Evolutionary Computation

Y1481702

January 13, 2018

Contents

1 Introduction

1.1 Background

Biomimicry, the imitation of nature for solving human problems has produced many examples of world class design. Some great examples of Biomimicry are Velcro(R), self-cleaning paints and Shinkansen high-speed trains[1]. In particular these examples imitate the form of nature's physical systems. Since the term biomimicry was first coined in 1997, this design philosophy has started to inspire new developments in various subjects such as the circular economy in product design[CITE] as well as several pieces of ongoing work in Computer Science such as swarm robotics[CITE].

Life has been around on earth for 3.8 billion years, this is a lot of R&D time to produce useful solutions which can be related to many of the current problems in various fields[CITE, VOX video].

Clearly Evolutionary Computation is an optimisation process inspired by nature in this way.

1.2 DEAP

Distributed **E**volutionary **A**lgorithms in **P**ython (DEAP) is an open source framework for Python[4]. This framework aims to be

1.3 EC for Snake

The task given is to create an evolutionary algorithm in Python using the DEAP frameowrk in order to play the classic video game, Snake. This is not a new task and has been the subject of previous work including an ...[2] and more recently a ...[3]. Despite this there is still room for further work, due to ongoing research into evolutionary computing and significant challenges because of the nature of the game.

There are a variety of available instances of Snake with an array of different rules. In this case, there is a square game board of 14x14 cells as shown in Figure. 1. The snake will continously move forwards in the direction it is facing, which can be changed by the user at any time. The aim is to collect food that appears in a random cell on the board when the previous one is eaten. Each time the snake eats a piece of food, it grows and takes up an extra cell. Once the snake fills every cell it can no longer grow. Hence, the highest possible score (185) is equal to the number of cells (196) less the initial length of the snake (11).

Implementing this task will involve facing all of the usual challenges in evolutionary computation such as maintaining diversity and reducing bloat. Snake also brings along new challenges. The random placement of the food on the board adds an element of stochasticity to the solution meaning algorithm will perform differently across multiple runs and care will need to be taken to ensure that it works in the general case. The solution will also need to balance the need for the snake to avoid crashing into walls or itself while still picking up the food before time runs out. Optimising both of these objectives at the same time might be difficult.

2 Methods

2.1 Representation

2.1.1 Available Options

The representation of the problem can significantly affect the algorithm's performance.

Genetic Algorithms

Evolutionary Programming

Evolutionary Strategy

Genetic Programming

Neuroevolution
Neuroevolution is a ... Unfortunately as the brief requires that the DEAP Python library is used, neuroevolution is not an available option as DEAP does not support it.
Grammatical Evolution? Is this explicitly supported in DEAP?[4] DEAP's transparency allows virtually any algorithm to be implemented but it's easier to use the built in structures.

2.1.2 Choice and Justification

Genetic Programming ¡— ??? Typing? Strong/Weak?

2.1.3 Physical Environments

Additional functions may help the snake to find a better solution, however we need to be careful of overparameterising the problem. This is a common trade-off in many forms of machine learning?

2.1.4 Initialisation Procedure

2.2 Population and Evaluation

2.2.1 General Discussion

Implementing a good fitness evaluation function is essential to evolving a solution to this problem. Without this, there will be no way to assess individuals in order to determine which will survive and reproduce. While there are two major objectives at play in the game, avoiding crashing the snake and getting to the food before the timer hits zero, these can be represented by the final size of the snake when the game ends. As such, we can simply consider this a single-objective problem where the objective is produce the maximum length of snake before either the snake crashes or the timer hits zero. This significantly simplifies the problem with minimal impact. The game itself does feature elements of stochasticity. In a given round the snake's food might spawn in a number of

random places. This adds an additional level of difficulty to the problem as the fitness of a given algorithm will likely change between runs.

2.2.2 Parent Selection

Selecting which individuals should have influence over the next generation is one of the most important factors for success in Evolutionary Computing. If only the best individuals are selected then diversity will be lost, the algorithm will get stuck in a local minima and be unable to find a good general solution. This is known as premature convergence. If too many of the less-promising solutions are selected then the algorithm may be unstable meaning good solutions will take too long to find and may be lost quickly. Striking a good balance between a diverse population and a stable algorithm is necessary for success in this task.

