Computational Intelligence

Unit # 5

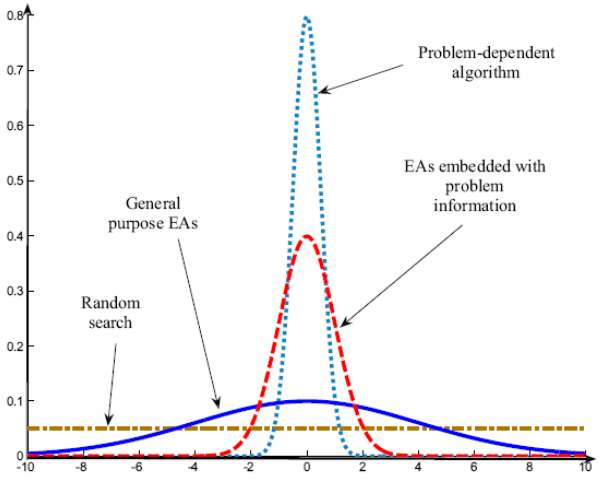
No Free Lunch Theorem

- Wolpert and Macready published a paper with a very strong title: “No Free Lunch Theorems for Optimization”. The key contents of the paper can be quoted as follows:
  - For both static and time dependent optimization problems, the average performance of any pair of algorithms across all possible problems is identical.

NFL Theorem (Cont'd)

- The more we understand the problem, the more specific technique we could design for solving it, and the better performance it will have, but the less robust it will be for other problems.
- We need to demonstrate that our algorithms are better than random search on the problem we face.
- General purpose EAs are reliable methods when you are doing a blind or near blind search in most cases.
- If problem information could be embedded into the encoding and decoding process and into operators, together with a problem-dependent local search method, the performance of the algorithm would be improved at the expense of lower adaptability for other problems.

One Possible Interpretation of NFL Theorem



NFL Theorem (Cont'd)

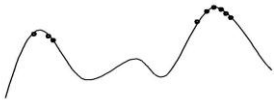
- To sum up, the No Free Lunch theorem is the sword of Damocles when you want to prove that your algorithm is an ideal one.

Typical behaviour of an EA

Phases in optimising on a 1-dimensional fitness landscape

Early phase:
quasi-random population distribution

Mid-phase:
population arranged around/on hills



Late phase:
population concentrated on high hills

Representation Type

- Binary
- Integer
  - Cardinal
- Floating (Real) Number
- Permutation

Mutation for Binary Representations

- The most common mutation operator used for binary encodings considers each gene separately and allows each bit to flip (i.e., from 0 to 1 and from 1 to 0) with a small probability.
- The actual number of values changed is thus not fixed, but depends on the sequence of random number drawn.

Mutation for Integer Representations

- Random Resetting
  - The "bit-flipping" mutation of binary encodings is extended to "random resetting", so that with probability p_m a new value is chosen at random from the set of permissible values in each position.
- Creep Mutation
  - This scheme was designed for ordinal attributes and works by adding a small (positive or negative) value to each gene with probability p.
  - Usually these values are sampled randomly for each position, from a distribution that is symmetric about zero, and is more likely to generate small changes that large ones.

Mutation for Floating-Point Representations

- It is common to change the allele value of each gene randomly with its domain by a lower Li and upper Ui bound, resulting in the following transformation:
- <x1, x2, ….., xn> → <x1`, x2`,….. xn`>
- Two types can be distinguished according to the probability distribution from which the new gene values are drawn: unifrom and nonuniform mutation.

Uniform Mutation

- For this operator, the values of xi` are drawn uniformly randomly from [Li, Ui].
- This is the most straightforward option, analogous to bit-flipping for binary encodings and the random resetting sketched above for integer encodings.

Nonuniform Mutation with a Fixed Distribution

- This option takes a form analogous to the creep mutation for integers.
- It is designed so that usually, but not always, the amount of change introduced is small.
- This is achieved by adding to the current gene value an amount drawn randomly, and then curtailing the resulting value to the range [Li, Ui] if necessary.

Mutation Operators for Permutation Representation

- Swap Mutation
  - This operator works by randomly picking two positions (genes) and swapping their allele values.
- Insert Mutation
  - This operators works by picking two alleles at random and moving one so that it is next to the other, shuffling along the others to make room.
- Scramble Mutation
  - Here the entire string, or some randomly chosen subset of values within it, have their positions scrambled.
- Inversion Mutation
  - Inverse mutation works by randomly selecting two positions in the string and reversing the order in which the values appear between those positions.

