MineNavigation:
Navigation tasks in a reinforcement learning framework

Author: Simone Azeglio

1.1 Navigation and Mapping

Navigation in unknown environments is a difficult open problem. The first difficulty of such an environment is that there is no a priori knowledge about it, and therefore a map can only be built while exploring.
One way in order to deal with this problem is to build an agent based model. Our approach considers using only visual information. The challenge for the agent is to acquire enough information about the environment (locations of landmarks and obstacles) so that it can move along a path from the starting location to the target.

1.2 Intelligent Agents

A major goal of artificial intelligence is to build *intelligent agents*. Perceiving their environment, understanding, reasoning and learning to plan, making decisions and acting upon them are essential characteristics of intelligent agents.

An *intelligent agent* is an autonomous entity that can learn and improve based on its interactions within its environment. An intelligent agent can analyze its own behavior and performance using its observations.

1.3 Learning Environments

The learning environment defines the problem or the task for the agent to complete.
A problem or task in which the outcome depends on a sequence of decisions made or actions taken is a sequential decision-making problem

1.4 The Platform: Project Malmo

Project Malmo is an AI experimentation platform from Microsoft which builds on top of Minecraft, a popular computer game. The platform is built keeping in mind recent advances in Deep Reinforcement Learning for Video Game playing; however, the project is meant to be very open ended allowing for

research into different topics such as Multi-Agent Systems, Transfer Learning, and Human-AI interaction.

1.5 Reinforcement Learning

Reinforcement learning is a kind of *hybrid way* of learning compared to supervised and unsupervised learning.

Reinforcement, as you know from general English (https://www.merriam-webster.com/dictionary/reinforcement), is the act of increasing or strengthening the choice to take a particular action in response to something, because of the perceived benefit of receiving higher rewards for taking that action. We, humans, are good at learning through reinforcement from a very young age. We could say: parents reward their kids with chocolate if these kids complete their homework on time after school every day. Kid learn the fact that they will receive chocolate (*a reward*) if they complete their homework every day.

2.1 Genetic Algorithms: a first strategy

A genetic algorithm (GA) is an example of "evolutionary computation" algorithm which is a family of AI algorithms that are inspired by biological evolution. These methods are regarded as a meta-heuristic optimization method which means that they can be useful for finding good solutions for optimization (maximization or minimization of a function, usually referred to as the *fitness function*) problems, but they do not provide guarantees of finding the global optimal solution.
In this framework we tipically refer to the term *chromosome* as a candidate solution to a problem, often encoded as a bit string.

2.2 GA Operators

The simplest form of genetic algorithm involves three types of operators: *selection*, *crossover*, and *mutation*.

*Selection**:*** This operator selects chromosomes in the population for reproduction. The fitter the chromosome, the more times it is likely to be selected to reproduce.

Crossover: This operator randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring. For example, the strings 10000100 and 11111111 could be crossed over after the third locus in each to produce the two offspring

10011111 and 11100100. The crossover operator roughly mimics biological recombination between two single−chromosome organisms.

Mutation: This operator randomly flips some of the bits in a chromosome. For example, the string 00000100 might be mutated in its second position to yield 01000100. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.01).

(Ref: An Introduction to Genetic Algorithms – M. Mitchell)

2.3 Hill Climbing Algorithm: another perspective

In the Hill Climbing technique we start with a sub-optimal solution of the problem which is going to be improved repeatedly until some condition is maximized.
The idea of starting with a sub-optimal solution can be compared, in an intuitively manner, as starting from the base of a hill, improving the solution to walking up the hill, and finally maximazing some condition to reaching the top of the hill.

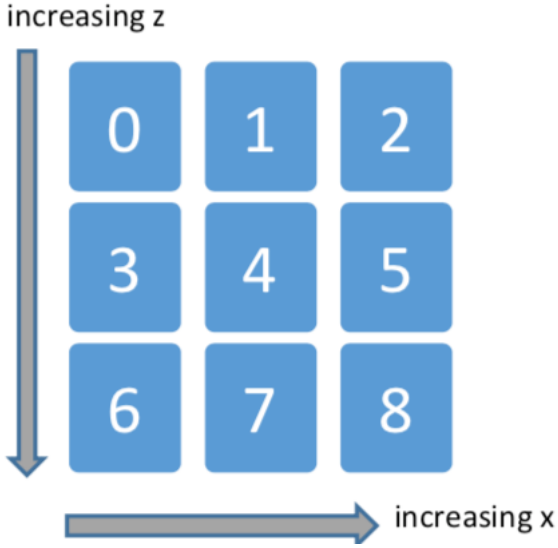
Hence the hill climbing algorithm can be considered as the following steps (pseudocode):

1) Evaluate the initial state
2) Loop until a solution is found or there are no new operators left to be applied:
    - Select and apply a new operator
    - Evaluate the new state:
        - Goal → Quit
        - Better than current state → New current state

2.4 Outlining our approach

This project teaches an agent to explore a contained but hostile environment. Our program currently uses the two local search algorithms defined in **2.1** and **2.3** to explore the environment. Each algorithm uses some heuristic functions to state which movement the agent is going to take. Those heuristic functions take as input the agent's observations.

