Weight Initialization for Backpropagation with Genetic Algorithms

Anoop Kunchukuttan
Sandeep Limaye
Ashish Lahane

Seminar Outline

- Weight Initialization Problem (Part – I)
- Introduction to Genetic Algorithms (Part – II)
- GA-NN based solution (Part – III)

Problem Definition

Part - I

Backpropagation

- Feed-forward neural network training method
- Minimizes Mean Square Error
- Gradient descent
- Greedy Method

Weight Initialization Problem

- Backpropagation is sensitive to initial conditions **[KOL-1990]**
- Initial conditions such as

  - initial weights

  - learning factor

  - momentum factor

Effect of Initial Weights

- Can get stuck in local minima
- May converge too slowly
- Achieved generalization can be poor

Solution

- Use of global search methods to get the final weights, such as: Genetic algorithms, Simulated Annealing etc.
- Use global search methods to get closer to global minimum & then run local search (BP) to get exact solution

We will concentrate on the latter method using Genetic algorithms

Genetic Algorithms
A primer

Part - II

Motivation

- Inspired by evolution in living organisms
- “Survival of the fittest”
- “Fitter” individuals in the population preferred to reproduce to create new generation

Other algorithms inspired by nature

- Neural networks – neurons in brain
- Simulated annealing – physical properties of metals
- GA, NN have partly overlapped application areas – pattern recognition, ML, image processing, expert systems

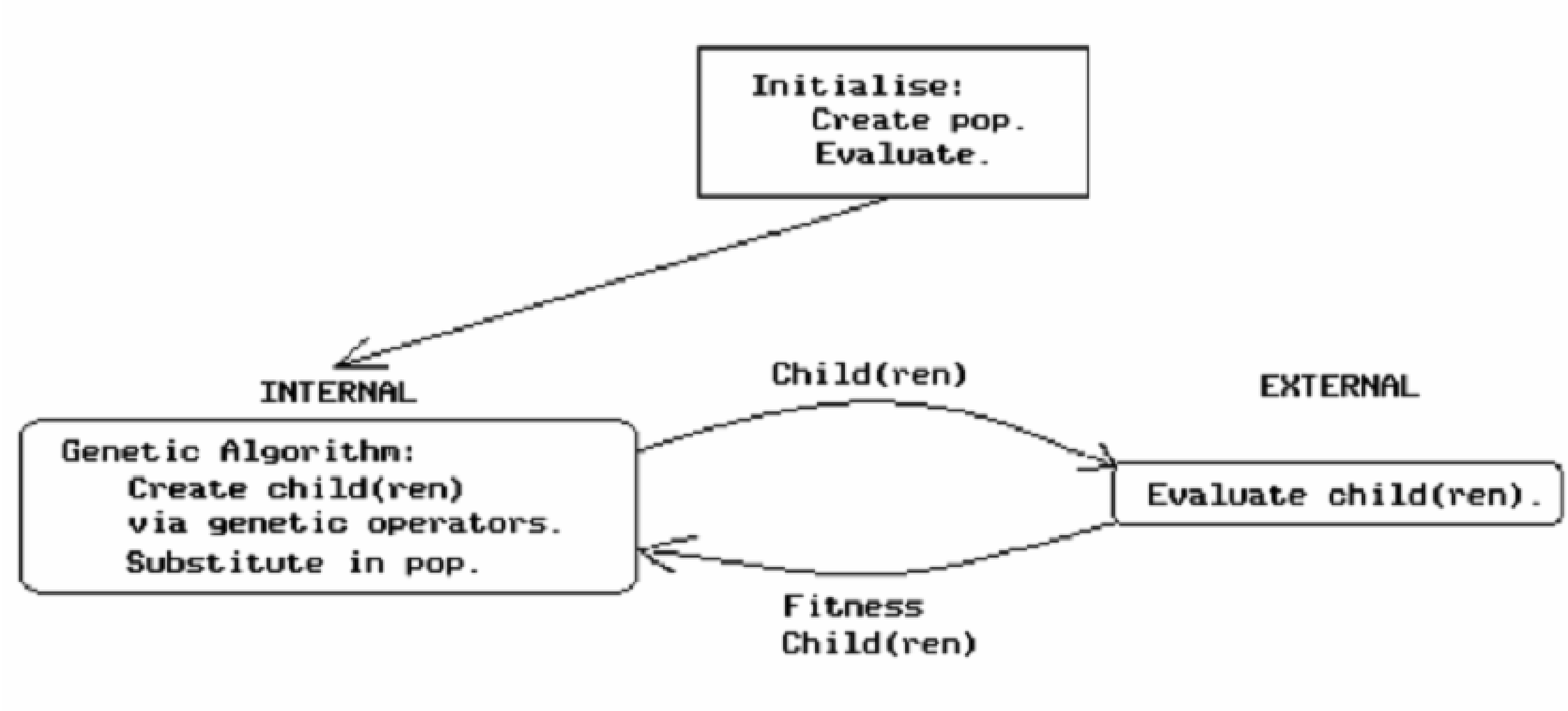
Basic Steps

- Coding the problem
- Determining “fitness”
- Selection of parents for reproduction
- Recombination to produce new generation