Recombination Operators for Binary Representations

- One-Point Crossover
  - It works by chosing a random number r in the range [1, l-1] (with l the length of the encoding), and then splitting both parents at this point and creating the two children by exchanging tails.
- N-Point Crossover
  - One-point crossover can easily be generlized to n-point crossover, where the representation is broken into more than two segments of contiguous genes, and then the offspring are created by taking alternative segments from the two parents.

Recombination Operators for Binary Representations (Cont'd)

- Uniform Crossover
  - The previous two operators worked by dividing the parents into a number of sections of contiguous genes and reassembling them to produce offspring. In constrast to this, uniform crossover works by treating each gene independently and making a random choice as to which parent it should be inherited from.

Recombination Operators for Integer Representations

- For representations where each gene has a higher number of possible allele values (such as integers) it is normal to use the same set of operators as for binary representations.

Recombination Operators for Floating-Point Representations

- Simple Recombination
  - First pick a recombination point k. Then, for child 1, take the first k floats of parent 1 and put them into the child. The rest is the arithmetic average of parent 1 and 2.
  - Child 2 is analogous, with the process reversed.
- Single Arithmetic Recombination
  - Pick a random allele k. At that position, take the arithmetic average of the two parents. The other points are the points from the parents.

Recombination Operators for Floating-Point Representations (Cont'd)

- Whole Arithmetic Recombination
  - This is the most commonly used operator and works by taking the weighted sum of the two parental alleles for each gene.
  - If weight = 0.5 then the two offspring will be identical for this operator.

Probability Distributions for Mutation

- All EAs follows a stochastic search process.
- Stochasticity is introduced by computing step sizes as a function of noise, η_{ij} , *sampled from some* probability distribution. The most popular distributions are
  - Uniform
  - Gaussian

Uniform

- Noise is sampled from a uniform distribution

$$\eta_{ij}(t) \sim U(x_{min}, x_{max})$$

- where x_{min} *and* x_{max} *provide lower and upper bounds for the values of* η_{ij}.

Gaussian

- For the Gaussian mutation operators, noise is sampled from a zero-mean, normal distribution.
- For completeness sake, and comparison with other distributions, the Gaussian density function is given as (assuming a zero mean)

$$f_G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/(2\sigma^2)}$$

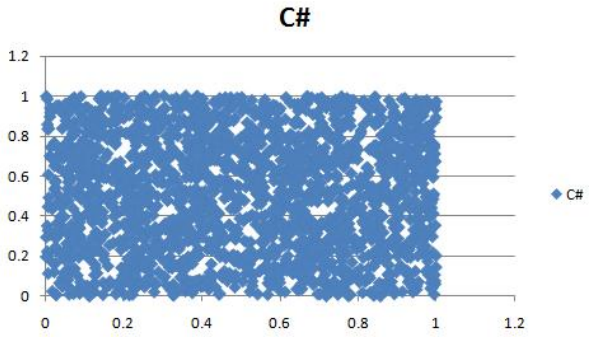
- where σ is the deviation of the distribution

C# Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         StreamWriter strOut = new StreamWriter("output.txt");

         Random r1 = new Random(1111);
          Random r2 = new Random(1729);

         for (int i = 0; i < 2000; i++)
            strOut.WriteLine(r1.Next(10000)/10000.0 + " , " + r2.Next(10000) / 10000.0);

