|  |
| --- |
| Task 1 |
| Genetic Algorithms |
| Dr: Ahmed Elngar |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Team Members   |  |  | | --- | --- | | - Mohamed Abdelrahman Aboelkassem | 180113 | | - Mohamed Gamal Yaseen | 180104 | | - Mohamed Essam Abdelmoneem | 180114 | | - Nourhan Mahmoud Hussein | 180146 | | - Ola Abdallah Mogawer | 180087 | |

* **Q1: How to generate initial population in genetic algorithm**

There are two primary methods to initialize a population in a GA. They are:

**Random Initialization:** Populate the initial population with completely random solutions through parameters called Parameter Representation Which are involved in this optimization problems to get handled by GA Framework.

**Heuristic initialization:** Populate the initial population using a known heuristic for the problem.

* When the entire population is initialized using Heuristic initialization, it can result in the population having similar solutions and very little diversity.
* It has been experimentally observed that the random solutions are the ones to drive the population to optimality. Therefore, with heuristic initialization, we just seed the population with a couple of good solutions, filling up the rest with random solutions rather than filling the entire population with heuristic based solutions.
* It is the diversity of the solutions which lead to optimality.
* Heuristic initialization effects the initial fitness of the population.
* **Q2: Compare Between Different GA Strategies (SGA, SSGA, MGA)**

1. **Simple Genetic Algorithm (SGA):**This model demonstrates the use of a genetic algorithm on a very simple problem. Genetic algorithms (GAs) are a biologically-inspired computer science technique that combine notions from Mendelian genetics and Darwinian evolution to search for good solutions to problems (including difficult problems). The GA works by generating a random population of solutions to a problem, evaluating those solutions and then using cloning, recombination and mutation to create new solutions to the problem.

**How it works:**

1. **initial population creation**

* Here one parameter called the GA parameter(N).so, initial population means it is a collection of solutions which is the size
* The programmer who decides the size
* If we follow large values of N may be that we can come into the butter solution, but at the cost of timing and if it is taking the population size less that means value of N is small then we may com quickly terminate the solution, but it may not give you the correct result always.so, population size is an important parameter

1. **evaluation each individuals**

Each solution is evaluated on the basis of how well it solves the problem. This measure of the "goodness" of the solution is called its "fitness". In this model, our goal is simply to find a solution

1. **A new generation of solutions is created from the old generation**, where solutions that have a higher fitness scores are more likely to be chosen as "parent" solutions than those that have low fitness scores.

* The selection method used in this model is called "tournament selection", with a tournament size of 3. This means that 3 solutions are drawn randomly from the old generation, and the one with the highest fitness is chosen to become a parent.
* Either one or two parents are chosen to create children. With one parent, the child is a clone or copy of the parent. With two parents, the process is the digital analog of sexual recombination -- the two children inherit part of their genetic material from one parent and part from the other.
* There is also a chance that mutation will occur, and some of the child's bits will be changed from "1"s to "0"s

1. Steps 2 and 3 above are **repeated** until a solution is found that successfully solves the problem

**Flowchart**

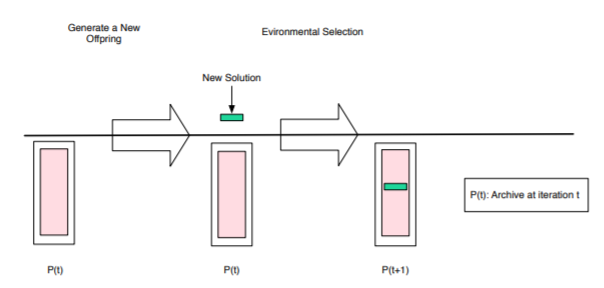


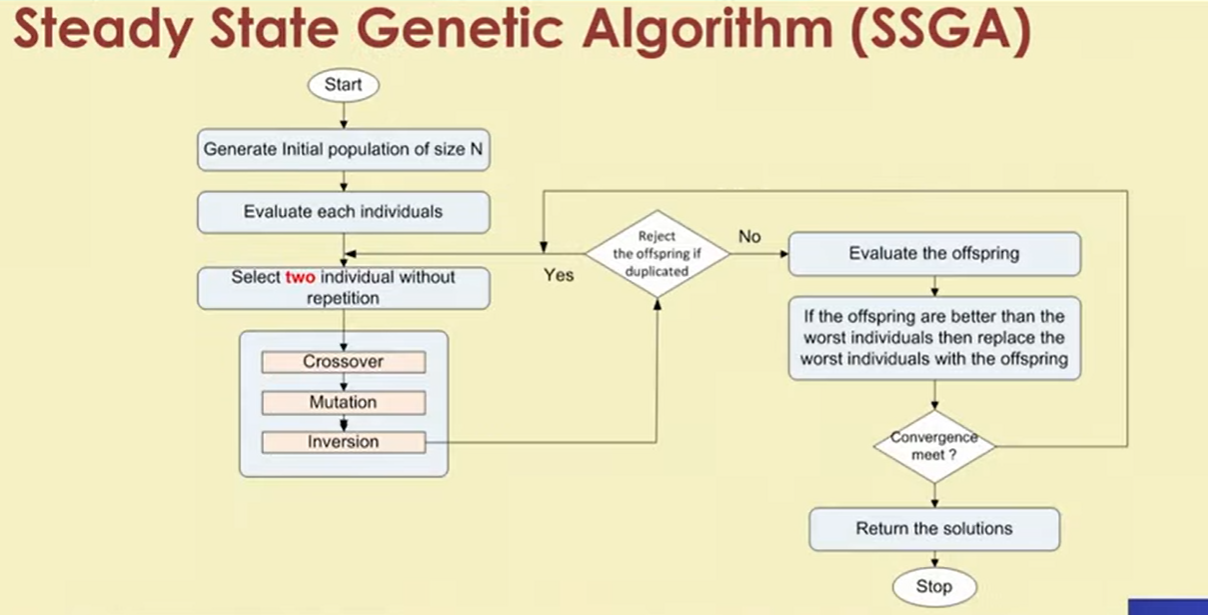
**Simple GA Features:**

* Have overlapping generation (only function of individuals are replace).
* Computational expensive
* Good when initial population size is large.
* In general, gives better results.
* Selection is biased toward more highly fit individuals; Hence, the average fitness (of overall population) is expected to increase succession.
* The best individual may appear in any iteration.

