FiniteStateAutomaton

Code Base by Andrew Ferdowsian
White Paper + Updates by Justin Oliver
May 13, 2016

GitHub

https://github.com/aaandrew152/FiniteStateAutomaton

Requirements

The graphical output of the library depends on `matplotlib` and `networkx`. The command line graphic also requires the `progress` library. All libraries can be installed using `$ pip install ...`

Usage

The desired simulation should be defined in `Parameters.py` using the guidelines outlined below. Once the simulation has been set up, simply navigate to the root directory and use `$ python main.py` to execute the simulation. If you wish to run the same simulation multiple times with one function call, use `$ python master.py` instead. Results will be generated according to the parameters specified and output in the desired format.

Parameters

Simulation Parameters

- `numSets`: the number of sets of individuals in the population
- `numGenerations`: the number of generations the simulation will be run for
- `collectGenerations`: the number of generations before the final generations to begin recording all actions played by individuals
  - must be less than or equal to the total number of generations
  - set to zero if this function is not desired
- `numSims`: the total number of simulations to be run (only relevant when using `master.py`)

Game Parameters

- `payoffMatrix`: the game's payoff matrix, i.e. ((2,0),(3,1)) for Prisoner's Dilemma
- `discountFactor`: the probability δ that a game is repeated (i.e. does not end) within a given generation
- `mutationProb`: the probability that a given individual undergoes mutation at the end of a generation, with probabilities of specific mutation types specified below
  - `mutation_addState`: given an individual is mutated, probability that a new state is added (random action chosen for the state, arrow randomly chosen from all existing arrows is reassigned to point to this new state)

  - `mutation_deleteState`: given an individual is mutated, probability that an existing state is deleted (all arrows currently pointing to this state are randomly reassigned to point to another state)
  - `mutation_changeArrow`: given an individual is mutated, probability that an existing arrow is randomly reassigned to a new target
  - `mutation_changeAction`: given an individual is mutated, probability that the action at an existing state is randomly reassigned (this probability cannot be specified, but is rather $1 - \sum$(previous probabilities))

- `startingStrategyDistribution`: the distribution of state actions at the first generation, i.e. [0.5,0.5] would state that half of all individuals begin playing each action
- `noise`: is there noise in the system (TO BE IMPLEMENTED)
- `mutationPrune`: if set to `True`, an individual's strategy will be reduced such that all states are accessible from the origin state *any* time a mutation occurs, if `False` this reduction will only occur at the final generation

Output Parameters

- `prevalence`: the threshold of prevalence in the population for a strategy to be recorded, all strategies with prevalences below this threshold will essentially be thrown out at the final generation, set to 0 if all strategies should be saved
- `saveOutput`: if `True` simulation results will be saved to a .txt file in a directory specified below, if `False` results will be printed to standard output
- `withGraphics`: if `True` graphics representing each strategy will be generated and saved in the appropriate directory
- `directory`: specify the directory for the output to be saved into, relative to the current location of the root folder
  - if left as `None`, the output will be saved in a subdirectory entitled `simulations/`
  - if the directory already exists, *it will be erased and replaced by the new results*
  - the specified directory will contain subfolders for each simulation run

Output

Summary

When simulation results are set to be saved (see `saveOutput` parameter), a file called 'summary.txt' will be generated in the specified directory. Otherwise the results will be printed to the standard output. These results contain:

- a summary of the parameters specified for the simulation
- the average amount all given actions were played by all individuals over the specified number of generations (see `collectGenerations` parameter)
- a list of all strategies present at the final generation and their prevalences

Each strategy in the list is a textual representation of the FiniteStateAutomaton object. Each item in the list represents a state as a tuple. Each state contains the following information (in order):

1. the number of the state (0 is the origin state from which all strategies begin)
2. the action to be played at the state
3. a list of arrows pointing away from the state, each of which is listed as a tuple of the form (target state, condition)

Graphics

When graphics are generated (see `withGraphics` parameter), a .png image will be generated for each strategy present at the final generation. The images are saved with the title 'Strategy#_prevalance%.png'. The graphics can be understood as follows:

- each state is a circle with the action at that state encoded as a color
- each arrow is a black line where the head of the arrow is thicker
- each arrow is labeled with the condition of its use, represented as the action (as a number) required to take that particular path
- the origin state (0) is marked with *