- *Grid* : a 3x3 blocks' grid centered at the block below the agent



- *Entity_distance*: the distance between the agent and each entity on the map (e.g. Diamonds)

Each algorithm uses the following heuristics functions:

- *Random_direction*: the agent moves in a random direction
- *Towards_item*: the agent moves towards the closest diamond

The agent has to maximize the score (*fitness function*) and he can do it in two different ways:

1) By exploring new blocks
2) By collecting diamonds
3) By maximizing the covered distance (exploring new cells instead of being steady on the same one)

At each junction in the maze, the algorithm decides which heuristic function to use, and then returns the movement action generated by that heuristic.
The agent takes an action and when the mission is ended the score is evaluated. In our case the score is a function of the amount of time (*seconds*) that the agent survived and of the number of diamonds collected and of the distance between the agent and the diamond.
We decided to include time and distance in order to detect when the performance increases even marginally – in this way the score is even continuous.

In detail, the score is calculated as:

$$score = diamonds*50 + t + \sum_d log_2(d+1)$$

The mission ends automatically after 30 seconds. The algorithms considers the score and updates its heuristics selection according to that.

2.5 Hands on GA in Navigation tasks

Our genetic algorithm creates a "*generation*" of strings of heuristic functions. Each generation has 8 strings. A normal distribution determines randomly the length of those strings. In every run of the mission, the agent relies on those strings to decide in which direction he has to run. For each move, he looks at the next heuristic in the string (formally a list of chars) and take the move generated by that heuristics.
If the agent get to the end of the string, he starts over at the beginning.

The algorithm is updated with the score that the agent receives after each run. When the agent has gone trough all the strings in a generation, the algorithm takes the top 5 highest scoring strings, and generates 8 new strings. It combines the most high-scoring strings at a randomly chosen crossover point. There is even a mutation probability ($p = 0.05$) that a given heuristics in a string will "*mutate"* into a different heuristic. The algorithm keeps on repeating this procedure with new generations, ideally obtaining higher scores each time.

2.6 Hands on Hill Climbing in Navigation tasks

Even the hill cllimbing algorithm operates on a string of heuristic functions. First, the hill-climbing algorithm runs a mission using one of these strings. After getting the score from this string, the algorithm runs a mission for each *adjacent* string in the search space. An adjacent string is a string differing by only one of these operations:

1) Addition of a heuristic from the string
2) Removal of a heuristic
3) Change of a heuristic

When every adjacent string is evaluated, the algorithm chooses the string with best score. It then repeats the same operations in order to "climb the hill", getting ideally higher scores each time.

3.1 Diving into the World of Minecraft [1]

As previously reported, Project Malmo builds on top of Minecraft. The first thing to do is to launch Minecraft's client from terminal.
The first window looks like this:

While it runs is possible to execute a Python script in order to "let our agent play". Basically the agent has to look for a diamond by circumvent an obstacle: a wall.

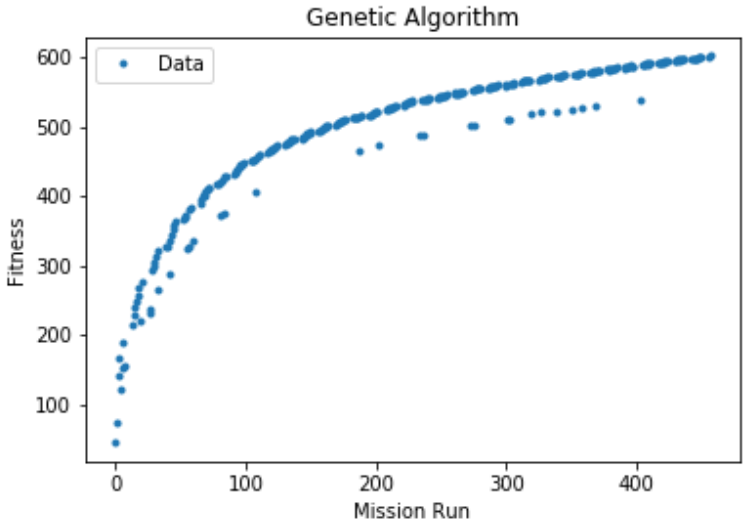
[1] For details, take a look at the official documentation of the project in supplementary materials

Our world is not huge at all, and after a few runs (each run takes 30 seconds) the agent should find the diamond.