1. **Steady State Genetic Algorithm (SSGA):**

Steady state GA is meaning that there are no generations. It differs from the generic GA in that tournament selection does not replace the selected individuals in the population, and instead of adding the children of the selected parents into the next generation, the two best individuals out of the two parents and two children are added back into the population so that the population size remains constant.



 **Flowchart**

**Pseudo code**

* generate a population of individuals randomly
* while stopping criterion has not been met:
* parent1 <- tournament\_selection(P)
* parent2 <- tournament\_selection(P)
* child1, child2 <- with probability cross\_rate crossover parent1, parent2
* child1 <- mutate child1
* child2 <- mutate child2
* best1, best2 <- get the two highest fitness individuals out of parent1, parent2, child1, child2
* replace parent1 with best1
* replace parent2 with best2

**SSGA features**

* Generation gap is small Only two offspring are produced in one generation.
* It’s applicable when
* population size is small
* Chromosomes are of longer length
* Evaluation operation is less computationally
* Expensive (compare to duplicate checking)

1. **Messy Genetic Algorithm (MGA):**

An mGA is an improved version of a GA such that tight building blocks are identified and used in order to locate the global optimum based on the schema theorem. An mGA is able to handle variable-length strings, and the gene representation used in an mGA gives a tight linkage to building blocks.

**How it works:**

A Messy Genetic Algorithm typically has three phases:

1. **Initialization**

During initialization a population containing one copy

of all substrings of length k is created The expectation is that recombination will find the proper building blocks and assemble them into good solutions Given a problem with size l and building block size k and this modified form of initialization does create all possible building blocks, so the initialization phase requires a population size.

1. **Primordial Phase**

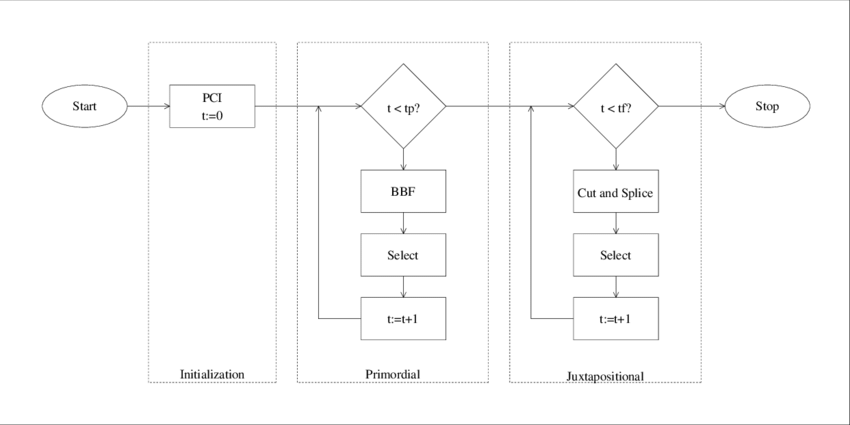
In a simplified primordial phase the error E is computed for each to the 2l triples from initialization phase. These triples are then sorted and some fraction of the best form the initial population In the experiments presented here the top 50 % of triples are used This simple selection of the better triples produces high quality building blocks.

1. **Juxtapositional phase**

During Juxtapostion selection is used together with two operators are cut and splice.

**Cut:** cuts the chromosome at random position

**Splice:** attaches two cut chromosomes together These two operators are the equivalents of crossover in a traditional GA It is here that the Messy Genetic algorithm begins to construct the match out of small building blocks that appear to be good matches to some subset of features in the model.

**Flowchart**

**Differences between messy GAs and simple GAs**

1. mGAs use variable-length codes that may be over or under specified with respect to the problem being solved.
2. mGAs use simple cut and splice operators III place of fixed-length crossover operators.
3. mGAs divide the evolutionary process into two phases: a primordial phase and a juxtapositional phase.
4. mGAs use competitive templates to accentuate salient building blocks.

**MGA Features**

* mGAs are ready for real-world applications, because they work, because they are efficient, and because they are practical.
* mGAs can converge to globally optimal results in the worst in all other problems with bounded deception.
* mGAs are a practical tool that can be used to climb a function's ladder of deception, providing useful and relatively inexpensive intermediate results along the way.

**Applications in Real World Problems:**

* **SGAs**
* Traveling salesman problem.
* Optimizing Cyclic-Steam Oil Production with Genetic Algorithms
* Genetic Programming and Genetic Algorithms for Auto-tuning Mobile Robot Motion Control.
* Feature Selection in Machine learning using GA.
* Genetic Algorithm as Automatic Structural Design Tool.
* Genetic Algorithm for Solving Site Layout Problem.
* Designing Texture Filters with Genetic Algorithms.
* **SSGAs**
* Multi-product supply chain network design using steady-state genetic algorithm.
* Steady State Genetic Algorithms for Generator Maintenance Scheduling Problems.
* Steady-state genetic algorithms for discrete optimization of trusses
* **MGAs**
* Messy Genetic Algorithms for Subset Feature Selection.
* Applications of MGAs in Design of FIR filters.
* Applications of MGAs Satellite image clustring.
* MGAs for solving the winner determination problem.