Course - IN104: Projet informatique

Natalia Díaz Rodríguez

April 2, 2019

1 Summary of the project: Genetic algorithms

This is one of the projects proposed for IN104 course Projet Informatique.

Genetic algorithms take inspiration from two concepts of the theory of evolution: the definition of a selection process and the use of random mutations to obtain a new set of solutions from solutions previously envisaged. The project aims to implement a genetic algorithm and use it to determine the largest subset of a set of relative integers such that the sum of the elements of this subset is 0.

The project will be implemented in Python 3. Support slides on GA are in the website of the project[1]

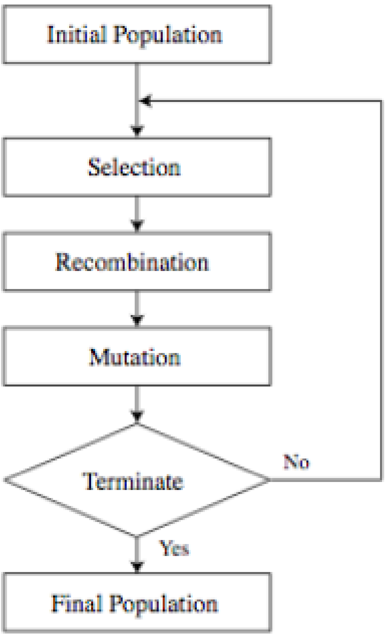

Figure 1: Example of genetic algorithms population evolution cycle

2 Genetic Algorithms

Because it can be challenging to search using methods like exhaustive search or exact optimization (e.g., linear programming) for finding the optimal solution in a high dimensional space, we draw inspiration from the field of genetic algorithms, which has shown to achieve efficient search across a variety of similar problems. The proposed design system of GAs can be shaped to mimic the processes that underlie these stochastic optimization methods.

[1] https://github.com/NataliaDiaz/IN_104-Projet-Informatique

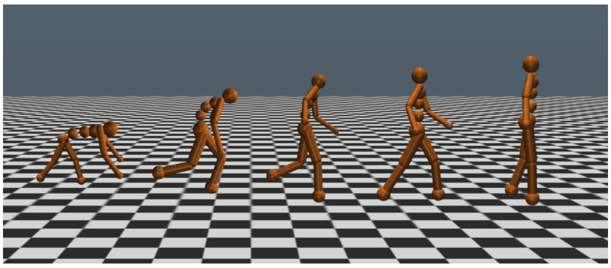

Figure 2: Deep Neuroevolution from Uber AI labs: https://eng.uber.com/deep-neuroevolution/

Genetic algorithms are a collection of methods inspired by genetics and natural selection. Within the metaphor, each candidate solution is conceptualized as having a chromosome composed of genes, where each gene's value is an allele.

- **Chromosome**: the string codification of a candidate solution (the code for an individual from the population)
- An **allele**: a feature (component, in our case an integer) that can be included as part of a chromosome string. In Biology, an allele is a variant form of a gene. Some genes have a variety of different forms, which are located at the same position, or genetic locus, on a chromosome. Humans are called diploid organisms because they have two alleles at each genetic locus, with one allele inherited from each parent.

2.1 Genetic Algorithm Operators

Crossover and mutation are two basic operators of GA. Performance of GA very often depends on them. Type and implementation of operators depends on encoding and also on the problem. There are many ways how to do crossover and mutation [1]:

- **Selection**:

  A fitness function filters (*selects*) a set of most fit individuals according to a score function of each individual

- **Crossover**: e.g., in *single point crossover*, one crossover point is selected; the binary string from the beginning of the chromosome to the crossover point is copied from one parent, and the rest is copied from the second parent. Example:

  11001011+11011**111 = 11001111**

- **Mutation**: An operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. For instance, *Bit string inversion* selected bits are flip (inverted) at random positions. E.g.

  Example:

  1**1**001001⇒ 1**0**001001

  The probability of a mutation of a bit is $1/L$ where L is the length of the binary vector.

- Uniform: This operator replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes.

2.2 Application Use case: An industry case study in fashion design at Stitch Fix Inc.

To illustrate how the GA operators work in a real problem, we consider the case in designing clothing from Stitch Fix [2]. In a fashion design use case, a blouse's chromosome is a vector of attributes corresponding to the dimensions of the search space [2]. Generating a new design can then be framed as an evolutionary process searching over a population of possible blouses, through a series of generations. Each generation passes through three stages:

- Selection: Each individual of the current generation is evaluated for its fitness. This implies access to an explicit objective function mapping attributes to outcomes or empirical measurements of each individual in the current generation. For instance, blouses might be selected to maximize style and fit feedback. These individuals from the current generation can then enter a mating pool using not-so-natural selection procedures such as selecting the N fittest individuals, or selecting individuals with a probability proportional to their fitness (e.g., biased roulette-wheel selection).
- Recombination (Crossover): Individuals in the mating pool are bred to produce new individuals that are hopefully desirable and novel because they were created by decomposing two good parents and using the resulting elements to compose a child. There are a variety of procedures for implementing recombination. They essentially boil down to using a random mix of features from each parent. An example of the latter in our system might be passing sleeve type, sleeve length, and sleeve fabric as a unit. We can then propose which genes to select from each parent using this broader knowledge.
- Mutation: Diversity and novelty are introduced into new generations by randomly changing alleles. These mutations search the vicinity of the solution space via a random walk. This vicinity can be more or less local depending on the type of mutations that are employed. For instance, the values of different alleles can be assigned proportional to the observed distribution of alleles or forced into their extremes to achieve a more aggressive search (e.g., boundary mutation). Finding a compromise between these is practical to retrieve underrepresented regions of the feature space.

3 This Project Problem: Find the largest sum of integers summing zero

This is an NP-complete problem. Given a set of integers, Does it exist a subset of integers whose sum is 0? If yes, provide the subset of integers of largest cardinal size. Example: Input:

```
E = [7, 3, 2, −5, 8]
```

Solution:

```
s = [3, 2, −5]
```

Common nomenclature to use:

- Class *Individual*: all information about a concrete solution. It includes min. the genetic code for that solution and its fitness
- *Chromosome*: the genetic information binary code associated to an individual
- Class *Population*: a set of individuals (or "gene pool") that will evolve in different generations

Minimal methods to implement that will be evaluated:

- *evaluate_fitness(a)*
- *select_individuals()*
- *crossover_individuals(a, b)*

- *mutate_individuals(a, b)*

Test driven development is strongly recommended as a development cycle. Some of the files and test data are included in the project repository[2]:

- Given test input data files: inputSet.zip
- Code for input set generation: genEnsDep.py.zip

3.1 Getting started

1. If you don't have github, apply for student github account in https://education.github.com.
2. Create a PRIVATE project called *GA-firstPartnerSurname-secondPartnerSurname*
3. Invite your partner and me as collaborators (with Master access if you use gitlab).
4. Start coding! Progress will be measured with github collaboration (commit, and push throughout the course)

Both gitlab.ensta.fr and github.com can be used. However, the latter is recommended (in which case, the project created needs to strictly be set as private).

4 Evaluation

The components of the final grade will be aggregated to provide a final mark according to the ENSTA scale.

- 25% **Source code**: features, tests, documentation, etc. Source code must be documented, the report including a link to the repository must be in a .zip file. The repository will contain the sources, as well as a plain text file README.md which indicates the actual operational features and limitations. Python 3 code should compile, the teaching assistants are not supposed to make significant corrections for it to compile: a code with few features, but that compiles and does not crash will be preferred to a more complete code but which is not directly operational!

- 25% **Defense** (7-10 minutes presentation, 3-5 minutes questions): The formal quality of the presentation will be an important element. The defense needs to include a demonstration on the basis of the source code, an analysis of the difficulties encountered and implemented solutions. It will not include a presentation of the problem or the method of resolution that the teaching assistant obviously knows already well. The defense is open to everyone (subject to the acceptance of the pair that will present), the chronological order of defense will be given by the list of each group.

- 20% **Practice Analysis**. In the report you will analyze and criticize the progress of your project, and its success and failure factors. This evaluation component also includes the oral treatment of these questions during the defense.

- 30% **Continuous Progress** of the practical work (mid-term evaluation) during the practical lab sessions (based on git commits thorough the whole course).

[2] https://github.com/NataliaDiaz/IN_104-Projet-Informatique

5 Reporting and deadline

You will return **ONLY** a **SINGLE pdf** (max. 5 pages) report to Natalia Díaz Rodríguez (natalia.diaz (at) ensta (dot) fr). The report must include a link to the git repository named IN104_NameA_SurnameA_NameB_SurnameB (for all team members) with the project code inside a folder called GA. Please do not send extra files by email (these should be all in the repository link to include in the pdf). The report needs to be submitted, at the latest, **5 days before the defense day** (happening the last day of the course). Late submissions will be penalized one point per day.

For this particular project we will evaluate the implementation of the methods *get_fitness(), select(), mutate(), crossover()*, plots based on the population size and the demo of the code running and changing diverse parameters. Implementation using a graphical interface library such as *tkinker* is optional. Further experiments evaluating the population evolution dynamics, parameter analysis plots and performance will be more valuable. The report should include insightful plots (e.g., using matplotlib) of the evolution of the population. Optionally, it will be positively valued if you get inspired from "The Era of Deep Neuroevolution" blog post in order to apply some new concepts to your project solution.

