Q: Discuss the primary data structure for genetic algorithms. How reproduction, crossover and mutation is performed on such data structure? explain with practical example.

Ans:-

GA Introduction:

- Search algorithms based on the mechanics of biological evolution.
- Developed by John Holland, University of Michigan
    - To understand the adaptive process of natural systems
    - To design artificial systems software that retains the robustness of natural systems.
- Can be used for finding solutions of:
    - Searching problem
    - Optimization problem

Data Structures:

1. **Population**
2. **Chromosomes**
3. **Gene**
4. **Allele**
5. **Genotype**
6. **Phenotype**
7. **Encoding & Decoding**
8. **Fitness Function**
9. **Genetic operators**
    a. **Selection**
    b. **Crossover**
    c. **Mutation**

Population:

- Subset of solution set
- Solution set – All possible solution member
- Also known as candidate solution
- Collection of chromosomes

Chromosomes:

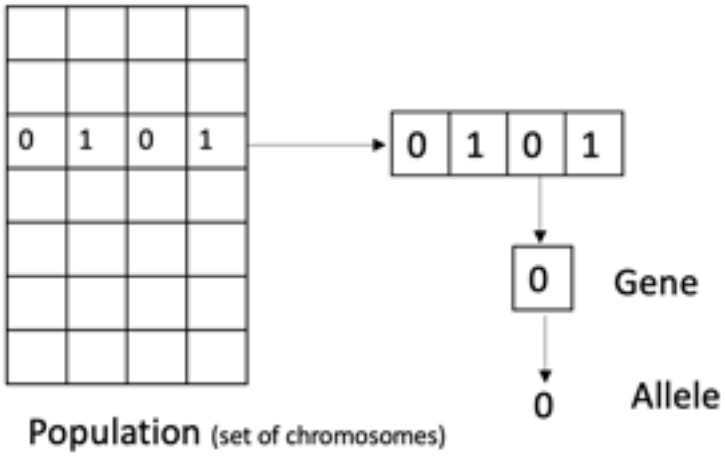
- Individual solution to the given problem
- Typically encoded as a string or an array of genes.

Gene:

- One element position of a chromosome.
- Each gene corresponds to a specific parameter or component of the solution.
- The arrangement of genes within a chromosome forms the candidate solution.

Allele:

- The positional value of gene

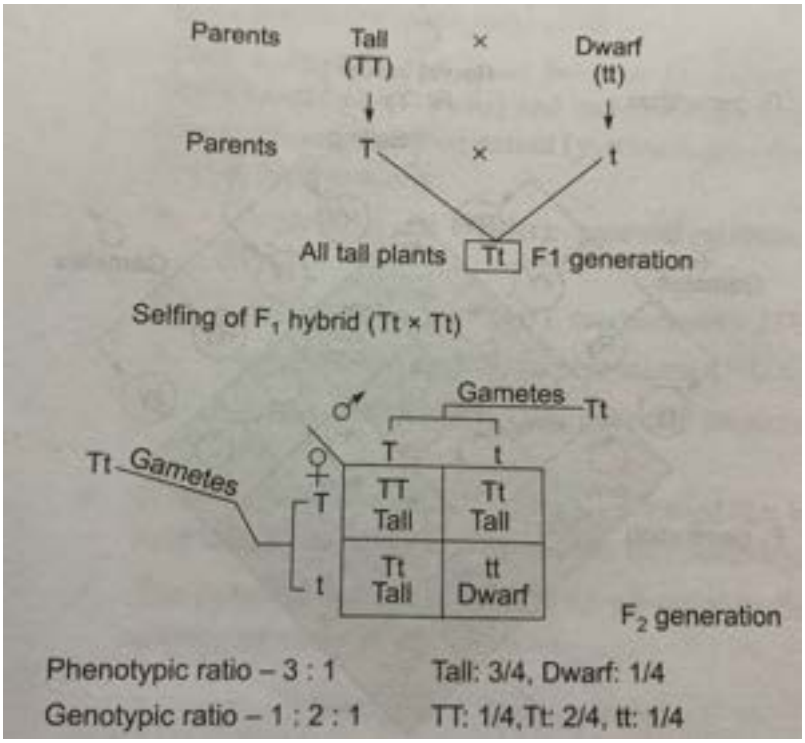


Genotype:

- Mathematical representation format
- Computation space
- Encoding of phenotype solutions

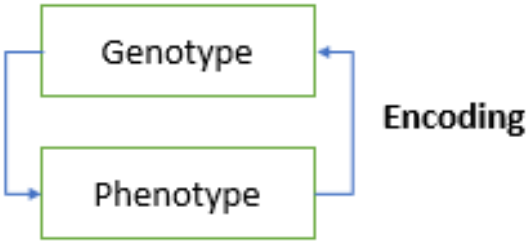
Phenotype:

- How actually it looks like in nature.



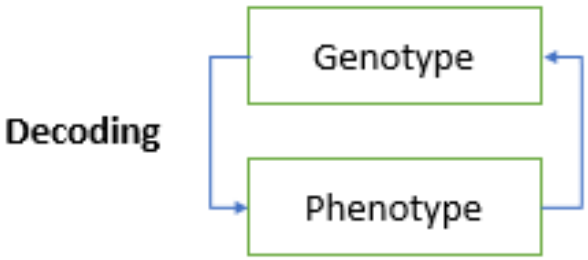
Encoding:

- Conversion of solution set from Phenotype to Genotype.
- It can be: (Genotype Representation)
  - **Binary representation**
    - Boolean 0(absent) or 1(Present) used
  - **Real value representation**
    - Using the actual value of gene (Floating point value)
  - **Integer representation**
    - Integer (whole number) used for representation
  - **Permutation**
    - Positional Significance – e.g. TSP



Decoding:

- Reverse of Encoding
- i.e. Conversion from genotype to phenotype.



Fitness Function:

- Function used to determine how fit a particular chromosome is.
- How fit the chromosome is toward the optimal solution?

Genetic Operators(3 Types):

1. **Reproduction**
2. **Crossover**
3. **Mutation**

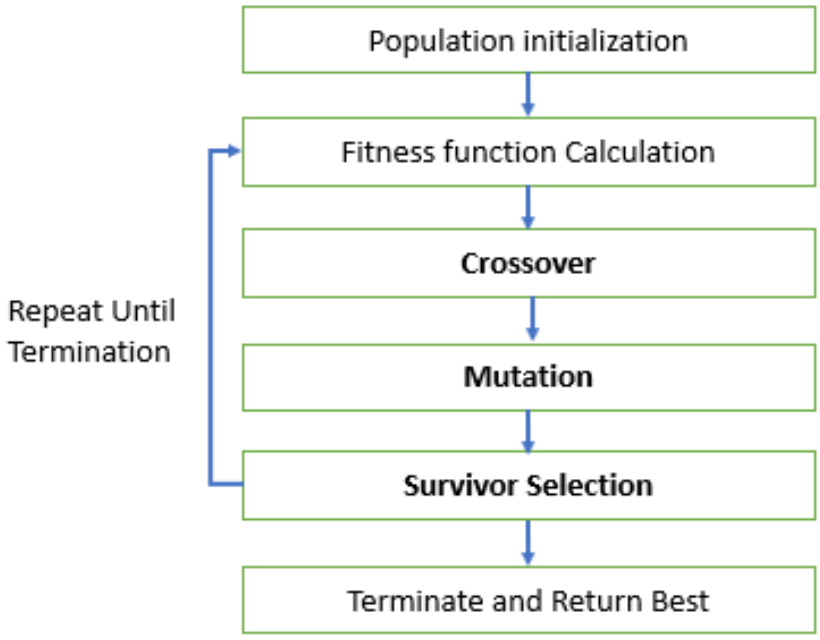
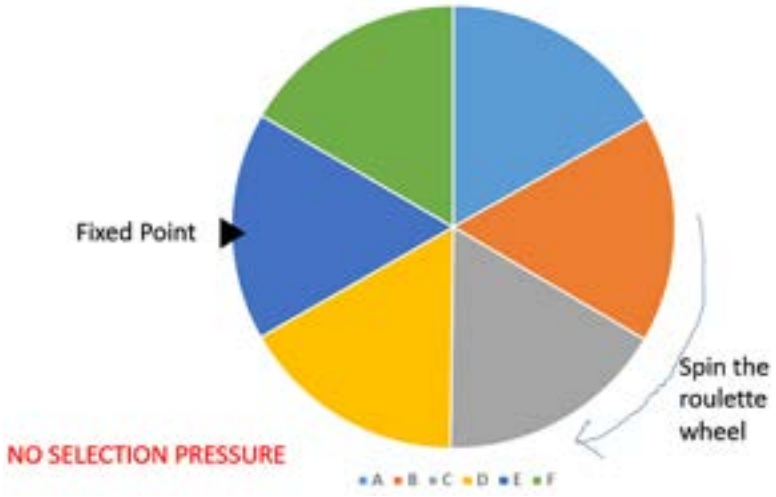


Fig: Genetic Operators in GA

Reproduction:

- Selecting parent chromosomes from the current population to create offspring for the next generation.
- The probability of selection is based on the fitness of each chromosome.
- Fitter chromosomes having a higher chance of being selected.
- Selection of parent for reproduction can be done using any of following:
    - **Roulette Wheel Selection**
    - **Stochastic Universal Sampling (SUS)**
    - **Tournament Selection**
    - **Rank Selection**

Roulette Wheel Selection:



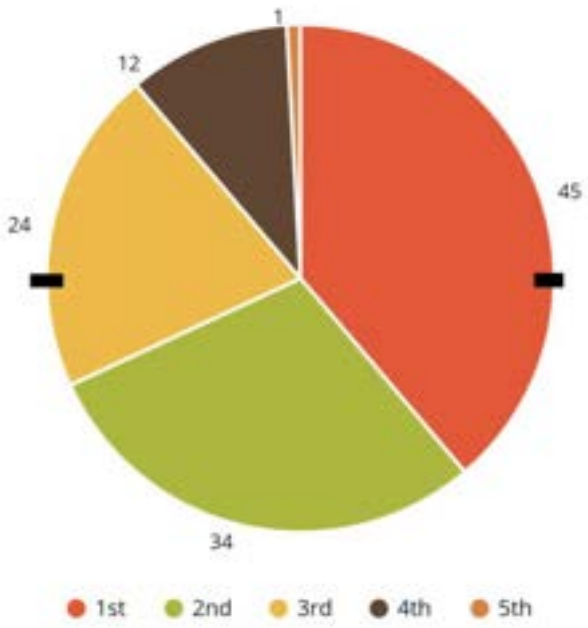
Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

Algorithm for Roulette Wheel:

- Calculate S = the sum of a finesses.
- Generate a random number between 0 and S.
- Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the chosen individual.

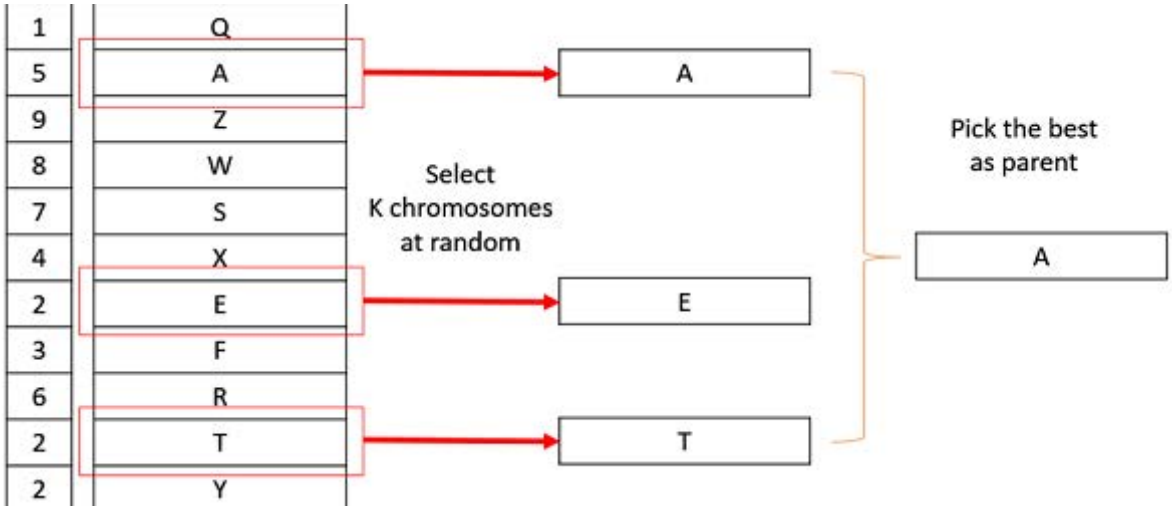
Stochastic Universal Sampling (SUS):

- Same as Roulette wheel selection
- We take two fixed point.



Tournament Selection:

Fitness Val. **Chromosome**

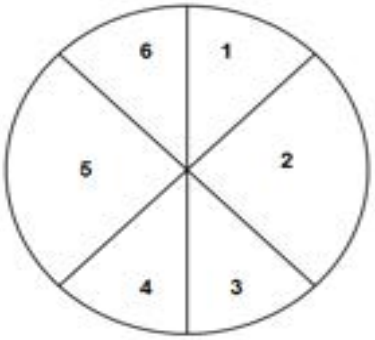


Rank Selection:

- (-ve) fitness can also be selected.
- Parent selected, based on the rank.

Chromosome	Fitness	Rank
1	6	2
2	9	1
3	-2	6
4	0	5
5	4	3
6	1	4

Here, -2 is not selected so we rank to select.



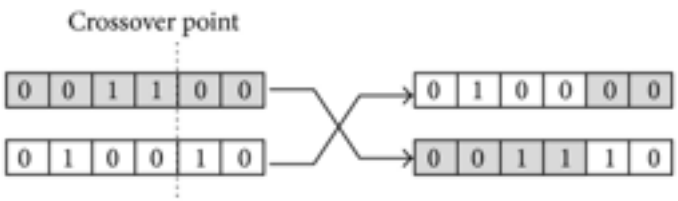
GA Crossover:

- Combining two parent chromosomes to create offspring chromosomes.

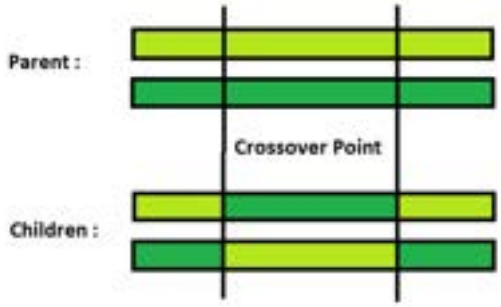
Types:

- **One - point Crossover**
- **Two - point Crossover**
- **Uniform Crossover**
- **Whole Arithmetic Recombination**
- **Davis' Order Crossover (OX1)**
- **Cyclic Crossover (CX)**
- **Partially Matched Crossover (PMX)**

One-Point Crossover:



Two – Point Crossover:

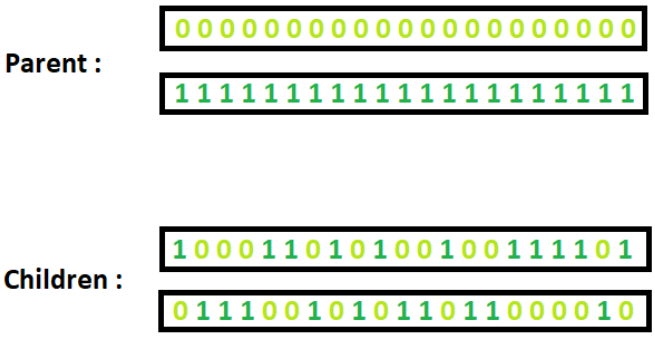


Chromosome1	11011|00100|110110
Chromosome2	10101|11000|011110
Offspring1	11011|11000|110110
Offspring2	10101|00100|011110

Two Point Crossover

Uniform Crossover:

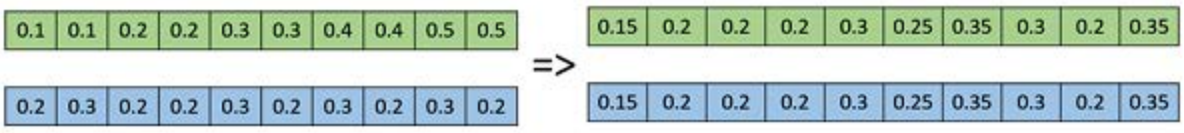
- Each gene (bit) is selected randomly from one of the corresponding genes of the parent chromosomes. Using tossing of a coin as an example technique.



Uniform Crossover

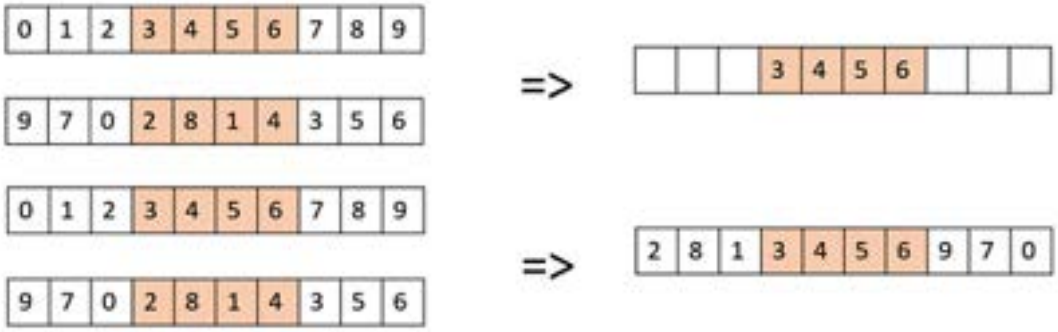
Whole Arithmetic Recombination:

- This is commonly used for integer representations and works by taking the weighted average of the two parents by using the following formulae –
- Child1 = α.x + (1-α).y
- Child2 = α.y + (1-α).x
- Obviously, if α=0.5, then both the children will be identical as shown in the following image.



Davis' Order Crossover (OX1):

- Used for permutation-based crossovers



Repeat the same procedure to get the second child

Cyclic Crossover (CX):

P1:	9	8	2	1	7	4	5	10	6	3
P2:	1	2	3	4	5	6	7	8	9	10

1st Cycle:

Ch1:	9	-	-	1	-	4	-	-	6	-
Ch2:	1	-	-	4	-	6	-	-	9	-

End of Cycle:

Ch1:	9	2	3	1	5	4	7	8	6	10
Ch2:	1	8	2	4	7	6	5	10	9	3

Partially Matched Crossover(PMX):

P1:	9	8	4	5	6	7	1	3	2	10
P2:	8	7	1	2	3	10	9	5	4	6

↓

Ch1:	-	-	-	2	3	10	-	-	-	-
Ch2:	-	-	-	5	6	7	-	-	-	-

↓

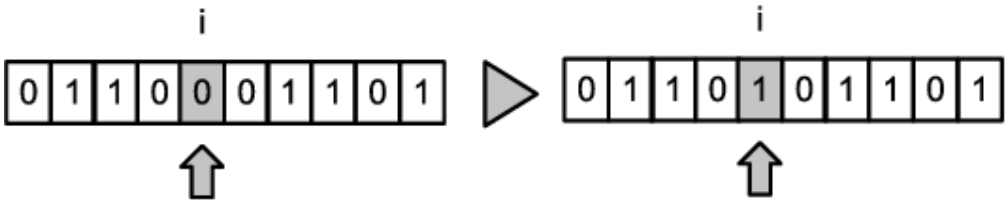
Ch1:	9	8	4	2	3	10	1	_	_	7
Ch2:	8	10	1	5	6	7	9	2	4	3

Mutation:

- Small random tweak in the chromosome to get new solution
- **Types:**
  - **Bit flip mutation**
  - **Random resetting**
  - **Swap mutation**
  - **Scramble mutation**
  - **Inversion mutation**

Bit Flip Mutation:

- **Select one or more bits and flip them.**
- **Used for binary encoded GAs.**

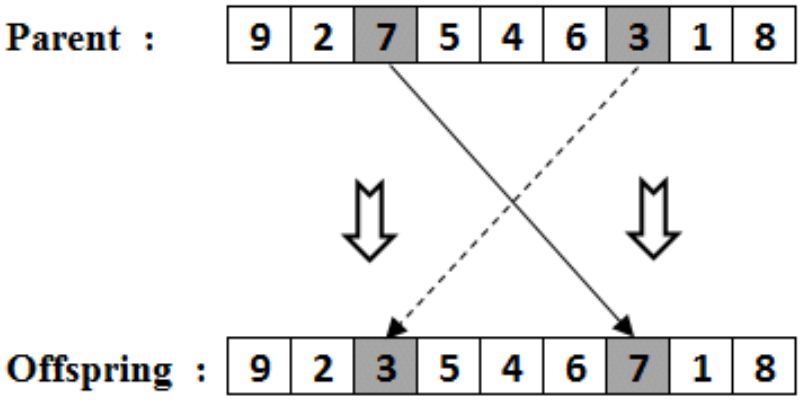


Random resetting:

- Extension of the bit flip for the integer representation.
- A random value from the set of permissible values is assigned to a randomly chosen gene.

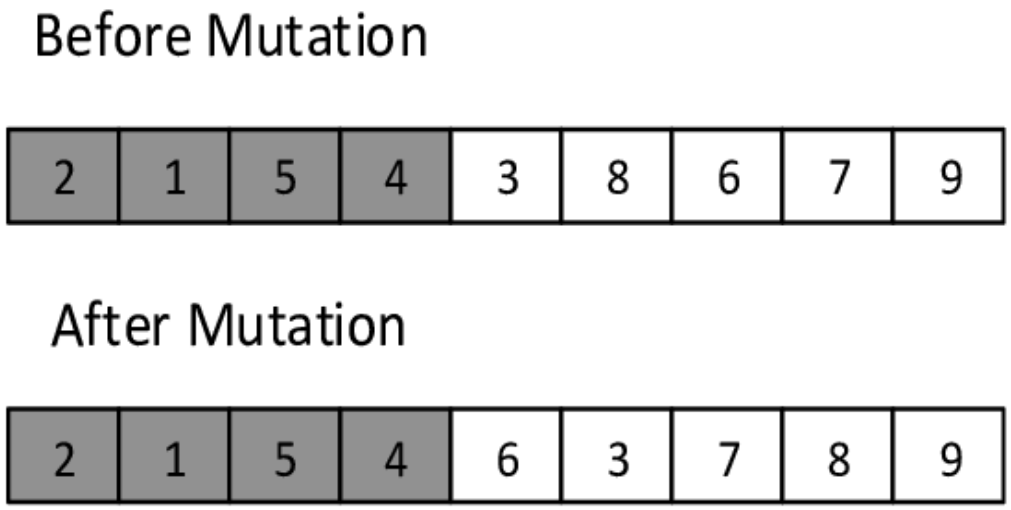
Swap mutation:

- Select two positions on the chromosome at random and interchange the values.



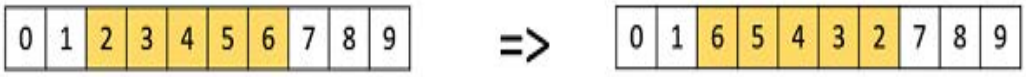
Scramble mutation:

- From the entire chromosome, a subset of genes is chosen, and their values are scrambled or shuffled randomly.
- popular with permutation representations.



Inversion mutation:

- Select a subset of genes like in scramble mutation
- But instead of shuffling the subset, we merely invert the entire string in the subset.