Coding

GA in nature	GA in CS
Genes	Parameters of solution
Chromosome = collection of genes	Chromosome = concatenated parameter values
Genotype = set of parameters represented in the chromosome	
Phenotype = Finished “construction” of the individual (values assigned to parameters)	

Generic Model for Genetic Algorithm (1)



Generic Model for Genetic Algorithm (2)

```
BEGIN /* genetic algorithm */
  generate initial population
  WHILE NOT finished DO
  BEGIN
      compute fitness of each individual
      IF population has converged THEN
          finished := TRUE
      ELSE
         /* reproductive cycle */
         apply genetic operators to produce children
      END
  END
END
```

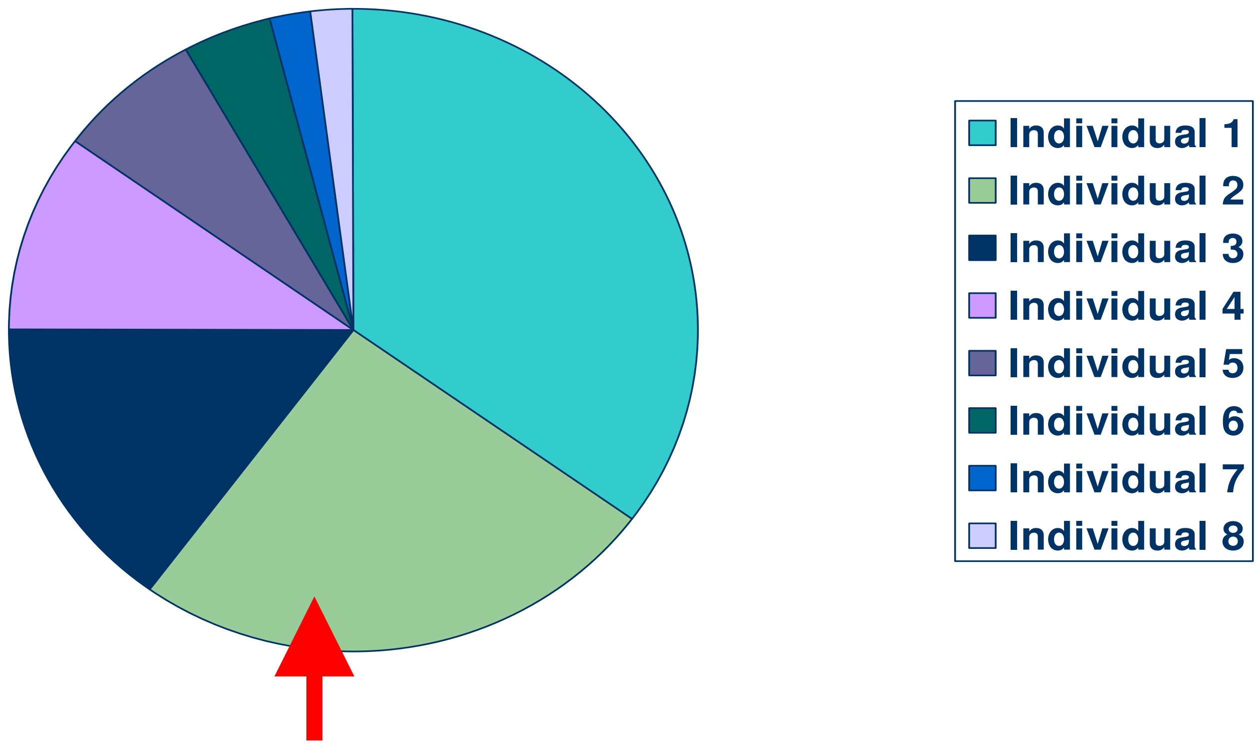
Determining fitness

- Fitness function
- Varies from problem to problem
- Usually returns a value that we want to optimize
- Example: Strength / Weight ratio in a civil engineering problem
- Unimodal / Multimodal fitness functions – Multimodal: several peaks

Selection

- Individuals selected to recombine and produce offspring
- Selection of parents based on “Roulette Wheel” algorithm
- Fitter parents may get selected multiple times, not-so-fit may not get selected at all
- Child can be less fit than parents, but such a child will probably “die out” without reproducing in the next generation.

