Biped Walker Using Genetic Algorithms

Neethu Renjith*

The aim of this project is to use Genetic algorithms to generate a walker who can travel on two legs with it's body upright. The body and motion of a walker is described as an array of real numbers. The fitness of the walker is evaluated using simulations. A group of walkers are randomly initiated and the most successful walkers are used to generate the next generation of walkers. The effectiveness of the algorithm is measured in terms of the performance of the population and how fast it improves.Several optimizations to the genetic algorithm has been tried and their results are presented in this paper.

I. Introduction

The problem of walking upright on two legs is quite complex. It has taken several centuries for it to develop in nature. Several studies have tried to understand the gait, posture and has proposed metrics to evaluate biped walking motion [1]. The distance covered in itself is not a complete measure; If upright motion is not encouraged, the optimization might converge to more stable crawling motions. Complex stabilization algorithms are available and can be engineered to meet the specifics needs of a bot[2]. The genetic algorithm approach treats this as an optimization problem. Though a less intuitive approach, this can be more easily adapted to generate gaits for different robots, by introducing constraints into the simulation.

The two main components of the project are: genetic algorithms[3] [4] - which decide how to create the individuals in the next generation; and simulation - which evaluates how good any given individual is. The simulation I have developed is based on the physics engine Box2D and assumes that the walker has finished it's run when anything other than it's lower legs touch the ground. All walkers in a single generation are simulated at the same time, but they cannot collide with each other. These components are described in detail in the following sections.

II. Genetic algorithms

A. Overview

A walker is described as a set of numbers called genes, each of which is an optimisable parameter, which describe it's motion or body. I have chosen the following parameters to optimize:

$$gene = [MOTORS - TORQUE, HIP - SPEED, KNEE - SPEED, BODY - HEIGHT, BODY - WIDTH] \quad (1)$$

The first three parameters describe the motion and the other 2 describe the dimensions of the walker. These can be easily customized for a different bot. The limb sizes are kept constant for all walkers, since that made the problem hard to optimize.

1) *Initialization:* The algorithm starts with a randomly initialized set of individuals.
2) *Fitness evaluation:* Each individual is evaluated using a fitness function which runs a simulation based on box2d.
3) *Selection:* The individuals with least fitness are removed
4) *Crossover:* The remaining individuals are randomly sampled with probability proportional to their fitness. The samples are interpolated to generate the next generation of individuals.
5) *Mutation:* Random exploration is introduced by adding some noise to the generation with a small probability.
6) *Iteration:* Repeat from step 2 on wards for a maximum number of iterations or till a good enough fitness is achieved.

B. Initialization

Each parameter is chosen from a given range. This simulates real life constraints; for instance the motor speed of the joint of a real robot would have limits. Gaussian and Cauchy distributions were tried and Cauchy is seen to give better results faster, perhaps because the Gaussian mostly tries the center values.

*Graduate student, Aero/Astro Department, Stanford University

C. Selection

Having a large number of individuals allows for a more widespread search. Simulating a generation of 20 (POPULATION SIZE) takes around 5 seconds on my system and increasing the number to 30 led to arbitrary stalls, so a balance has to be found . Maintaining 10 individuals in a population and selecting 7 (SAMPLE SIZE) of them have given me good convergence reasonably fast. I have used truncation selection, where the individuals are ordered according to fitness and less fit ones are trimmed. I have also tried tournament selection, where SAMPLE SIZE number of individuals are randomly selected multiple times and the fittest one from each sample is kept.

D. Crossover

These are methods to combine 2 genes to get 1 gene. I have tried the following crossover techniques: single-point, two-point, uniform and interpolation. None of them were especially better than the other.

E. Mutation

The gene is slightly modified art random. Exploration is increased by increasing the mutation rate. For real valued numbers, the general mutation used is to add gaussian noise to the entire gene. This gives reasonable good results. For boolean genes there is bitwise mutation, where bits are flipped with a low probability. I tried a bitwise gaussian approach where gaussian noise is added to a gene value, with low probability. The probability used was (1- 1/length of gene). This is because gassian mutation was adding too much randomness to the exploration and taking away from exploitation as can be seen from the fact that the best in a population oscillates a lot.

III. Simulation

A. Overview

The simulation consists of the world, which is the surface on which the walkers move and the walkers themselves. The individual walkers cannot touch each other. They only interact with the world. All walkers have been assigned constant values for physical constants such as friction, restitution, etc. These could also be turned into optimisable parameters, and I did not want to make the problem extremely difficult to solve. The walkers update their states based on feedback from the physics engine, after each update of the world.