Example:

Given a TSP,

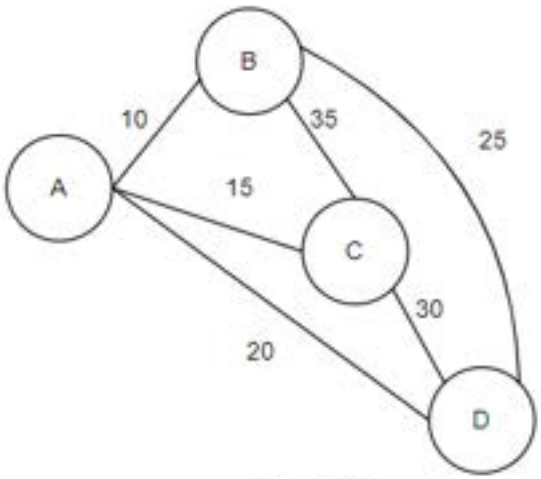


Fig: TSP

Now, **Initial Population (Chromosomes):**

- Individual 1: | A | B | C | D |
- Individual 2: | C | B | A | D |
- Individual 3: | D | A | C | B |

Fitness Calculation (lower distance is better):

Individual 1: Total distance = 10 + 35 + 30 + 25 = 100 units

Individual 2: Total distance = 15 + 35 + 10 + 25 = 85 units

Individual 3: Total distance = 20 + 10 + 30 + 35 = 95 units

Roulette Wheel Selection:

Calculate the selection probabilities based on fitness values.

Randomly select individuals for reproduction. For simplicity, let's choose Individual 2 and Individual 3.

Single Point Crossover (Crossover Point at index 2):

Offspring1: | C | B | A | D |

Offspring2: | D | A | C | B |

Swap Mutation (with a low probability, e.g., 10% chance):

Mutate Offspring1 by swapping two random cities: | B | C | A | D |

Mutate Offspring2 by swapping two random cities: | D | C | A | B |

Again, let's calculate the total distance for the mutated offspring1 and offspring2:

Mutated Offspring1: | B | C | A | D |

- Distance from B to C: 35 units
- Distance from C to A: 15 units
- Distance from A to D: 20 units
- Total distance for Mutated Offspring1: 35 + 15 + 20 = 70 units

Mutated Offspring2: | D | C | A | B |

- Distance from D to C: 30 units
- Distance from C to A: 15 units
- Distance from A to B: 10 units
- Total distance for Mutated Offspring2: 30 + 15 + 10 = 55 units

These mutated offspring, along with the rest of the population, form the new generation. The process repeats for multiple generations, gradually improving the solutions. The algorithm continues until a termination criterion is met, such as a maximum number of generations or a

specific fitness threshold. Over time, the population should converge to a near-optimal solution for the TSP, which is the shortest route to visit all cities.

Genetic Algorithms - Survivor Selection

The Survivor Selection Policy determines which individuals are to be kicked out and which are to be kept in the next generation. It is crucial as it should ensure that the fitter individuals are not kicked out of the population, while at the same time diversity should be maintained in the population.

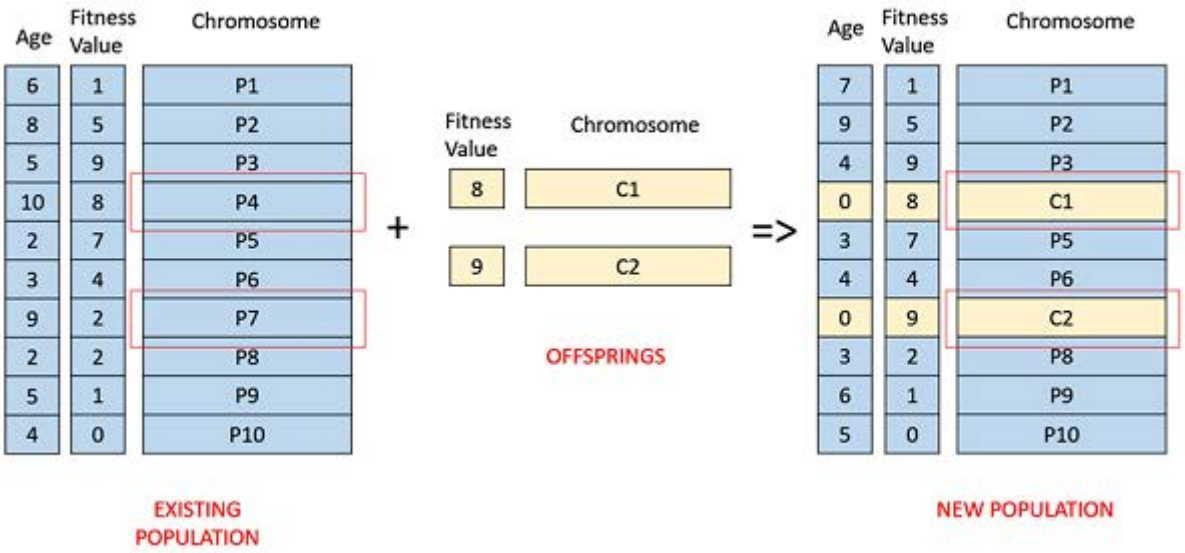
Some GAs employ **Elitism**. In simple terms, it means the current fittest member of the population is always propagated to the next generation. Therefore, under no circumstance can the fittest member of the current population be replaced.

The easiest policy is to kick random members out of the population, but such an approach frequently has convergence issues, therefore the following strategies are widely used.

Age Based Selection

In Age-Based Selection, we don't have a notion of a fitness. It is based on the premise that each individual is allowed in the population for a finite generation where it is allowed to reproduce, after that, it is kicked out of the population no matter how good its fitness is.

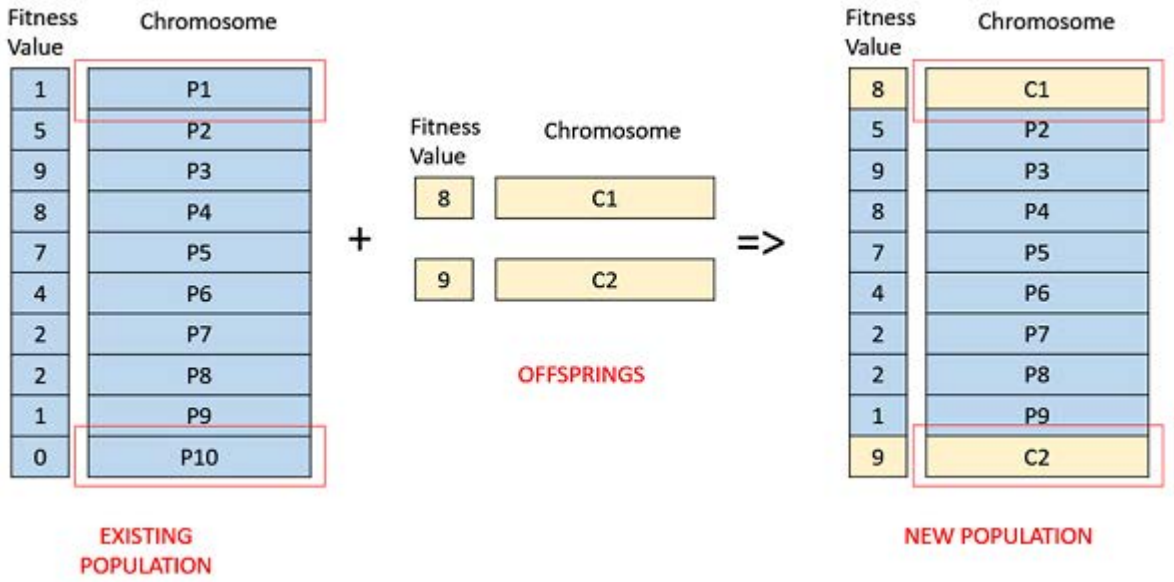
For instance, in the following example, the age is the number of generations for which the individual has been in the population. The oldest members of the population i.e. P4 and P7 are kicked out of the population and the ages of the rest of the members are incremented by one.



Fitness Based Selection

In this fitness-based selection, the children tend to replace the least fit individuals in the population. The selection of the least fit individuals may be done using a variation of any of the selection policies described before – tournament selection, fitness proportionate selection, etc.

For example, in the following image, the children replace the least fit individuals P1 and P10 of the population. It is to be noted that since P1 and P9 have the same fitness value, the decision to remove which individual from the population is arbitrary.



Genetic Algorithms - Termination Condition

The termination condition of a Genetic Algorithm is important in determining when a GA run will end. It has been observed that initially, the GA progresses very fast with better solutions coming in every few iterations, but this tends to saturate in the later stages where the improvements are very small. We usually want a termination condition such that our solution is close to the optimal, at the end of the run.

Usually, we keep one of the following termination conditions −

- **When there has been no improvement in the population for X iterations.**
- **When we reach an absolute number of generations.**
- **When the objective function value has reached a certain pre-defined value.**

For example, in a genetic algorithm we keep a counter which keeps track of the generations for which there has been no improvement in the population. Initially, we set this counter to zero. Each time we don't generate off-springs which are better than the individuals in the population, we increment the counter.

However, if the fitness any of the off-springs is better, then we reset the counter to zero. The algorithm terminates when the counter reaches a predetermined value.

Like other parameters of a GA, the termination condition is also highly problem specific and the GA designer should try out various options to see what suits his particular problem the best.

TRIBHUVAN UNIVERSITY INSTITUTE OF SCIENCE AND TECHNOLOGY

Central Department of Computer Science and Information Technology

Kirtipur, Kathmandu

Genetic Algorithm Assignment

“ Mathematical foundation of genetic algorithm and building block hypothesis with reference to schema theorem. Highlight its importance in GA ”

Submitted to

Asst. Prof. Ram Krishna Dahal

CDCSIT, TU

Submitted By

Dharmendra Thapa (29/077)

Sandesh Bista (24/077)

1 Present the mathematical foundation of genetic algorithm and building block hypothesis with reference to schema theorem. Highlight its importance in GA.

1.1 Genetic Algorithm

A Genetic Algorithm (GA) is a heuristic optimization technique inspired by the principles of natural evolution and genetics. It is used to solve complex optimization and search problems by mimicking the process of natural selection. GAs work by evolving a population of potential solutions over multiple generations to find the best or near-optimal solution to a given problem.

GAs are particularly effective for solving complex problems with a large solution space where traditional optimization methods might struggle due to the high dimensionality or nonlinearity of the problem. GAs can handle both continuous and discrete optimization problems, and they are known for their ability to perform global search by maintaining diversity in the population. While GAs are powerful optimization techniques, they require careful parameter tuning and problem-specific adjustments to ensure their success. Practitioners often experiment with different parameters, selection mechanisms, crossover and mutation rates, and population sizes to find the right configuration for a given problem.

Overall, Genetic Algorithms provide a versatile and efficient approach to solving optimization problems across various domains, ranging from engineering and finance to artificial intelligence and beyond.

1.2 Mathematical Foundation of Genetic Algorithm

During the early 1950s many scientists began pioneering genetics and evolution-based computation processes. John Holland is generally credited as having invented genetic algorithms. In 1970s, he proposed schema theory. Holland's schema theorem, also called the fundamental theorem of genetic algorithms, is an inequality that results from coarse-graining an equation for evolutionary dynamics. The theorem was initially widely taken to be the mathematical foundation for explanations of the power of genetic algorithms.

Genetic Algorithms operate based on a population of candidate solutions (chromosomes) that evolve over generations to improve their fitness in solving a given optimization problem. The key operators are:

1. **Selection:** Choosing individuals from the population for reproduction based on their fitness. The probability of selection is proportional to fitness.

2. **Crossover:** Combining genetic material from two parent solutions to create offspring. This process is inspired by biological recombination.

3. **Mutation:** Introducing small random changes to individual solutions to maintain diversity and explore new regions of the solution space.

1.3 Building Block Hypothesis

The Building Block Hypothesis states that short, low-order schemata (building blocks) tend to form the basis of solutions in GAs. Schemata are strings of genes (bits) that share specific positions. For instance, in a binary representation, a schema might be defined by the pattern "1**0" where '*' represents any bit value.

1.4 Schema Theorem

The Schema Theorem, introduced by John Holland, connects the building block hypothesis to the behavior of GAs. The theorem provides insights into how schemata of different lengths and compositions propagate through generations. It establishes that the expected proportion of a schema in the next generation is a function of its fitness, length, and the probability of crossover and mutation. It consist 2 Properties.

1. **Schema Order o(H):** schema order in GAs refers to the length of building blocks or patterns under consideration in the population. The order of schema H, denoted by o(H), is the number of fixed positions present in the template.

   Examples

   `0 1 1 * 1 * 1` => the order of the schema is 5

2. **Defining Length δ(H):** The defining length of a schema H, denoted by δ(H), is the distance between the first and last specific string position.

   Examples

   `0 1 1 * 1 * 1` => the length of the schema is 6. because the last specific position is 7 and the first specific position is 1, and the distance between them is, `Length = 7-1 = 6`

1.5 Effect on schemata

Effect of reproduction on schemata

Suppose at a given time step t there are m examples of a particular schema H contained within the population $A(t)$ where $m = m(H, t)$. During the reproduction, a string is copied according to its fitness or more precisely a string A_i gets selected with probability given by,

$$p_i = \frac{f_i}{\sum f_i}$$

After picking a non-overlapping population of size n with replacement from the population $A(t)$, we expect to have $m(H, t + 1)$ representative of the schema H in the population at time $t + 1$ as given by the equation:

$$m(H, t + 1) = m(H, t).n.\frac{f}{\sum f_j}$$

where $f(H)$ is the average fitness of the strings representing schema, H, at time t. If the average fitness of the entire population may be written as:

$$\bar{f} = \frac{\sum f_j}{n}$$

then we may rewrite the reproductive schema growth equation as:

$$m(H, t+1) = m(H, t).\frac{f(H)}{\bar{f}}$$

A particular schema grows as a ratio of the average fitness of the schema to the average fitness of the population. That is, schemata with fitness values above the population average will receive an increasing number of samples in the next generation, while schemata with fitness value below the population average will receive a decreasing number of samples. In other words, all the schemata in a population grow or decay according to their schema averages under the operation of the reproduction alone.

Suppose we assume that a particular schema H remains above average an amount c $\bar{f}$ with c a constant. Under this assumption we can rewrite the schema difference equation as follows:

$$m(H, t+1) = m(H, t).\frac{\bar{f} + c\bar{f}}{\bar{f}} = (1 = c).m(H, t)$$

Starting at t = 0 and assuming a stationary value of c, we obtain the equation:

$$m(H, t) = m(H, 0).(1 + c)^{`}$$

This equation is a geometric progression or the discrete analog of an exponential form. The reproduction allocates exponentially increasing (decreasing) number of trials to above (below) average schemata. The reproduction alone does nothing to promote exploration of new regions of the search space, since no new points are searched; if we copy old structures without change.

Effect of crossover on schemata

Crossover is a structured yet randomized information exchange between strings. It creates new structure with a minimum of disruption to the allocation strategy dictated by reproduction alone. This results in exponentially increasing (or decreasing) proportions of schemata in a population on many of the schemata contained in the population. To see which schemata are affected by crossover and which are not, consider a particular string of length l = 7 and two representative schemata within that string:

A	1	1	1	1	0	0	0
H_1	*	1	*	*	*	*	0
H_2	*	*	*	1	0	*	*

The two schemata H_1 and $H2$ are represented in the string A. Suppose string A has been chosen for mating and crossover. In this string of length 7, suppose we choose the crossing site between positions 3 and 4. The effect of this cross on our two schemata $H1$ and $H2$ can be seen in the following example, where the crossing site has been marked with the $||$:

A	1	1	1	1	0	0	0
H_1	*	1	*	*	*	*	0
H_2	*	*	*	1	0	*	*

Unless string A's mate is identical to A at the fixed positions of the schema, the schema $H1$ will be destroyed because the 1 at position 2 and the 0 at position 7 will be placed in different offspring. With the same cut point, schema $H2$ will survive because the 1 at position 4 and the 0 at position 5 will be carried intact to a single offspring. The schema $H1$ is less likely to survive crossover than schema $H2$ because on average the cut point is more likely to fall between the extreme fixed positions. The schema $H1$ has a defining length of 5, so the crossover site is selected uniformly at random among l – 1 = 7 -1 = 6 possible sites, then clearly schema $H1$ is destroyed with probability:

$$p_d = \frac{\delta(H_1)}{l-1} = \frac{5}{6}$$

It survives with probability:

$$p_s = 1 - p_d = \frac{1}{6}$$

Similarly, the schema H2 has defining length $\delta(H2) = 1$, and it is destroyed during that one event in six where the cut site is selected to occur between positions 4 and 5 such that $p_d = \frac{1}{6}$ or the survival probability is $p_s = 1–p_d = \frac{5}{6}$.

The survival probability under simple crossover is given by,

$$p_s = 1 - \frac{\delta(H)}{l-1}$$

If crossover is itself performed by random choice, say with probability p_c at a particular mating, the survival probability may be given by the expression:

$$p_s \geq 1 - p_c.\frac{\delta(H)}{l-1}$$

which reduces to the earlier expression when $p_c = 1.0$.

The combined effect of reproduction and crossover may now be considered. Assuming the independence of the reproduction and crossover operations, the number of particular schema H expected in the next generation can be estimated as:

$$m(H, t+1) \geq m(H, t).\frac{f(H)}{\bar{f}}.\left[1 - p_c.\frac{\delta(H)}{l-1}\right]$$

Schema H grows or decays depending upon a multiplication factor. With both crossover and mutation, that factor depends upon two things:

- whether the schema is above or below the population average and
- whether the schema has relatively short or long defining length

Those schemata with both above-average observed performance and short defining lengths are going to be sampled at exponentially increasing rates.

Effect of mutation on schemata

Mutation is a random alteration of a single position with probability p_m. In order for a schema H to survive, all of the specified positions must themselves survive. Therefore, since a single allele survives with probability $(1–p_m)$, and since each of the mutations is statistically independent, a particular schema survives when each of the $o(H)$ fixed positions within the schema survives. The probability of surviving mutation is $(1–p_m)^o(H)$. For small values of $pm(pm << 1)$, the schema survival probability may be approximated by the expression $1–o(H)\ p_m$. Therefore a particular schema H receives and expected number of copies in the next generation under reproduction, crossover and mutation as given by the following equation:

$$m(H, t+1) \geq m(H, t).\frac{f(H)}{\bar{f}}.\left[1 - p_c.\frac{\delta(H)}{l-1} - o(H)p_m\right]$$

This concludes that the short, low-order and above-average schemata receive exponentially increasing trials in subsequent generations

1.6 Schema processing example

Consider the problem of maximizing the function $f(x) = x^2$, where x is permitted to vary between 0 and 31. To use a genetic algorithm we must first code the decision variables of our problem as some finite-length string. For this problem, we will code the variable x simply as a binary unsigned integer of length 5.

To start off, we select an initial population at random. We select a population of size 4 by tossing a fair coin 20 times. Looking at this population, shown on the left-hand side of Table 1, we observe that the decoded x values are presented along with the objective values$f(x)$.

String No.	Initial Population	x value	$f(x) = x^2$	$pselect_i = \frac{f_i}{\sum f}$	Expected Count = $\frac{f_i}{\bar{f}}$	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 $(\sum f)$			1170	1	4	4
Average $(\frac{\sum f}{n})$			293	0.25	1	1
Max			576	0.49	1.97	2

Table 1: String Processing

A generation of the genetic algorithm begins with reproduction. We select the mating pool of the next generation by spinning the weighted roulette wheel four times. Actual simulation of this process using coin tosses has resulted in string 1 and string 4 receiving one copy in the mating pool, string 2 receiving two copies, and string 3 receiving no copies, as shown in Table 2. Comparing this with the expected number of copies $(npselect_i)$, we have obtained: the best get more copies, the average stay even, and the worst die off.

Mating Pool after Reproduction	Mate	Crossover Site	New Population	x Value	$f(x) = x^2$
0 1 1 0 | 1	2	4	0 1 1 0 0	12	144
1 1 0 0 | 0	1	4	1 1 0 0 1	25	625
1 1 | 0 0 0	4	2	1 0 0 1 1	27	729
1 0 | 0 1 1	3	2	1 0 0 0 0	26	256
Sum					1756
Average					439
Max					729

Table 2: String Processing

With an active pool of strings looking for mates, simple crossover proceeds in two steps:

- trings are mated randomly, using coin tosses to pair off the happy couples, and
- mated string couples cross over, using coin tosses to select the crossing sites

Referring to Table 2, random choice of mates has selected the second string in the mating pool to be mated with the first. With a crossing site of 4, the two strings 0 1 1 0 1 and 1 1 0 0 0 cross and yield two new strings 0 1 1 0 1 and 1 1 0 0 1. The remaining two strings in the mating pool are crossed at site 2; the resulting strings are shown in the table. Finally, mutation is performed on bit-by-bit basis. We assume that the probability of mutation is 0.001. With 20 transferred bit positions we should expect 20·0.001 = 0.01 bits to undergo mutation during a given generation. Simulation of this process indicates that no bits undergo mutation for this probability value. As a result, no bit positions are changed from 0 to 1 or vice versa during this generation.

H_1	1	*	*	*	*
H_2	*	1	0	*	*
H_3	1	*	*	*	0

During the reproduction phase, the strings are copied probabilistically according to their fitness values. Looking at the first column of the Table 1, the strings 2 and 3 are both representatives of the schema 1 * * * *. After reproduction, three copies of the schema have been produced (strings 2, 3, 4 in the mating pool column) as shown in Table 3.

Before Reproduction				After Reproduction		
		String Representatives	Schema Average Fitness f(H)	Expected Count	Actual Count	String Representatives
H_1	1 * * * *	2,4	469	3.20	3	2,3,4
H_2	* 1 0 * *	2,3	320	2.18	2	2,3
H_3	1 * * * 0	2	576	1.97	2	2,3

Table 3: Schema Processing

From the schema theorem, we compute the expected number of copies of schema after reproduction as:

$$m(H_1, t+1) = m(H_1, t).\frac{f(H_1)}{\bar{f}} = 2 * \frac{469}{293} = 3.20$$

$$m(H_2, t+1) = m(H_2, t).\frac{f(H_2)}{\bar{f}} = 2 * \frac{320}{293} = 2.18$$

$$m(H_3, t+1) = m(H_3, t).\frac{f(H_3)}{\bar{f}} = 2 * \frac{576}{293} = 1.97$$

Comparing these to the actual number of schemata, we see that we have the correct number of copies. Next, we apply crossover with crossover probability, p_c = 1, and we obtain the results as show in Table 4.

After Reproduction and Crossover			After Reproduction, Crossover, and Mutation		
Expected Count	Actual Count	String Representatives	Expected Count	Actual Count	String Representatives
3.20	3	2,3,4	3.20	3	2,3,4
1.64	2	2,3	1.64	2	2,3
0	1	4	0	1	4

Table 4: Schema Processing

We compute the expected number of copies of schema after reproduction and crossover as:

$$m(H_1, t+1) \geq m(H_1, t).\frac{f(H_1)}{\bar{f}}\left[1 - p_c.\frac{\delta(H_1)}{l-1}\right] = 3.20 * \left[1 - 1 * \frac{0}{4}\right] = 3.20$$

$$m(H_2, t+1) \geq m(H_2, t).\frac{f(H_2)}{\bar{f}}\left[1 - p_c.\frac{\delta(H_2)}{l-1}\right] = 2.18 * \left[1 - 1 * \frac{1}{4}\right] = 1.64$$

$$m(H_3, t+1) \geq m(H_3, t).\frac{f(H_3)}{\bar{f}}\left[1 - p_c.\frac{\delta(H_3)}{l-1}\right] = 1.97 * \left[1 - 1 * \frac{4}{4}\right] = 0$$

Furthermore, with the mutation rate set at $p_m = 0$ we expect to have no bits changed within the three schema copies in the three strings. As a result, we observe that for schema H_1, we obtain the expected exponentially increasing number of schemata as predicted by the schema theorem. For the short schema, schema H_2, the two copies are maintained even though crossover has occurred. Because the defining length is short we expect crossover to interrupt the process only one time in four (l–1 = 5–1 = 4). As a result, the schema $_H2$, survives high probability. And because H_3 has the long defining length ($\delta(H_3) = 4$), crossover destroyed this schema.

