
'Fitness function = f(x) = (x/c) ^ n

Function ObjFunc(x As Single) As Single 'coef = (2 ^ 30)-1 = 1073741823
'coef is chosen to normalize the x parameter when a bit string of
 length lchrom=30 is   chosen

 'since the x value has been normalized, the maximum value of the fcn wil be f(x)=1,
 'when x=(2^30)-1, for the case when lchrom=30

  Const coef  As Single = 1073741823  'coefficient to normalize domain
  Const   n  As Single = 10 'power of x

  ObjFunc = (x / coef) ^ n

End Function
```

Generation of Chromosomes

SelectIndividual()

```
Function SelectIndividual(PopSize As Integer, SumFitness As Single, Pop() As
IndividualType) As Integer
  Dim RandPoint As Single
  Dim PartSum As Single
  Dim j As Integer

  PartSum = 0
  j = 0
  RandPoint = Random * SumFitness

  Do 'find wheel slot
       j = j + 1
       PartSum = PartSum + Pop(j).Fitness

  Loop Until ((PartSum >= RandPoint) Or (j = PopSize))
  SelectIndividual = j
End Function
```

Select a single individual or offspring for the next generation via roulette wheel selection

Generation of Chromosomes

```
Function CrossOver(Parent1() As Boolean, Parent2() As Boolean, Child1() As Boolean, Child2() As Boolean,
lchrom As Integer, NCross As Integer, NMutation As Integer, jcross As Integer, Pcross, PMutation As Single)

  Dim j As Integer
  If (Flip(PCross)) Then
      jcross = Rndx(1, lchrom - 1) 'cross-over site is selected between 1 and the last cross site
      NCross = NCross + 1
  Else   ‘ use full-length string l, and so a bit-by-bit mutation will take place despite the absence of a cross
      jcross = lchrom
  End If

  For j = 1 To jcross    ‘ 1st exchange, 1 to 1 and 2 to 2
     Child1(j) = Mutation(Parent1(j), PMutation, NMutation)
     Child2(j) = Mutation(Parent2(j), PMutation, NMutation)
  Next j

  If jcross <> lchrom Then  ‘ 2nd exchange, 1 to 2 and 2 to 1
    For j = jcross + 1 To lchrom
       Child1(j) = Mutation(Parent2(j), PMutation, NMutation)
       Child2(j) = Mutation(Parent1(j), PMutation, NMutation)
    Next j
  End If
End Function
```

```
CrossOver OldPop(Mate1).Chromosome,  OldPop(Mate2).Chromosome, _
                    NewPop(j).Chromosome,  NewPop(j + 1).Chromosome, _
                    lchrom, NCross, NMutation, jcross, PCross, PMutation
```

Generation of Chromosomes

Mutation()

```
'Mutate an allele with PMutation, count number of mutations

Function Mutation(Alleleval As Boolean, PMutation As Single, NMutation As Integer)
As Boolean
  Dim Mutate As Boolean

  Mutate = Flip(PMutation)
  If Mutate Then
      NMutation = NMutation + 1
      Mutation = Not Alleleval
  Else
      Mutation = Alleleval
  End If
End Function
```

```
Function Flip(Probability As Single) As
Boolean
  If Probability = 1 Then
    Flip = True
  Else
    Flip = (Rnd <= Probability)
  End If
End Function
```

Generation of Chromosomes

Generation()

```
Sub Generation()
  Dim j As Integer
  Dim Mate1 As Integer
  Dim Mate2 As Integer
  Dim jcross As Integer

  j = 1
  Do
    'Pick a pair of mates
    Mate1 = SelectIndividual(PopSize, SumFitness, OldPop)
    Mate2 = SelectIndividual(PopSize, SumFitness, OldPop)

