Module Interface Specification for AgnoLearn

Yiding Li
April 17, 2024

1 Revision History

Date	Version	Notes
2024-03-15	1.0	Initial draft
2024-04-17	1.1	Rewrite

2 Reference Material

This section records information for easy reference.

Table 1: Abbreviations and Acronyms

Symbol	Description
MG	Module guide
SRS	Software requirement specification
AC	Anticipated change
UC	Unlikely change
HHM	Hardware-hiding modules
BHM	Behaviour-hiding modules
SHM	Software-hiding modules
ADT	Abstract data type

3 Introduction

The following document details the Module Interface Specifications for Agnolearn.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at github.com/Tan630/Agolearn/issues.

4 Notation

Not used.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Table 2: Module Decomposition

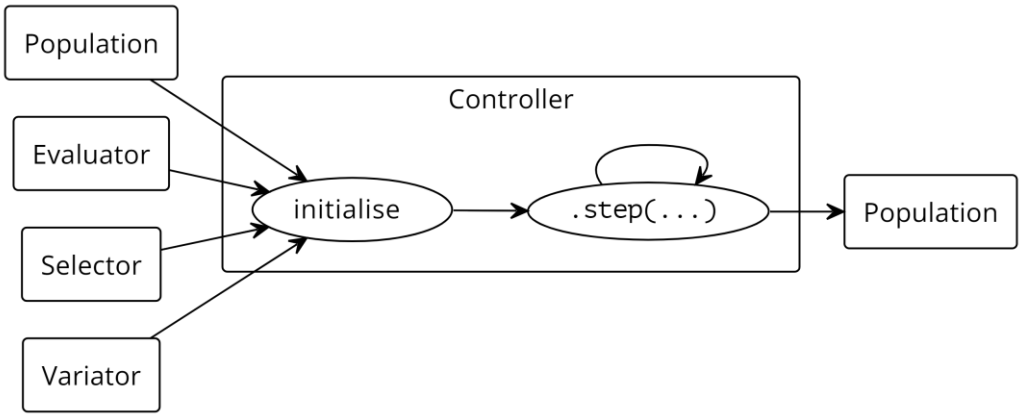
Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	`Controller, Genome, Population, ParentPool, Evaluator, Selector, Variator`
Software-Decision	`TestEvaluator, TestSelector, TestVariator, TestController, TestAlgorithm, AnalyseEvaluator, AnalyseSelector, AnalyseVariator. AnalyseAlgorithm, OneMax, FP, GP, LP, TPG`

6 MIS of Controller

Module	`Controller`
Uses	Genome, Population, ParentPool, Evaluator, Selector, Variator

6.1 Syntax

The controller manages the learning process of an evolutionary algorithm. An instance of the controller receives (a) a population, (b) an evaluator, (c) a selector, and (d) a variator. The method `.step(…)` runs one iteration of the optimization algorithm, then outputs the result.



6.1.1 Exported Constants

This module exports no constant.

6.1.2 Exported Access Programs

Name	In	Out	Exceptions
initialise	Population[T], Evaluator[T], Selector[T], Selector[T], Variator[T], Controller[T]	Controller[T]	
.step	--	state change, Population	

6.2 Semantics

6.2.1 State Variables

The controller is generic. The controller maintains a population of genomes, as well as several evolutionary operators.

Variable	Type
T	type
population	Population[T]
evaluator	Evaluator[T]
parent_selector	Selector[T]
child_selector	Selector[T]
variator	Variator[T]

6.2.2 Access Routine Semantics

(i) initialise

The initialise routine registers its inputs as state variables. The routine also evaluates the initial population, so that subsequent steps begin with an evaluated population.

routine initialise(pop, eval, p_selector, s_selector, var)
  initialise self
  self.population ← pop
  self.evaluator ← eval
  self.parent_selector ← p_selector
  self.survivor_selector ← c_selector
  self.variator ← var
  self.evaluator.eval_population(self.population)
  return self
end **routine**

(ii) step

The step routine performs one iteration of the optimisation algorithm.

```
routine step()
    evaluator.eval_population(population)
    parents ← parent_selector(evaluator)
    children ← variator.vary_pool(parents)
    evaluator.eval_population(population)
    survivors ← survivor_selector(population)
    return population
end routine
```

7 MIS of Genome

Module	Genome
Uses	--

7.1 Syntax

7.1.1 Exported Constants

This module exports no constant.

7.1.2 Exported Access Programs

Name	In	Out	Exception
initialise	T	Genome	
copy	--	Genome	
set_fitness	float	state change	
get_fitness	--	float	

7.2 State Variables

The genome is generic. It contains one value of the given type, as well as a fitness score.

Variable	Type
T	type
value	T
fitness	float

7.3 Access Routine Semantics

7.3.1 initialise

The initialise routine registers its input as a state variable.

```
routine initialise(val)
    initialise self
    self.value ← val
    return self
end routine
```

7.3.2 copy

The copy routine deep-copies the contained value, so that changes made on the returned value do not affect the object.

```
routine copy()
    return initialise(deep_copy(self.value))
end routine
```

7.3.3 set_fitness

The set_fitness routine sets the fitness score of this genome.

```
routine set_fitness(fit)
    fitness ← fit
end routine
```

7.3.4 .get_fitness

The .get_fitness routine returns the fitness score of this genome.

```
routine get_fitness()
    return fitness
end routine
```

8 MIS of Population

Module	Population
Uses	Genome

8.1 Syntax

8.1.1 Exported Constants

This module exports no constant.