1.7 Importance of the Schema Theorem in GAs

1. **Explain Convergence and Diversity:** The theorem explains how building blocks can lead to both convergence (focus on promising regions of the solution space) and diversity (exploration of various solutions). It clarifies how these dynamics change over time.

2. **Guide Operator Choices:** The theorem provides guidance on how to design crossover and mutation operators to preserve and propagate valuable building blocks while exploring new solutions effectively.

3. **Predict Algorithm Behavior:** By understanding how schemata evolve, practitioners can predict how GAs will behave on specific problems, helping them fine-tune parameters and strategies for optimal performance.

4. **Enhance Algorithm Design:** The theorem guides the design of selection mechanisms, population sizes, and genetic operators to strike a balance between exploration and exploitation.

5. **Analyze Performance:** The Schema Theorem enables the analysis of algorithm performance, aiding researchers in assessing the efficiency and effectiveness of GAs for different problem domains.

Submitted By:
Dharnidhar Thakur (23/077)
Rishav Acharya (01/077)

Q.n(3).

Discuss and present the situations that the objective function shall map to fitness function. Explain the various ways of fitness scaling.

Ans:

The objective function is the function that the genetic algorithm is trying to optimize. It is a mathematical expression that measures the quality of a solution. The fitness function is a way of interpreting the objective function so that it can be used by the genetic algorithm.

The objective function and the fitness function must be carefully mapped to each other. The fitness function must be able to accurately measure the quality of a solution, and it must be able to do so in a way that the genetic algorithm can understand.

There are a few things to keep in mind when mapping the objective function to the fitness function:

- The fitness function should be computationally efficient. The genetic algorithm will need to evaluate the fitness function many times, so it is important that it does not take too long to compute.
- The fitness function should be scalable. The genetic algorithm can be used to solve problems of varying sizes, so the fitness function should be able to handle problems of different sizes.
- The fitness function should be differentiable. The genetic algorithm uses gradient descent to optimize the objective function, so the fitness function must be differentiable in order for gradient descent to work.

Here is an example of how the objective function and the fitness function can be mapped to each other:

- The objective function is to find the shortest path between two points in a graph.
- The fitness function can be defined as the length of the path.

Submitted By:
Dharnidhar Thakur (23/077)
Rishav Acharya (01/077)

- This is a simple example, but it illustrates the basic principles of mapping the objective function to the fitness function. In more complex problems, the mapping may be more difficult, but the same principles still apply.

Here are some other examples of fitness functions:

- In a problem of finding the optimal placement of sensors, the fitness function could be the sum of the distances between each sensor and the points that need to be monitored.
- In a problem of finding the optimal investment portfolio, the fitness function could be the expected return of the portfolio.
- In a problem of finding the optimal design of a circuit, the fitness function could be the power consumption of the circuit.
- The choice of fitness function is problem-specific. There is no single fitness function that is best for all problems. The best fitness function is the one that best measures the quality of the solutions to the problem.

There are also numerical concepts involved in the mapping of the objective function to the fitness function. Here are a few examples:

1. Normalization: The fitness function should be normalized so that it has a consistent scale. This is important because the genetic algorithm uses a roulette wheel selection mechanism, and the probability of an individual being selected is proportional to its fitness. If the fitness function is not normalized, then the individuals with the highest fitness may not always be selected.
2. Scaling: The fitness function may need to be scaled to handle problems of varying sizes. For example, the fitness function for the shortest path problem could be scaled by the number of nodes in the graph.
3. Differentiation: The fitness function must be differentiable in order for gradient descent to work. This is because gradient descent uses the derivative of the fitness function to update the parameters of the model.

Submitted By:
Dharnidhar Thakur (23/077)
Rishav Acharya (01/077)

In addition to these numerical concepts, there are also some more abstract concepts that are important to consider when mapping the objective function to the fitness function. These concepts include:

a. Measurability: The fitness function must be able to accurately measure the quality of a solution. This is important because the genetic algorithm will need to evaluate the fitness function many times, and it is important that the results are reliable.
b. Interpretability: The fitness function must be interpretable by the genetic algorithm. This means that the genetic algorithm must be able to understand how the fitness function works and how to use it to improve the solutions.
c. Robustness: The fitness function should be robust to noise and outliers. This is important because the genetic algorithm will be working with noisy data, and it is important that the fitness function is not unduly affected by noise.

The choice of fitness function is a critical decision in the design of a genetic algorithm. The fitness function must be carefully chosen to ensure that the genetic algorithm can find good solutions to the problem.

For Fitness Scaling

The main purpose of fitness scaling is to reduce the effect of fitness outlier in the population The process of scaling is intended to maintain selective pressure as the overall performance rises within the population. Scaling Mechanism is used to distinguish the performance of the individual when improvement among the population is minute. Instead of scaling parent based on the raw fitness value (x) we select parent based on some function of the row fitness f(x). Fitness Scaling is used to scale the row fitness value so that the GA sees a reasonable amount of difference in the scaled fitness value of the best versus worst individual. The fitness Scaling is useful to avoid the premature convergence and slow finishing

The fitness scaling can be done is various ways. The types of fitness scaling are

1. Linear Scaling

Submitted By:
Dharnidhar Thakur (23/077)
Rishav Acharya (01/077)

2. Sigma Scaling / Sigma Truncation
3. Power law
4. Rank Scaling

1. Linear Scaling:

Linear Scaling adjust the fitness value of all the string such that the best individual get the fixed number of expected offspring. Other value are altered so as to ensure that the correct number of new string are produced (i.e an average individual will still expect one offspring). Exceptionally fit individual are prevented to take over quickly (i.e without linear scaling best fit individual can easily take over quickly. A linear relation between the scaled fitness f and raw fitness f of the form

$f' = af + b$ where a and b are scaling coefficient.

- The coefficient a and b are chosen in number of way.
- However in all cases the $f'_{avg} = f_{avg}$ in order to ensure the presence of an average individual in next generation
- In order not to allow dominance by super individual the number of copies assigned to them is controlled by taking

$$f'_{max} = C * f_{avg}$$

Where C is number of copies of highly fit individual.

For typical population size of n=50 to 100, C= [1.2 , 2] has been used successively

There are 2 different cases to choose the value of C

<u>Case I:</u>

- Initially C is chosen any desired value

Submitted By:
Dharnidhar Thakur (23/077)
Rishav Acharya (01/077)

- If $f_{min} > \frac{(C*f_{avg}-f_{max})}{C-1}$ then

$$a = \frac{f_{avg}(C-1)}{f_{max}-f_{avg}} \qquad b = \frac{f_{avg}(f_{max}-C*f_{avg})}{f_{max}-f_{avg}}$$

else,

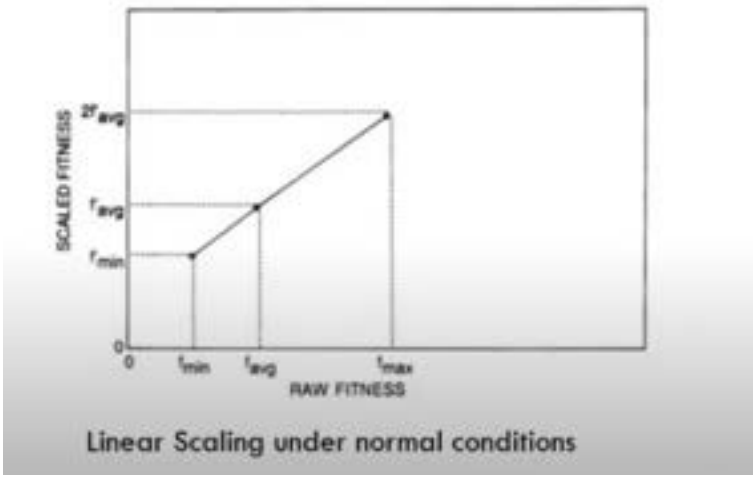
$$a = \frac{f_{avg}}{f_{avg}-f_{min}} \qquad b = \frac{-f_{avg}*f_{min}}{f_{avg}-f_{min}}$$

Case II:

- For entire run we take C=2
- If $f_{min} > (2 * f_{avg} - f_{max})$ then

$$a = \frac{f_{avg}}{f_{max}-f_{avg}} \qquad b = \frac{-f_{avg}*f_{min}}{f_{max}-f_{avg}}$$

Under Normal Condition:



2. Sigma Scaling/Sigma Truncation:

- Linear Scaling gives negative scaling fitness unless special steps are taken Negative scaled fitness results at matured runs due to one or two very week members (low fitness value).

Submitted By:
Dharnidhar Thakur (23/077)
Rishav Acharya (01/077)

- Sigma Scaling helps to overcome the problem.
- Sigma Scaling keeps the baseline near average.
- Setting the baseline s standard deviation (σ) below the mean where s is the scaling factor.
- Individuals below this score assigned fitness 0
- Sigma Truncation" discards such off the average member
- Linear scaling is then applied to the remaining members.

$$f'' = f - (f_{avg} - s\sigma) \text{ if RHS} > 0$$

$$= 0, \text{otherwise}$$

- After the Linear Scaling is applied without the danger of negative fitness

$$f' = af'' + b$$

Thus, fitness scaling can proceed without the danger of negative results.

3. Power Law

- In Power law scaling, the Scaled fitness is given by

$$f' = f^k(\, raw\ fitness\ f)$$

- K-problem dependent constant
- In machine vision application, Gillies took k=1.005

4. Rank Scaling:

1. Rank selection first ranks the population and then every chromosome receives fitness from this ranking.
2. The worst will have fitness 1, second worst 2 etc. and the best will have fitness N (number of chromosomes in population).

Submitted By:
Dharnidhar Thakur (23/077)
Rishav Acharya (01/077)

3. After this all the chromosomes have a chance to be selected.
4. Rank-based selection schemes can avoid premature convergence.
5. But can be computationally expensive because it sorts the populations based on fitness value.
6. But this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones

Thus , the situations for the objective function mapping to fitness function and different ways of fitness scaling are described above.

TRIBHUVAN UNIVERSITY

Central Department of Computer Science and Information Technology
Kirtipur

Assignment on :

Genetic Algorithms

Submitted by:
Tulsi Acharya (Roll No. : 12)
Jina Chaudhary (Roll No. : 20)
M.Sc. CSIT Fourth Semester

Submitted To:
Asst. Prof. Ram Krishna Dahal
CDCSIT, TU

August, 2023

Question: *Discuss and present the issues posed by the two arm BANDIT problem and its implementation in GA with reference to enhancement of security of Computer Network.*

Answer:

The Two-Armed Bandit Problem:

The Two-Armed Bandit Problem is a classic dilemma in the field of probability theory and decision-making i.e; it is an important and classical problem of decision theory. It is a classic reinforcement learning problem in which a player is faced with two slot machines, each with a different payout distribution and has to choose between them to maximize rewards .

It models a situation where a gambler (or decision-maker) is faced with a choice between two options (arms) from which to draw rewards. Each arm provides a stochastic reward, and the decision-maker's goal is to maximize their cumulative reward over a sequence of trials while learning about the arms' reward distributions.

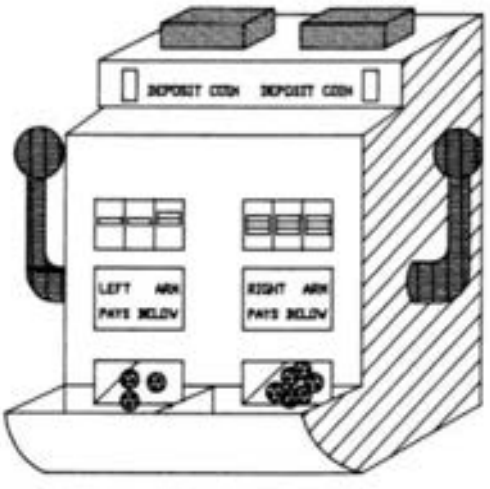
In the context of network security, this problem can be seen as selecting the best security measures to deploy in a dynamic environment.

Multi-armed or k-armed bandit problem involves making a series of decisions in a situation where each decision has uncertain rewards associated with it.

The two armed Bandit problem can be formalized as follows:

- There are two arms (A and B).
- Pulling arm A yields a reward from a probability distribution P(A).
- Pulling arm B yields a reward from a probability distribution P(B).
- The goal is to maximize the cumulative reward over a fixed number of trials.

Suppose we have a two-armed slot machine, as depicted in figure with two arms namedLEFT and RIGHT. Furthermore, let's assume we know that one of the arms pays an award μ_1 with variance σ_1^2 and the other arm pays an award μ_2 with variance σ_2^2 where $\mu_1 \geq \mu_2$. Which arm should we play? Clearly, we would like to play the arm that pays off more frequently (the arm with payoff μ_1), but therein lies the rub. Since we don't know beforehand which arm is associated with the higher expected reward, we are faced with an interesting dilemma. Not only must we make a decision about which arm to play, but we must at the same time collect information about which is the better arm. This trade-off between the exploration for knowledge and the exploitation of that knowledge is a recurrent and fundamentally important theme in adaptive system theory.



Issues Posed by the Two-Armed Bandit Problem:

1. **Exploration vs. Exploitation Dilemma:**
   Choosing an arbitrary action when the outcome is unknown is called **exploration**. Taking actions according to the expectations and using prior knowledge to get immediate rewards is called **exploitation**. The dilemma of choosing what the agent already knows vs. gaining additional knowledge is called the **exploration vs. exploitation tradeoff**.
   The key challenge in the Two-Armed Bandit problem is the trade-off between exploration (trying out both arms to learn their reward distributions) and exploitation (choosing the arm that seems to have the highest reward based on current knowledge). Striking the right balance is essential for maximizing long-term rewards.
   Exploration can lead to higher reward in the long run while sacrificing short term rewards. Conversely, exploiting ensures an immediate reward in the short-term, with uncertainty in the long run.
2. **Regret Minimization:**
   Regret refers to the difference between the rewards that could have been earned if the best arm were chosen every time and the rewards actually earned. The objective is to minimize this regret, which represents the cost of suboptimal decision-making.
3. **Non-stationary rewards:**
   The payout distributions of the two slot machines may change over time. This means that the player must constantly be learning about the machines in order to make the best decisions.
4. **Limited trials:**
   The decision-maker initially has limited information about the arms' reward distributions. Learning the optimal arm requires making informed choices while gradually updating beliefs based on observed rewards.
5. **Prior knowledge:**
   The player may have some prior knowledge about the two slot machines, such as the payout distributions of the machines. This prior knowledge can be used to improve the player's chances of winning.
6. **Multiple players:**
   There may be multiple players playing the two-armed bandit problem. In this case, the players must compete with each other to maximize their cumulative rewards.
7. **Hidden rewards:**
   The rewards from the two slot machines may not be visible to the player. In this case, the player must learn about the rewards by trial and error.

Enhancement of Security of Computer Networks:

Some of the challenges in Network Security are:

- Rapidly changing threats.
- Uncertain outcomes of security measures.

- Need for smarter decision-making.

Enhancing the security of a computer network involves various measures to protect the network and its components from unauthorized access, cyberattacks, and other threats. This includes implementing firewalls, intrusion detection systems, encryption protocols, access controls, and more.

Genetic Algorithms (GAs) are a type of optimization algorithm inspired by the process of natural selection. They can be applied to various optimization problems, including enhancing the security of computer networks.

The Implementation the Two-Armed Bandit Problem in GA:

1. **Define Parameters:**
   It involves identifying various security configurations or measures that can be considered as "arms." These could be different firewall settings, intrusion detection rules, access control policies, etc.
2. **Reward Function:**
   It involves defining a reward function that measures the effectiveness of each security configuration or strategy. This could be based on metrics such as successful intrusion prevention, rapid threat detection, minimal disruption to network operations, etc.
3. **GA:**
   It involves the following steps:

- Initialization: Create a population of security configurations. Each configuration is a list of rules that the security analyst can use to choose the best security measures.
- Selection: Choose configurations from the population based on their potential effectiveness (fitness) as evaluated by the reward function. The fittest configurations are then chosen. The fitness of a strategy is a measure of how well it can enhance security.
- Crossover and Mutation: The selected configurations are then combined and modified to create new ones.
- Evaluation: The reward function is used to evaluate the fitness of the new configurations.
- Repeat: The steps selection, crossover, mutation, and evaluation steps are repeated until the GA converges on the best configuration.

4. **Adaptive Security:**
   Adaptive security involves adjusting and evolving security measures over time in response to changing threats and vulnerabilities. In the context of genetic algorithms, this concept is applied by allowing the algorithm to gradually favor configurations that have demonstrated better performance in terms of network security. As the algorithm progresses through generations, it learns from past experiences and adapts its approach based on which configurations are more effective at countering attacks.
5. **Feedback Loop:**
   The feedback loop involves a continuous process of assessing the network's security performance, incorporating the results into the optimization process, and updating the

genetic algorithm's parameters or reward function to reflect the changing threat landscape. This loop ensures that the algorithm's decisions remain relevant and responsive to emerging security challenges.

Genetic Algorithms for Enhancement of Computer Network Security:

Genetic Algorithms (GAs) can be applied to various areas in the enhancement of computer network security.

Here are some areas where GAs have been used or can potentially be applied:

1. **Routing**:
   In computer networks, routing refers to the process of determining the optimal paths for data to travel between nodes. GAs can be applied to optimize routing strategies for improved network security
2. **Intrusion Detection Systems (IDS):** GAs can be used to train IDSs to detect new or evolving attack patterns, optimize the configuration of IDSs, and develop new IDS techniques. GAs can also be used to optimize the configuration of IDSs by determining the optimal thresholds for triggering alerts, the types of traffic to monitor, and the response strategies to take when an alert is triggered.
3. **Firewall Configuration:** GAs can help in optimizing firewall rules and policies to efficiently filter and block malicious traffic while allowing legitimate traffic to pass through.
4. **Vulnerability Assessment:** GAs can assist in prioritizing vulnerabilities for patching by considering factors such as the severity of the vulnerability, potential impact, and available resources.
5. **Network Topology Design**: GAs can aid in designing network topologies that are inherently more secure, by optimizing the placement of security devices, segregating critical components, and minimizing attack surfaces.
6. **Key Management and Cryptography:** GAs can optimize the generation and distribution of cryptographic keys, enhancing the security of communication channels and data protection.
7. **Resource Allocation:** GAs can optimize the allocation of security resources (such as computational power and bandwidth) to different network segments based on their current threat levels and importance.
8. **Security Policy Optimization:** GAs can be used to optimize security policies, access controls, and authentication mechanisms to strike a balance between usability and security.
9. **Malware Detection:** GAs can be applied to improve the accuracy of malware detection by evolving signatures and behavioral patterns.
10. **Network Traffic Analysis:** GAs can help in identifying patterns of malicious network traffic by optimizing the analysis of large volumes of data.

11. **DDoS Attack Mitigation:** GAs can be used to mitigate DDoS attacks by dynamically adjusting network traffic routing and filtering rules.

Give and illustration of Dominance, Diploidy, and Abeyance along with its importance and application with reference to GA.

Prepared by:

Sita Sapkota (8)

Utsav Baral (10)

Dominance:

Mendel's law of dominance states that in a heterozygote, one trait will conceal the presence of another trait for the same characteristic. Rather than both alleles contributing to a phenotype, the dominant allele will be expressed exclusively.

In the context of Genetic Algorithms (GAs),

- dominance refers to the superiority of one individual over another in terms of their fitness or objective value
- Crucial role in selection processes within GAs,
- Determining which individuals are chosen to contribute their genetic material to the next generation.
- Here's an illustration of how dominance works in a Genetic Algorithm:
- Let's consider a simple optimization problem where we're trying to find the maximum value of a mathematical function: f(x) = x^2. We'll represent individuals (potential solutions) as numerical values for the variable x. The fitness of an individual will be determined by its corresponding function value, f(x).
- Initialization:

- Generate a population of individuals with random values of x. For example, {2.5, 1.0, 4.2, 3.8, 0.9}.
- Fitness Evaluation:
- Calculate the fitness of each individual by evaluating the function f(x) = x^2. The fitness values might be {6.25, 1.0, 17.64, 14.44, 0.81}.

Selection:

- Dominance comes into play during selection. The individuals with higher fitness values are more dominant because they represent better solutions to the optimization problem.
- A common selection method is tournament selection. In a tournament, a few individuals are randomly selected from the population, and the most dominant individual (with the highest fitness) is chosen to proceed to the next stage (reproduction)
- Example: Let's say we perform a tournament with two randomly chosen individuals: 2.5 (fitness 6.25) and 1.0 (fitness 1.0). The individual with a fitness of 6.25 wins the tournament due to its dominance and moves on to the next step.

Crossover and Mutation:

- The dominant individuals selected from the previous step are used to create offspring for the next generation. Crossover and mutation operations introduce genetic diversity and help explore the solution space.

New Generation:

- The offspring, which inherit genetic material from dominant parents, form the new population for the next generation.

- By selecting dominant individuals for reproduction, Genetic Algorithms bias the search towards promising areas of the solution space. Over successive generations, dominant traits are more likely to propagate, leading to the evolution of better solutions.
- In summary, dominance in Genetic Algorithms refers to the preferential selection of individuals with higher fitness values for reproduction. This mimics the natural process of selecting individuals with advantageous traits for survival and reproduction, and it allows GAs to efficiently explore and optimize complex solution spaces.

Importance of dominance:

- The concept of dominance plays a crucial role in Genetic Algorithms (GAs) and is important for several reasons:
- Preserving Good Traits: Dominant individuals possess genetic material that has proven to be successful in solving the problem at hand. By prioritizing these individuals for reproduction, GAs increase the chances of preserving and propagating beneficial traits through successive generations. This leads to a gradual improvement in the quality of solutions over time.
- Exploitation of Fitness Landscape: Dominance helps GAs exploit the fitness landscape by gradually concentrating the population around areas with higher fitness values. This exploitation phase is crucial for fine-tuning solutions and reaching the global optimum or a high-quality local optimum.
- Diversity Maintenance: While dominance emphasizes selecting better solutions, it's essential to strike a balance between exploitation and exploration. GAs need to maintain genetic diversity in the population to prevent premature convergence and avoid getting stuck in local optima.

However, the dominance-based selection mechanism ensures that the exploration is still guided by promising solutions.

- Parallelism in Evolution: Dominance provides a means to parallelize the evolution process. Dominant individuals can be selected and reproduced concurrently, enabling faster convergence and better utilization of computational resources.

Diploidy

- Diploidy is a concept borrowed from genetics that has been applied to various optimization and search algorithms, including Genetic Algorithms (GAs). In a genetic context, diploidy refers to the presence of two sets of genetic information in an individual, typically represented by pairs of chromosomes. Each chromosome in the pair carries a slightly different set of genetic instructions for a given trait.
- Here's an example of how diploidy can be applied in a genetic algorithm:

Example: Evolutionary Optimization of a Function

- Let's consider a simple optimization problem where we want to find the minimum value of a mathematical function, such as f(x) = x^2. In this case, x is a real-valued parameter that we want to optimize.
- Initial Population: The genetic algorithm starts with an initial population of candidate solutions (chromosomes). Each chromosome represents a potential value of the parameter x.
- Fitness Evaluation: Each chromosome's fitness is evaluated by calculating the value of the function f(x) for the corresponding parameter value. The fitness indicates how well a solution performs in the optimization task.