    'Crossover and mutation - mutation embedded within crossover
    CrossOver OldPop(Mate1).Chromosome,  OldPop(Mate2).Chromosome, _
                    NewPop(j).Chromosome,  NewPop(j + 1).Chromosome, _
                    lchrom, NCross, NMutation, jcross, PCross, PMutation

    'Decode string, evaluate fitness & record parentage date on both children
    With NewPop(j)
      .x = Decode(.Chromosome, lchrom)
      .Fitness = ObjFunc(.x)
      .Parent1 = Mate1
      .Parent2 = Mate2
      .XSite = jcross
    End With
    With NewPop(j + 1)
      .x = Decode(.Chromosome, lchrom)
      .Fitness = ObjFunc(.x)
      .Parent1 = Mate1
      .Parent2 = Mate2
      .XSite = jcross
    End With

    j = j + 2 'increment population index

  Loop Until (j > PopSize)
End Sub
```

Why use Fitness Scaling?

At the start of the GA run, it is common to have a few extraordinary individuals in a population of mediocre colleagues.

If left to the selection rule

$$\text{pselect}_i = \boxed{\frac{f_i}{\sum f}}$$

the extraordinary individuals would take over a significant proportion of the finite population in a single generation.

This is undesirable, leading to a premature convergence!

Why use Fitness Scaling?

