Outline

1. **Implementation of BGA**
   - Data structure ✓
   - Input ✓
   - Random initial population ✓
   - Fitness assignment ✓
   - Binary tournament selection ✓
   - Single-point crossover operator ✓
   - Bit-wise mutation operator ✓
   - Survival strategy ✓
2. **Implementation of RGA** ✓
   - Data structure ✓
   - Input ✓
   - Random initial population ✓
   - SBX crossover operator ✓
   - Polynomial mutation operator ✓
3. **Closure**

Implementation of BGA

Generalized Framework of EC Techniques

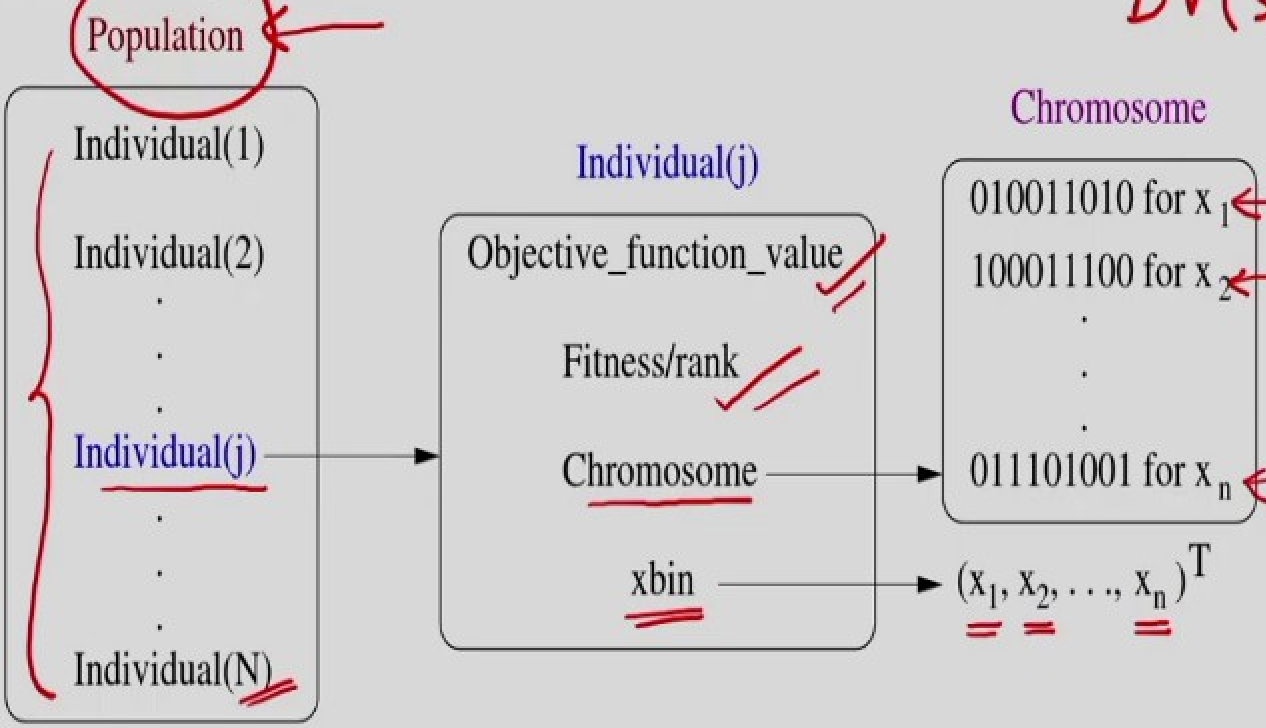
Algorithm 1 Generalized Framework for BGA

```
Solution representation                                          %binary string
Input: $t := 1$ (Generation counter), Maximum allowed generation $= T$
Initialize random population $(P(t))$;                           %Parent population
Evaluate $(P(t))$;          %Evaluate objective, constraints and assign fitness
while $t \leq T$ do
    $M(t) :=$ Selection$(P(t))$;                                 %Survival of the fittest
    $Q(t) :=$ Variation$(M(t))$;                                 %Crossover and mutation
    Evaluate $Q(t)$;                                             %Offspring population
    $P(t+1) :=$ Survivor$(P(t), Q(t))$;                          %Survival of the fittest
    $t := t + 1$;
end while
```

Data Structure for BGA

- Data Structure for population

DV(s)



Population

Individual(1)

Individual(2)

.
.
.

Individual(j)

.
.
.

Individual(N)

Individual(j)

Objective_function_value

Fitness/rank

Chromosome

xbin

Chromosome

010011010 for x_1

100011100 for x_2

.
.
.