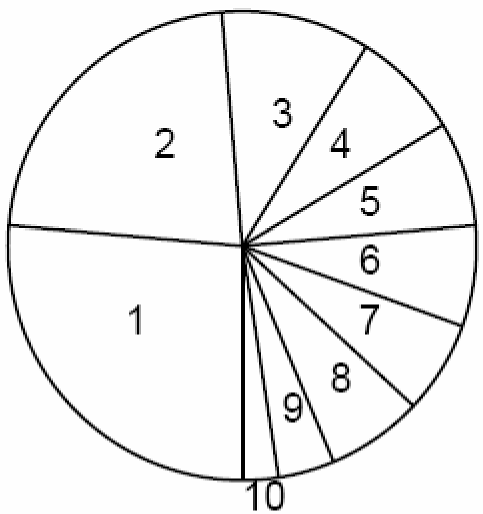
Roulette Wheel selection



Selection - variations

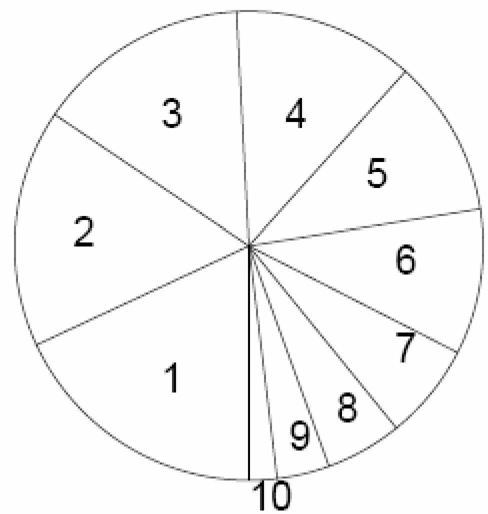
- Fitness-based v/s rank-based

Fitness-based selection

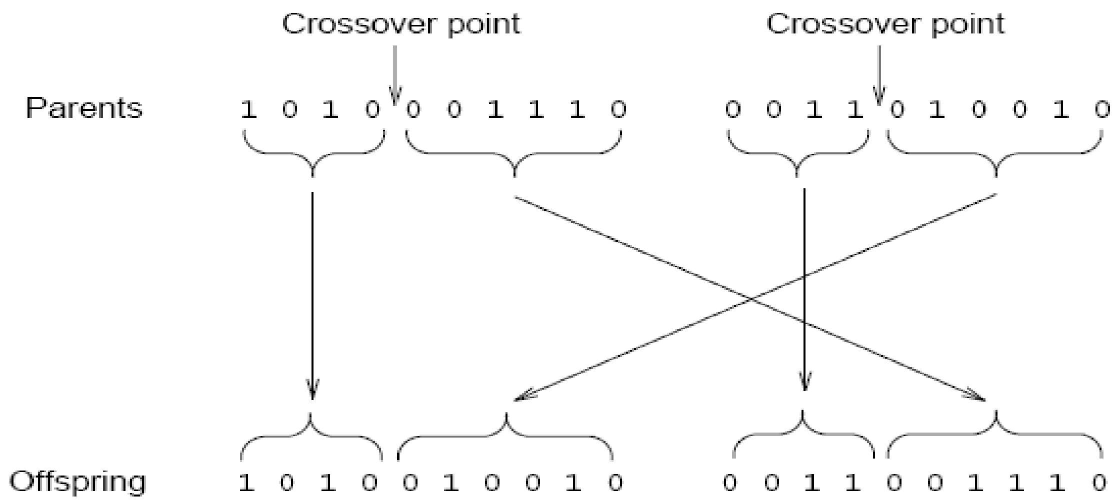


rank	evaluation
#1	5.3
#2	4.4
#3	2.1
#4	1.5
#5	1.4
#6	1.4
#7	1.3
#8	1.3
#9	0.8
#10	0.5

Rank-based selection



Recombination - Crossover



- Crossover probability (typically $0.6 < CP < 1$)
- If no crossover, child is identical to parent

Crossover – variations (1)

- One-point crossover

Parent 1:	001010011	0101001010101110
Parent 2:	010101110	1010101101110101
Child:	001010011	1010101101110101