- Crossover and Mutation: Crossover and mutation operations are performed on the chromosomes to create new offspring. Crossover involves combining genetic material from two parent chromosomes to produce one or more child chromosomes. Mutation introduces small random changes to the chromosomes.
- Crossover with Diploidy: During crossover, the algorithm selects two parent solutions and combines their genetic material to create one or more offspring solutions. In this case, the algorithm can take genetic material from both sets of chromosomes for each parent, and combine them to create offspring with a blend of the two parameter values.
- Mutation with Diploidy: Similarly, during mutation, the algorithm can introduce random changes to both sets of chromosomes, allowing for exploration of a broader solution space.
- Selection and Survival: The offspring, along with some of the parents, are selected for the next generation based on their fitness values. The process of crossover, mutation, and selection continues over multiple generations.
- The introduction of diploidy in this genetic algorithm example allows for greater diversity and adaptability in the population, as each candidate solution is represented by two parameter values. This can lead to improved exploration of the solution space, as well as increased robustness against premature convergence to suboptimal solutions.

Applications of Dominance, Diploidy, Abeyance

Dominance

- Parent Selection: Individuals with higher fitness values (dominant individuals) are more likely to be selected as parents for the next generation.

This ensures that the genetic material of better solutions is passed on, contributing to the improvement of the population's overall fitness.

- Convergence to Optimal Solutions: Dominance ensures that individuals with better fitness values are given preference during selection. This bias towards better solutions promotes convergence towards optimal or near-optimal solutions over generations.
- Remove the conflict of Redundancy: When we have a pair of genes describing each function, something must decide which of two values to choose because the phenotype cannot have two alternating characters at the same time so to remove this conflict, dominance is used.

Elitism: Elitism is a strategy where the best-performing individuals from the current generation are directly copied to the next generation. This maintains the best solutions and ensures they are not lost during reproduction. Elitism is based on the concept of dominance, as the best individuals are dominant in terms of fitness.

Diploidy

- Dynamic Environments: In scenarios where the problem changes over time, diploidy can provide an advantage. Having duplicate gene copies allows individuals to adapt more effectively to sudden changes by preserving functional traits even if one copy experiences sudden changes. This helps the algorithm maintain performance in dynamic or uncertain environments.
- Exploration and Exploitation Balance: Diploidy can contribute to a better balance between exploration and exploitation in genetic algorithms by enhancing their ability to explore new areas of the solution space while still exploiting known promising solutions.

Preservation of Diversity: Diploidy introduces redundancy by maintaining duplicate copies of chromosomes within each individual in the population. This redundancy helps prevent the loss of diversity during reproduction and genetic operations.

TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Central Department of Computer Science and Information Technology

Kirtipur, Kathmandu

Genetic Algorithm Assignment

"**Niche and Speciation in Genetics Algorithms**"

Submitted to:

Asst. Prof. Ram Krishna Dahal

CDCSIT, TU

Submitted By:

Apil Adhikari (22/077)

Sudhan Kandel (2/077)

Table of Contents

1 Niche

In computer science, a niche is like a perfect spot in a puzzle where some solutions fit really well. It's all about making sure we have different kinds of solutions to solve a problem effectively. A niche is a specific role or region in a problem space where solutions are well-suited or well-adapted. It focuses on diversity of solutions. It diverse the solution by

- Increasing Mutation Rate
- Crossover Variations.
- Fitness sharing
- Increase Population Size.
- Random initialization
- Speciation.

1.1 Increasing Mutation Rate.

Increasing the mutation rate in computer algorithms, like genetic algorithms, means making more random changes to potential solutions. This helps explore a wider range of possibilities, potentially finding better solutions or avoiding getting stuck in suboptimal ones. Here are the simple examples showing how increasing in mutation diverse the solutions.

1. No Mutation:

Chromosome: 1100110011

Gen:

1100110011 (Itself).

Possible solution:1

Niche1:1100110011

2. Increasing Mutation Rate:

Let's simulate the effect of increasing mutation rates on the diversity of the solution. We'll start with a low mutation rate (10%), then medium (30%), and finally high (60%).

Mutation Rate: Low (e.g., 10%)

Chromosome: 1100110011

Number of gene is to muted:1

Gen:

0100110011

1000110011

……...

……

Total number of possible solution=10

Niche1:1100110011(itself)

Niche2:0100110011, 1000110011, ……. (1 bit different)

Mutation Rate: Medium (e.g., 30%)

Chromosome: 1100110011

Number of gene is to muted:3

Gen:

0100010010

1011110011

0010110011

……

Total number of possible solution is =120

Niche1:1100110011(itself)

Niche2:**0**100110011, 1000110011, …… (1 bit different)

Niche3:0100010010, 1011110011, …… (3 bit different)

Mutation Rate: High (e.g., 60%)

Chromosome: 1100110011

Number of gene is to muted:6

Gen:

1101001101

0011000011

1111001111

……..

Total number of possible solution=210

Niche1:1100110011(itself)

Niche2:**0**100110011, 1000110011, (1 bit different)

Niche3: 0100010010, 1011110011, …… (3 bit different)

Niche4:1101001101, 0011000011, 1111001111

Summary:

In the case of no mutation, only the initial chromosomes remain in its niche/solution.

As the mutation rate increases, new individuals with slight variations appear in additional niches.

Higher mutation rates lead to greater diversity in the solution.

1.2 Crossover variations

Crossover variations diversify solutions by mixing and matching different parts of solutions to create new and potentially better ones, increasing the variety of possible answers. This examples show how small changes in crossover points changes the solutions. Suppose two chromosomes 1and 2.

Chromosome 1: 1100110011

Chromosome 2: 1011010101

Step 1: Identify the Crossover Point

Crossover Point: 6

Chromosome 1: 110011|0011

Chromosome 2: 101101|0101

Step2: Create Offspring

Offspring 1: 1100110101

Offspring 2: 1011010011

Chromosome 1: 110011|0011

Chromosome 2: 101101|0101

Step 1: Identify the Crossover Point

Crossover Point:4

Chromosome 1: 1100|110011

Chromosome 2: 1011|010101

Step2: Create Offspring

Offspring 1: 1100010101

Offspring 2: 1011110011

Chromosome 1: 110011|0011

Chromosome 2: 101101|0101

Step 1: Identify the Crossover Point

Crossover Point:2

Chromosome 1: 11|00110011

Chromosome 2: 10|11010101

Step2: Create Offspring

Offspring 1: 1111010101

Offspring 2: 1000110011

Summary:

In the case of no crossover, only the initial chromosomes remain in its niche/solution.

As the cross over variates, new individuals with slight variations appear in additional solutions. Higher variations lead to greater diversity in the solution.

1.3 Fitness Sharing

Fitness sharing is a concept used in evolutionary algorithms, such as genetic algorithms, to promote diversity in a population of candidate solutions. It is particularly useful when dealing with optimization problems where multiple solutions are similar and might lead to premature convergence to a suboptimal solution.

Examples:

Suppose two chromosomes A and B has raw fitness **100**.

Similarity between them is **4.**

Total no of gene in each chromosomes is **10.**

Similarity calculation as

$\alpha = \frac{4}{10} = 0.4$

the formula for fitness sharing or penalty is $\mathbf{f} = \frac{Raw\ Fitness}{1+\alpha}$

$\mathbf{f} = \frac{100}{1+0.4}$

=71.42(The fitness value is degraded as certain percentage according to similarity).

1.4 Increasing Population

In Genetic algorithms, increasing population can diversify the solution space in several ways:

Exploration of a Broader Solution Space:

A larger population means there are more candidate solutions being considered simultaneously. This allows the algorithm to explore a wider range of possibilities in the solution space, increasing the chances of finding diverse and potentially better solutions.

Diverse Initial Solutions: A larger population often involves a greater variety of initial solutions. This diversity in the starting population can be beneficial because it provides a broader foundation for the optimization process. It's possible that some initial solutions are already close to optimal, while others are far from it, which encourages exploration.

Preventing Premature Convergence: With a larger population, there is less likelihood of the algorithm prematurely converging to a suboptimal solution. Smaller populations can get stuck in local optima, whereas larger populations have a better chance of escaping these local optima and finding globally better solutions.

Robustness to Noise: In real-world optimization problems, noise or uncertainty can affect the fitness evaluations of solutions. A larger population can provide robustness to such noise by reducing the impact of individual noisy evaluations, allowing the algorithm to make more informed decisions.

1.5 Random Initialization

Generate the initial population by creating individuals (chromosomes) randomly. This means randomly selecting values or configurations for each gene within each chromosome. The randomness ensures diversity in the initial population, as each individual represents a different potential solution.

Why is Niche Required?

Niche is required for

Promoting diversity and exploration: Initiating the algorithm with a diverse set of solutions or employing techniques like mutation and niche preservation to encourage exploration of different regions in the search space.

Avoiding premature convergence: Preventing the algorithm from settling too quickly on a suboptimal solution by maintaining diversity in the population and balancing exploration with exploitation.

Handling multimodal and complex landscapes: Adapting the algorithm to deal with problems that have multiple peaks (modes) in the solution space, ensuring it explores and exploits various modes effectively.

Robustness and Adaption: Developing algorithms that can adapt to changing problem conditions or noisy environments, ensuring consistent performance in a wide range of scenarios.

Parallel Exploration: Utilizing parallel processing to explore multiple branches of the search space simultaneously, improving efficiency and potentially finding better solutions.

Addressing Fitness Conflict: Managing situations where the objectives of optimization conflict with each other, requiring specialized algorithms or techniques to balance conflicting goals effectively.

2 Speciation

In a genetic algorithm, speciation is a technique used to promote diversity within a population of candidate solutions (also known as individuals or genomes). This diversity helps maintain a healthy exploration-exploitation balance during the evolution process, which is essential for finding optimal or near-optimal solutions to complex optimization problems. Speciation is particularly useful in problems where the solution space is highly multimodal or contains multiple suboptimal solutions.

In the speciation approach, the population is divided into distinct subpopulations or species, each focusing on evolving solutions in a specific region of the solution space. This prevents premature convergence to a single solution. Requirements of speciation is listed below.

- Diversity Preservation
- Partitioning into Species
- Reduced Inter-Species Competition
- Genetic Diversity
- Crossover and Mutation within Species

It's achieved by introducing additional parameters and mechanisms to the GA, such as tracking genetic similarity or dissimilarity between individuals. Speciation ensures that solutions that are distinct from each other have a chance to evolve separately, potentially leading to a more diverse and comprehensive set of solutions.

2.1 Types of Speciation

a) **Allopatric speciation:** This type of speciation occurs when a population is physically separated from the rest of its species, such as by a mountain range or a river.

b) **Sympatric speciation:** This type of speciation occurs without physical separation of the population. This can happen through a process called genetic drift, in which random changes in the gene pool of a population can led to the development of new adaptations.

2.2Steps of Speciation in Genetic Algorithm

1. **Initialization:** Begin with an initial population of individuals, each representing a potential solution to the optimization problem.
2. **Fitness Evaluation:** Evaluate the fitness of everyone in the population based on how well it solves the problem. The fitness function quantifies how close each solution is to the desired goal.
3. **Speciation:** To encourage diversity, you introduce a concept of species. Individuals within the same species are more like each other in terms of their genetic makeup, while individuals from different species are more dissimilar. Speciation can be achieved using various methods, including:
   a) **Distance-Based Speciation**: Calculate a genetic distance (e.g., Hamming distance, Euclidean distance, or other custom metrics) between individuals. If the distance between two individuals is below a specified threshold, they belong to the same species.
   b) **Clustering Algorithms:** Use clustering algorithms like k-means to group individuals into species based on their genetic traits.
   c) **Niching Techniques:** Employ niching techniques such as sharing functions or crowding to give individuals within the same species a competitive advantage. This helps maintain diversity by preventing closely related solutions from dominating the population.
4. **Reproduction:** Perform reproduction operations (e.g., selection, crossover, and mutation) separately within each species. This means that individuals within the same species are more likely to mate and produce offspring.
5. **Next Generation:** Create a new generation of individuals by applying genetic operators to the selected individuals within each species. Keep the best individuals from each species and apply the operators to produce a diverse set of offspring.
6. **Repeat:** Continue the evolutionary process, including fitness evaluation, speciation, reproduction, and creating new generations, for a predefined number of generations or until a termination condition is met (e.g., convergence or a maximum number of iterations).

2.3 Example

Optimize the function f (x, y) = x^2 + y^2, where x and y are real numbers in the range [-5, 5].

1. Individual A: x = 3, y = 4, fitness = 25
2. Individual B: x = -2, y = 1, fitness = 5
3. Individual C: x = 1, y = -3, fitness = 10
4. Individual D: x = -4, y = 2, fitness = 20
5. Individual E: x = 2, y = 3, fitness = 13

For simplicity, let's divide the population into two species based on the sign of their x-coordinate. Species 1 will have individuals with positive x-coordinates, and Species 2 will have individuals with negative x-coordinates.

Species 1:

Individual A: x = 3, y = 4

Individual C: x = 1, y = -3

Individual E: x = 2, y = 3

Species 2:

Individual B: x = -2, y = 1

Individual D: x = -4, y = 2

2.4 Advantage of Specification

By dividing the population into species based on the value of x, we ensure that individuals with similar characteristics remain together in their own subpopulations. This can lead to several advantages:

1. **Diversity Maintenance:** Individuals in each species focus on exploring a specific region of the solution space. This prevents premature convergence to a local optimum.
2. **Preservation of Traits:** If certain regions of the solution space are more promising, species will evolve solutions that exploit those regions effectively.
3. **Improved Exploration:** Each species can explore its own region of the solution space independently, potentially discovering unique solutions.
4. **Preventing Extinction:** If a species is struggling to find good solutions, it might continue to exist due to isolation from other species, preventing its traits from being lost.

2.5Niche Vs Speciation

Niche	Speciation in Genetic Algorithms
Focuses on species' interactions with the environment and other species.	Focuses on maintaining diversity within a population to prevent premature convergence.
Helps understand how species coexist, adapt, and interact within ecosystems.	Aims to maintain population diversity to improve the genetic search space exploration.
Describes how a species fits into an ecosystem and uses available resources.	Addresses how individuals are grouped and diversified to explore different solutions.
Not directly related to genetic algorithms, more relevant in ecology.	A strategy used within genetic algorithms to enhance diversity.
Describes how species adapt to their environment over time.	Facilitates efficient exploration and exploitation of the search space in optimization.

Multi Objective optimization in GA & Knowledge Based Techniques

Tribhuvan University

Central Department of Computer Science and Information Technology (MSC.CSIT), 4th Semester

Genetic Algorithm

Presented By:

Manoj Saru (49/077)

Prakash Paudel (25/077)

Q.N. Explain the multi-objective optimization in GA with an example. Discuss the knowledge-based technique for initializing a GA.

Multi-objective Problems:

Multi-objective problems are problems with multiple objectives or targets to achieve. There are several criteria in a single problem, so they are called multi-criteria problems. Most real-world problems are multi-objective. For example: buying a mobile phone where we want to minimize the cost but maximize the features and durability, Travelling Salesman Problem (TSP) where we want to minimize cost, time, distance, etc. In multi-objective problems, the objective can be:

1. **minimization only,**
2. **maximization only or**
3. **A Combination of both.**

Mathematically, we can express multi-objective functions as:

$$\min(f_1(x), f_2(x)\ldots, fk(x))$$

where xϵX is a feasible set of solutions that should satisfy some constraints.

and K>=2 is the number of objectives.

In some cases, maximization can be treated as equivalent to the inverse of minimization or its negative value.

Multi-objective Optimization:

Multi-objective optimization refers to the simultaneous optimization of the multiple objectives of a problem. Multi-objective optimization is also called as multicriteria optimization, vector optimization, or Pareto optimization. Generally, these objective functions are measured in different units and are often competing and conflicting. In a single objective optimization problem, most of the time the criteria are transformed to a fitness function without any additional explanation and try to seek the best optimal (highest or lowest) value for that particular function but in multi-objective optimization, it is not possible and wise to combine several objective functions into a single one. Also, we need to interrelate the relative values of different criteria. The result of multi-

objective optimization is not a single solution, but a set of nearly optimal solutions which is called the Pareto optimal solution set. The goodness of the solution can be measured using the concept called dominance.

The multi-objective optimization used the concepts of:

1. **Superiority (dominance),**
2. **Inferiority and**
3. **Non-dominant/non-inferiority solutions.**

Superiority: If x and y are the solution sets then y is superior to x if x is partially less than y i.e. (xp < y) i.e.,

- if x is worse than y in all objectives (for all i, xi<=yi) and
- x is strictly worse than y in at least one objective.
- Here y is superior or dominant to x and x is the inferior.

Non-dominant/non-inferiority: x and y are said to be non-inferior if x is neither superior nor inferior to y. non-dominance ensures that there are no other better solutions than x and y. There is also a tradeoff between the values of competing objectives.

#For example:

A widget manufacturing company wants to manufacture widgets with minimized accidents and minimized cost. 5 possible ways of running a plant are generated which resulted in the following widget cost and accident count:

A= (2,10) (widget cost, widget accident)

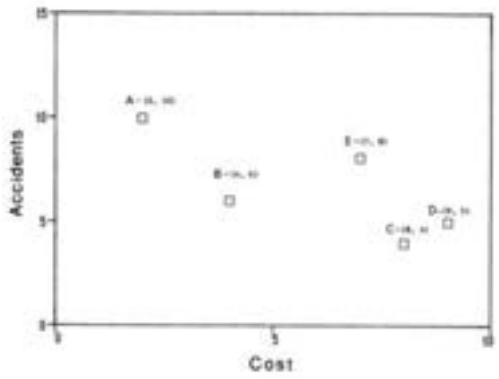
B= (4,6)

C= (8,4)

D= (9,5)

E= (7,8)

Plotting these solutions in a graph gives the following results:



Here solutions A, B, and C are considered as best solutions as they are non-dominant with each other and there are no other better solutions than them. They form a set called Pareto optimal set. There is a trade-off between each other since there is gain along one dimension and loss along another direction.

For the points D and E, D is dominated by point C since 8<9 and 4<5. Also, point E is dominated by B since 4<7 and 6<8. So, D and E are inferior to A, B, and C.

In this example, the Pareto optimal Set is **P= {A, B, C}.**

The curve formed by joining the solutions in the Pareto optimal solution forms a Pareto optimal curve. The decision maker must make a value analysis among the alternatives to reach a particular solution.

Multi-objective Optimization in GA (MOGA):

The practical scheme for using GA for multi-objective optimization was developed by Schaffer (1984), after 17 years of Rosenberg's (1967) study where he suggested using multiple properties in his simulation of the genetics and chemistry of a population of single-celled organisms. Schaffer extended Greenstreet's GENESIS program to include multicriteria functions in his Vector Evaluated Genetic Algorithm (VEGA). In his work, the selection step is modified so that at each generation a number of sub-populations is generated by performing proportional selection according to each objective function in turn.

For a problem with the k objective and M population size, there will be a k sub-population with M/k size each. These sub-populations are shuffled together to generate a new population of size M, on which GA will apply crossover, and mutation operators in a usual way.

The process is shown below:

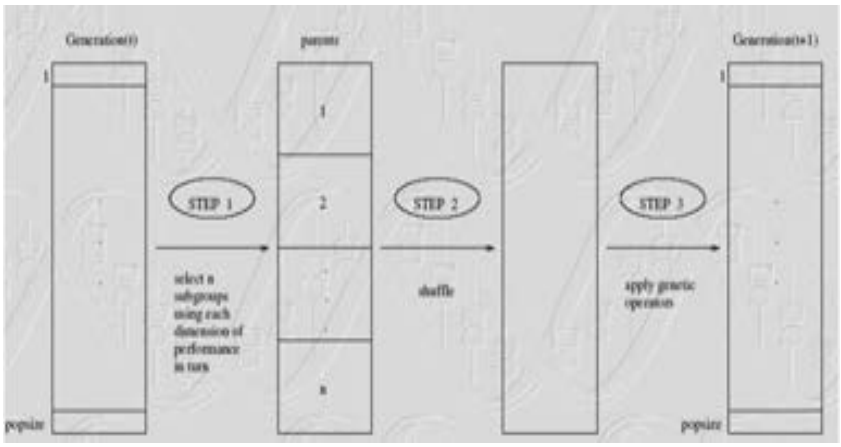


Fig of: VEGA multi-objective process

Schaffer tried VEGA on seven different functions taken from the literature of De Jong, Vincent, Grantham, etc. to see the VEGA's typical performance. Although this scheme is simple to implement, his experiments showed that it caused a bias against middling points (like in the previous example, B is a middling point which is good but not excellent).

Example:

To see VEGA's typical performance, let's look at schaffer's second function (F2).

This function is a two-valued function of a single parameter.

$$F_{21}(t) = t^2$$

$$F_{22}(t) = (t-2)^2$$

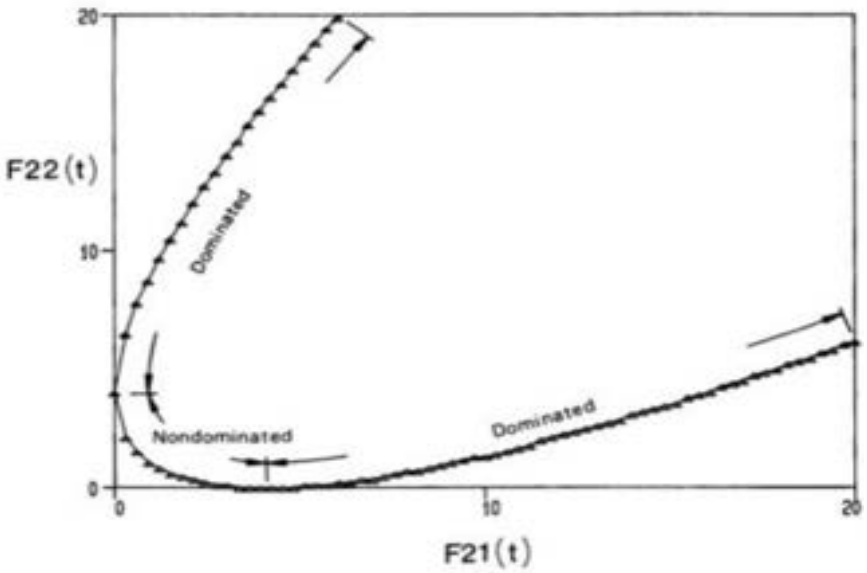


Fig of: Schaffer's second function (F2)

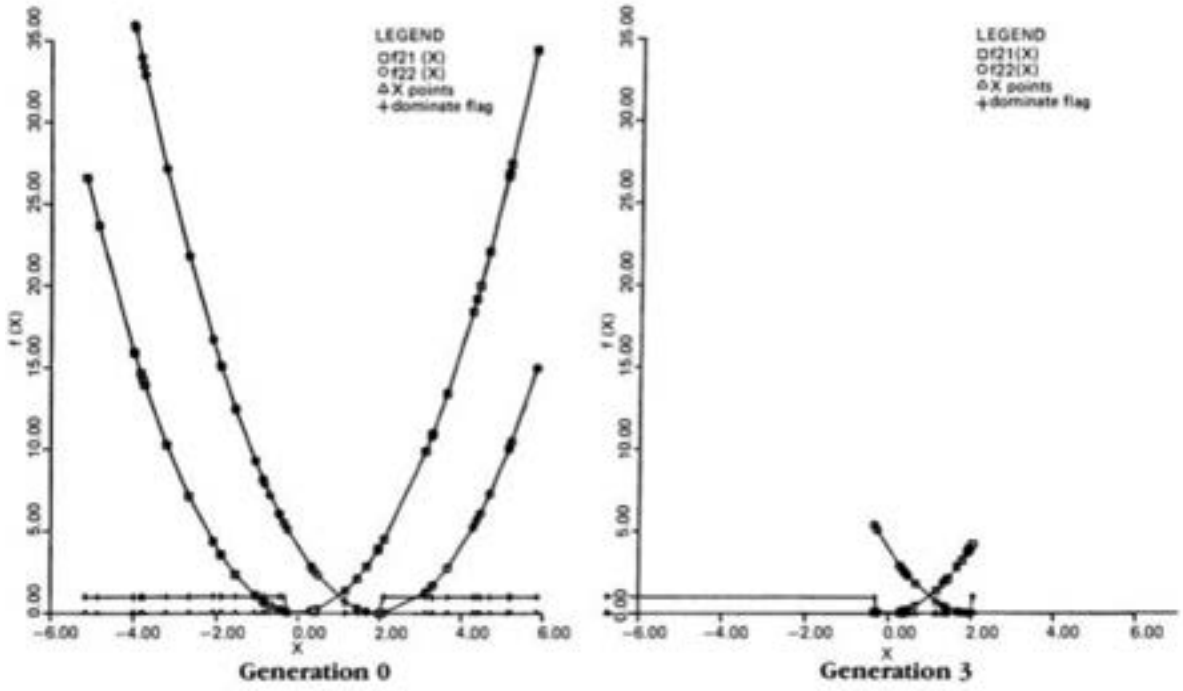


Fig: Comparison of generation 0 vs generation 3

Problem Generated:

There has been There has been some tendency to ignore the middling points which shows the bias against locally non-dominated individuals.