011101001 for x_n

$(x_1, x_2, \ldots, x_n)^T$

- datatype **Population** parent_population, offspring_population;
  - parent.individual(j).objective_function_value;

Input to BGA

Algorithm 2 Input

```
Population size: $N$
Number of generations: $T$
Number of binary variables: $n$
for $(j = 1; j \leq n; j++)$ do                              %Each binary variable
    Binary string length: $l_j$
    Lower and upper bounds on $x_j$ that are $x_j^{(L)}$ and $x_j^{(U)}$
end for
Probability of crossover over: $p_c$
Probability of mutation: $p_m$
```

Initialize random population

Algorithm 3 Initialize random population

```
Input: $N$: population size, $n$: number of variables, $l_j$: binary string length of an individual($j$)
for ($i = 1; i \le N; i++$) do                      %Each individual in the population
    for ($j = 1; j \le n; j++$) do                  %Each variable of a solution
        for ($k = 1; k \le l_j; k++$) do            %Each bit of a variable $j$
            if ($random\_no \le 0.5$) then
                Assign 0
            else
                Assign 1
            end if
        end for                                     %Binary string for variable $j$ as 0 1 1 0 0 1
    end for                                         %Chromosome of all variables
end for
```

- Decode the binary string for each variable ($j \in \{1, \ldots, n\}$)
- Calculate real value (x_j) of each variable and store in the data-structure of individual

scaling formula

Evaluate Population

Algorithm 4 Evaluate Population

```
Input: $P(t)$: population, $N$: population size, $n$: number of variables
for ($j = 1; j \le N; j++$) do                                    %Each individual in the population
    Evaluate $f(x^{(j)})$        %Extract $x^{(j)} = (x_1, \ldots, x_n)^T$ from the data structure of an individual(j)
end for                                                  %Assign fitness same as the function value
```

- parent.individual(j).objective_function_value $= f(x_1, \ldots, x_n)$;
- parent.individual(j).fitness $=$ parent.individual(j).objective_function_value;

Selection Operator

Algorithm 5 Binary Tournament Selection Operator

```
Input: $x^{(1)}$: Individual 1 and $x^{(2)}$: Individual 2
if $(F(x^{(1)}) < F(x^{(2)}))$ then        %$F(x^{(i)})$ is the fitness of individual $i$. Extract this value from the data
    structure of an individual. We assume minimization of fitness.
    return($x^{(1)}$)                      %Individual 1 is selected.
else if $(F(x^{(1)}) > F(x^{(2)}))$ then
    return($x^{(2)}$)                      %Individual 2 is selected.
else
    if $(random\_no \leq 0.5)$ then
        return($x^{(1)}$)                  %Individual 1 is selected.
    else
        return($x^{(2)}$)                  %Individual 2 is selected.
    end if
end if
```

Crossover Operator

Algorithm 6 Single-point crossover operator

```
Input: parent-1, parent-2, offspring-1, offspring-2, $n$: number of binary variables
if ($random\_no \le p_c$) then
    for ($j = 1; j \le n; j++$) do                                  %Each variable of an individual
        $site = random\_no(1, l_j - 1)$                             %Random site for crossover
        for ($k = 1; k \le site; k++$) do                           %Each bit of a variable
            Copy $k-$th bit of parent-1 individual to $k-$th of offspring-1 individual
            Copy $k-$th bit of parent-2 individual to $k-$th of offspring-2 individual
        end for
        for ($k = site + 1; k \le l_j; k++$) do                     %Each bit of a variable
            Copy $k-$th bit of parent-1 individual to $k-$th of offspring-2 individual
            Copy $k-$th bit of parent-2 individual to $k-$th of offspring-1 individual
        end for
    end for
else
    Copy parent-1 binary string to offspring-1
    Copy parent-2 binary string to offspring-2
end if
```

1 2 3 4 5

1 0 0 1 1 0

Mutation

Algorithm 7 Bit-wise mutation operator

```
Input: offspring, $n$: number of binary variables
for ($j = 1; j \leq n; j++$) do                      %Each variable of an individual
    for ($k = 1; k \leq l_j; k++$) do                %Each bit of a variable
        if ($random\_no \leq p_m$) then
            if ($k -$ th bit is 0) then
                Mutate $k-$th bit of offspring to 1
            else
                Mutate $k-$th bit offspring to 0
            end if
        end if
    end for
end for
```

Survival

Algorithm 8 $(\mu + \lambda)$−strategy

```
Input: $P(t)$: parent population, $Q(t)$: offspring population
$C(t) = P(t) \cup Q(t)$                                   %combine both population
Sort $C(t)$ in an ascending order of fitness values       %Quick sort algorithm
Copy the first $N$ solutions from $C(t)$
```