2.2.3 Maintaining Diversity

Within this evolutionary computation task, each individual of the population corresponds to a point in the search space of all possible programs. Diversity corresponds to having these points reasonably spread out throughout this space. A diverse population is, by definition, exploring more of this search space than one that is not.
Textbook: 'Premature convergence is the well-known effect of losing population diversity too quickly and getting trapped in a local optimum.'
Diversity is difficult to quantity in an objective manner and, as such, no single measure for it exists. A range of different measures including the number of different fitness values, number of different phenotypes/genotypes or entropy may be used[5].
Elitism is used to ensure that the best solution isn't randomly removed from the population. If no better solution is ever found, this solution will want to be present in the final population once evolution is complete.
In biological evolution, old generations die to free up resources for newer generations. Disregarding solutions due to age ...
Parent Selection can be random, so (if this is the case) good solutions may well die out.

2.2.4 Bloat

Bloat, the increase of the size of a genetic program without significant increase in fitness, is a difficult problem to overcome. Bloat is linked to, but different from, overfitting, a common problem in many forms of machine learning[6]. Controlling bloat while maximising fitness turns the evolution into a multi-objective optimisation problem. The two most commonly used solutions to bloat are parsimony pressure and double tournaments. Parsimony pressure subtracts a value based on the size of the program tree from the individual's fitness, this value does not necessarily need to be linear. Double tournaments use a secondary It has been shown that the order of a double tournament has no significant bearing on the end result. [cite, practical?]
[7]

2.3 Variation Operators

2.3.1 Mutation

Mutation is the operator that adds the novel, 'off-the-wall' ideas into the population. Without it, genes would only be shuffled around.. A trade-off is required here, too much mutation will result in a

highly unstable algorithm. Far too much mutation would Care needs to be taken when implementing mutation so as not to create biases. This is particularly the case with real-valued mutation..

2.3.2 Recombination

2.3.3 Termination Condition

3 Results

3.1 Calibration

4 Conclusion

4.1 Main Findings

4.2 Further Work

5 References

[1] J. M. Benyus, *Biomimicry: Innovation Inspired by Nature.* HarperCollins, May. 1997.

[2] T. Ehlis. (2000, Aug) Application of genetic programming to the snake game. [Online]. Available: https://www.gamedev.net/articles/programming/artificial-intelligence/application-of-genetic-programming-to-the-snake-r1175/

[3] J.-F. Yeh, P.-H. Sun, S.-H. Huang, and T.-C. Chiang, "Snake game ai: Movement rating based functions and evolutionary algorithm-based optimization," Nov. 2016.

[4] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, Jul. 2012.

[5] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing.* Springer, 2007.

[6] L. Vanneschi, M. Castelli, and S. Silva, "Measuring bloat, overfitting and functional complexity in genetic programming," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 877–884. [Online]. Available: http://doi.acm.org/10.1145/1830483.1830643

[7] E. D. de Jong, R. A. Watson, and J. B. Pollack, "Reducing bloat and promoting diversity using multi-objective methods," pp. 11–18, 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=2955239.2955241

6 Appendix

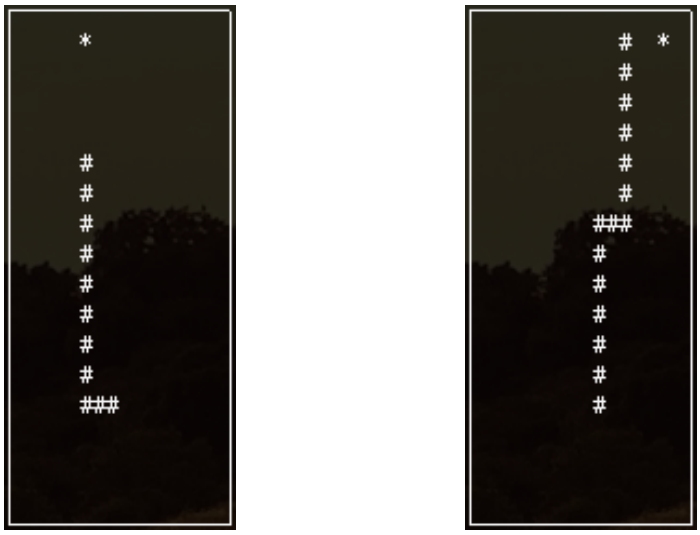

Figure 1: The 14x14 Game Board