         strOut.Flush();
         strOut.Close();
      }
   }
}
```

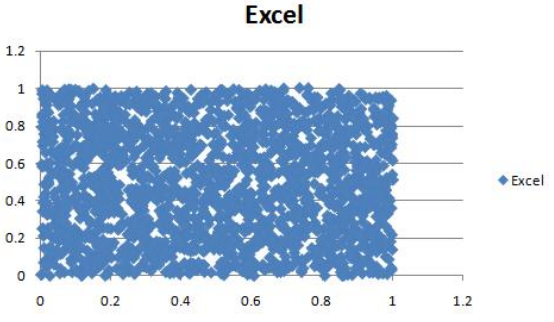
C# Random Numbers

	Random 1 Series	Random 2 Series
Q1	0.253525	0.267425
Median	0.5056	0.5123
Q3	0.749675	0.751325



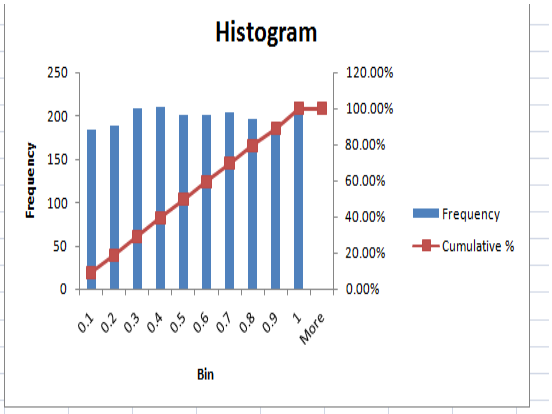
Excel Random Numbers

	Random 1 Series	Random 2 Series
Q1	0.234332	0.24797
Median	0.495124	0.486396
Q3	0.748964	0.743278



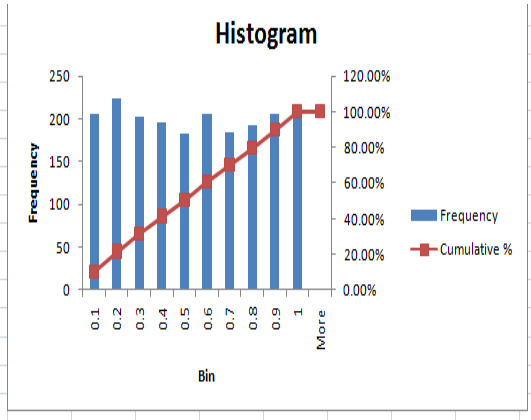
C# Random Numbers Histogram

Bin	Frequency	Cumulative %
0.1	184	9.20%
0.2	188	18.60%
0.3	208	29.00%
0.4	210	39.50%
0.5	201	49.55%
0.6	201	59.60%
0.7	204	69.80%
0.8	196	79.60%
0.9	192	89.20%
1	216	100.00%
More	0	100.00%



Excel Random Numbers Histogram

Bin	Frequency	Cumulative %
0.1	205	10.25%
0.2	224	21.45%
0.3	202	31.55%
0.4	195	41.30%
0.5	182	50.40%
0.6	205	60.65%
0.7	185	69.90%
0.8	192	79.50%
0.9	206	89.80%
1	204	100.00%
More	0	100.00%

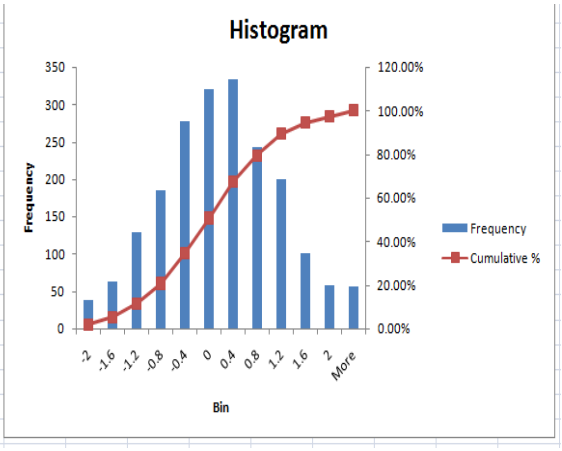


Normal (Gaussian) Random Numbers

- The most important transformation functions is known as the **Box-Muller** (1958) transformation.
- It allows us to transform uniformly distributed random variables, to a new set of random variables with a Gaussian (or Normal) distribution.
- The most basic form of the transformation looks like:
  - y1 = sqrt(- 2 ln(x1)) cos(2 pi x2)
  - y2 = sqrt(- 2 ln(x1)) sin(2 pi x2)
- We start with *two* independent random numbers, x1 and x2, which come from a uniform distribution (in the range from 0 to 1).
- Then apply the above transformations to get two new independent random numbers which have a Gaussian distribution with zero mean and a standard deviation of one.

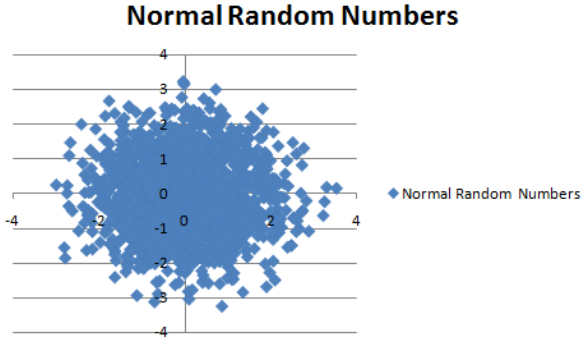
Normal Distribution

Bin	Frequency	Cumulative %
-2	38	1.90%
-1.6	63	5.05%
-1.2	128	11.45%
-0.8	185	20.70%
-0.4	277	34.55%
0	321	50.60%
0.4	333	67.25%
0.8	242	79.35%
1.2	199	89.30%
1.6	101	94.35%
2	57	97.20%
More	56	100.00%



Normal Random Numbers

	Random 1 Series	Random 2 Series
Q1	-0.66701	-0.69655
Median	-0.00948	0.022777
Q3	0.624935	0.69396



A Canonical EA (Source: DeJong)

- This canonical EA
  - maintains a fixed-size population of individuals,
  - each of which is a fixed-length vector (chromosome) of real-valued parameters (genes), and
  - whose objective fitness is determined by calling a "landscape" evaluation function.
- After randomly generating and evaluating the fitness of the members of the initial population, EA
  - effects population evolution by repeatedly selecting a member of the current population at random (uniformly) as a parent and
  - produces a clone offspring.

A Canonical EA (Cont'd)

- EA then introduces some variation in the offspring via a simple mutation operator which, on average picks one gene in the genome to modify by randomly adding/subtracting a small value to the current value of the selected gene.
- The new offspring is forced to immediately compete for survival against an existing member of the population selected at random.
- If the objective fitness of the child is greater than the selected member, the child survives and the old member dies off. Otherwise, the child dies without ever residing in the population.

Steady-State Model of EA

- This form of population management in which offspring are produced one at a time and immediately compete for survival is called an incremental or "steady state" model.
- An alternate model is one in which a large number (a batch) of offspring are produced from a fixed pool of parents. Only after the entire batch is produced there is competition for survival into the next generation. Such models are often referred to as "batch" or "generational" algorithms.

Limitations

- It is quite possible that some members of the current population never produce any offspring in spite of the fact that they may have high fitness.
- Similarly, by randomly picking members from the current population to compete with children for survival into the next generation, it is quite possible that weak individuals may survive by the luck of the draw and not out of merit.