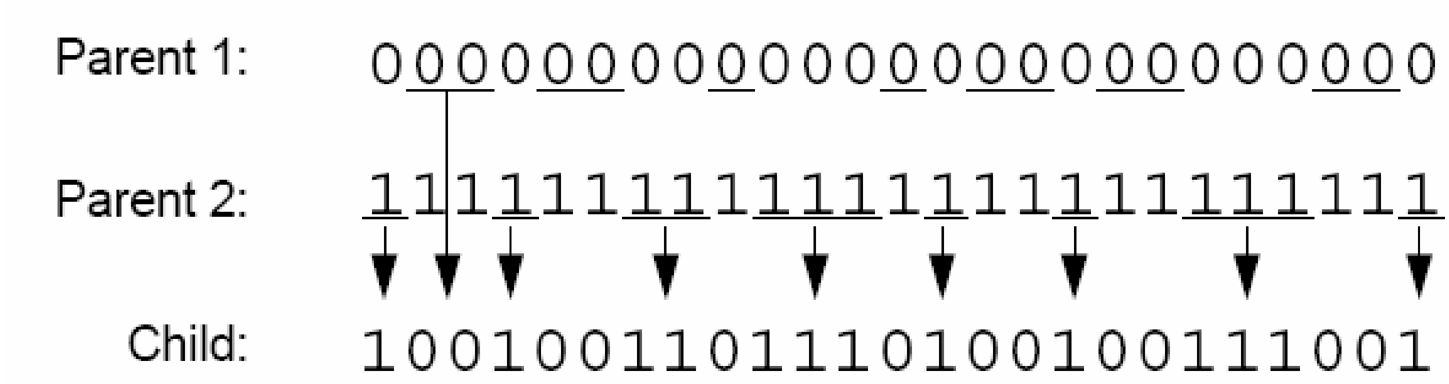
Crossover – variations (2)

- Two-point crossover

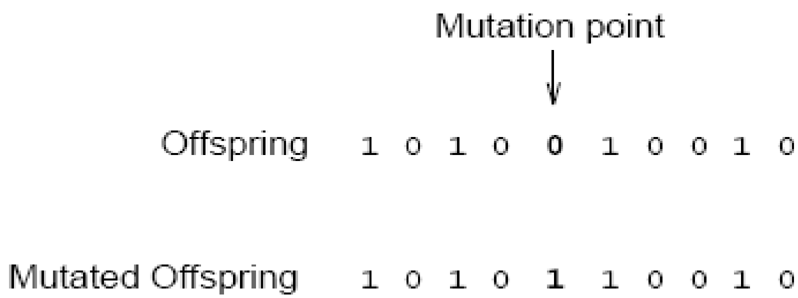
Parent 1:	001010011	01010010	10101110
Parent 2:	010101110	10101011	01110101
Child:	001010011	10101011	01110101

Crossover – variations (3)

- Uniform crossover



Recombination - Mutation



- Crossover – rapidly searches a large problem space
- Mutation – allows more fine-grained search

Convergence

- Gene convergence: when 95% of the population has the same value for that gene
- Population convergence: when all the genes reach convergence

Design Issues

- Goldberg's principles of coding (building block hypothesis):
  - Related genes should be close together
  - Little interaction between genes
- Is it possible, (and if yes, how) to find coding schemes that obey the principles?
- If not, can the GA be modified to improve its performance in these circumstances?

GA - Plus points

- Robust: Wide range of problems can be tackled
- “Acceptably good” solutions in “acceptably quick” time
- Explore wide range of solution space reasonably quickly
- Combination of Exploration and Exploitation
- Hybridizing existing algorithms with GAs often proves beneficial

GA - shortcomings

- Need not always give a globally optimum solution!
- Probabilistic behavior

Weight Initialization using GA

Part - III

Applying GA to Neural Nets

- Optimizing the NN using GA as the optimization algorithm **[MON-1989]**
- Weight initialization
- Learning the network topology
- Learning network parameters

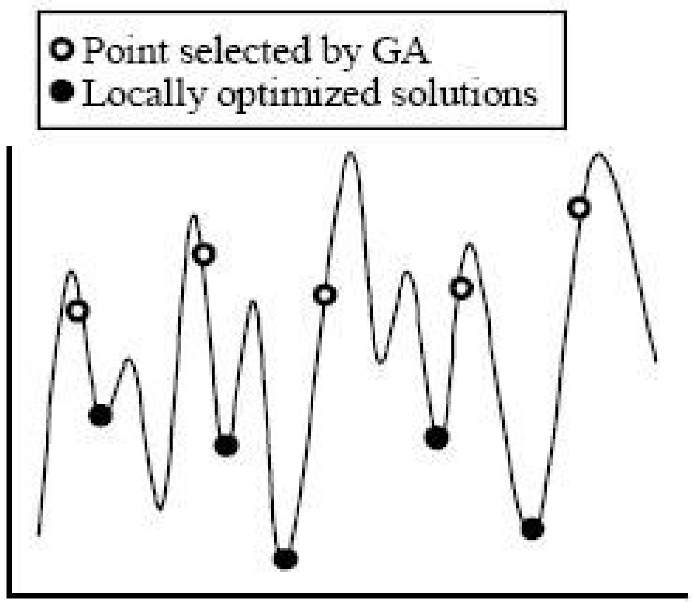
Why use genetic algorithms?

- Global heuristic search (Global Sampling)
- Get close to the optimal solution faster
- Genetic operators create a diversity in the population, due to which a larger solution space can be explored.