8.1.2 Exported Access Programs

Name	In	Out	Exception
initialise	Sequence of Genome[T]	Population[T]	
.get_genomes	--	Sequence of Genome[T]	

8.2 State Variables

The population is generic. The population maintains a list of genomes.

Variable	Type
T	
genomes	Sequence of Genome[T]

8.3 Access Routine Semantics

8.3.1 initialise

The initialise routine receives a sequence of genomes. The routine then registers these values as state constants.

routine initialise(vals)
  initialise self
  self.genomes ← vals
  return self
end routine

8.3.2 get_genomes

The get_genomes routine returns the contained value.

routine get_genomes(vals)
  return genomes
end routine

9 MIS of ParentPool

Module	ParentPool
Uses	Genome

9.1 Syntax

9.1.1 Exported Constant

This module exports no constant.

9.1.2 Exported Access Programs

Name	In	Out	Exception
initialise	Sequence of Genome[T], arity : int	Population[T]	
get_arity	--	int	
pop_tuple	--	Tuple of Genome[T]	
is_empty	--	bool	

9.2 State Variables

Variable	**Type**
.genome_tuples	Sequence of tuples pf Genome[T]

9.3 Access Routine Semantics

9.3.1 initialise

The initialise routine receives a sequence of genomes. The routine splits these genomes into tuples of a given size, then discards left-over values.

```
routine initialise(vals, arity)
    initialise self
    while length(vals) > 2 do
        new_tuple = tuple()
        for i := 1, ..., (.get_arity()) do
            add vals.next() to new_tuple,
                then remove that value from vals
        end for
    end while
    return self
end routine
```

9.3.2 get_arity

The get_arity routine returns the arity of contained tuples.

```
routine .get_arity()
    return .arity
end routine
```

9.3.3 pop_tuple

The pop_tuple routine returns a contained tuple, then removes that tuple from the collection of contained tuples.

```
routine .pop_tuple
    out_tuple = .genome_tuples.next()
    remove out_tuple from genome_tuples
    return out_tuple
end routine
```

9.3.4 is_empty

The is_empty routine returns true if the collection of contained tuples is empty; otherwise, this routine returns false.

```
routine .is_empty
    return length of genome.tuples
end routine
```

10 MIS of Evaluator

Module	Evaluator
Uses	Population Genome

10.1 Syntax

10.1.1 Exported Constant

This module exports no constant.

10.1.2 Exported Access Programs

Name	In	Out	Exception
.evaluate_genome	Genome	float	
.evaluate_population	Population[T]	float	

10.2 State Variables

This module has no state.

10.3 Access Routine Semantics

10.3.1 evaluate_genome

The evaluate_genome routine returns the fitness of one genome.

routine .evaluate_genome(genome)
  return *fitness of* genome
end routine

10.3.2 evaluate_population

The evaluate_population routine receives a population, then assigns a fitness score to each item in the population.

```
routine evaluate_population(population)
    for each item in population do
        item.set_fitness(.evaluate_genome(item))
    end for
end routine
```

11 MIS of Selector

Module	Selector
Uses	Population Genome

11.1 Syntax

11.1.1 Exported Constant

This module exports no constant.

11.1.2 Exported Access Programs

Name	In	Out	Exception
select_pool	GenomePool[T], int^+	Population[T]	
select_population	Population[T], int^+, int^+	Population[T]	

11.2 State Variables

This module has no state.

11.3 Access Routine Semantics

11.3.1 select_population

The select_population method receives a population, then returns a population with high-quality items in the input. The way this is done depends on implementation.

routine .select_pool(genome_pool)
  return *items in* genome_pool *with good fitness*
end routine

12 MIS of Variator

Module	Variator
Uses	Population
	Genome

12.1 Syntax

12.1.1 Exported Constants

This module uses no constant.

12.1.2 Exported Access Routines

Name	In	Out	Exception
vary_tuple	Tuple of Genome[T]	Sequence of Genome[T]	
vary_pool	GenomePool[T]	Population[T]	

12.2 Semantics

12.2.1 State Variables

This module has no state.

12.2.2 Access Routine Semantics

(i) vary_tuple

The vary_tuple routine receives one tuple of genomes, then creates a Genome using its input as parents.

> **routine** vary_tuple(genome_tuple)
>
> > **return** *a* Genome *that is produced with* genome_tuple *as parents*
>
> **end routine**

(ii) vary_population_pool

The vary_population_pool routine receives a GenomePool, then, for each tuple in the input, creates a Genome using that tuple as parents. The results are collected into a Population.

```
routine vary_population_pool(genome_pool)
    genome_collection ← [ ]
    for each genome_tuple in genome_pool do
        genome_collection.add(vary_tuple(genome_tuple))
    end for
    return Population(genome_collection)
end routine
```