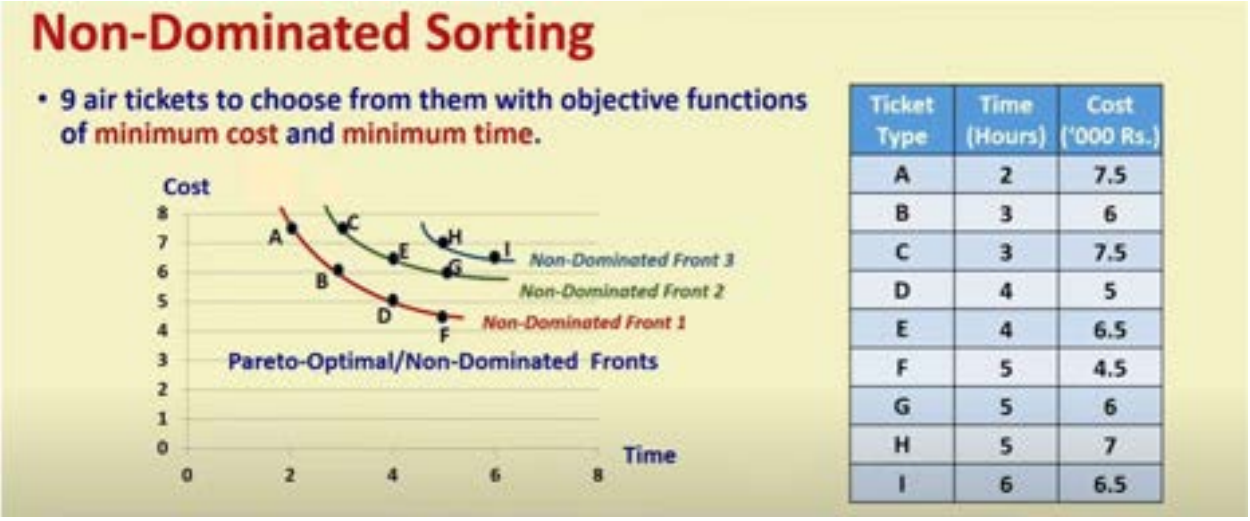
How to Optimized:

To overcome this problem, Rank Based fitness assignment on the basis of non-dominance is introduced which operates as follows:

1. All non-dominated individuals in the current population are identified and flagged.
2. These are placed at the top of the list and assigned a rank of 1 and have the highest fitness.
3. Solutions in the same front have the same rank and same fitness.
4. Now, these points are then removed from the list and the next set of non-dominated individuals is identified and assigned rank 2.
5. This process continues until the entire population is ranked.
6. Thereafter, reproduction count values or selection probabilities may be assigned according to rank.

This will help to maintain the diversity among the solutions. For better performance, it can be conjugated with the Niche Speciation technique which stabilizes the multiple subpopulations that arise along Pareto optimal front thus preventing excessive competition among distant population members.

Example:



Ticket Type	Time (Hours)	Cost ('000 Rs.)
A	2	7.5
B	3	6
C	3	7.5
D	4	5
E	4	6.5
F	5	4.5
G	5	6
H	5	7
I	6	6.5

Note: When two solutions have the same non-domination rank (belong to the same front), the one located in a less crowded region of the front is preferred.

Knowledge-Based Techniques:

In knowledge-based techniques, we use problem-specific knowledge or heuristics in GA. Unlike humans, GA is a blind search procedure that only exploits the coding and objective function value to determine plausible trials in the next generation. This is both a blessing and a curse. This may either gives GA its broad competence (a procedure that works well without any specific knowledge about the problem) or may put GA at a competitive disadvantage (not using all the knowledge about specific problem) with methods.

There are several knowledge-based techniques like:

- **Hybrid Approach**
- **Knowledge Augmented technique**
- **Approximate function evaluation methods**

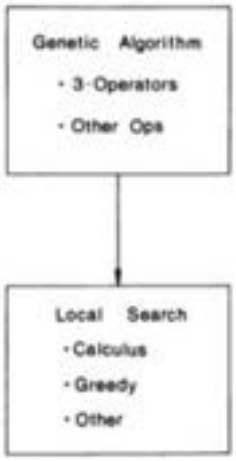
1. **Hybrid Approach**

   The simple concept of a hybrid approach is to create hybrid GA by crossing it with various problem-specific search techniques and local optimization techniques, thus overcoming the drawbacks of GA in terms of local search ability and premature convergence. Hybridization of GA can be done using various approaches like:

   - **Batch approach**
   - **Parallel approach**
   - **G-bit improvement approach**

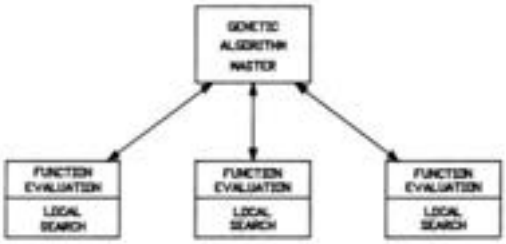
- **Batch Approach:**

The main idea of a batch approach is to allow GA to run to substantial convergence and then permit the local optimization procedure to take over which perhaps will search from the top 5% or 10% of points in the last generation. This is useful in maintaining diversity within the GA population and allowing stable sub-population.



- **Parallel Approach:**

In the parallel approach, function evaluations for different strings within a generation can be carried out simultaneously on numerous parallel processors. It performs fitness value computation and occasional iterations of the local search schemes to improve the current string.



- **G-Bit improvement:**

In the Gradientlike-Bitwise improvement technique, the similarity between changing single bits and gradient information (proposed by Goldberg) can be used to obtain a more general local search procedure that can be used regardless of coding or problem structure.

The steps in this approach include:

- Select one or more best strings from the current population
- Sweep bit by bit performing successive one-bit changes to string retaining the better of the last two alternatives
- At the end of the sweep, enter the best (or k-best) structures into the population and continue a normal genetic search

It converges to the best solution of a deterministic, bitwise linear function. The method is enhanced by keeping an explicit record of successful bits and determining whether to proceed or not on the basis of that record. G-bit improvement can also be extended to include all two- and three-bit experiments.

2. Knowledge-Augmented Approach

In this approach, problem-specific knowledge can be used to augment random choice in main operators like crossover, and mutation. This was applied in the crossover and tried in the Travelling Salesman Problem based on Greedy Heuristics (Grefensette, Gopal, Rosmaita, and Van). The authors examined the TSP solution representation in 3 ways: Ordinal representation, a path representation, and adjacency representation and concluded following points.

- Ordinal representation preserves tours (current order of the city in stack to be visited next) under crossover but a small change in the coding a cause massive reordering of a tour as a result meaningless building block (not good for GA).
- Other representation (path and adjacency representation) maintains more meaningful building blocks but simple crossover operating on either of these representations creates nontours.
- To solve this problem, they used an adjacency representation and a heuristic crossover (a greedy crossover) that constructs an offspring by choosing the better of two parental edges.

So, to solve this problem they used adjacency representation with greedy heuristic crossover (a greedy crossover) that constructs an offspring by choosing the better of two parental edges in which the shorter edge among the two adjacent edges in the parents is

chosen and continued until the tour is complete. If the shorter edge selection cause cycle, then extend the tour by a random edge. The authors generated good results on problems up to 200 cities with the proposed approach.

3. Approximate function evaluation methods

Problem Specific Knowledge allows us to construct approximate models of our problem. The model created can be more or less accurate approximation to our objective function.

- With GA, this knowledge can be used to reduce the number of full cost function evaluations at the cost of some error. Approximate function evaluation also helps in saving computational time and more evaluations can be performed in the same time.

#Example:

Approximate function evaluation technique in the image registration work (By Grefenstette and Fitzpatrick):

- The function evaluation was an accumulated pixel by pixel difference taken between two images, one before dye injection and one after dye injection of an artery.
- Initially, full function evaluation was expensive as the image consists of 100x100= 10,000 pixels.
- After a number of experiments, an approximate function evaluation based on a random sample of only 10 yields the best solution.

Also, to approximate an offspring's fitness, parents must be recognized and their model information should be used in the child's function evaluation. There are several ways to do so:

- Use the closest parent,
- Use the weighted average of parents and
- Use the most recently evaluated parents.

Thereafter, parents may pass on the Jacobian matrix to their offspring to propagate an approximation model as well as approximate fitness value. Other possibilities exist for

using population data to obtain a better approximate model than that obtained from the parents alone.

Tribhuvan University

Institute of Science and Technology

Report on

Concept of Genetic Based Machine Learning (GBML) along with classifier system and rule based system

Submitted to

Asst. Prof Mr. Ram Krishna Dahal

Central Department of Computer Science & Information Technology

Tribhuvan University, Kirtipur

Kathmandu, Nepal

Submitted by

Shrilata Wagle

(Roll no: 43/077)

Hari Lal Chalise

(Roll no: 57/077)

September, 2023

Discuss and present the concept of GBML (Genetic Based Machine Learning) along with a classifier system and rule-based system.

Genetic Algorithm Overview:

A genetic algorithm (GA) is an algorithm used to find approximate solutions to solve problems through application of the principles of evolutionary biology to computer science. Genetic algorithms use biologically-derived techniques such as inheritance, mutation, natural selection, and recombination. Genetic algorithms are a particular class of evolutionary algorithms.

Machine Learning Overview:

Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", to make predictions or decisions without being explicitly programmed. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field focusing on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Genetics Based Machine Learning

Genetics based machine learning (GBML) is the application of evolutionary algorithms (EAs) to machine learning. It describes a framework for GBML, which includes ways of classifying GBML algorithms and a discussion of the interaction between learning and evolution. Genetic Algorithms seem more humanlike a search mechanism than others we commonly encounter; they are speculative, seeking better alternatives through the juxtaposition of hunches; they are inductive, breathing fresh air into a world filled with ploddingly deductive procedures. Machine learning

systems that use genetic search as their primary discovery heuristic are described. Let us review the most common GBML architecture, the so-called classifier system.

Classifier System:

A CS obtains its information from an outside world by means of its detector and has a number of effectors that are used to select system action. The outside world is represented as an environmental object to be modified by application. An *animat* studied extensively by Wilson, is a good example to describe CS. An *animat* lives in a digital microcosm and is incarnated into the real world in the form of an autonomous robot vehicle with sensors/detectors (camera eyes, whiskers, etc.) and effectors (wheels, robot arms, etc.). A LCS algorithm acts as the "brain" of this vehicle, connecting the hardware parts with the software learning component.

The animat world is an artificially created digital world; for example, in Booker's Gofer system, a 2-dimensional grid that contains "food" and "poison" and the Gofer itself, which walks across this grid and tries to learn to distinguish between these two items, and survive well fed. Imagine a very simple animat: A simplified model of a frog, called "Kermit" living in little ponds—the environment—and having Eyes (i.e., sensorial input detectors) and Hands and legs (i.e. environment-manipulating effectors), which is a spicy-fly-detecting-and-eating device. A single "if-then" rule can be referred to as a "classifier" and encoded into binary strings. Similarly, a set of classifiers (rules) is called a "classifier population" and is used to GA-generate new classifiers from the current population.

A CFS starts from scratch, that is, without any knowledge, using a randomly generated classifier population, and we let the system learn its program by induction. This reduces the input stream to recurrent input patterns that must be repeated over and over again to enable the animat to classify its current situation/context and react on events appropriately. Kermit lives in its digital micro wilderness where it moves randomly. Whenever a small flying object appears that has no stripes, Kermit should eat it because it is very likely a spicy fly, or other flying insect. If the insect has stripes (possibly a wasp, hornet, or bee), then Kermit should leave it. If Kermit encounters a large, looming object, it immediately uses its effectors to jump away as far as possible. So, part of these behavior patterns within a specified virtual environment, in this case "pond," are referred to as a "frame" in AI.

In summary, a classifier system is a rule based problem solving machine learning system that learns through credit assignment (bucket brigade algorithm) and rule discovery (the genetic algorithm).

A classifier system includes:

• Production rules (classifiers)

• Working memory

• Input sensors (decoders)

• Outputs (effectors)

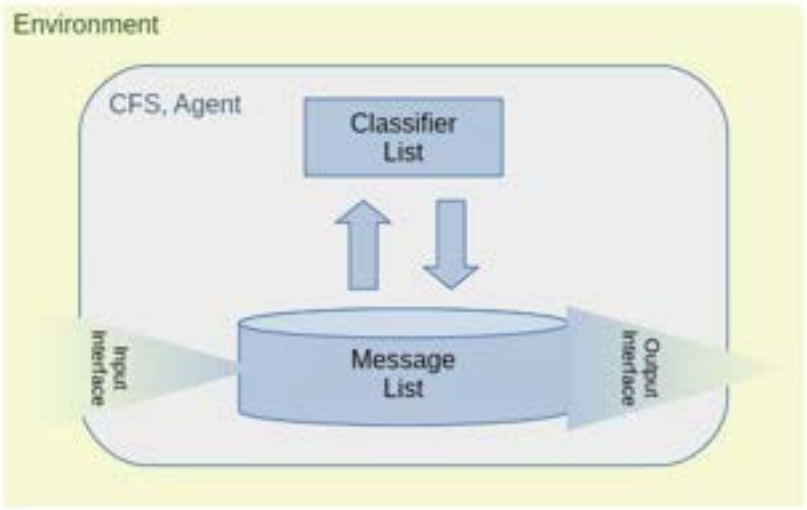
Requirements:

• an environment

• receptors/sensors

• effectors/actuators

• system itself



Components of Classifier System:

I. Rule and message system

II. Credit Assignment Algorithm

III. Genetic algorithm

I. Rule and Message System:

The rule and message system of a classifier system is a special kind of production system. The rules are generally of the following form:

if <condition> then <action>

The meaning of a production rule is that when the condition is satisfied then the action is taken (that means the rule is "fired").

Classifier list is a list of classifiers: *IF cond-1 AND cond-2 ... AND cond-N THEN action*

cond-1, cond-2, ... cond-N / action

(we use a shorter notation ',' means *AND*)

Rule based Classifier:

• Technique for collecting records using a collection of

IF….. THEN ……. Rules

• Dataset: Vertebrata

Name	**Body Temp. (BT)**	**Gives Birth (GB)**	**Aq. Creature**	**Aerial Creature**	**Legs**	**Hibernates**
Lemur	Warm Blooded (WB)	Y	N	N	Y	Y
Turtle	Cold Blooded (CB)	N	Semi Aquatic	N	Y	N
Shark	Cold Blooded (CB)	Y	Y	N	N	N

R1: (GB = N) ∧ (Aerial Creature = Y) -> Birds

R2: (GB = N) ∧ (Aq. Creature = Y) -> Fish

R3: (GB = Y) ∧ (Body Temp. = WB) -> Mammals

R4: (GB = N) ∧ (Aerial Creature = N) -> Reptiles

Let us consider the example of dataset: vertebrata to explain the basic concept of Rule based classifier. Here there are three animals that need to be classified to the category they best belong. Each of the animals have their respective attributes that describe their features. The attributes include body temperature, aquatic creature, aerial creature, legs, etc.

There are also rules defined in the system, which is the most important part because the classification of the animals into their category depends on these rules. Basically the rules checks the attributes of the animals with the conditions in the system and if the attributes of the animals match with that of the conditions in the system then the rule is fired and the animal is classified into the respective category of the animal. This is the simple example of the Rule Based Classifier.

Block Diagram of Classifier System

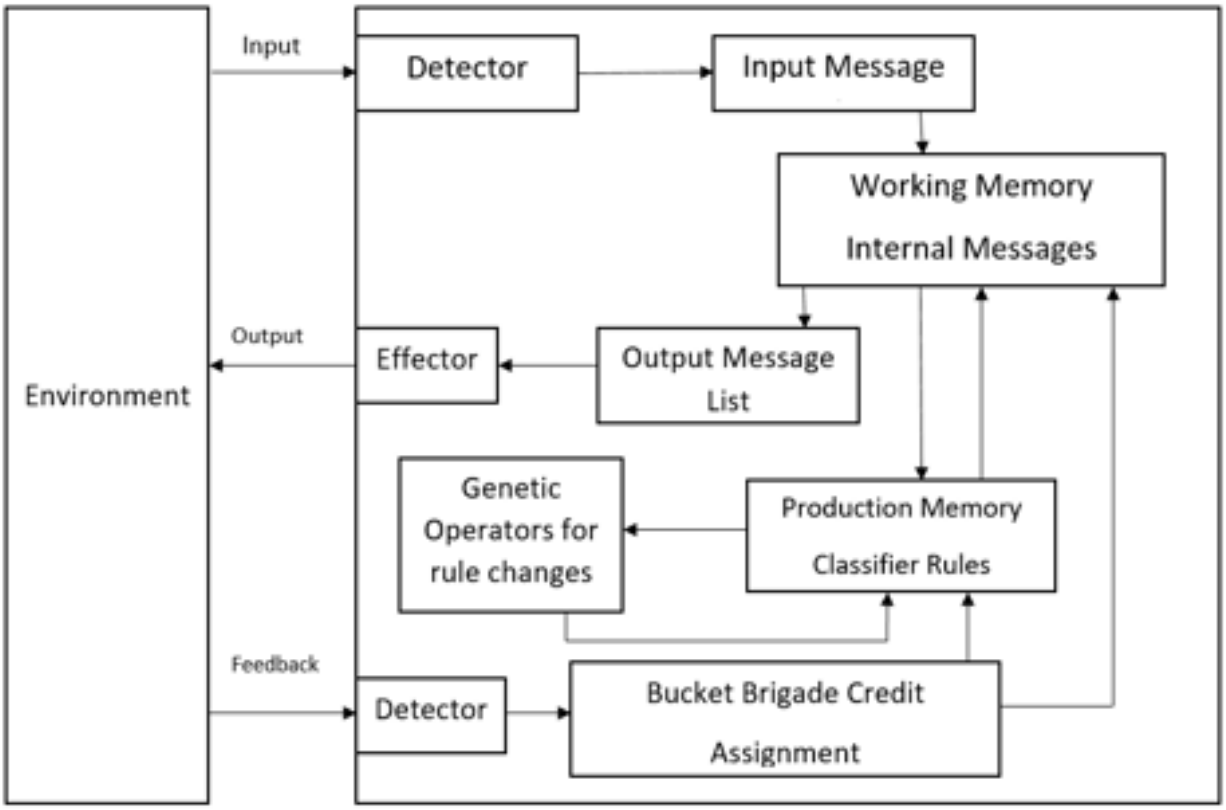


Figure 2 : Block Diagram of Classifier System

The above figure shows the block diagram of the Classifier system. The input message from the environment is passed to the system with the help of detectors. The input message is then passed to the working memory that temporarily holds the input message. The input message is then passed to the production memory that consists of the classifier rules which compares the input message from the environment with the condition of the system. If the input message from the environment matches with that of the conditions of the system then the rule is fired and the action of the respective rule is passed to the environment with the help of effector. The environment then decides whether the action is appropriate or not. The environment sends the message (as feedback) to the system with the help of the detectors. The feedback can be positive as well as negative. If the environment finds the action appropriate then the positive feedback (reward) is sent and if the action is not appropriate then negative feedback (punishment) is sent. The Bucket Brigade Credit Assignment then pass either reward or punishment to the production memory that modifies the rules of the classifier.

Classifier System Major Cycle

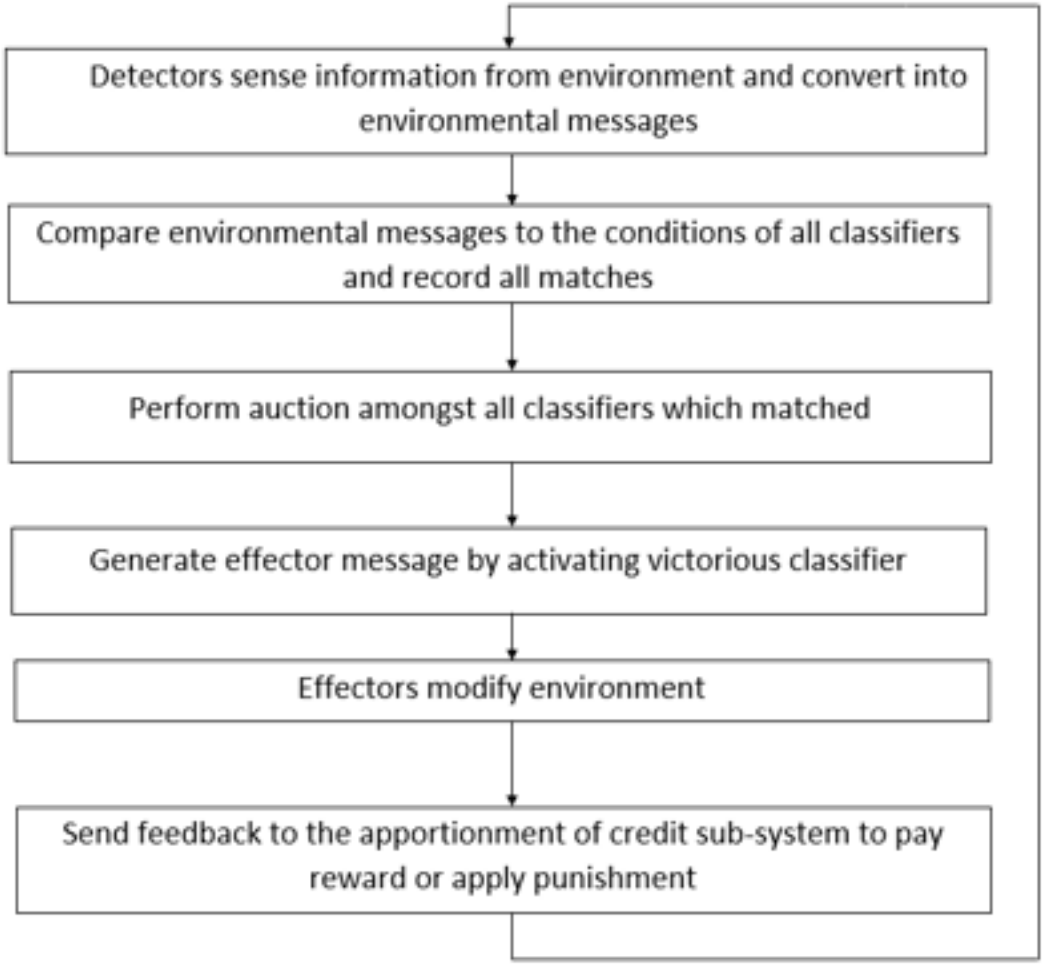


Figure 3: Classifier System Major Cycle

As shown in the above figure, the information through detectors is converted to the environmental messages. The environmental message is compared to the conditions of all the classifiers and all the matches are recorded. After then the auction amongst all classifiers that matched the condition takes place. The classifiers take part in auction through bidding process. The classifier having the highest bid is selected as victorious classifier. The effector message is generated by victorious classifier and the effectors modify the environment. The environment send feedback to the apportionment of credit subsystem to pay reward or apply punishment.