The Intuition

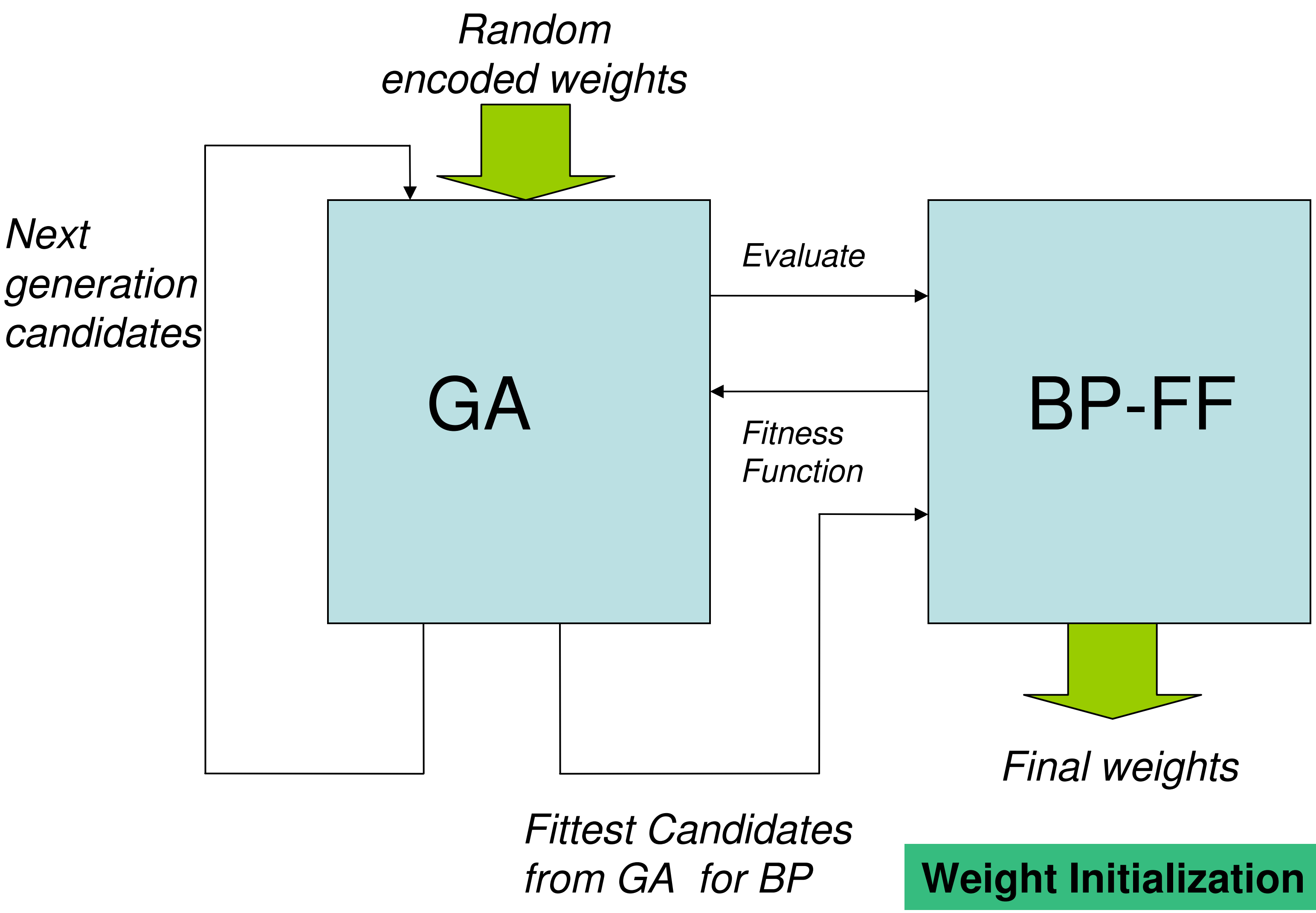
- Optimizing using only GA is not efficient.
  - Might move away after getting close to the solution.
- Gradient Descent can zoom in to a solution in a local neighbourhood.
- Exploit the global search of GA with local search of backpropagation.
- This hybrid approach has been observed to be efficient.

The Hybrid (GA-NN) Approach



GA provides ‘seeds’ for the BP to run.

Basic Architecture



Weight Initialization using GA

Considerations

- Initially high learning rate
  - To reduce required number of BP iterations
- Learning rate and number of BP iterations as a function of fitness criteria:
  - To speed up convergence
  - To exploit local search capability of BP
- Tradeoff between number of GA generations and BP iterations.