Late in the run, there may still be significant diversity within the population. However, the population's **average fitness** may be close to the
population's **best fitness**.

If this is left alone,

- **average members** get nearly the same number of copies in future generations, and
- the survival of the fittest necessary for improvement becomes a **random walk** among the **mediocre**.

In both cases, at the beginning of the run, and as the run matures, **fitness scaling** can help.

Why use Fitness Scaling?

Linear Scaling

$$f' = a\,f + b$$

Scaled fitness

Raw fitness

In all cases, we want $f'_{ave} = f_{ave}$ because subsequent use of the selection procedure will insure that each average population member contributes one expected offspring to the next generation.

Why use Fitness Scaling?

To control the number of offspring given to the population **member with the maximum raw fitness**, we choose the other scaling relationship to obtain a **scaled maximum fitness**.

Scaled Maximum Fitness:

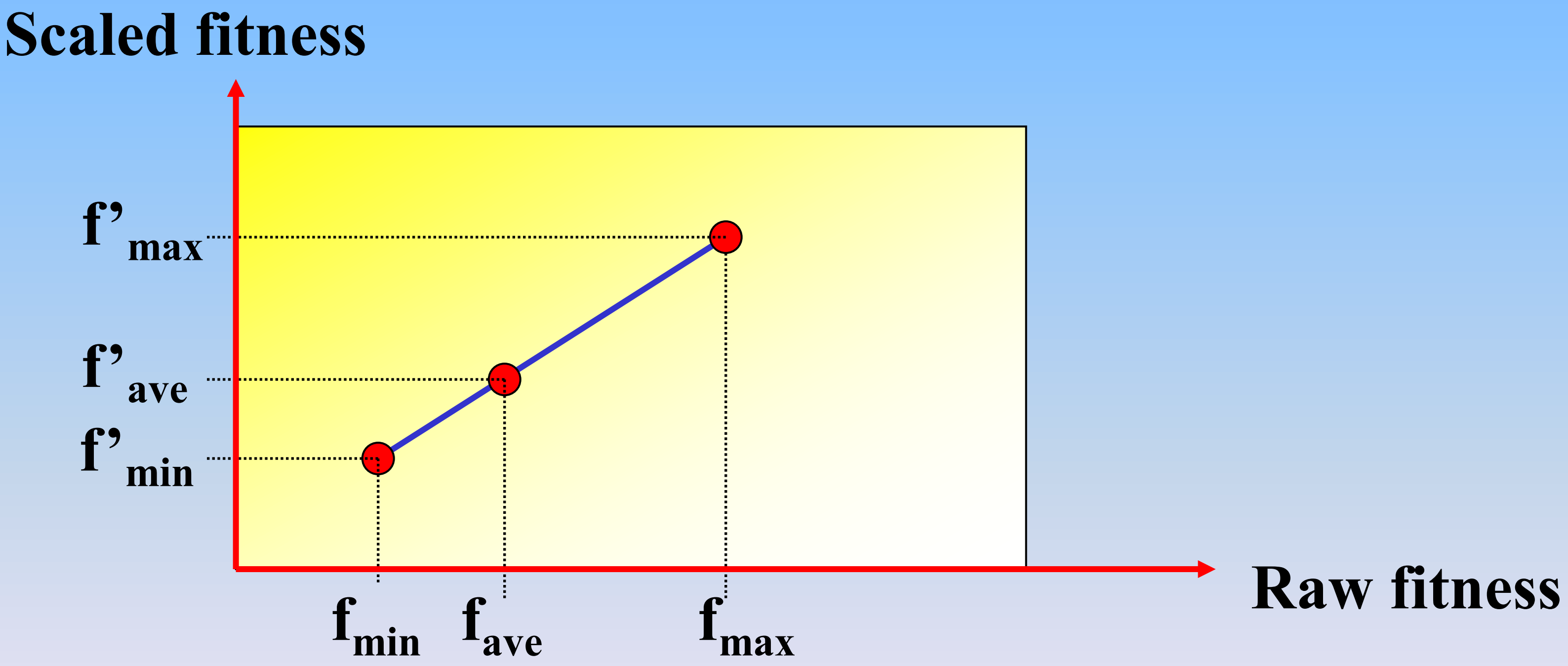
$$f'_{max} = c_{mult} * f_{ave}$$

Number of expected copies desired for the best population member

For a typical population size of **n = 50 to 100**, C_{mult} **= [1.2, 2]** has been used successfully

