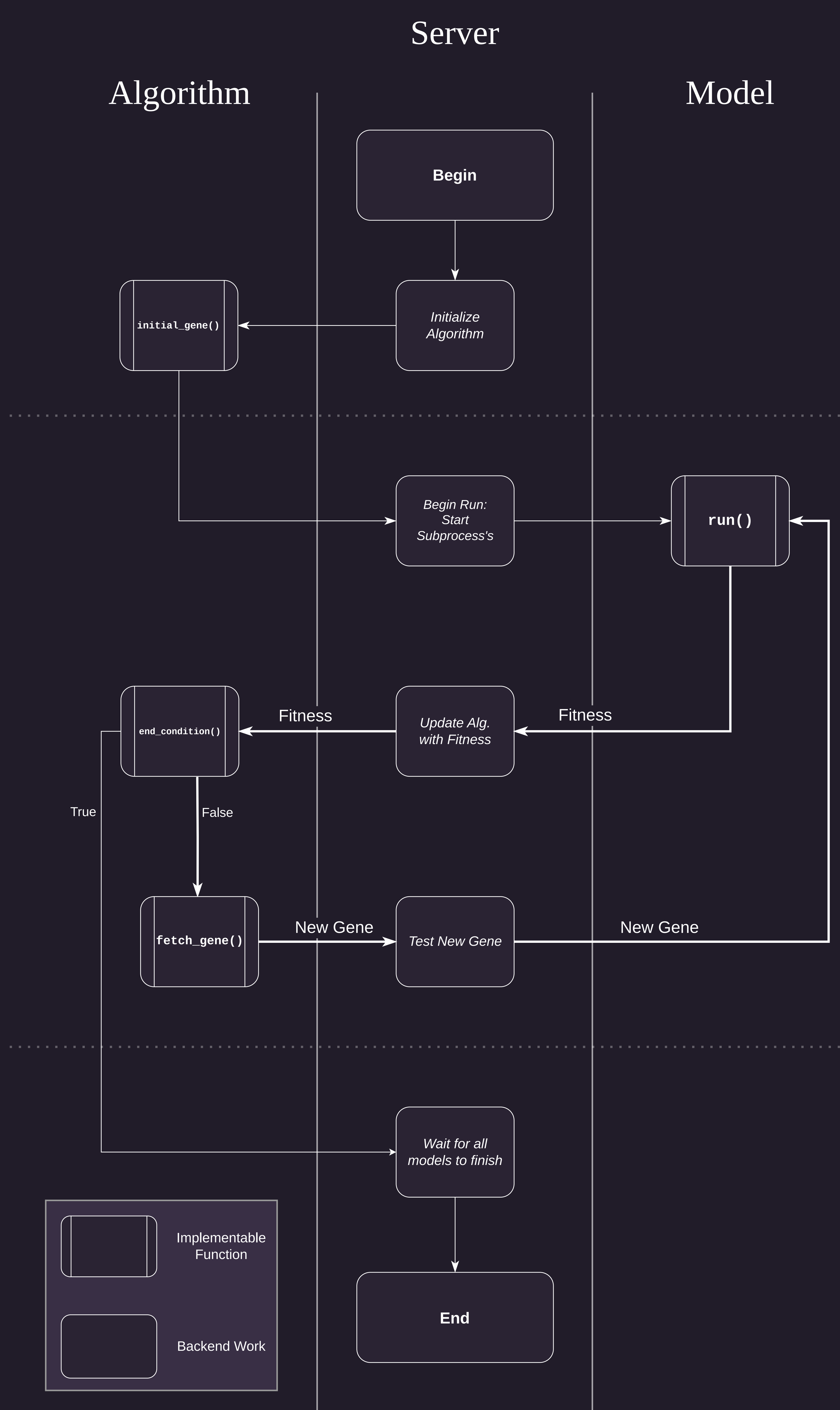
DGA Runtime Flowchart



Initialization

Repeatedly use Algorithm.initial_gene() to populate the gene pool (self.pool).

Asynchronous Main Loop

Splitting into subprocess's is not shown. After 'Test Initial Genes', all subprocesses called to test genes will run in parallel. The loop is run for every subprocess. That is, each subprocess will run its own instance of this loop (generating and testing new genes), but all will update the *same* genetic algorithm and state variables. State variables include things like gene pool, current iteration, current mutation rate (if using decay), etc.

Implementable functions can be found in details to the right

Ending

After an end flag is sent, the server will wait for all running models to return before ending.

Model: Functions

`Model.run()`	Inputs
	`gene: np.array`
	
	Returns
	`float`
	
	Function that should contain the model you want to test. The input gene are the parameters for the model, and the fitness of those parameters after being tested in the model should be returned as a float.
	
	Example: model is a Torch ANN, with the input (gene) being the weight matrices. In this case, in the run() function I would load the ANN with the given weights and test it. A good fitness value to return would be the testing accuracy of the model with those weights.