Encoding Problem

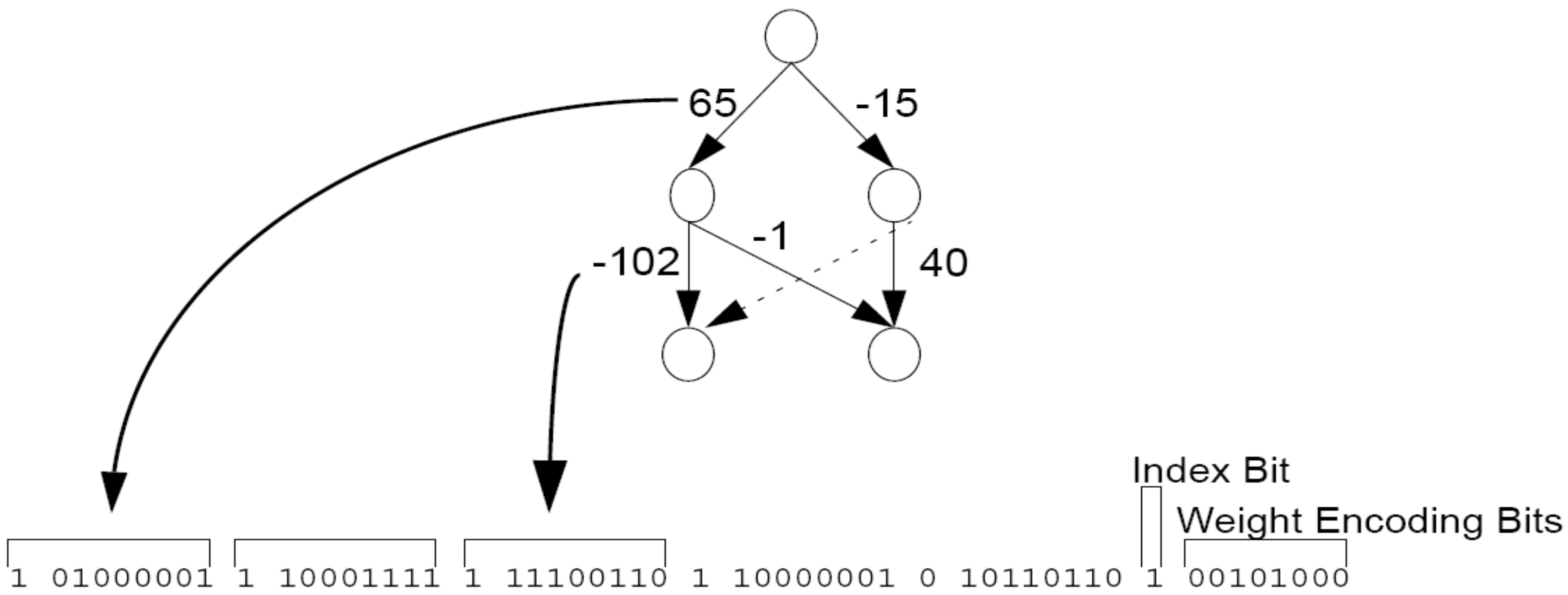
Q. What to encode?

Ans. Weights.

Q. How?

Ans. Binary Encoding

Binary Weight Encoding



Issues

- Order of the weights
- Code length
- Weights are real valued

Code Length '$\mathcal{L}$'

- Code length determines
  - resolution
  - precision
  - size of solution space to be searched

- Minimal precision requirement → minimum '$\mathcal{L}$' → limits the efforts to improve GA search by reducing gene length

