USING MISE

Contents

Version 1.0, SPJ 10 June 2013

Introduction

`mise` is a tool used to optimise the value of some measured quantity in an experiment by varying the value of certain globals. `mise` must be run on the same computer as `runmanager`.

Before beginning an optimisation routine you must decide what the goal is. `mise` will optimise parameters of you choosing to increase the overall "fitness" of your experiment. This may come from a single- or multi-shot experiment. Each run of the experiment by `mise` is called an "individual". In your `lyse` script you may calculate the fitness of an individual however you like. A bigger fitness is seen as being better (there are no bounds). If you wish to optimise for a low value of some measurement then you can simply add a negative sign to that value before reporting the fitness.

Once all shots comprising an individual have run, its fitness should be reported to `mise`. This should only be done once per individual (the first reported value for that individual will be final).

The mutation rate is the standard deviation of the mutations to be applied to the value of a global for newly created individuals. Upper and lower bounds for each global to be varied are also set.

Writing a fitness reporting lyse script

Your fitness reporting script can be a single- or multi-shot script, depending on how you calculate the fitness. At the beginning of the script, import the `mise` API (`import mise`. Once the fitness of an individual has been calculated simply call "`mise.report_fitness(id, fitness, host)`", where `id` is an integer corresponding to the individual's id number, `fitness` is a float representing that individual's fitness, and `host` is a string containing the hostname of the computer on which `mise` is running. You should ensure that if individuals comprise more than one shot then the fitness is not reported until all the shots for that individual have been analysed.

Beginning the optimisation routine

In `runmanager`, select the globals that you want to be varied and enter the following command:

```
MiseParameter(min,max,mutation_rate=None,initial=None)
```

The `min` and `max` arguments correspond to the bounds you want to impose on the global, and `mutation_rate` is the standard deviation of mutations each generation. If the mutation rate is not set it will be default to 10% of the span between `min` and `max`. If the `initial` keyword argument is set then the first generation will be distributed around the initial value, with a standard deviation of the mutation rate. If `initial` is not set then the first generation will be randomly distributed throughout the span set by `min` and `max`.

Multiple globals may be optimised together by using the `MiseParameter` command on each of them. If globals that are not being optimised contain lists of values then an individual for that optimisation will contain shots for each point in the resulting parameter space, each with the same value for the parameters being optimised.

To begin optimisation, select the "send to `mise`" option at the top of the `runmanager` window, ensure the hostname is set to "localhost" (you can run `mise` from a different computer if your script is on a shared drive, but we don't recommend it) and click "engage". Individuals will appear in the `mise` window, and begin compiling and being sent to BLACS. You can pause the compilation by clicking pause.

Altering parameters during optimisation

During an optimisation, you may change the bounds and mutation rate for each global being used. Note that this will only take effect for the next generation. You may select individuals to be re-executed if you suspect a bad shot, or removed from the optimisation completely. Note however that only the current generation of shots will affect the values given to the next generation to be created. Individuals may also be added to the gene pool manually.

The population size may be altered in `mise` to change the number of individuals used per generation.

Optimisation will continue forever. We recommend plotting the average fitness (and its spread) vs generation to see how your optimisation is going. You could also program your `lyse` script to stop sending fitness reports to `mise` once some threshold condition was reached, preventing the next generation from being generated.