Copy Solution

Algorithm 9 Copy solution

```
Input: individual 1, individual 2, $n$: no. of binary variables, $l_j$: string length of $j-$th binary variable
Copy objective function value of individual 1 to individual 2
Copy fitness/rank of individual 1 to individual 2
for $(j = 0; j \leq n; j++)$ do                                %For each variable of an individual
    for $(k = 0; k \leq l_j; k++)$ do                          %For each bit of a variable
        Copy $k-$th bit of individual 1 at $k-$th bit individual 2
    end for
    Copy $x_j$ of individual 1 to $x_j$ of individual 2        %Copy real value of a variable
end for
```

- Copy the complete data structure

Implementation of RGA

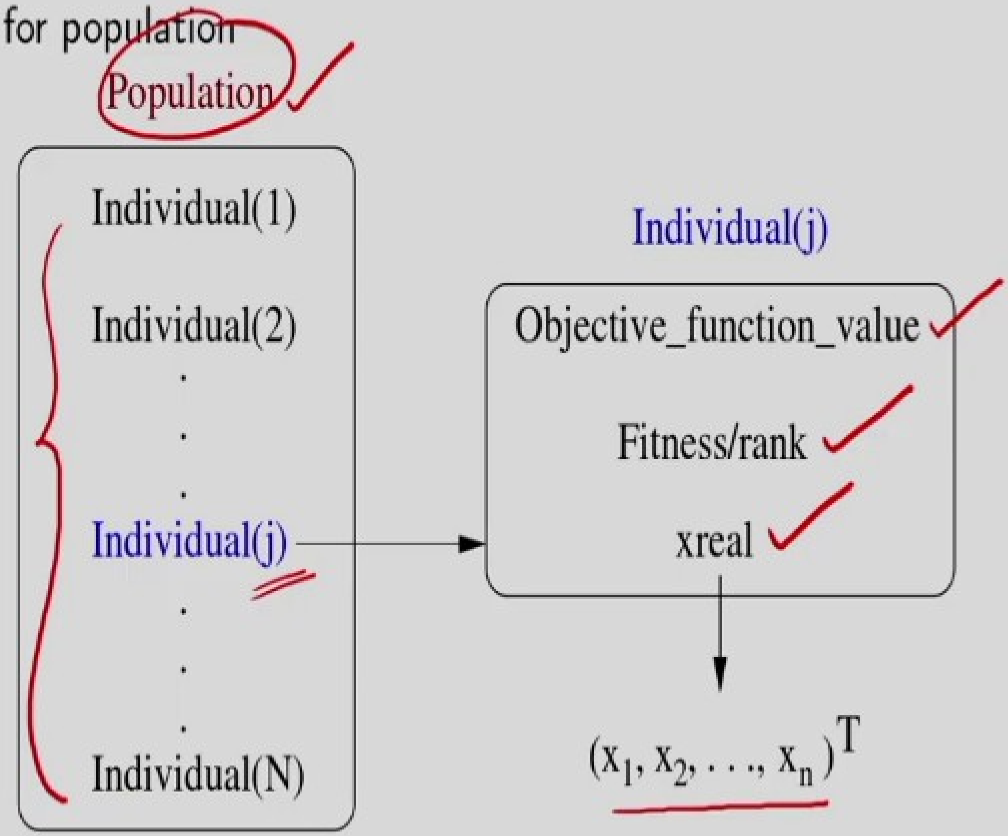
Generalized Framework of EC Techniques

Algorithm 10 Generalized Framework for RGA

```
Solution representation                                   %real number
Input for RGA;
Evaluate $(P(t))$;   %Call Algo. 4
while $t \le T$ do
    $M(t) :=$ Selection$(P(t))$;   %Call Algo. 5
    $Q(t) :=$ Variation$(M(t))$;                           %Crossover and mutation
    Evaluate $Q(t)$;   %Call Algo. 4
    $P(t+1) :=$ Survivor$(P(t), Q(t))$;   %Call Algo. 8
    $t := t + 1$;
end while
```

Data Structure for RGA

- Data Structure for population



Population

Individual(1)

Individual(2)

.

.

.

Individual(j)

.

.

.