B. biped walker

The biped walker consists of a body and limbs. Initially I had added a face as well. But balancing a head on a neck in biped motion is in itself a challenge and it led to a lot of instability. The motion of the limbs is controlled by a state machine, which decides when a limb should change direction, depending on the current state of the walker. How the leg moves is dependent on the walker, ie it's controlled by the gene. The state of the walker is described as

$$state = \begin{pmatrix} hullanglespeed, \\ angularvelocity, \\ horizontalspeed, \\ verticalspeed, \\ positionof joints, \\ jointsangularspeed, \\ legscontactwithground, \\ 10 lidarrangefindermeasurements \end{pmatrix}$$

and the walker is described by genes as
$gene = [MOTORS - TORQUE, HIP - SPEED, KNEE - SPEED, BODY - HEIGHT, BODY - WIDTH]$

The statemachine tries to mimic human motion, in that it coordinates arm movements with the opposite leg, the legs move one after the other, etc.Please refer to github code for more details .The lidar range measurements are from a lidar located in the lower body of the walker, these help adjust the state machine according to the terrain. This can be easily adjusted to include other sensors for a custom bot. This repository was used as reference.

C. Environment world

The world consists of a smooth wavy terrain. The walkers are created and stored as part of the world. For each simulation, the world is created once and the walkers from each generation starts at the beginning of the terrain. I've created a fixed length of the terrain, resulting in an upper limit for the maximum reward a walker can earn. After each rendering of the world in the physics engine, the state of the walkers are read.The state machine actions update the walker and the cycle continues till all the walkers have stopped.

D. Stopping conditions

Stopping condition for a generation is when all the walkers in the generation are done. A walker stops if one of the following happens:

- Any body part other than lower legs touch the ground. According to the current state machine, a walker cannot recover from this.
- A walker hasn't passed it's improved it's position in the last 250 (tuned) renderings. The position considered is the x- coordinate of the hull. This removes walkers who are travelling backward or are stuck in some kind of oscillatory motions
- A maximum number of renderings have passed

IV. Optimizations

These are some of them optimizations I thought of and the reasoning behind them.

- For crossover, the parents are chosen based on a probability distribution proportional to their fitness. This would ensure that the fitter genes are more likely to survive.
- When the previous generation has not improved the global best performance, the gene corresponding to the global best is inserted back into the population, before selection. This would bias the population towards a possible optima. This also reduced the oscillation of the best per population.

V. Results

I have used average, best and minimum reward in a population to evaluate the algorithm. Following are the plots for different runs using different approaches to each component of the genetic algorithm such as crossover, mutation and selection.

From the plots we can see that:

- Running the simulation for a long time generally doesn't provide much improvement.
- Increasing the population above a limit doesn't give much better results but takes very long to run
- Following combination works well: (Truncation, interpolation, bitwise gaussian)

VI. Conclusion

Through this project I've used genetic algorithms to solve a problem with real world implications. For mass production of customized bots, this could prove very useful. Since the code has been appropriately separated into simulation and optimizer, it can be easily adapted to an entirely different problem with little modification. Though a of more tuning is left to be done, I've implemented and tested out a large combinations of approaches and has found a reasonably good combination. The code can be found here: github code. A video of the algorithm learning can be found here: link.

Acknowledgments

Thank you Mykel and TAs for an awesome course. You guys inspired me to take on a challenging project and do it completely for myself (thanks to the relaxed grading criteria). The developer of this simulation Rafael Matsunaga for the idea.