Real Value Encoding Methods

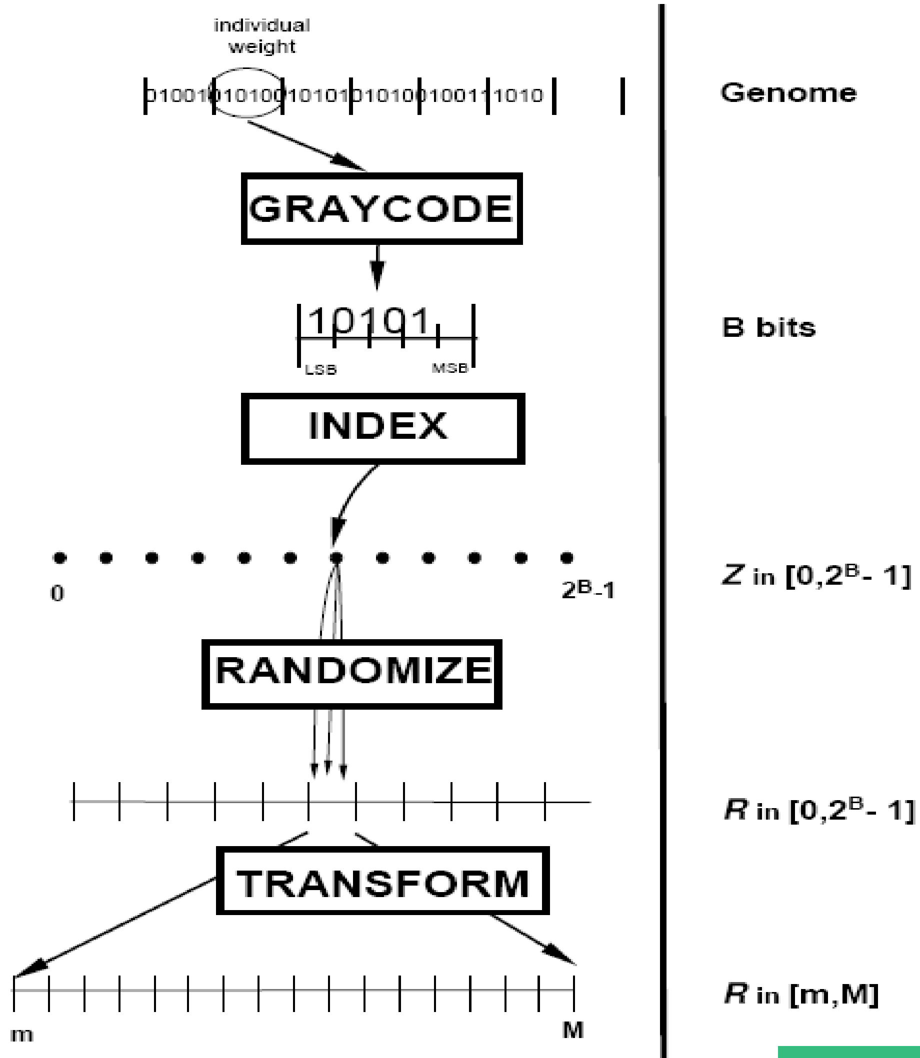
- Gray-Scale
- DPE (Dynamic Parameter Encoding)

Gray-Code Encoding

- Gray-code : Hamming distance between any two consecutive numbers is 1
- Better than or at least same as binary encoding
- Example

Number	Gray Codes	Binary Codes
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

Real-Value Encoding Using Gray-code



Weight Initialization using GA

Randomization + Transformation

$$T(i) = a + (b - a)\frac{i + X}{2^l}$$

[a,b) – the interval
i – parameter value read from chromosome
ℓ – code length
X – random variable with values from [0,1)

Drawbacks

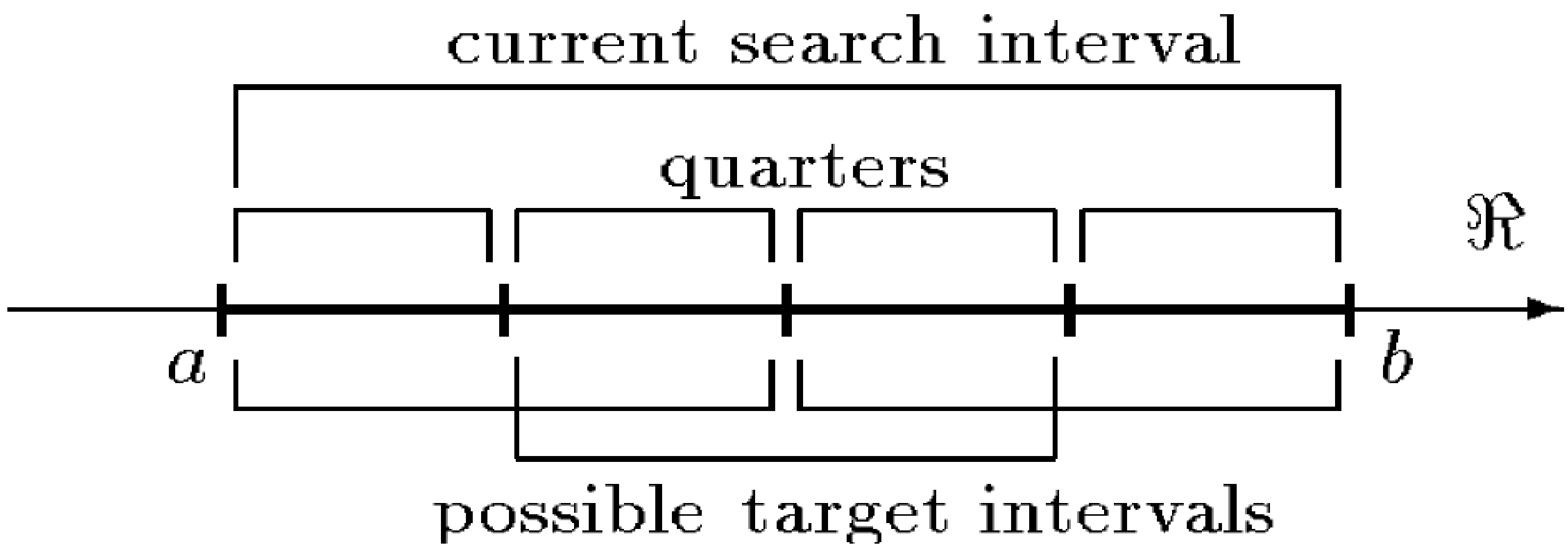
- Too large search space → large 'ℓ' → higher resolution → high precision → long time for GA to converge
- Instead: search can proceed from coarse to finer precision: i.e. DPE

