Computational Intelligence

Unit # 6

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.

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.

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.