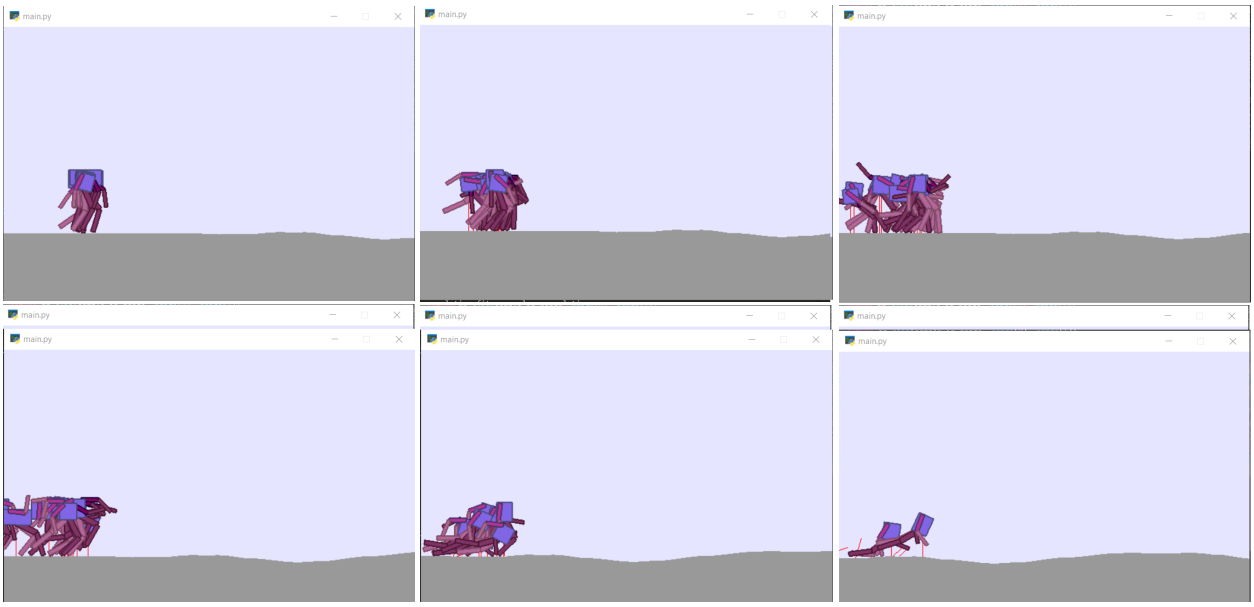
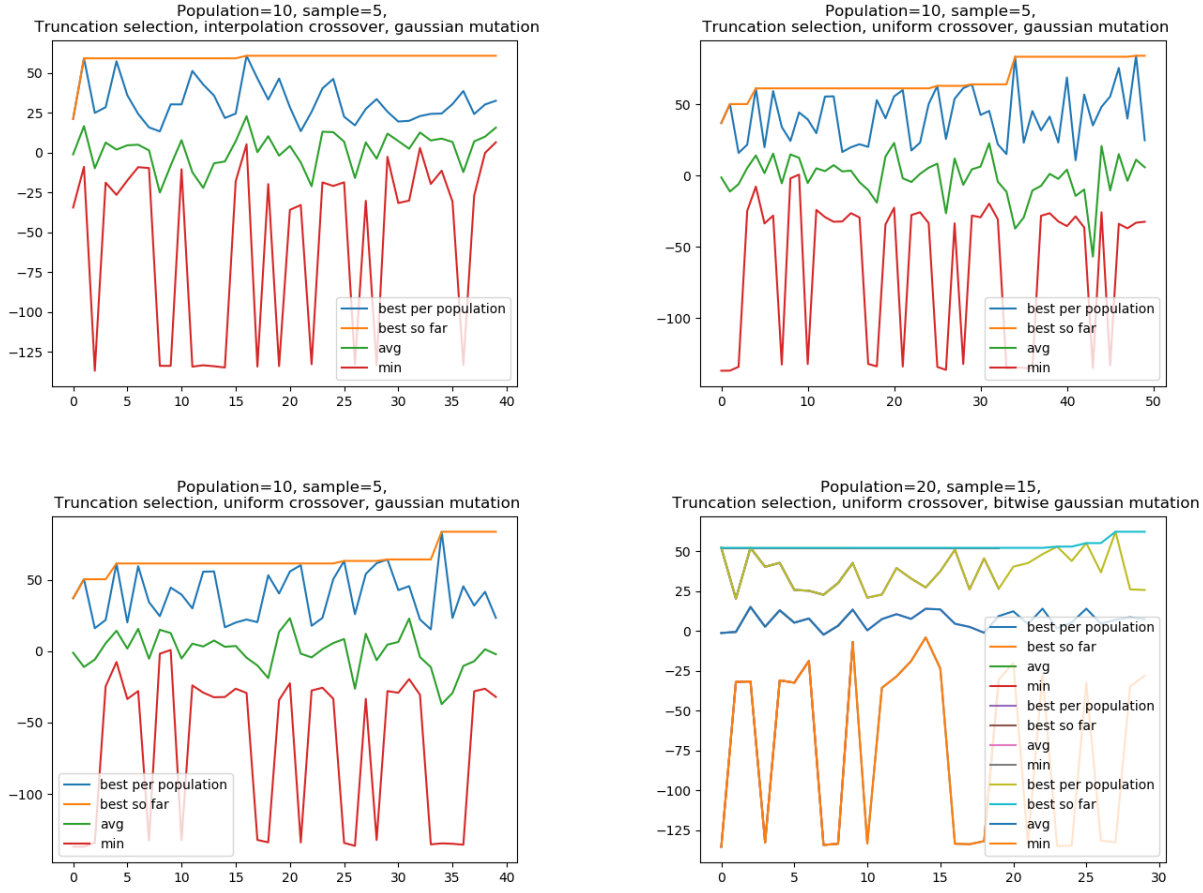
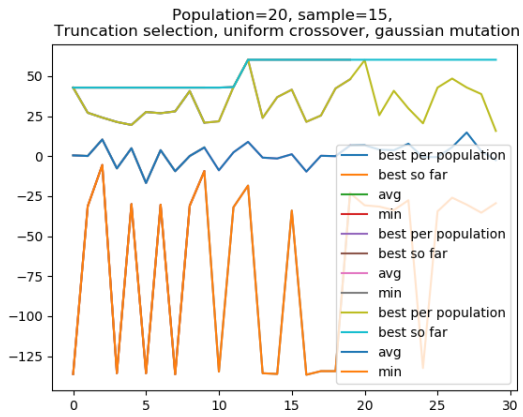