Individual(N)

Individual(j)

Objective_function_value

Fitness/rank

xreal

$(x_1, x_2, \ldots, x_n)^T$

- datatype **Population** parent_population, offspring_population;
  - parent.individual(j).objective_function_value;

Input to RGA

Algorithm 11 Input

```
Population size: $N$ ✓
Number of generations: $T$ ✓
Number of real variables: $n$ ✓
for ($j = 1; j \leq n; j++$) do                                        %For each variable
    Lower and upper bounds on $x_j$ that are $x_j^{(L)}$ and $x_j^{(U)}$
end for
Probability of crossover over: $p_c$ ✓
Probability of mutation: $p_m$ ✓
In case of SBX crossover operator: $\eta_c$ ✓
In case of polynomial mutation operator: $\eta_m$ ✓
```

Initialize random population

Algorithm 12 Initialize random population

```
Input: $N$: population size, $n$: number of variables
for ($i = 1; i \leq N; i++$) do                                  %Each individual in the population
    for ($j = 1; j \leq n; j++$) do                              %Each variable of a solution
        $x_j$=Generate real number randomly between $x_j^{(L)}$ and $x_j^{(U)}$
    end for
end for
```

Crossover Operator

Algorithm 13 SBX crossover operator

```
Input: parent-1, parent-2, offspring-1, offspring-2, $n$: number of real variables
if ($random\_no \leq p_c$) then
    for ($j = 1; j \leq n; j++$) do                                        %Each variable of an individual
        Check $p_1 = x_j^{(1)} < x_j^{(2)} = p_2$. If not, interchange the values.
        Calculate $\beta^{(L)} = \frac{p_1+p_2-2x_i^{(L)}}{|p_2-p_1|}$ and $\beta^{(U)} = \frac{2x_i^{(U)}-p_1-p_2}{|p_2-p_1|}$ corresponding to the lower and upper bounds on $x_j$
        Generate random number ($u_1$) and calculate $\beta'_1$. Generate random number ($u_2$) and calculate $\beta'_2$
        Calculate offspring solutions as
            $(\text{offspring-1})_j = 0.5\left[(p_1+p_2) - \beta'_1(p_2-p_1)\right]$
            $(\text{offspring-2})_j = 0.5\left[(p_1+p_2) + \beta'_2(p_2-p_1)\right]$
        If $((\text{offspring})_j < x_j^{(L)})$, $(\text{offspring})_j = x_j^{(L)}$. If $((\text{offspring})_j > x_j^{(U)})$, $(\text{offspring})_j = x_j^{(U)}$
    end for
else
    Copy $(x_1, \ldots, x_n)^T$ of parent-1 to offspring-1
    Copy $(x_1, \ldots, x_n)^T$ of parent-2 to offspring-2
end if
```

Mutation

Algorithm 14 Polynomial mutation operator

```
Input: offspring, $n$: number of real variables
if ($random\_no \leq p_m$) then
    for ($j = 1; j \leq n; j++$) do                                %Each variable of an individual
        Generate random number $r_j$ and calculate
            $\bar{\delta}_j = \begin{cases} (2r_j)^{1/(\eta_m+1)} - 1, & \text{if } r_j < 0.5, \\ 1 - [2(1-r_j)]^{1/(\eta_m+1)}, & \text{if } r_j \geq 0.5. \end{cases}$
        Mutate offspring as
            $(\text{offspring})_j = (\text{offspring})_j + (x_j^{(U)} - x_j^{(L)})\bar{\delta}_j$
        If ($(\text{offspring})_j < x_j^{(L)}$), $(\text{offspring})_j = x_j^{(L)}$. If ($(\text{offspring})_j > x_j^{(U)}$), $(\text{offspring})_j = x_j^{(U)}$
    end for
end if
```

Copy Solution

Algorithm 15 Copy solution

```
**Input**: individual 1, individual 2, $n$: no. of real variables, $N$: population size
Copy objective function value of individual 1 to individual 2
Copy fitness/rank of individual 1 to individual 2
**for** $(j = 0; j \leq n; j++)$ **do**                %For each variable of an individual
    Copy $x_j$ of individual 1 to $x_j$ of individual 2    %Copy real value of a variable
**end for**
```

- Copy the complete data structure

Closure

- Implementation of BGA
  - Data structure for BGA
  - Input to BGA
  - Random initial population
  - Fitness evaluation
  - Binary tournament selection
  - Single-point crossover operator
  - Bit-wise mutation operator
  - Survivor strategy

RGA

- Implementation of ~~B~~GA
  - ✓ Data structure for RGA
  - ✓ Input to BGA
  - ✓ Random initial population
  - ✓ SBX crossover operator
  - ✓ Polynomial mutation operator
  - Fitness evaluation, Binary tournament selection and Survivor strategy will remain the same as with BGA.

Closure

- Implementation of BGA
  - Data structure for BGA
  - Input to BGA
  - Random initial population
  - Fitness evaluation
  - Binary tournament selection
  - Single-point crossover operator
  - Bit-wise mutation operator
  - Survivor strategy

- Implementation of BGA
  - Data structure for RGA
  - Input to BGA
  - Random initial population
  - SBX crossover operator
  - Polynomial mutation operator
  - Fitness evaluation, Binary tournament selection and Survivor strategy will remain the same as with BGA.

- One of the implementations was discussed.
- Independent of programming language: c/c++, java, matlab, python, etc.