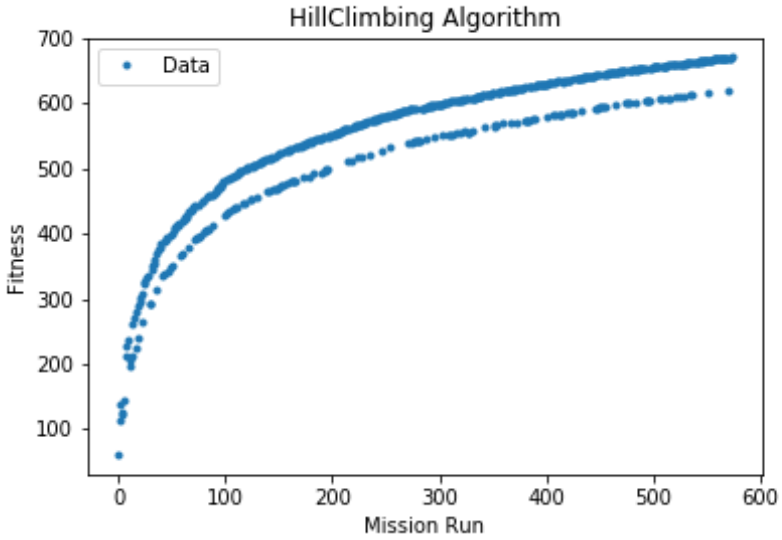
The learning process is not ended yet. The agent has to maximize the travelled distance by exploring new blocks.

3.2 A glimpse to Fitness plots

How do we quantify learning? The score function gives us a hand.
More formally the score is called *fitness* and it is a distinctive signature of the learning process. We plotted the fitness function for each algorithm against mission runs.



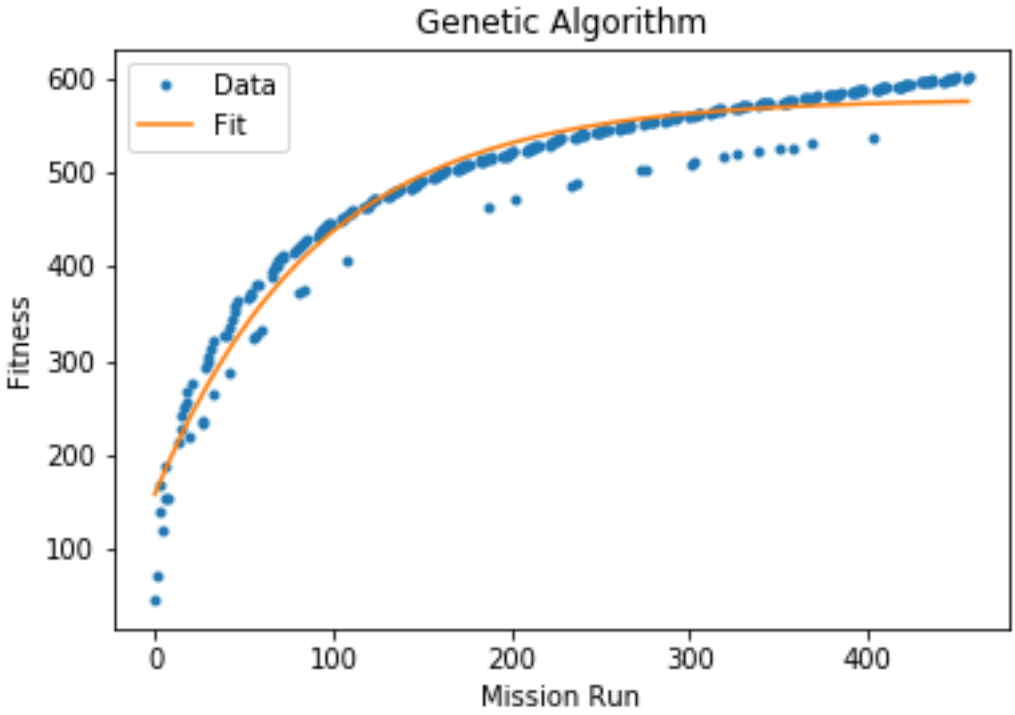
Regarding the genetic algorithm we can see that there are not many fluctuations (e.g. the agent does not find the diamond) and the algorithm is going to converge as soon as the agent has explored the whole space.