Fitness Scaling

Linear Scaling Under Normal Conditions



Problem with Linear Scaling

Toward the end of a run, the choice of **C_{mult}** stretches the raw fitness values significantly.

This may in turn cause difficulty in applying the linear scaling rule.

The effects of the Linear Scaling rule works during the **initial run** of the GA:

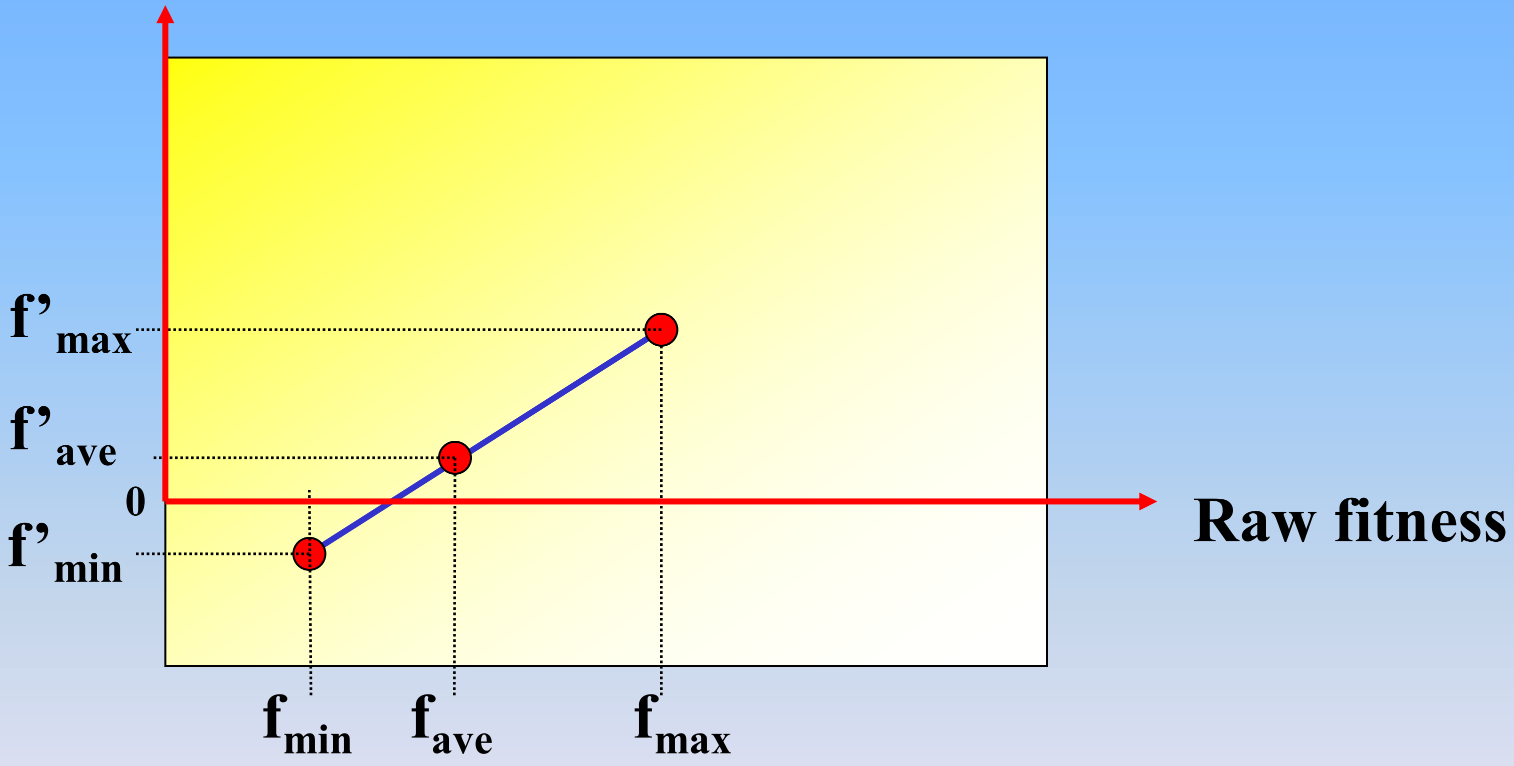
- few extraordinary individuals get scaled down, and
- the lowly members of the population get scaled up

The problem: As the run matures, points with **low fitness** can be scaled to **negative values**!

The stretching required on the relatively close average and maximum raw fitness values causes the low fitness values to go negative after scaling.
See for yourself, TestGA2.xls

Why use Fitness Scaling?

Difficult situation for a linear scaling in mature run



Negative fitness violates non-negativity requirement!

Fitness Scaling

If it's possible to scale to the desired multiple, **Cmult**

Then

Perform linear scaling

Else

Scaling is performed by pivoting about the average value and stretching the fitness until the minimum value maps to zero.

Scaling

Non-negative test:

```
If( min > (fmultiple*avg - max) / (fmultiple - 1.0) ) {
    Perform Normal Scaling
}
```

Eliminating negative fitness values

Solution:

When we cannot scale to the desired cmult, we still maintain equality of the raw and scaled **fitness averages** and we map the minimum raw fitness $\mathbf{f_{min}}$ to a scaled fitness $\mathbf{f'_{min} = 0}$.

Description of Routines:

Prescale – takes the **average**, **maximum** and **minimum** raw fitness values and calculates linear scaling of the coefficients **a** and **b** based on the logic described previously. It takes into account whether the desired $\mathbf{c_{mult}}$ can be reached or not.

Scalepop – called after Prescaling is done. It scales all the individual raw fitness values using the function **Scale**.

Fitness Scaling

```
procedure scalepop(popsize:integer; var max, avg, min, sumfitness:real;
                   var pop:population);
{ Scale entire population }
var j:integer;
    a, b:real;    { slope & intercept for linear equation }
begin
 prescale(max, avg, min, a, b);  { Get slope and intercept for function }
 sumfitness := 0.0;
 for j := 1 to popsize do with pop[j] do begin
   fitness := scale(objective, a, b);
   sumfitness := sumfitness + fitness;
  end;
end;
```

```
function scale(u, a, b:real):real;
{ Scale an objective function value }
begin  scale := a * u + b end;
```

Fitness Scaling

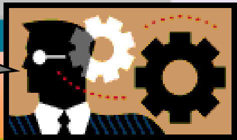
```
{ scale.sga: contains prescale, scale, scalepop for scaling fitnesses }

procedure prescale (umax, uavg, umin:real; var a, b:real);
{ Calculate scaling coefficients for linear scaling }
const fmultiple = 2.0;     { Fitness multiple is 2 }
var   delta:real;          { Divisor }
begin
 if umin > (fmultiple*uavg - umax) / (fmultiple - 1.0) { Non-negative test }
    then begin  { Normal Scaling }
      delta := umax - uavg;
      a := (fmultiple - 1.0) * uavg / delta;
      b := uavg * (umax - fmultiple*uavg) / delta;
  end else begin   { Scale as much as possible }
       delta := uavg - umin;
       a := uavg / delta;
       b := -umin * uavg / delta;
 end;
end;
```

Linear Scaling

Stretch fitness until Minimum maps to zero.

Let's try to solve an example using a stored GA run.



Why Scaling?

Simple scaling helps prevent the early domination of extraordinary individuals, while it later on encourages a healthy competition among near equals.

Multiparameter Code

Constructed from concatenated, mapped, fixed point codes

```
procedure extract_parm(var chromfrom, chromto:chromosome;
                       var jposition, lchrom, lparm:integer);
{ Extract a substring from a full string }
var j, jtarget:integer;
begin
       j := 1;
       jtarget := jposition + lparm - 1;
       if jtarget > lchrom then jtarget := lchrom; { Clamp if excessive }
       while (jposition <= jtarget) do begin
                chromto[j] := chromfrom[jposition];
                jposition := jposition + 1;
                j := j + 1;
        end;
end;
```

Multiparameter Code

Constructed from concatenated, mapped, fixed point codes

```
procedure decode_parms(var nparms, lchrom:integer;
                       var chrom:chromosome;
                       var parms:parmspecs);
var j, jposition:integer;
    chromtemp:chromosome; { Temporary string buffer }
begin
 j := 1; { Parameter counter }
 jposition := 1; { String position counter }
 repeat
  with parms[j] do if lparm>0 then begin
    extract_parm(chrom, chromtemp, jposition, lchrom, lparm);
    parameter := map_parm( decode(chromtemp, lparm),
                           maxparm, minparm, power(2.0, lparm)-1.0 );
   end else parameter := 0.0;
  j := j + 1;
 until j > nparms;
end;
```

```
function map_parm(x, maxparm, minparm, fullscale:real):real;
{ Map an unsigned binary integer to range [minparm,maxparm] }
begin
map_parm :=  minparm + (maxparm -minparm)/fullscale*x
end;
```

Multiparameter Code

```
function decode(chrom:chromosome; lbits:integer):real;
{ Decode string as unsigned binary integer - true=1, false=0 }
var j:integer;
    accum, powerof2:real;
begin
 accum := 0.0; powerof2 := 1;
 for j := 1 to lbits do begin
   if chrom[j] then accum := accum + powerof2;
   powerof2 := powerof2 * 2;
  end;
 decode := accum;
end;
```

References

Genetic Algorithms: Darwin-in-a-box
Presentation by Prof. David E. Goldberg
Department of General Engineering
University of Illinois at Urbana-Champaign
deg@uiuc.edu

Neural Networks and Fuzzy Logic Algorithms
by Stephen Welstead

Soft Computing and Intelligent Systems Design
by Fakhreddine Karray and Clarence de Silva

References

A Genetic Algorithm Tutorial

Darrell Whitley

Computer Science Department