History of Evolutionary Algorithms

- Several efforts were started in parallel during 1960s
  - Evolution Strategies (Berlin Technical University)
  - Genetic Algorithms (University of Michigan)
  - Evolutionary Programming (UCLA)
- During 1990s the above communities agreed to the term "Evolutionary Computation".

Evolution Strategies

- At the Technical University Berlin, Rechenberg and Schwefel (1965 paper) began formulating ideas about how evolutionary processes could be used to solve difficult real-valued parameter optimization problems.
- From these early ideas emerged a family of algorithms called "*evolution strategies*" which today represent some of the most powerful evolutionary algorithms for function optimization.

Evolutionary Programming

- At UCLA during the same period Fogel (1966 paper) saw the potential of achieving the goals of artificial intelligence via evolutionary techniques.
- These ideas were initially explored in a context in which intelligent agents were represented as finite state machines, and an evolutionary framework called "*evolutionary programming*" was developed which was quite effective in evolving better finite state machines (agents) over time.

Blondie24

- **Blondie24** is an artificial intelligence checkers-playing computer program.
- The screen name was used on the The Zone, an internet boardgaming site, during 1999. During this time, Blondie24 played against some 165 human opponents and was shown to achieve a rating of 2048, or better than 99.61% of the playing population of that web site.
- The design of Blondie24 is based on a minimax algorithm of the checkers game tree in which the evaluation function is an artificial neural network.
- The neural net receives as input a vector representation of the checkerboard positions and returns a single value which is passed on to the minimax algorithm.

Blondie24 (Cont'd)

- The neural net weightings were obtained by an evolutionary algorithm, in this case by having a population of Blondie24-like programs play against each other and later eliminating those with fewest earned points in which the players earned +1 for a win, 0 for a draw, and -2 for a loss, and repeating the process with a new population derived from the winners. The result was an evolutionary process selecting the programs that played better checkers games.

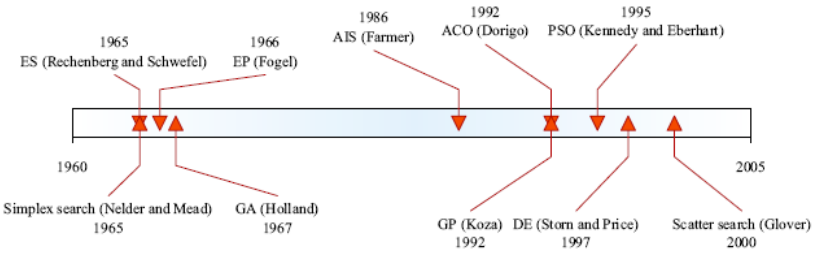
Genetic Algorithms

- At the University of Michigan, Holland (1962 paper) saw evolutionary processes as a key element in the design and implementation of robust adaptive systems, capable of dealing with an uncertain and changing environment.
- His view emphasized the need for systems which self-adapt over time as a function of feedback obtained from interacting with the environment in which they operate.
- This lead to an initial family of "reproductive plans" which formed the basis for what we call "*simple genetic algorithms*" today.

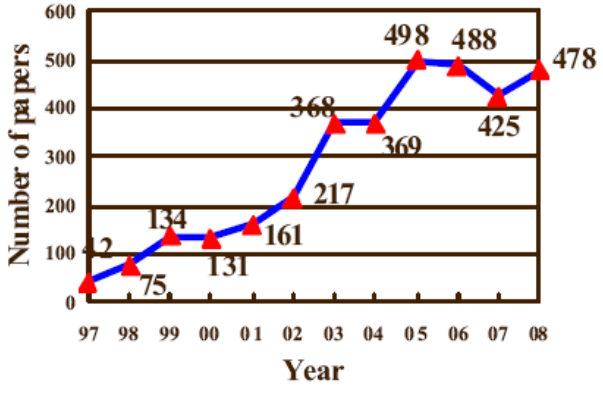
The Unifying 90s

- Till 90s, much of the R&D was done independently and in parallel without much interaction among the various groups.
- The emergence of the various EA conference in the late 1980s and early 1990s, however, changed all that as representative from various EA groups met, presented their particular viewpoints, challenged other approaches, and were challenged in return.
- The immediate result was an agreement on the term "evolutionary computation" as the name of the field and a commitment to start the field's first journal, Evolutionary Computation.

Recap: History of Evolutionary Computation



Number of EC Papers Published in ISI Indexed Journals



Canonical Evolutionary Programming

- For I = 1 to m Do:
  - Use parent I to produce 1 child and add it to the offspring population
- End Do
- From the combined pool of 2m parents and children, select only the m individuals with highest fitness to survive.