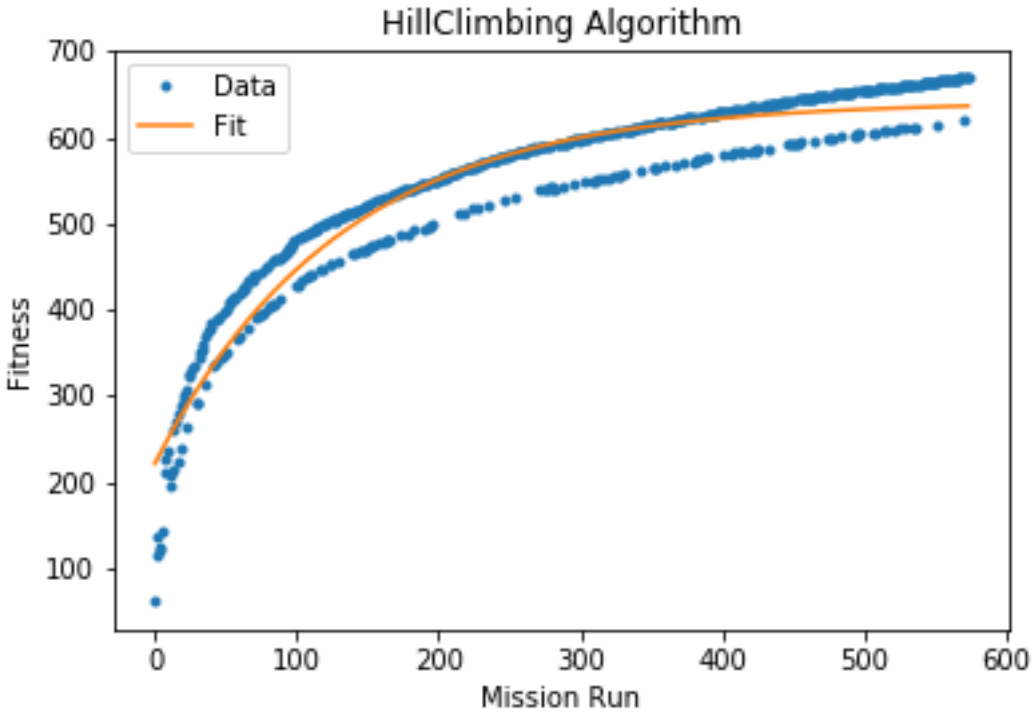


With respect to the hill-climbing algorithm we can notice that there are more fluctuations: this is related to the nature of local space searching in some way. Regardless to fluctuations the behavior is similar to the genetic one.

We can do something more in order to understand which algorithm converges faster: *curve fitting*.

We have fitted our curves with the following function: $y = a * (1 - b * e^{-c*x})$, by inserting the *maximum time fluctuation* (*2.0s*) as the error on the fitness function.





Respectively we have:

- *Genetic* a = $5.77 * 10^{2}$, b = $7.26 * 10^{-1}$, c = $1.09 * 10^{-2}$
- *Hill-climbing* a = $6.41 * 10^{2}$, b = $6.54 * 10^{-1}$, c = $7.70 * 10^{-3}$

Which basically means that the Genetic algorithm is faster.

4.1 Conclusions

In this paper we have described a toy model approach for exploration in reinforcement learning, and evaluated them on navigation tasks implemented within Minecraft.
Minecraft provides an attractive framework to develop visual versions of standard RL tasks.
We showed one example here, but the opportunity to translate other tasks that stress and highlight various learning abilities of an agent is a very exciting proposition for future work.
Among these possibilities one of the most exciting could be integrating Convolutional and Recurrent Neural Networks in order to exploit video frames from agent's first-person-view. In fact, this is one of the tools in autonomous driving research and more generally in autonomous mobile robot navigation methods. In this way a future direction could be developing models for Deep Reinforcement Learning problems.

5. References:

1) The Malmo Platform for Artificial Intelligence Experimentation, M.Johnson, K.Hoffman, T.Hutton, D. Bignell (IJCAI-16)
2) Exploratory Gradient Boosting for Reinforcement Learning in Complex Domains, D. Abel, A. Agarwal, F. Diaz, A. Krishnamurthy, R. Schapire (arXiv:1603.04119 [cs.AI])