Auction in Classifier System

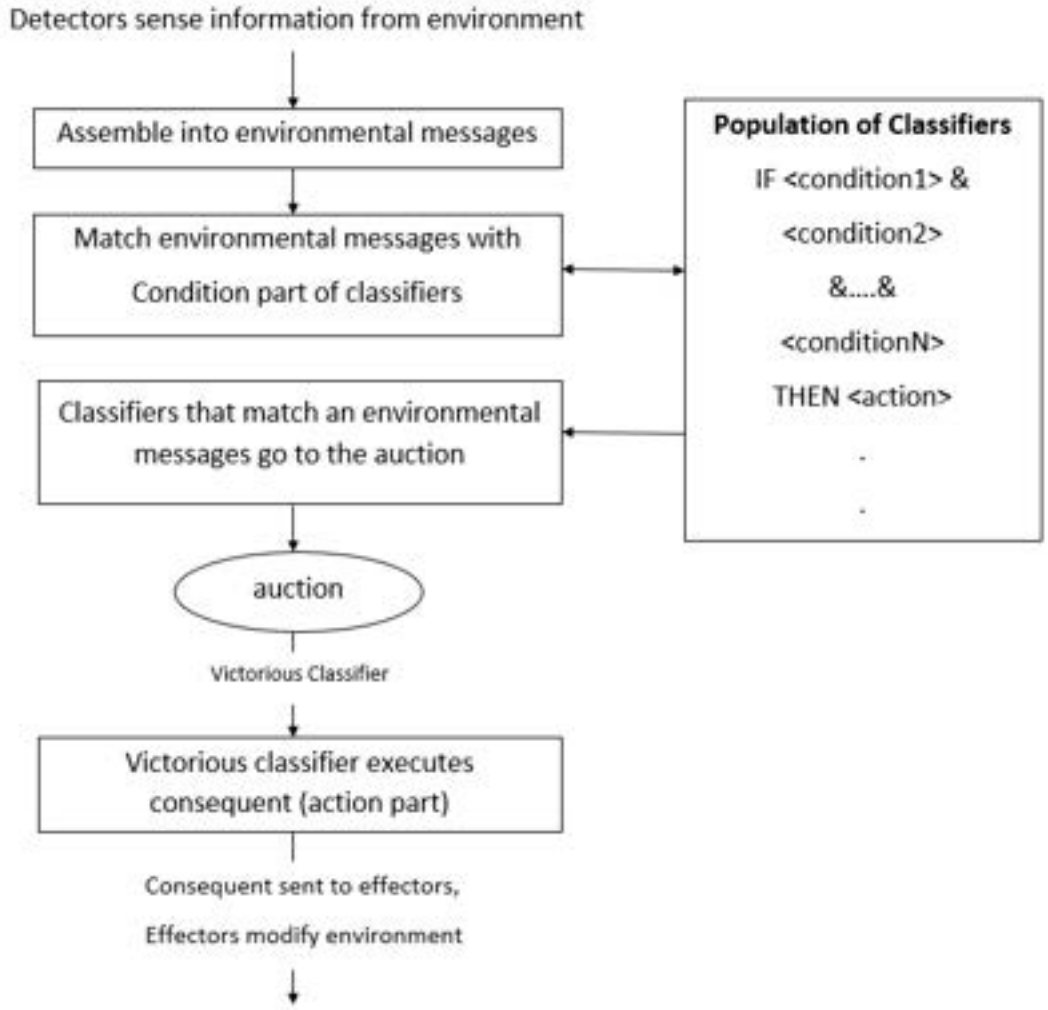


Figure 4: Auction in Classifier System

The detectors sense information from the environment and assemble into environmental messages. The environmental messages match with the condition part of the classifiers and the classifiers that match the environmental messages go to the auction. The victorious classifier is selected based on the bid. The classifier having the highest bid is selected which is termed as victorious classifier. After then the victorious classifier executes action through effectors. The action sent by the effectors modify the environment.

II. Credit Assignment Algorithm (Bucket Brigade Algorithm)

In this step, classifier systems implement a form of reinforcement learning that assigns credit and blame to rules during learning based on feedback. Feedback from the outside environment provides a means of evaluating the fitness of candidate classifiers. Credit assignment algorithm passes the reward or penalty back to classifier rules.

III. Genetic Algorithm (Classifier Discovery)

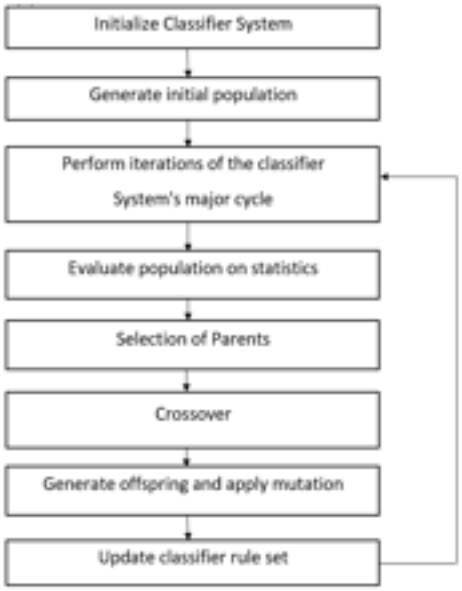


Figure 5: Genetic Algorithm (Classifier Discovery)

In classifier discovery step, the classifier system is initialized first and then the initial population is generated. After then the classifier's system major cycle is performed and the population statistics is evaluated. And then the selection of the parents is done based on the classifier's strength (bid). The crossover is done between the classifiers to generate the offspring and mutation is applied. After then the rule of the classifier set is updated and the iterations of the classifier's system major cycle is performed until the maximum iteration is reached.

Classifier System

Figure 6:Classifier System

Figure 6 shows the overall working of the classifier system. As mentioned earlier, the input is taken from the environment with the help of detectors and the environmental messages are matched with the condition of the classifiers; for this a number of classifiers takes part in the process. The classifier that take part in auction submits bids to the credit assignment system. The classifiers whose condition match with the environmental messages take part in the auction for the selection of the best classifier (also known as victorious classifier). The action of the victorious classifier is passed to the environment through effectors. After then the environment gives the feedback to the credit assignment system. The environment updates the strength of the classifier based on the original strength, bids and feedback. The classifier with the updated strength checks the environmental messages. During this process, Genetic Algorithm generates new set of classifiers with the help of crossover and mutation. And these set of classifiers take part in the auction followed by the bidding process. This process continues until maximum iteration is reached or the best classifier is discovered.

Learning Classifier System

Learning classifier systems, or LCS, are a paradigm of rule-based machine learning methods that combine a discovery component (e.g. typically a genetic algorithm) with a learning component (performing either supervised learning, reinforcement learning, or unsupervised learning). Learning classifier systems seek to identify a set of context-dependent rules that collectively store and apply knowledge in a piecewise manner in order to make predictions (e.g. behavior modeling, classification, data mining, regression, function approximation, or game strategy). This approach allows complex solution spaces to be broken up into smaller, simpler parts.

The architecture and components of a given learning classifier system can be quite variable. It is useful to think of an LCS as a machine consisting of several interacting components. Components may be added or removed, or existing components modified/exchanged to suit the demands of a given problem domain (like algorithmic building blocks) or to make the algorithm flexible enough to function in many different problem domains.

Basic Building Blocks of LCS

The three basic components to construct an LCS are:

- A rule and message system, which allows the machine to interact with its environment (accepting information from the environment and generating actions towards the environment).
- A reward mechanism to evaluate the rule set. This mechanism allows separating successful rules from unsuccessful or meaningless rules.
- A GA which is used to generate new (more optimal) rules for the rule set.

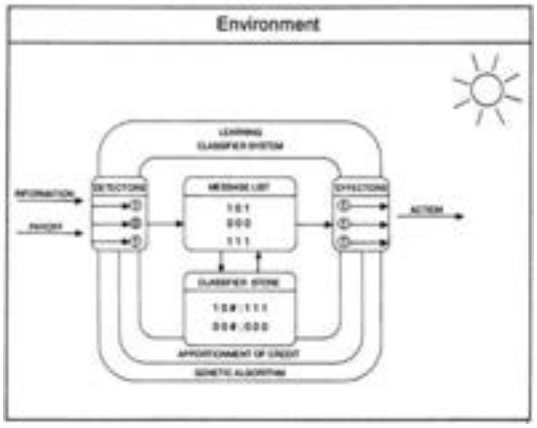


Fig: A Learning Classifier System interacts with its environment

A Rule and Message System

The message system is the interface of the LCS with its environment. Using detectors environment messages are imported into a *message list*. These messages are then interpreted by the rule set and finally an environment message (action) can be generated (using the effectors) to influence the environment.

A rule which accepts an environment message, can generate:

- An action (immediately sent to the environment)
- Or a new internal message which is sent to the (internal) message list

Using the second possibility (internal messages) a chain of internal messages can be created before a rule (finally) generates an environment action. The format of a rule is given by:

IF < condition > THEN < action >

The following definitions can be given when the messages and actions (together called 'classifiers') are binary coded:

< classifier>=<condition>:< message >

< condition >={0,1,#}k

< message >={0,1}k

where {0, 1}k or {0, 1,#}k is a concatenation of the symbols 0, 1 and 0, 1, # into a string of length k.

Suppose the following rule set (k = 4):

0##0 : 1 1 1 1

1#01 : 0 1 1 0

##10 : 0 1 0 0

#10# : 0 0 0 0

and an incoming message {1 0 1 0}.

This message corresponds only with the condition part of the third classifier which generates the internal message {0 1 0 0}. As both the condition parts of classifier (1) and classifier (4) correspond with this message, two new messages are generated: {1 1 1 1} and {0 0 0 0}. The message {0 0 0 0} will subsequently also generate the message {1 1 1 1} by means of classifier

(1). The resulting message {1 1 1 1} could be interpreted as an environment action (it is supposed here that a 1 on the least significant position indicates an environment action).

The Reward Mechanism

To separate successful classifiers from less successful (or unsuccessful) classifiers an evaluation mechanism needs to be implemented. To this end an evaluation parameter Si(t), called strength, is associated with each classifier i at a time t. The strength is modified by a certain reward mechanism, increasing the strength for successful classifiers and decreasing the strength for unsuccessful classifiers.

A well-known reward mechanism is the so-called Bucket Brigade Algorithm (BBA). However, other reward mechanisms have been developed and used successfully. The next paragraph will be limited to introducing the BBA. The BBA is constructed in analogy to a simple economical system where one or more fitting participants with a high bid are allowed a chance to make a 'profit' (this could also lead to a loss for the active participants).

In an LCS, the participants are classifiers and their 'financial' status is given by Si(t): In other words: when a certain message is presented to the message list, all classifiers with the corresponding condition part, can make a bid (for example proportionate to their strength). The highest bidding classifier(s) are activated and have their bid deducted from their strength. There are now two possibilities:

- The selected classifier generates an internal message for the message list of the LCS. A possible reward could come from the next cycle where a new bid is done by other classifiers based on the new generated message.
- The selected classifier generates an environment action which is immediately evaluated and possibly rewarded by this environment.
-

$$S_i(t+1) = S_i(t) - P_i(t) - T_i(t) + R_i(t) \quad (1)$$

with

$S_i(t)$ *the strength of a classifier i at time t*

$P_i(t)$ *the bid of a selected classifier i at time t*

$T_i(t)$ *the tax of a classifier i at time t*

$R_i(t)$ *the reward of the selected classifier i at time t*

and

$$P_i(t) = \alpha.Si(t) \quad (2)$$

$$Ti(t) = \beta.Si(t) \quad (3)$$

The tax component may be necessary to reduce the strength of classifier which are never activated (in other words to annihilate unnecessary or useless classifiers) and is always applied to each classifier regardless of bidding or activation.

Example:

0##0: 1111

1#01: 0110

##10: 0100

#10#: 0000

and an incoming message {1 0 1 0}

$\alpha = 0.1$,

$\beta = 0.0$ (no tax component), an environment reward of 20 for the action {1 1 1 1} $\forall_i$, $S_i(0)=100$

Table 1
Classifier strength using the BBA

i	$S_i(0)$	$P_i(0)$	$S_i(1)$	$P_i(1)$	$R_i(1)$	$S_i(2)$	$P_i(2)$	$R_i(2)$	$S_i(3)$
(1)	100		100	10	20	110	11	20	119
(2)	100		100			100			100
(3)	100	10	90		10 + 10	110			110
(4)	100		100	10		90		11	101

The generated results are shown in Table 1 for the first three time steps. Classifier (3) is activated by the environment message {1 0 1 0} and pays a bid of 10 (going towards the environment). The internal message {0 1 0 0} activates classifiers (1) and (4), each of them bidding an amount of 10. These bids will go to the classifier which generated the internal message in the previous cycle, resulting in a total strength of 110 for classifier (3). The message {1 1 1 1} generated by the classifier (1) is however an action which is immediately rewarded by the environment with an amount of 20, resulting in a strength of 110 for classifier (1).

The internal message generated by classifier (4) {0 0 0 0} activates classifier (1) again in the next cycle. Classifier (1) bids 11 (going to classifier (4) and receives immediately 20 from the environment. Classifier (1) finishes with the highest strength (after sending two successful actions

to the environment). Classifier (3) which started the chain of (successful) classifiers has the second highest strength. It should be noted that classifier (2) has never been activated and still remains at its original strength. The use of a tax component would have reduced the strength of classifier (2) during the 3 aforementioned cycles.

Adaptation by Rule Discovery

In addition to credit assignment, in order to learn we need a way to introduce new classifiers to the system. Evolutionary algorithm can be used to optimize and adapt a CFS in two ways:

- Considering the classifier list as a single individual whose chromosome is obtained by concatenating the conditions and actions of all classifiers (the Pittsburgh approach, DeJong)
- Considering each classifier as a separate individual (the "Michigan" approach, Holland)

In the Pittsburgh approach, the fitness of each CFS is determined by observing the behavior of the system for a certain amount of time or on some test data. The EA optimizes the CFS by breeding and making compete different sets of classifiers.

The Michigan approach requires a fitness measure for each classifier. If used with the bucket brigade algorithm, the strength of the classifier can be taken as its fitness. In this case, the EA optimizes the CFS by breeding and making compete and co-operate different classifiers.

Tribhuvan University

Institute of Science and Technology

Presentation Discussion Report on Genetic Algorithm

GA implementation techniques like -

- Asynchronous concurrent network,
- Object based model,
- Synchronous master slave and
- Semi-synchronous master slave models

Submitted By

Bed Prasad Pant (58/077)

Khagaraj Paneru (04/077)

2023

1] Parallelization Basics:

Parallelization is the act of designing a computer program or system to process data in parallel. Normally computer programs compute data serially. They solve one problem, and then the next, then the next. If such program is parallelized, it breaks a problem down into smaller pieces to be independently solved simultaneously by discrete computing resources.

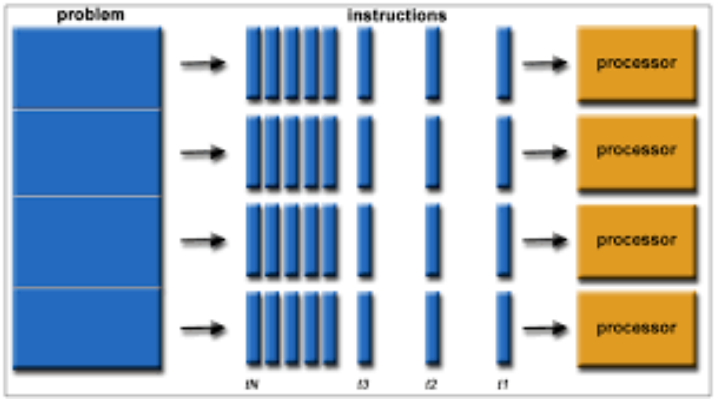


Figure 1: Breaking of task for parallel processing

In figure above we have a problem and we break this problem into smaller sub-problems and for executing each sub-problem we have separate processers. Note that each sub-problem may contain lots of statements within it and each statement may be executed sequentially by each assigned processor. But if we see this entire process globally we are successfully able to execute multiple sub-problems concurrently. This is the very basic concept for parallelization. Parallel processing of task is possible only if we are able to break the task into smaller subtask and each subtask can be executed concurrently. Similarly another requirement for parallelization is we must have a multiple computing resources like-

- Multiple processors
- Multiple cores
- Multithreading Operating System support
- Or combination of above

2] Communication Models:

In order to execute any task there is a simple requirement for multiple processing elements that we already have discussed in above section. Now if we require multiple processing elements for parallelizing the task, at the same time we also require some communication mechanism between each computing resource in order to maintain consistency and better coordination among them. There are basically two fundamental model for communication and these are as follows-

A] Shared Memory Model:

In shared memory model a common shared memory is established between cooperating processor. They can then exchange information by simply reading and writing data to the shared region.

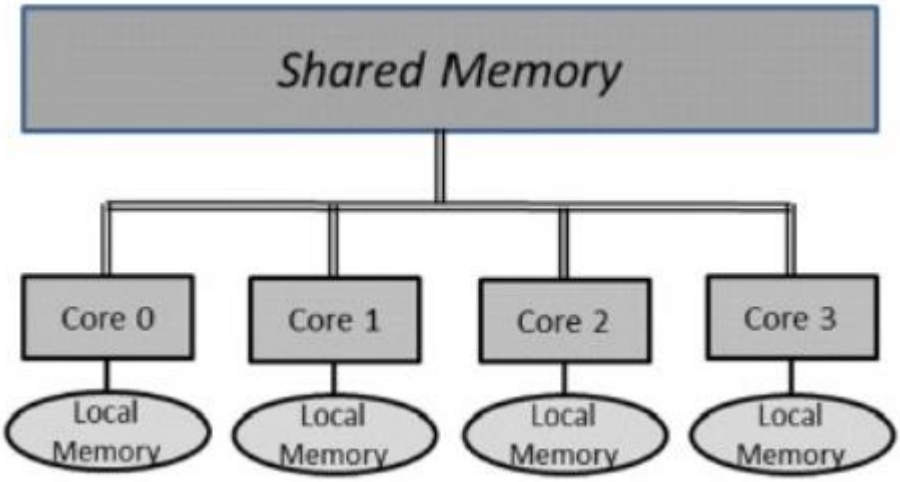


Figure 2: Shared memory model

In figure above it is clearly seen that we have 4 different cores each with its own local memory and they all are connected with a single common shared memory. This common memory is used for establishing communication between each cooperating cores. Let us suppose core 0 wants to communicate with core 2 then core 0 simply write its message in some specified memory location within the shared memory and in order to read this message core 2 will have to access the same location.

B] Message Passing Model:

In message passing model data is shared by sending and receiving messages between cooperating processes using system call. It is especially useful in distributed environment where communicating processes may reside on different network connected system. In this model each process has access to two function send( ) and receive( ) for sending and receiving the messages with in the network.

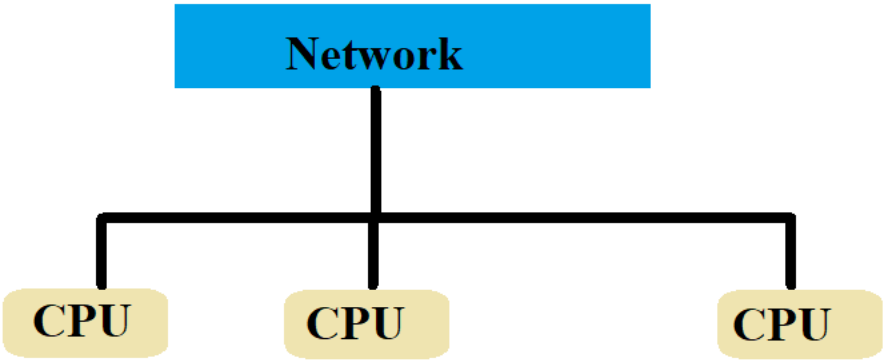


Figure 3: Message passing model

Again if the system is distributed over a multiple location and if it is a client server system then communication can be established by using sockets, Remote Procedure Call (RPC) and using pipe technology.

3] Parallel Implementation of GA:

Parallelization can be implemented in various ways, depending on the problem at hand, the available hardware and software resources and the desired level of parallelism. When we talk about the parallelization of task the most popular model that comes in mind is pipelining. In pipelining we divide our complete process or task in to multiple separate independent tasks and for each subtask separate processor is used. When talking about genetic algorithm we cannot use pipelining for parallel processing so easily. Because GA is a sequential algorithm, and each steps in GA are not completely decomposable. There exists some inter task dependences. So without proper coordination or proper control mechanism it is not easy for the parallel implementation of

GA. In context of GA we require some other parallelization strategies to enhance the optimization process. Here are common models to implement parallelization to genetic algorithm.

A] Synchronous master slave model:

Synchronous master-slave model is a parallelization technique used in GA to improve the efficiency of optimization process. It is based on a single master process and multiple slave processes. Master process coordinates and controls the overall execution, while the slave processes perform the computation in parallel.

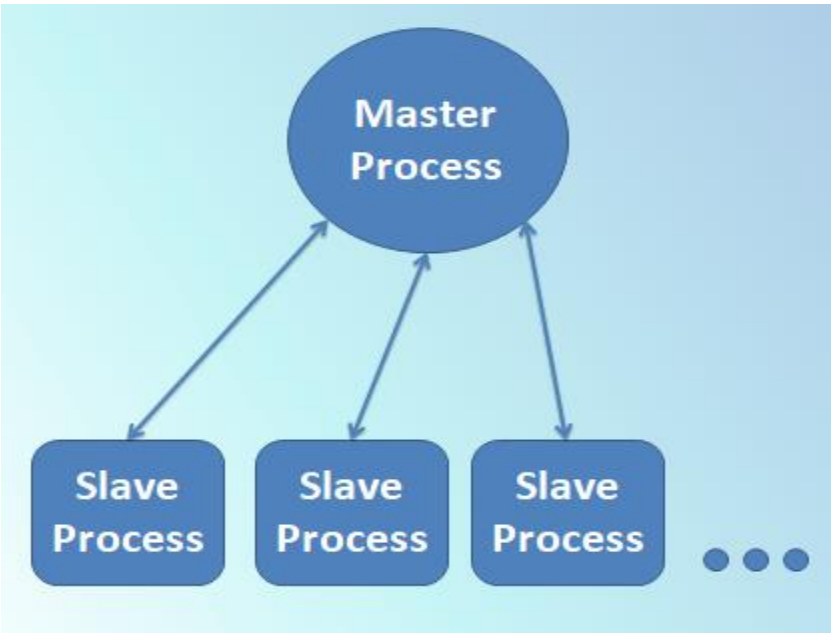


Figure 4: Synchronous master-slave model

Master Process:

Master process is responsible for managing the entire optimization process. It divides the complete task into a subtask and distributes the subtasks to the slave processes. It also monitors their progress and collects their results. It is responsible for the operations like control selection, mating and the performance.