Canonical EP (Cont'd)

- Producing a very small number of offspring from a parent is not a very reliable sample of its potential for producing useful progeny.
- Computationally, it is quite easy to extend the EP paradigm to encompass this idea by just allowing the size of the offspring population n to be greater than m, the size of the parent population.

Canonical Evolution Strategy

- Having done so raises two interesting questions:
  - How many offspring should each parent produce, and given that we are now using parents more efficiently,
  - Can we reduce the size of the parent population
- These ideas correspond directly to the early Evolutionary Strategy algorithms.
- Perhaps most striking about this early work was the focus on $(1 + \lambda)$-ES models in which the entire next generation was produced from a single parent.

EP Differences

- EP emphasizes phenotypic evolution, instead of genotypic evolution.
- Due to the above, EP does not make use of any recombination operator. There is no exchange of genetic material.
- Selection is based on competition. Those individuals that perform best against a group of competitors have a higher probability of being included in the next generation.
- Parents and offspring compete for survival.
- The behavior of individuals is influenced by strategy parameters, which determine the amount of variation between parents and offspring.

EP - Mutation

- As mutation is the only means of introducing variation in an EP population, it is very important that the design of a mutation operator considers the exploration–exploitation trade-off.
- The variation process should facilitate exploration in the early stages of the search to ensure that as much of the search space is covered as possible.
- After an initial exploration phase, individuals should be allowed to exploit obtained information about the search space to fine tune solutions.

EP – Mutation (Cont'd)

- In general, mutation is defined as

$$x'_i(t) = x_i(t) + \Delta x_i(t)$$

- where $x'_i(t)$ is the offspring created from parent $x_i(t)$ by adding a step size $\Delta x_i(t)$ to the parent.
- The step size is noise sampled from some probability distribution, where the deviation of the noise is determined by a strategy parameter, σ_i .
- Generally, the step size is calculated as

$$\Delta x_i(t) = \Phi(\sigma_i(t))\eta_i(t)$$

- where $\Phi : R \rightarrow R$ is a function that scales the contribution of the noise, $\eta_i(t)$.

EP – Scaling Functions

- Based on the characteristics of the scaling function, Φ, EP algorithms can be grouped into three main categories of algorithms:
- ***non-adaptive EP,*** in which case Φ(σ) = σ. In other words, the deviations in step sizes remain static.
- ***dynamic EP,*** where the deviations in step sizes change over time using some deterministic function, Φ.
- ***self-adaptive EP,*** in which case deviations in step sizes change dynamically.

ES - Recombination

- The basic recombination scheme in evolution strategies involves two parents that create one child.
- To obtain λ offspring recombination is performed λ times.
- There are two recombination variants distinguished by the manner of recombining parent alleles.
  - Using discrete recombination one of the parent alleles is randomly chosen with equal chance for either parents.
  - In intermediate recombination the values of the parent alleles are averaged.

ES – Step-Size Adjustment

- Theoretical studies motivated an on-line adjustment of step sizes by the famous 1/5 success rule of Rechenberg.
- This rule states that the ratio of successful mutations to all mutations should be 1/5.
- If the ratio is greater then 1/5 the step size should be increased to make a wider search of the space, and if the ratio is less than 1/5 then it should be decreased to concentrate the search more around the current solution.

ES - Survival Selection