`Model.load_data()`	Inputs `None` Returns `None` 'Safely' load data from disk for your model. Use class args (self.your_arg) to store these datas. For Server and SLURM_Server, asynchronously running models share the same disk. Hence, this function will only *safely* load the data using file locks.

`Model.logger()`	Inputs
	`fitness: float, iteration: int`
	
	Returns
	`dict`
	
	Given just-tested fitness & iteration for logging. Override to add personal things to log. Logs stored in run_name/logs/model_i.log, where 'i' is the id for the subprocess being logged.

`fetch_gene()` from `Genetic_Algorithm` class

Genetic_Algorithm is an optimizer provided with DGA. Here is the fetch_gene function from this class as an example for how to write your own:

```
def fetch_gene(self, **kwargs) -> tuple:
  self.current_iter += 1    # Increment iteration

  # If pool is unitialized, add new gene
  if len(self.pool.items()) < self.num_genes:
    ...

  # Need at least 2 genes in pool to create new gene
  elif len(self.valid_parents.items()) < 2:
    ...

  # Otherwise, create a new offspring
  else:
    ...
```

`create_new_gene()` from `Genetic_Algorithm` class

Here is the create_new_gene function from the Genetic_Algorithm class as an example for how to use the different Algorithm functions to generate your own genes:

```
def create_new_gene(self, **kwargs):
  # Parents of new gene
  p1, p2 = self.select_parents()

  # New gene created
  new_gene = self.crossover(p1, p2)

  # Apply mutations
  new_gene = self.mutate(new_gene)
  return new_gene
```

Algorithm: Functions

`Algorithm.__init__()`	Inputs
	`num_genes: int, gene_shape: tuple, mutation_rate: float, iterations: int`
	
	Returns
	`None`
	
	Use this constructor to add your own personalized algorithm variables. For example, a dictionary to track the top n highest fitness genes tested so far.

`Algorithm.initial_gene()`	Inputs
	`None`
	
	Returns
	`np.array()`
	
	Generate and return an initial gene. Ideally a randomly generated array.

`Algorithm.fetch_gene()`	Inputs
	`None`
	
	Return:
	`np.array`
	
	Function that is called to create new gene. Function should decide what to do given the current state of the algorithm. Should contain an 'If' statement that handles different state conditions. Returns a newly created gene.

`Algorithm.create_new_gene()`	Inputs
	`None`
	
	Returns
	`np.array`
	
	Logic to create new gene. Intended use: utilize select_parents(), crossover(), mutate(), to generate a new gene. Returns new gene.

`Algorithm.remove_weak()`	Inputs `None` Returns `None` Used to prune the pool. Intended to be called during fetch_gene() (because this is the function that iterates the algorithm). Note the pool (self.pool) is a class object, so no args)

`Algorithm.select_parents()`	Input
	`None`
	
	Returns
	`[ np.array ]`
	
	Selects two or more genes as 'parent genes', intended for 'breeding' new genes. Ideally, parents selected for higher fitness ('survival of the fittest' strategy). Returns two+ parent genes.

`Algorithm.crossover()`	Inputs
	`p1: np.array, p2: np.array`
	
	Returns
	`np.array`
	
	Input is two genes representing 'parent' genes for 'breeding'. Sections of these genes should be spliced to create a new 'child' gene that's returned.

`Algorithm.mutate()`	Inputs
	`gene: np.array`
	
	Returns:
	`np.array`
	
	Take in a gene and apply a mutation to it. Ideally some random variation applied to a small part of the gene. Return mutated gene

`Algorithm.end_condition()`	Inputs
	`None`
	
	Returns
	`bool`
	
	Checks state of algorithm for an end condition, returns True if that condition is met. Otherwise returns False.

Provided Server & Algorithm Classes

Listed here are the Server and Algorithm classes provided with DGA. These objects can be imported into your own file and used as follows:

```
from DGA.Algorithm import Plateau_Genetic_Algorithm
from DGA.Algorithm import Genetic_Algorithm
from DGA.Server import Server
from DGA.Server import Server_SLURM
```

DGA.SLURM_Server

`Server_SLURM()`

Description

This is a server object intended to run on a SLURM system. It operates exactly like the basic server, except it will queue nodes using a provided sbatch script to test models.

Arguments

`run_name`: `str`
Used to name directory with run info

`algorithm`: `Algorithm`
Used for optimizing

`model`: `Model`
Contains model to be optimized

`num_parallel_processes`: `int`
Number of parallel processes to call during testing

`sbatch_script`: `str`
Path to SLURM sbatch script that will deploy your model

Example Usage:

```
Server_SLURM(
  run_name="My_SLURM_run",
  algorithm=My_Algorithm(...),
  model=My_Model(...),
  sbatch_script="sbatch_script.sh",
  num_parallel_processes=5,
)
```

Full script using this object:

/scripts/SLURM_examples/SLURM_GA...ple.py

DGA.Server

`Server()`

Description

This is a basic 'server', which manages the asynchronous subprocess's and state of the algorithm. This will run the optimizer (algorithm) on the given model, utilizing cores on the CPU for each subprocess.