Slave Process:

In master slave model multiple slave process works under the guidance of master process. Master process assigns subset of a population to each slave to work on. They are responsible for operation like fitness evaluation and other assigned by master.

In synchronous master-slave model, the algorithm progresses in synchronized steps-

- Master process initializes the population and divides it into subsets and assigns each subset to a slave process.
- At each generation, the master process sends instruction to the slave processes to perform operations on their assigned individuals.
- Slave process executes their tasks concurrently, following the masters instructions.
- Once all slave processes have completed their tasks, they synchronize and send back their result to master process.

Advantages:

1. **Parallelization:**
   Parallelization is achieved because fitness evaluation is performed concurrently by slaves.

2. **Controlled Coordination :**
   The master process ensures that tasks are assigned, coordinated, and executed in a controlled manner.

3. **Consistency :**
   All slave works on the same generation simultaneously maintaining a consistent state.

4. **Global operation :**
   All the global operations are handled by master process.

Limitations:

1. **Dependency:**
   Highly depends of master process. If master goes down the system halts.

2. **Waste of time:**
   Fair amount of time is wasted if there is much variance in time of function evaluation.

3. **Poor utilization of resources :**
   If master is busy then all the slaves have to wait.

4. **Restriction on parallelization**
   Parallelization is achieved only at certain level.

B] Semi-Synchronous Master Slave Model:

In order to solve the limitations of synchronous master slave model, another approach is purposed which combines the aspects of both synchronous and asynchronous execution. In this model the master process still plays a coordinating role, but it does not need to wait for all slave processes to complete their tasks before moving to the next generation. The master process can advance to the next generation as soon as a specified number of slave processes have finished their tasks.

Advantages:

1. **Increased Parallelism:**
   Slave processes can move to the next generation independently of other slave processes, allowing for increased parallelism compared to the fully synchronous model.

2. **Reduced Wait Time**
   Slave processes do not need to wait for slower processes to finish before moving forward, reducing idle time.

3. **Partial Synchronization**

   While the master process still coordinates, the synchronization requirements are less strict, enabling more concurrent execution.

Limitations:

1. **Dependency:**

   Highly depends of master process. If master goes down the system halts.

2. **Communication Complexity :**

   Communication become complex and require careful management of communication to ensure the data consistency.

   .

3. **Poor utilization of resources :**

   If master is busy then all the slaves have to wait.

C] Object Based Model:

In an object-based design procedure, two design models are considered: a community model and a plant pollination model.

Community Model:

In community model the genetic algorithm is mapped to a set of interconnected communities. The communities consist of a set of homes connected to the centralized, interconnected towns. Parents give birth to offspring in their homes and perform function evaluations there. The children are sent on to a centralized singles bar or in town where they meet up with prospective mates. After mating, the couples go to the town's real estate broker to find a home. Homes are auctioned off to competing couples. If town is currently crowded, the couples may also consult the broker about homes in other communities, and if necessary they may go to the bus station to move to another community.

Now, imagine that each community or town corresponds to a separate core or processor. Each core is responsible for evolving its own subset of the population within its community. Within each community, individuals are referred to as a home. These homes represent potential solutions. Parents give birth to offspring in their respective homes, reflecting the genetic operations (selection, crossover and mutation) taking place. Function evaluation determines fitness of individuals occur within the home. Each home evaluates its own residents. If community or core become overcrowded or individuals want to explore solution space, they can communicate with other core to consider the migration.

Plant Pollination Model:

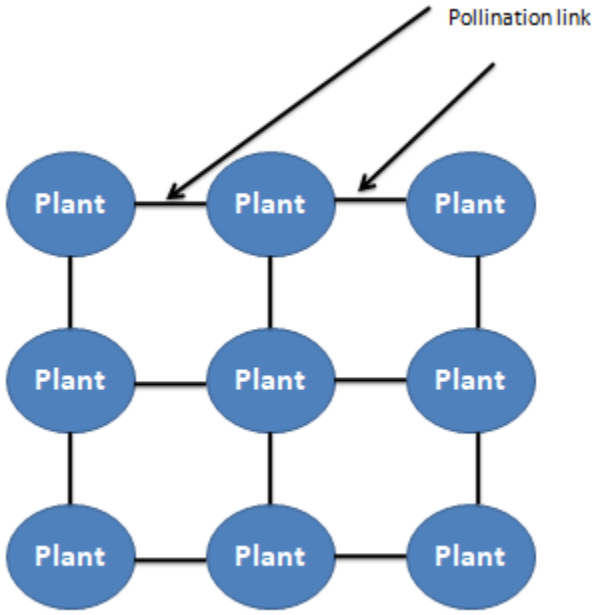


Figure 5: Plant pollination model

The **plant pollination model** consists of a series of plant nodes connected by a pollination network. Seeds grow up to become full plants that generate pollen that is cast out on the pollination network. Associated with each pollination network link is a probability of pollen transmission. This capability permits plants subpopulations to be more or less isolated from one another. Plant pollen fertilizes mature plants, creating more seed. Selection occurs locally by selecting the local best seeds to become mature plants in a probabilistic fashion.

Plant node:

- Pollination model is structured as a network of plant nodes.
- Each node represents a potential solution to the optimization problem.

Seed growth and pollen generation

- At the beginning of the optimization process, the population is initialized with a collection of seeds, each corresponding to potential solution to the optimization problem.
- Seeds within the plant nodes grow into mature plants.
- Once mature, these plants generate pollen, which carries genetic information that can contribute to new solution.

Network links and transmission probability

- Pollination network consists of links connecting plant nodes.
- Each link is associated with a probability of pollen transmission between connected nodes.
- This probability reflects the likelihood that genetic information from one solution is shared with another.

D] Asynchronous Concurrent Network Model:

Asynchronous Concurrent Model:

- K – Identical processors perform both genetic operations and function evaluations independently of one another, accessing a common shared memory.
- They don’t need to wait for other to finish their tasks before proceeding to the next steps.
- The shared memory requires that the processes avoid simultaneous hits on identical memory locations; otherwise, there are no further timing requirements for this configuration.

- On a shared-memory multiprocessor, the population could be stored in shared memory and each processor can read the individuals assigned to it and write the evaluation results back without any conflicts.
- While the units operate independently, some level of information sharing might occur among neighboring units to exchange potentially better solutions or genetic material.

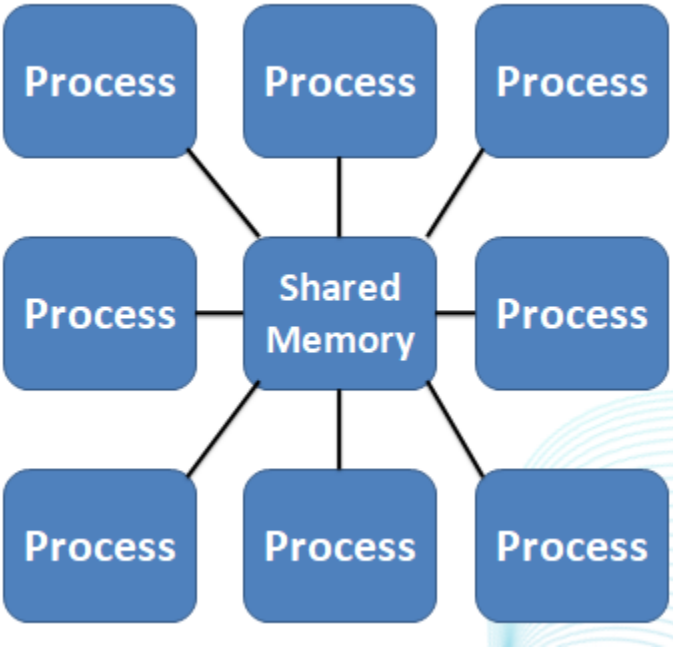


Figure 6: Asynchronous concurrent model

Network Model:

- K independent simple genetic algorithms run with independent memories, independent genetic operations and independent function evaluations.
- K processes work normally, with the exception that the best individuals discovered in a generation are broadcast to the other subpopulations over a communication network.

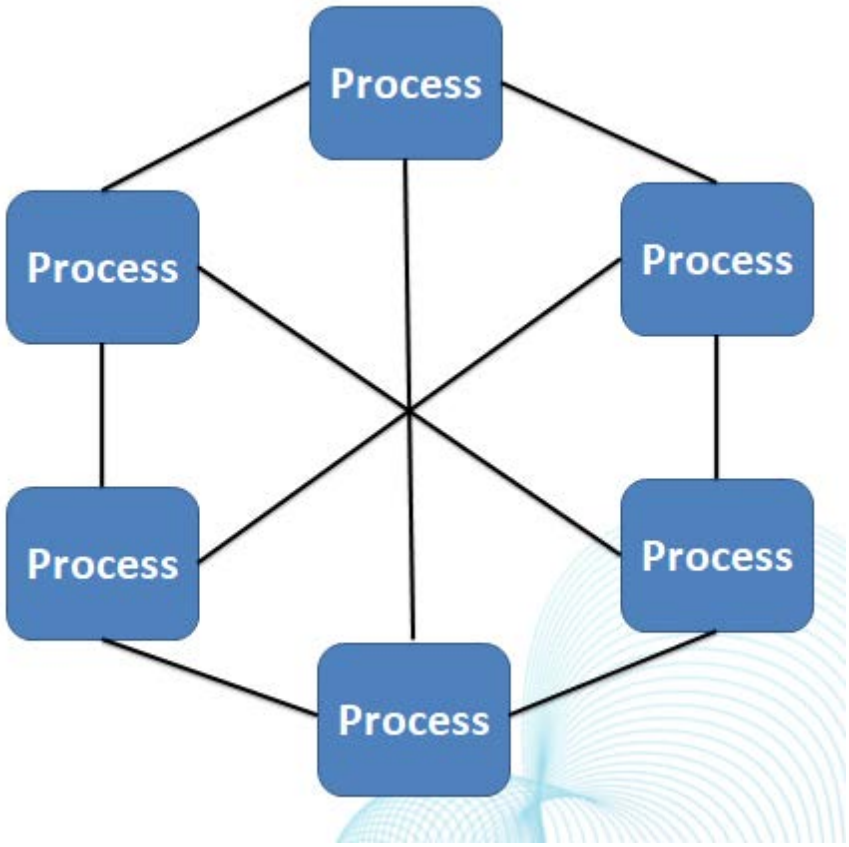


Figure 7: Network prototype

Tribhuvan University

Institute of Science and Technology

A Report On

"Application of GBML & Rise of GBML with development of CS-1"

Submitted to

Assistant Professor Ram Krishna Dahal

Central Department of Computer Science and Information Technology

Tribhuvan University, Kirtipur

Kathmandu, Nepal

Submitted By

Nivaja Ranjit

Class Roll No. 40/077

Saroj Shahi

Class Roll No. 52/077

(M.Sc. CSIT), 3rd Semester

Table of Contents

List of Figures

1. The Rise of GBML

The influence of biological analogy and metaphor was at an all-time high, the level of neural network activity, and the interest in cellular automata, and computational analogs of self-reproduction. A number of people began to investigate relationships between natural evolution and artificial adaptation.

1.1 Schemata and Their processors

Holland proposed the development of schemata processors in four phases. The prototypes increased in complexity from prototype I, a simple stimulus-response machine, to prototype IV, a complex automaton with internal states and modifiable detectors and effectors.

1.1.1 Prototype I

Stimulus-Response (SR) processor that would link environmental schemata (conditions) with particular action effectors.

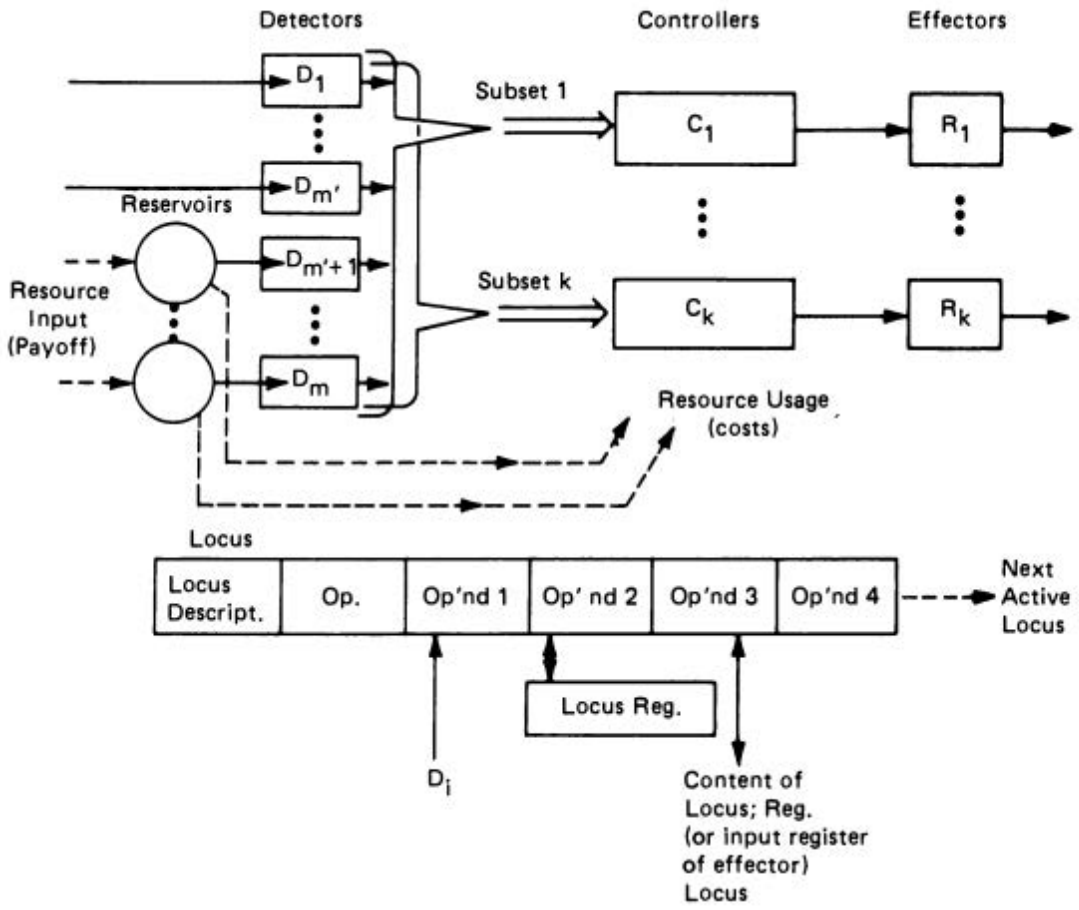


Figure 1 Prototype I: Stimulus-Response

1.1.2 Prototype II

Prototype II extended prototype I by adding internal effectors (internal states).

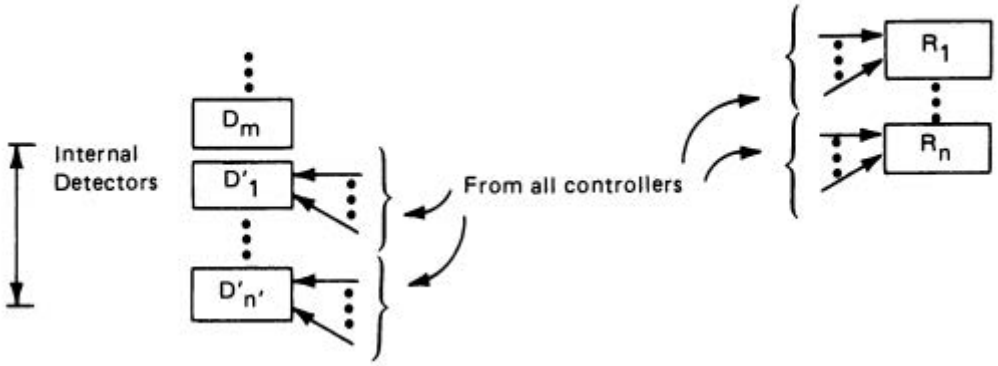


Figure 2 Prototype II: Internal Detectors

1.1.3 Prototype III

Prototype III is built upon prototype I and prototype II by including explicit environmental state prediction (a model of the real world) and an internal evaluation mechanism.

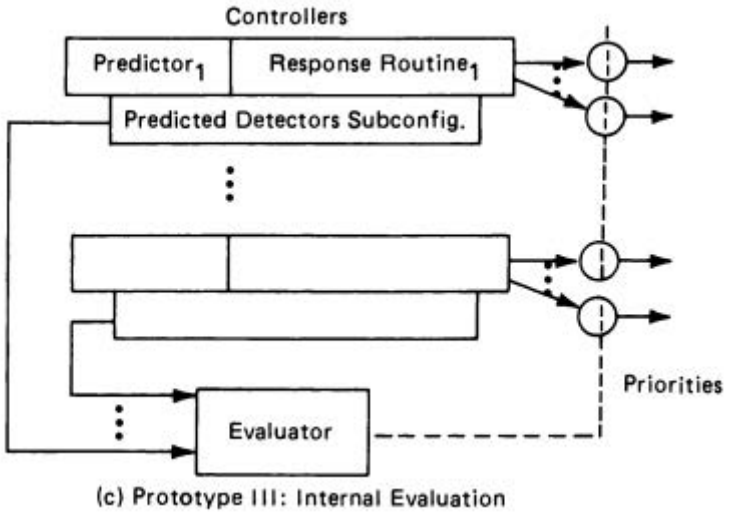


Figure 3 Prototype III: Internal Evaluation

1.1.4 Prototype IV

Prototype IV extended the other types by incorporating the capability to modify its own effectors and detectors, permitting greater (or lesser) range of data detection and a large behavioral response.

1.2 The Broadcast Language

Holland proposed a Post production system over a 10 letter alphabet called Broadcast Language. He suggested a creation of broadcast unit (String rules) with one or two antecedents (Condition) and a single consequent (action). The alphabet contains the following 10 characters:

Λ = {0,1, * , : , ◇, ▼, ▽, △, p, ’}

- Here, 0 & 1 are the basics symbol for specifying symbols
- The “*” is basic delimiter for broadcast units (BUs), Everything between a pairs of stars is to be interpreted as BU.
- The “:” is the intra BU punctual symbol, separating condition and action.

There are four types of Broadcast Units:

1. * I_1:I_2 If I_1 is matched, broadcast I_2.
2. * : I_1:I_2 If I_1 is not matched, broadcast I_2.
3. *I_1::I_2 If I_1 is matched, delete persistent I_2.
4. * I_1:I_2:I_3 If I_1 and I_2 matched, broadcast I_3

The I’s (I_1, I_2, I_3) are constructed from any characters of the alphabet where the other symbols are interpreted as follows:

- ◇ (diamond) Single character don’t care symbol (or terminating multiple character don’t care)
- ▽ (clear nabla) Initial or terminating multiple character don’t care symbol (ignored in other positions) with pass through
- ▼ (solid nabla) Same as ▽, permits concatenation on pass through
- △ (clear delta) Single character don’t care symbol with pass through
- p (letter p) A signal marked by a p persists for all time until deleted by a type 3 broadcast unit
- ’ (single quote) A single character preceded by a ’ is taken literally

2. Development of CS-1, the First Classifier System

Schemata Processor and The Broadcast Language, along with elements of theory of genetic algorithm, led to development of first classifier system- called as CS-1(Cognitive System).

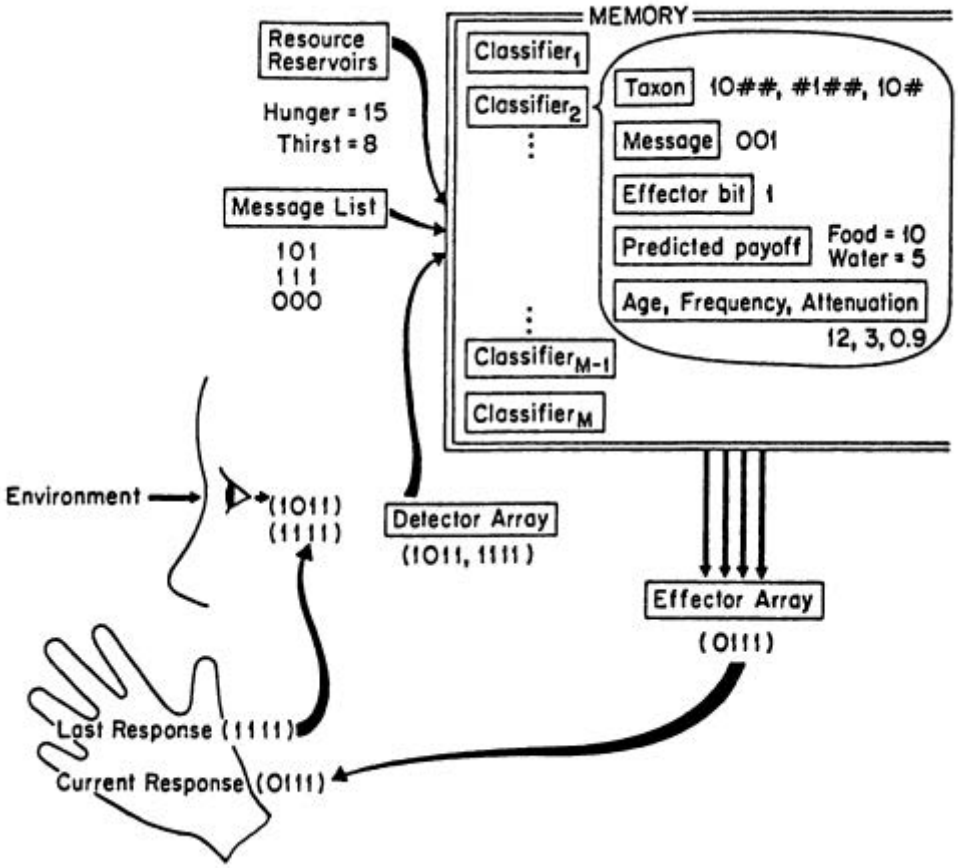


Figure 4 Schematic of Cognitive System One, the first classifier system

The system combined a performance system based on:

- Classifiers (simple string rules)
- Epochal Apportionment of credit system
- A genetic Algorithm (Similar to Generic Classifier System)

The figure above shows, an overall structure similar to that of the generic classifier system. A classifier store (the memory) contains a finite set of classifiers. The classifiers conditions, or taxa (plural for taxon) are constructed over the ternary alphabet {0,1,#} where the 0 and 1 are basic symbols and the # is a don't care character.

2.1 Segmentation of Classifiers

In CS-I conditions are segmented so that

- a portion pays attention to environmental signals,
- a portion pays attention to the last action and
- a portion pays attention to a separate internal message list.

This makes it different from the generic classifier system where all communications are posted as messages to the message list. The scheme advocated in the generic description is perhaps the more unified viewpoint. Also, a classifier in generic system seeks pay-off and that pay-off is distributed to classifiers through bucket-brigade algorithm.