DPE
(Dynamic Parameter Encoding) [SCH-1992]

- use small '$\mathcal{l}$' → coarse precision → search for most favored area → refine mapping → again search → iterate till convergence with desired precision achieved

- Zooming operation

Zooming


current search interval
quarters
$\Re$
a
b
possible target intervals

Fitness Function

- Mean Square Error
- Rate of change of Mean Square Error

Genetic operators – problem-specific variations (1)

- UNBIASED-MUTATE-WEIGHTS
  - With fixed p, replace weight by new randomly chosen value
- BIASED-MUTATE-WEIGHTS
  - With fixed p, add randomly chosen value to existing weight value – biased towards existing weight value – exploitation

Genetic operators – problem-specific variations (2)

- MUTATE-NODES
  - Mutate genes for incoming weights to *n* non-input neurons
  - Intuition: such genes form a logical subgroup, changing them together may result in better evaluation
- MUTATE-WEAKEST-NODES
  - Strength of a node =

    (n/w evaluation) – (n/w evaluation with this node “disabled” i.e. its outgoing weights set to 0)
  - Select *m* weakest nodes and mutate their incoming / outgoing weights
  - Does not improve nodes that are already “doing well”, so should not be used as a single recombination operator

Genetic operators – problem-specific variations (3)

- CROSSOVER-WEIGHTS
  - Similar to uniform crossover
- CROSSOVER-NODES
  - Crossover the incoming weights of a parent node together
  - Maintains “logical subgroups” across generations

Genetic operators – problem-specific variations (4)

- OPERATOR-PROBABILITIES
  - Decides which operator(s) from the operator pool get applied for a particular recombination
  - Initially, all operators have equal probability
  - An adaptation mechanism tunes the probabilities as the generations evolve

Computational Complexity

- Training time could be large due to running multiple generations of GA and multiple runs of BP.
- ***Total Time = Generations*PopulationSize*Training Time***
- There is a tradeoff between the number of generations and number of trials to each individual

CONCLUSION & FUTURE WORK

- Hybrid approach promising
- Correct choice of encoding and recombination operators are important.

- As part of course project
  - Implement the hybrid approach
  - Experiment with different learning rates and number of iterations and adapting them

REFERENCES (1)

- [BE1-1993] David Beasley, David Bull, Ralph Martin: **An Overview of Genetic Algorithms – Part 1, Fundamentals,** University Computing, 1993. *http://ralph.cs.cf.ac.uk/papers/GAs/ga_overview1.pdf*

- [BE2-1993] David Beasley, David Bull, Ralph Martin: **An Overview of Genetic Algorithms** – Part 2, Research Topics**,** University Computing, 1993. *http://www.geocities.com/francorbusetti/gabeasley2.pdf*

REFERENCES (2)

- [BEL-1990] Belew, R., J. McInerney and N. N. Schraudolph (1990). **Evolving networks: Using the genetic algorithm with connectionist learning.** CSE Technical Report CS90-174, University of California, San Diego. *http://citeseer.ist.psu.edu/belew90evolving.html*

- [KOL-1990] Kolen, J.F., & Pollack, J.B. (1990). **Backpropagation is sensitive to the initial conditions.** *http://citeseer.ist.psu.edu/kolen90back.html*

- [MON-1989] D. Montana and L. Davis, **Training Feedforward Neural Networks Using Genetic Algorithms**, *Proceedings of the International Joint Conference on Artificial Intelligence*, 1989. *http://vishnu.bbn.com/papers/ijcai89.pdf*

REFERENCES (3)

- [SCH-1992] Schraudolph, N. N., & Belew, R. K. (1992) **Dynamic parameter encoding for genetic algorithms.** Machine Learning Journal, Volume 9, Number 1, 9-22. *http://citeseer.ist.psu.edu/schraudolph92dynamic.html*

- [WTL-1993] D. Whitley, **A genetic algorithm tutorial,** Tech. Rep. CS-93-103, Department of Computer Science, Colorado State University, Fort Collins, CO 8052, March 1993. *http://citeseer.ist.psu.edu/whitley93genetic.html*

- [ZBY-1992] Zbygniew Michalewicz, **Genetic Algorithms + Data Structures = Evolution Programs**, Springer-Verlag, 1992. *Text Book.*