5.1 Before submitting your project

- Please make sure you have reported in a table in the report, the set cardinal of your algorithms best solutions found for each of the datasets you were able to run
- Be sure to provide a plot with the evolution of the population across simulated generations.
- Please use the following line at the first line of your Python files to avoid encoding errors across different Python versions: `# coding=utf-8`.
- Make sure to have implemented methods *selection(), crossover()* and *mutate()*.
- Extra points: It is preferable to provide extra implementations of *selection(), crossover()* and *mutate()* than providing an interactive GUI.
- All information to run your program should be in your repository README.md file.

6 Useful links and references

1. www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php
2. Genetic algorithms in Fashion Personalization at Stitch Fix (data-driven fashion design): http://multithreaded.stitchfix.com/blog/2016/07/14/data-driven-fashion-design/
3. https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)
4. GIT:
   - Lecture 0, 1, 2 Intro to Git, OOP and Unit Tests[3]
   - Antonin Raffin tutorials - Intro to Git: http://slides.com/antoninraffin/git and Git intermediate: http://slides.com/antoninraffin/git-intermediate
   - http://users.humboldt.edu/smtuttle/s12cis492/492guide-to-git.pdf
   - https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf
   - https://github.com/git-tips/tips#everyday-git-in-twenty-commands-or-so
   - https://tutorialzine.com/2017/11/10-useful-git-tips
5. Install Python libraries and Master Python: http://musicinformationretrieval.com/python_basics.html

[3] https://github.com/NataliaDiaz/IN_104-Projet-Informatique

6. Python Numpy http://cs231n.github.io/python-numpy-tutorial/ and IPython tutorials http://cs231n.github.io/ipython-tutorial/

7. Iterate fast installing Jupyter notebooks http://jupyter.org/install and get good at IPython: http://musicinformationretrieval.com/get_good_at_ipython.html

8. The quartet of NumPy, SciPy, Matplotlib, and IPython is a popular combination in the Python world. Numpy Basics: http://musicinformationretrieval.com/numpy_basics.html, Numpy Tutorial: http://scipy.github.io/old-wiki/pages/Tentative_NumPy_Tutorial

9. Nice machine learning tutorials and notebooks: Image processing, k-means, matplotlib and numpy (French): http://wwwdi.supelec.fr/liesse/html/tp_k_means/tp_k_means.html

10. Useful for permutations and combinations with replacement: itertools: https://docs.python.org/2/library/itertools.html

11. On the length limits of a Python string: http://stackoverflow.com/questions/1739913/what-is-the-max-lengt

7 Appendix

7.1 Recap from pre-requirement lectures

Remember to add your ssh-key to your gitlab/github account (each of the team members) using [1]. There are some subtleties found to work easier in Github versus Gitlab; I recommend using github:

1. Create a new account as in [1] and a repository

2. Add your ssh-key as in [2].

3. Provide master access to your team member and me (username '*nataliadiaz*' in Github, '*diaz*' in Gitlab) via the *Team* option settings of the project.

For those using gitlab, access denied problems should be solved by giving your partner (and your teaching assistant), *master* access instead of *developer* access.

Easy to use editors are *Atom* [3] (open-source) or *sublime* [4] (recommended).

7.1.1 Test Driven Development

unittest has some internal magic that:

- discovers all classes inheriting from *unittest.TestCase*
- then runs its *setUp* function
- then runs all methods that are named *test_**
- then prints a summary (passed vs failed tests)

And more. We've already seen other methods to do unitary tests (using the __*name*__==__*main*__ trick in the program containing some _*test_func*). unittest is just more practical when there is a large set of tests as here. See all unittest frameworks online[4].

Running tests

If you use PyTest (which Can run unittest and nose test suites out of the box)[5].

Option A) (preferred) To run a given test:

```
python −m unittest test_name
```

[4] http://docs.python.org/3/library/unittest.html

[5] https://docs.pytest.org/en/latest/

e.g.:

```
python -m unittest tests_binary
```

if tests_binary.py is in the main binary repository

Option B) Run all tests in tests folder (-v for verbose mode):

```
py.test -v
```

7.1.2 Links

[1] Github (alternative to gitlab, easier):[6] ->request student account for individual, using your ensta email. Option b: create gitlab account[7].

[2] Create and add your ssh key to gitlab[8] or github[9]

[3] Atom editor[10]

[4] Sublime editor[11]

[5] Python overloading operators API: https://docs.python.org/3.4/library/operator.html

7.1.3 Q & A Troubleshooting

- Q: Using gitlab:

  ```
  bash: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)
  fatal: protocol error: bad line length character: No s
  ```

  A: You can try setting the locale language:

  ```
  export LC_ALL=fr_FR.UTF-8
  ```

  You can also try run the command *ssh-add* (this commands adds the ssh key to the ssh agent).

[6] https://education.github.com/

[7] http://gitlab.ensta.fr/

[8] https://gitlab.ensta.fr/help/ssh/README

[9] https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/

[10] https://atom.io

[11] https://www.sublimetext.com/3