2.2 Distribution of Pay-Offs

In CS-1, payoff and its distribution are handled somewhat differently.

- First, the system maintains separate reservoirs for a finite number of resources corresponding to a number of system needs: in the example schematic we see two resources, food (hunger) and water (thirst). These resource levels are depleted uniformly with time and must be replenished. In this way current resource levels are used to determine current demand, and these demand levels are then used in the decision process to determine which rules to activate.
- Second, CS-1 does not distribute payoff with a bucket brigade. Instead, an epochal algorithm is used. Here an epoch is defined as the time period between payoff events, and elaborate recordkeeping is performed to track a rule's usage and accuracy. The information is then used to revise a rule's decision parameters. To understand how this is done, we need to examine the parameters and their use.

The primary decision parameters for a CS-1 classifier are its predicted payoff values, the u values. CS-1 maintains a separate value for each resource relevant to the system (food and water in the example system). To determine which rule or rules to activate on a given cycle, CS-1 takes a classifier's predicted payoff values u_i and the current system demand values d_i (where the d function specifies an increasing demand level for decreasing resource reservoir level) and calculates an appropriateness value a for each classifier according to the following equation:

$$\alpha = \sum_i d_i u_i$$

Here the summation is taken over all resources i. The spin of a weighted roulette wheel determines the decision winner where the product of appropriateness α and match score M size the wheel slots (here match score is a measure that increases with increasing rule specificity).

As the matching and rule activation proceed, the epochal apportionment of credit algorithm system tracks the accuracy of a classifier's predicted payoff values through the use of three parameters: age, frequency and attenuation. The age parameter is incremented by 1 during each computational cycle; however, as the classifier receives reward (at the completion of an epoch), the classifier's age is reduced by an amount that increases with increasing rule usefulness.

The frequency parameter is incremented each time a rule is activated It is used in the weight adjustment scheme to give greater emphasis to more heavily used rules. The attenuation parameter is a number between 0 and 1. Initially a classifier's attenuation is set to 1.0. It is decreased whenever a rule has a predicted payoff value higher than that of the rules successor. An attenuation function as shown in figure 5 is used as a multiplier to decrease a rule's accumulated attenuation with increasing error. When payoff actually enters the system, it is distributed according to attenuation and frequency.

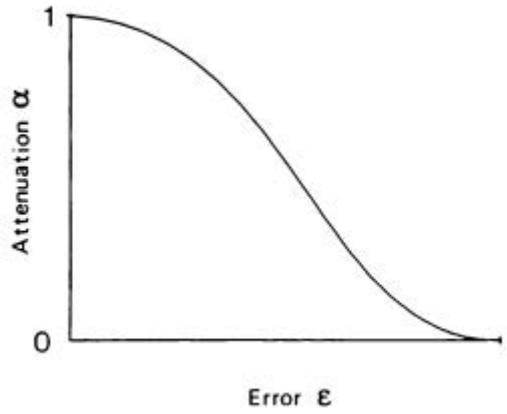


Figure 5 Attenuation Function as a function of negative error as used in CS-1

Therefore, the epochal apportionment of credit scheme continues from pay-off to pay-off adjusting predicted pay-off values to agree with actual payoff values.

2.3 CS-1 in Operation

In order to see the concepts of CS-I operationally, it was programmed in Fortran on an IBM 1800 at the University of Michigan.

The implementation contained the following limitations and simplifications:

1. Twenty-five positions per condition with eight bits for the environment, one bit for the last effector, and 16 bits for internal signals.
2. One effector with two settings, 0 and 1.
3. Two resources (needs) reservoirs.
4. Eight bit maze node names.
5. CS-1 was faced with the two maze-running tasks depicted schematically in figure 6.

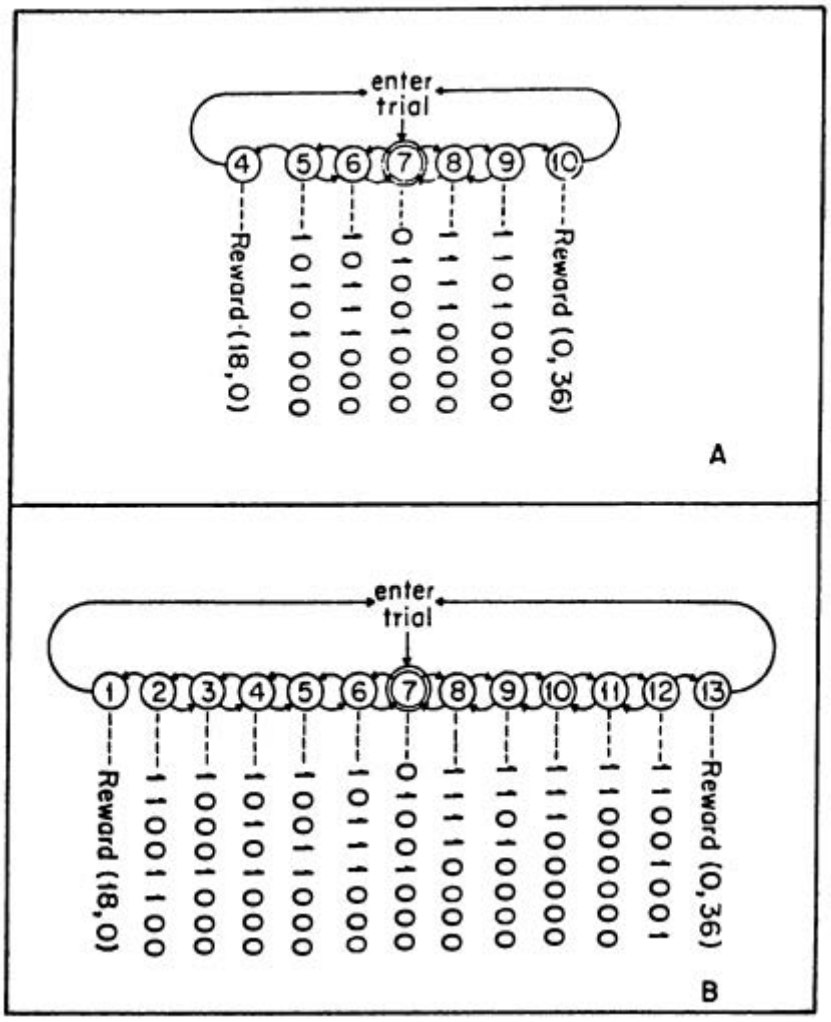


Figure 6 Schematic of (a) initial test environment, a seven-node maze and (b) the transfer test environment, a thirteen-node maze

In Figure 6 (a), we see a seven node maze where 18 units of food are available at the left end and 36 units of water are available at the right end. In Figure 6 (b), we see the transfer task that consists of the original seven-node maze with six additional nodes, three on each end.

2.4 Performance of CS-1

There are three cases of comparison for the performance of CS-1 on the two tasks depicted in figure 6.

- Random Walk
- CS-1 without Genetic Algorithm
- CS-1 with Genetic Algorithm

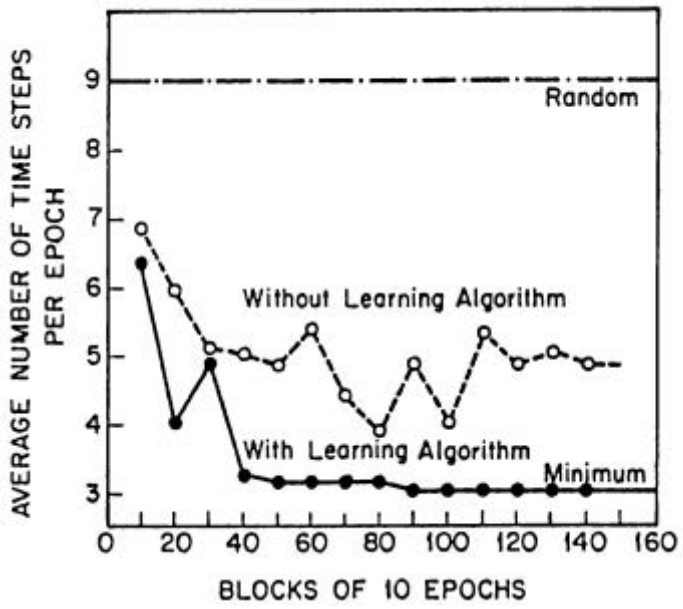


Figure 7 Performance of CS-1 in initial test environment (seven-node maze) with and without genetic algorithm

The following observations are seen:

- The run with GA outperforms than that without GA.
- Both (with GA and without GA) outperforms a random walk

As per the expectation, the lower payoff for food (18 as compared to water), to seek food twice, as it seeks water, was seen verified in the experiment.

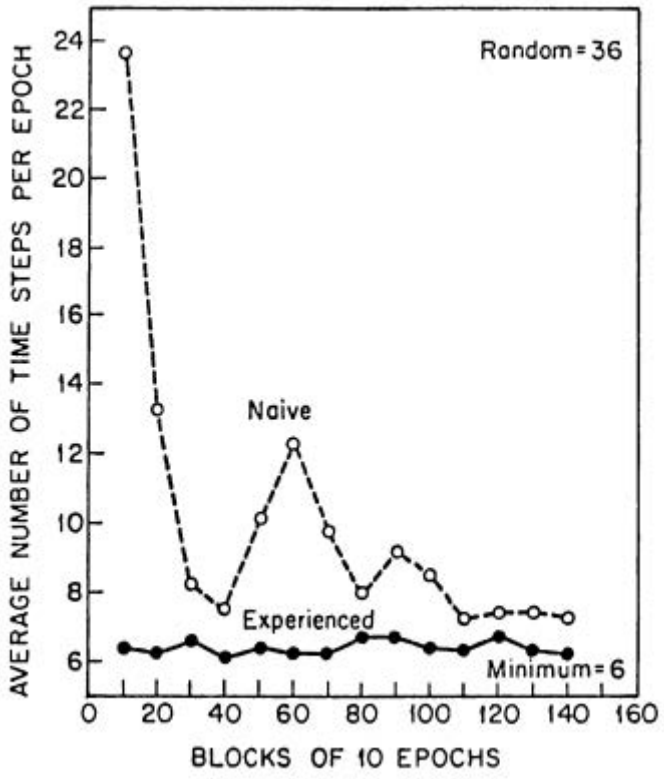


Figure 8 Comparison of naive versus experienced CS-1 on transfer test (thirteen-node maze) environment

In transfer experiment, we can see the significant difference in performance between a naïve run (random) and experienced run (CS-1 trained on 7-node maze first). In this task, the knowledge learned in the first set of experiments does indeed transfer to the more complex task as evidenced by the immediate convergence to near optimal performance in the experienced trace.

3. Applications of GBML

3.1 BOOLE: A Classifier System Learns a Difficult Boolean Function

Wilson developed a system called BOOLE that learns increasingly difficult multiplexer problems.

The 6-multiplexer in disjunctive normal form as follows:

$F6 = a'_0a'_1d_0 + a'_0a_1d_1 + a_0a'_1d_2 + a_0a_1d_3$

Where,

- Multiplication is a Boolean AND operation,
- Addition is a Boolean OR operation and
- Prime is a Boolean NOT operation.

This problem may be extended to larger multiplexer. In general, for k address lines there exist multiplexer with $k + 2^k$ lines. Each classifier is immediately rewarded (or not) as a result of its current action.

3.2 CL-ONE: Parallel Semantic Networks in a Classifier Framework

Forest has offered an existence of proof that classifier systems can emulate the complex models of symbolic AI. To understand the system (CL-ONE), we examine its overall structure. There are four major components:

- Parser and classifier generator
- Symbol table manager
- External command processor
- Classifier system

The classifier generator takes a description of a KL-ONE network and translates it into a set of classifiers. In so doing, it generates a symbol table for possible use in the future when new concepts are added to the network. A user query is translated into message form by the external command processor. This in turn is presented to the classifier system for computation.

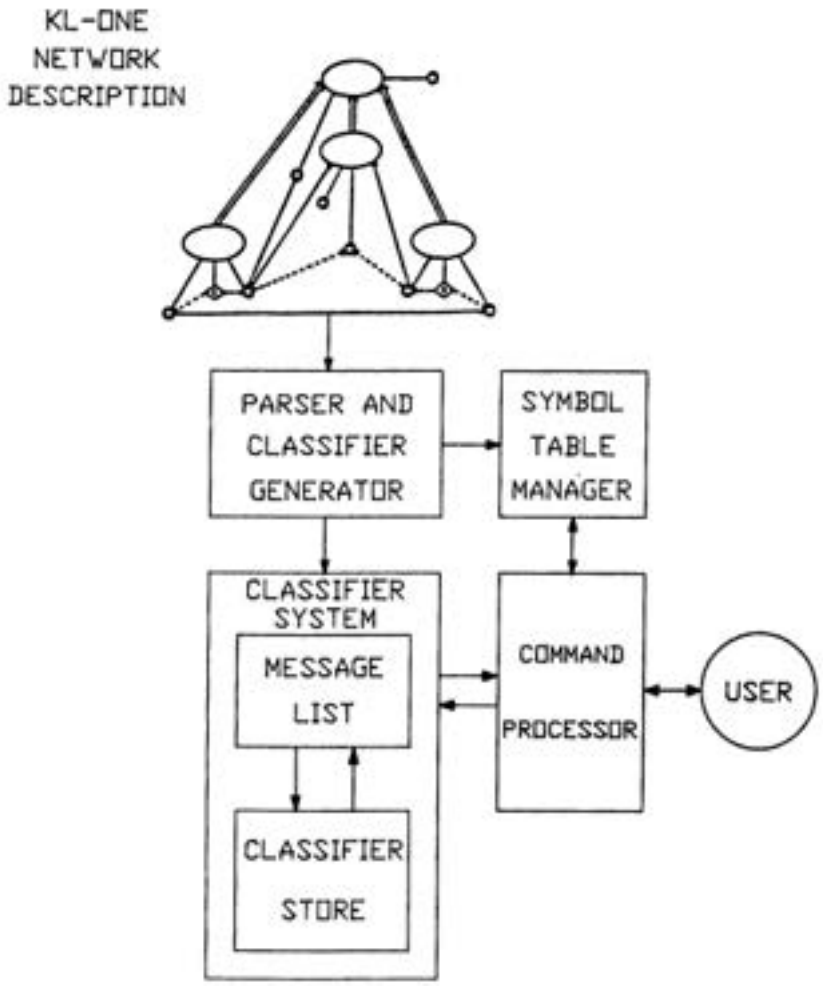


Figure 9 CL-ONE schematic shows interconnection of the classifier generator, symbol table manager, command processor and classifier system

3.3 Learning Simple Sequential Programs: JB and TB

Cramer's started form the Turing-equivalent language PL, removed go-to statements, and devised a simple language (PL --) for the calculation of primitive recursive functions. He devised two different coding schemes for the language, JB and TB. He applied modified reproduction and modified genetic operators in a search for a simple binary multiplication program. We examine the language, coding, the operators, and results of these experiments.

The PL− language has three primitives and two derived operations, as shown below in LISP-like notation:

```
1. (:INC VAR); Increment variable VAR by one (primitive).
2. (:ZERO VAR); Set the variable VAR to zero (primitive).
3. (:LOOP VAR STAT); Repeat statement STAT, VAR times
                     (primitive).
4. (:SET VAR1 VAR2); Assign VAR1 the value VAR2 (derived).
5. (:BLOCK STAT1 STAT2); Sequentially perform statements STAT1
                         and STAT2 (derived).
```

For example, to implement the multiplication expressed by the Pascal-like notation $V5 := V4 * V3$ we might write the following PL− program:

```
(:ZERO V5)
(:LOOP V3 (:LOOP V4 (:INC V5)))
```

The program works because the variable $V5$ is incremented by 1, $V4$ times, a total of $V3$ times, thus leaving the product of $V3$ and $V4$.

Crammer created a language coding called JB that took ordered triples of integer as instructions:

(x,y,z)

Where, x is the instruction code, y is the first operand and z is the second operand.

The five instruction assigned number are:

- :BLOCK = 0
- :LOOP = 1
- :SET = 2
- :ZERO = 3
- :INC = 4

Under JB, variable operands name the index of the desired variable and statement operands name ordered triple indexes. Furthermore, unnecessary operands and leftover integers are ignored.

For example, the program

```
(0 0 1 3 5 8 1 3 2 1 4 3 4 5 9 9 2)
```

is interpreted as five ordered-triples as follows:

```
(0 0 1); main statement → (:BLOCK STAT1 STAT2)
(3 5 8); statement 0    → (:ZERO V5); operand 8 ignored
(1 3 2); statement 1    → (:LOOP V3 STAT2)
(1 4 3); statement 2    → (:LOOP V4 STAT3)
(4 5 9); statement 3    → (:INC V5); operand 9 ignored;
                          leftover 9 and 2 ignored
```

Cramer implemented tree based coding that preserved the desired halting characteristics, called TB coding. TB uses parentheses to group instructions to any finite level of string. Multiplication program can be written in TB as follows:

(0 (3 5) (1 3 (1 4 (4 5))))

12. Discuss the impact of fitness function on reproduction, crossover and mutation. Highlight the issues of implementation of neuro-genetic algorithm with example.

Discuss the impact of fitness function on reproduction, crossover and mutation.

A key component of a genetic algorithm is the fitness function. Each member of the population is assessed using this method, and those that score the highest on fitness are more likely to be chosen for reproduction. This indicates that the genetic algorithm's capacity to identify effective solutions to the issue it is attempting to solve is directly influenced by the fitness function.

There are several ways in which the fitness function might influence the process of reproduction. For instance, the genetic algorithm might not be able to identify effective solutions to the problem if the fitness function is inaccurate. The genetic algorithm might not be able to function well if the fitness function is not achievable. Furthermore, it could be challenging for the user to comprehend how the genetic algorithm functions if the fitness function has no significance.

For example; if the fitness function is not accurate, then the genetic algorithm may not be able to find good solutions to the problem. If the fitness function is not feasible, then the genetic algorithm may not be able to run efficiently. And if the fitness function is not meaningful, then it may be difficult for the user to understand how the genetic algorithm is working.

This means that the fitness function has a direct impact on the genetic algorithm's ability to find good solutions to the problem it is trying to solve.

Impact of fitness function on reproduction are as:

- Selection Pressure: The selection of individuals for reproduction is influenced by the fitness function. Parents who score higher on fitness are more likely to be selected to raise the next generation. Fitter people are more likely to pass on their genetic makeup to the following generation, which creates a sort of selection pressure. The scale and selection technique of the fitness function determine how much pressure there is.
- Preservation of variety: The population's variety may be impacted by the fitness function selected. Premature convergence is the tendency for the population to converge to a suboptimal solution too soon if the fitness function is improperly designed. Conversely, a well-thought-out fitness function can promote diversity by awarding a variety of approaches.

Impact of fitness function on crossover are as:

- Exploitation vs. Exploration: The practice of giving preference to well-established effective solutions or promising regions within the solution space is known as "exploitation." In contrast, exploration refers to the process of conducting a thorough search throughout the solution space in order to find novel and maybe superior solutions. Exploration aids in locating globally optimal or nearly ideal solutions, whereas exploitation is helpful when you have a solid initial assumption. Creating a fitness function for optimization algorithms that strikes a balance between exploration and exploitation is a difficult issue. The decision is based on the particular problem, its features, and the optimization process' objectives.
- Parent Selection: Crossover people are chosen as parents based on their fitness function. Parents tend to favor fitter people, and this preference can facilitate more effective search space exploration. To encourage diversity, a well-designed fitness function should, however, also permit fewer fit people to add to the gene pool.

Impact of fitness function on mutation are as:

- Mutation Rate: The frequency with which individuals in the population undergo random modifications is dictated by the mutation rate. By influencing the convergence of the population, the fitness function can have an indirect impact on the rate of mutation. A greater mutation rate might be required to bring diversity

and break free from the local optimum if the fitness function favors the population in that direction.

- Mutation Strength: The size of mutations can also be influenced by the fitness function. For instance, in adaptive mutation strategies, an individual's fitness may be correlated with the strength of a mutation. Less fit people could be given stronger mutations to help them more efficiently explore the solution space.

Highlight the issues of implementation of neuro-genetic algorithm with example.

Neuro-genetic algorithm (NGA) is a type of genetic algorithm that uses neural networks to represent solutions to optimization problems. The neural networks are used to learn the relationship between the input and output variables of the problem, and the genetic algorithm is used to search for the best combination of weights and biases for the neural networks. Neural networks are a type of machine learning algorithm that are inspired by the structure of the human brain. They consist of a network of interconnected nodes, called neurons. Each neuron receives input from other neurons, processes that information, and then outputs a signal. The connections between neurons are weighted, and the weights are adjusted over time to learn the patterns in the data.

Neuro-genetic algorithm (NGA) can be used to solve a variety of optimization problems, including:

- Function optimization: Finding the minimum or maximum value of a function.
- Machine learning: Training a machine learning model to perform a task.
- Scheduling: Finding the best way to allocate resources to tasks.
- Pathfinding: Finding the shortest path between two points

Some issues of implementation of neuro-genetic algorithm are as:

1. Computational complexity: Neuro-genetic algorithms can be computationally expensive, especially for large problems. This is because they require repeated evaluation of the fitness function, which can be computationally expensive for complex problems.

Some ways to reduce the computational complexity of NGAs:

- Use a smaller population size.
- Use a fewer number of generations.
- Use a parallel implementation.
- Use a simpler fitness function.
- Use a more efficient representation of the solutions.
- Use a more efficient search algorithm.

2. Local minima: Neuro-genetic algorithms can be trapped in local minima, which are sub-optimal solutions. This is because they search for solutions by iteratively exploring the search space, and it is possible to get stuck in a local minimum that is not the global minimum.

Example;

Let's say we are trying to use a neuro-genetic algorithm to optimize the parameters of a neural network for a classification task. The neural network has 100 parameters, and we want to find the combination of parameters that gives the best classification accuracy. However, the search space is very large, and it is possible that the neuro-genetic algorithm could get trapped in a local minimum. This means that the algorithm might find a combination of parameters that is not the best, but it is the best that it can find in the local region that it is exploring.

Annealing is a method for escaping from local minima. It works by gradually reducing the temperature of the search, which makes it less likely that the algorithm will get stuck in a local minimum.

3. Parameter tuning: The performance of a neuro-genetic algorithm can be sensitive to the choice of parameters, such as the population size, mutation rate, and crossover rate. This can make it difficult to tune the algorithm to achieve good performance.

The parameters that need to be tuned include:

- Population size: A larger population size can lead to better solutions, but it can also be more computationally expensive.
- Number of generations: A larger number of generations can lead to better solutions, but it can also take longer to run the algorithm.
- Mutation rate: A higher mutation rate can lead to more diversity in the population, which can help the algorithm to find better solutions. However, a too high mutation rate can also lead to the algorithm getting stuck in local minima.

- Crossover rate: A higher crossover rate can help the algorithm to recombine good genes from different individuals, which can lead to better solutions. However, a too high crossover rate can also lead to the algorithm losing good genes.
- Selection method: The selection method is the way that the algorithm chooses the individuals to be used to create the next generation. There are a number of different selection methods, each with its own advantages and disadvantages. So, the user must be careful with the type of appropriate selection method for the NGA.

4. Data requirements: Neuro-genetic algorithms require a large amount of data to train. This is because they need to learn the relationship between the input and output variables. Genetic algorithms might inadvertently select neural networks that overfit the training data. Implementing techniques like regularization and early stopping is essential to mitigate this issue.