Intended more as a base class to define 'Server' behavior.

Arguments

`run_name`: `str`
Used to name directory with run info

`algorithm`: `Algorithm`
Used for optimizing

`model`: `Model`
Contains model to be optimized

`num_parallel_processes`: `int`
Number of parallel processes to call during testing

Example Usage:

```
Server(run_name="My_Run",
       algorithm=My_Alg(...),
       model=My_Model(...),
       num_parallel_processes=5,)
```

Full script using this object:

/scripts/GA_examples/Simple_GA_Example.py

DGA.Local

`Synchronized()`

Description

Same useage as Server (trains a Model with an Algorithm), except it does so using a single process.

Same args as Server, but *no* num_parallel_process argument.

DGA.Genetic_Algorithm

`Genetic_Algorithm()`

Description

This is simple genetic algorithm which maintains a 'gene pool' of tested parameters and their fitness's. On each iteration (each fetch_gene() call), a new gene is created, and the weakest gene is ejected from the pool. This way, the pool stays a consistent size while still optimizing for fitness.

When create_new_gene() is called, the new gene is created by selecting 2 parents, crossing them at a random point (all values before random point from parent 1, all values after from parent 2), and finally a random mutation that can occur with probability mutation_rate.

Run ends at maximum iterations ('iterations' argument)

Arguments

`gene_shape`: `tuple`
Shape of gene (assumed genes are array-like)

`num_genes`: `int`
UMax number of genes in the gene pool

`mutation_rate`: `float`
Probability of mutation occuring

`iterations`: `int`
Total number of genes to test (100 iterations means 100 genes generated & tested)

Example Usage:

```
Genetic_Algorithm(
  gene_shape=(100, 100),
  num_genes=25,
  mutation_rate=0.25,
  iterations=100,
)
```

Full script using this object:

/scripts/GA_examples/Simple_GA_Example.py

DGA.Plateau_Genetic_Algorithm

`Genetic_Algorithm()`

Description

The intent with this algorithm is to search until the tested genes are no longer improving. This is done by looking for a 'plateau' in the learning curve (details below). When a plateau is detected, it's assumed a local minima in the parameter space has been reached. The algorithm then does the following:

On Plateau Detection:
1. Top performing gene added to 'founders pool'
2. Reset pool (New random genes)

To evaluate if a 'plateau' is reached, the most recently tested plateau_sample_size models are retrieved and plotted in the order they were tested. A line is regressed over these points, and if the regression coefficient is small enough (aka, line is flat enough), a plateau is detected.

Arguments

`gene_shape`: `tuple`
Shape of gene (assumed genes are array-like)

`num_genes`: `int`
UMax number of genes in the gene pool

`mutation_rate`: `float`
Probability of mutation occuring

`mutation_decay`: `float`
Decay rate, applied to mutation_rate at every iteration. Ideally in (0, 1]

`plateau_sensitivity`: `float`
Slope value that will triggers plateau detections

`plateau_sample_size`: `int`
Number of fitness's used for regression when detecting plateaus.

`iterations_per_epoch: int`
If a plateau is not found, a new epoch is autoamtically started after the current iterations surpasses iterations_per_epoch

`epochs: int`
Number of epochs to run for (number of plateaus)

Example Usage:

```
Plateau_Genetic_Algorithm(
  gene_shape=(100, 100),
  num_genes=25,
  mutation_rate=0.25,
  iterations=100,
)
```

Full script using this object:

/scripts/GA_examples/Simple_GA_Example.py

Example Scripts

ANN_Example

`/Distributed/scripts/ANN_example/ANN_Example.py`

Trains small ANN on MNIST using the DGA.Genetic_Algorithm optimizer. Good example of how to use Model.load_data() if you're not sure how.

GA_Examples

`/Distributed/scripts/GA_example/Complex_GA_Example.py`

Simple_GA_Example.py trains a simple vector-matching model using the DGA.Genetic_Algorithm optimizer. A single vector is randomly generated and set as the 'target', and the optimizer must try to find/estimate this target.

`/Distributed/scripts/GA_exampl/Simple_GA_Example.py`

Complex_GA_Example.py trains a complex vector-matching model using DGA.Plateau_Genetic_Algorithm optimizer. *n* vectors are randomly generated, and the plateau algorithm must find all vectors.

Local_Example

`/Distributed/scripts/Local_example/Local_Example.py`

This example trains a vector-matching model using a DGA.Synchronized.

SLURM_Example

`/Distributed/scripts/SLURM_example/SLURM_Example.py`

Need SLURM based system to run this. Trains a vector-matching model using SLURM. Nodes are queued according to the sbatch_script.sh provided in the same directory.