- After creating λ offspring and calculating their fitness, the best μ of them are chosen deterministically, either from the offspring only, called (μ, λ) selection, or from the union of the parents and offspring, called $(\mu + \lambda)$ selection.
- Both schemes are strictly deterministic and are based on rank rather than an absolute fitness value.

ES - Survival Selection (Cont'd)

- (μ, λ) is typically preferred over $(\mu + \lambda)$ for the following reasons
  - The (μ, λ) discards all parents and is there in principle able to leave (small) local optima, so it is advantageous in the case of multimodal topologies
  - If the fitness function is not fixed, but changes in time, the $(\mu + \lambda)$ selection preserves outdated solutions, so it is not able to follow the moving optimum well.
  - $(\mu + \lambda)$ selection hinders the self-adaptation mechanism because mis-adapted parameter may survive for a relatively large number of generations

Summary EP

Representation	Real-valued vectors
Parent Selection	Deterministic (each parent creates one offspring via mutation)
Recombination	None
Mutation	Gaussian perturbation
Survival Selection	Probabilistic (mu + mu)

Summary of ES

Representation	Real-valued vectors
Parent Selection	Uniform random
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Survival Selection	(mu, lambda) or (mu + lambda)

Genetic Algorithms

- In the previous schemes, individuals die off only when replaced by younger individuals with higher fitness.
- This results in a significant loss of diversity in the population and can increase the likelihood of becoming trapped on a false peak.
- One way to handle this problem is to allow new individuals to replace existing individuals with higher fitness.
- A more direct method is to use generational model in which parent survive for exactly one generation and are completely replaced by their offspring.
- This is the form that standard GA take and also the form that (μ , λ) – ES model take.

Genetic Algorithms (Cont'd)

- GA also implement the biological notion of fitness in a somewhat different fashion than we have seen so far (fitness proportional scheme).
- The EP and ES systems all used objective fitness to determine which offspring survive to adulthood.
- However, once in the parent pool, there is no additional bias with respect to which individuals get to reproduce – all parents have an equal chance.

Genetic Algorithms (Cont'd)

- Another important difference between GAs and the other models is the way in which offspring are produced.
- The basic idea is that offspring inherit gene values from more than one parent.
- This mixing of parental gene values along with an occasional mutation provides the potential for a much more aggressive exploration of the space.

Genetic Algorithms (Cont'd)

- Crossover provides an additional source of variation involving larger initial steps, improving the initial rate of convergence.
- Since crossover does not introduce new gene values, its influence diminishes as the population becomes more homogenous, and the behavior of GA with crossover becomes nearly identical to a GA with no crossover.

Universal Genetic Code

- Holland emphasized the importance of a universal string-like "genetic" representation to be used internally by a GA to represent the genome of an individual.
- He initially suggested a binary string representation in which each bit internally is viewed as a gene, and the mapping to the external phenotype left unspecified and problem specific.
- Crossover and mutation operate as before at the gene level except at a much finer level of granularity.
- In the case of a binary representation, mutation simplifies to a "bit flipping" operator.

Example (Goldberg)

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

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	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

Summary of GA

Representation	Binary strings (Genotype)
Parent Selection	Fitness Proportional
Recombination	N-Crossover
Mutation	Bit-flip
Survival Selection	Generational