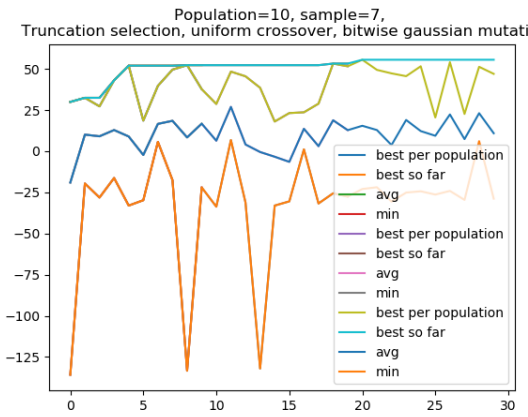
Fig. 1 The figures above show the rendering of a population in the simulation. I've also put up the videohere

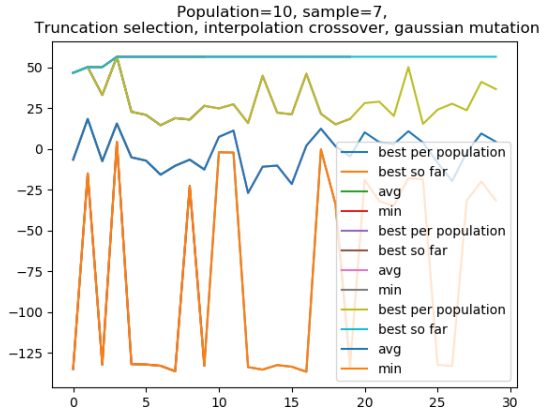


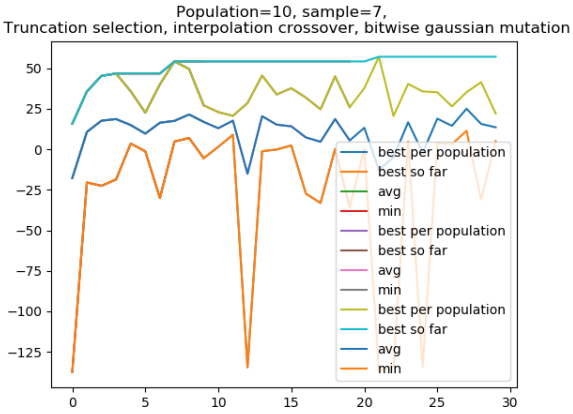
References

[1] Shen, L. X. T. H. e. a., K., "Analyses of biped walking posture by dynamical-evaluating index," *Artif Life Robotics 23, 261–270 (2018)*, ????





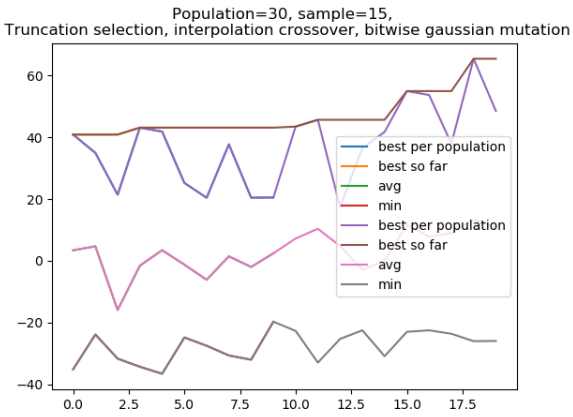


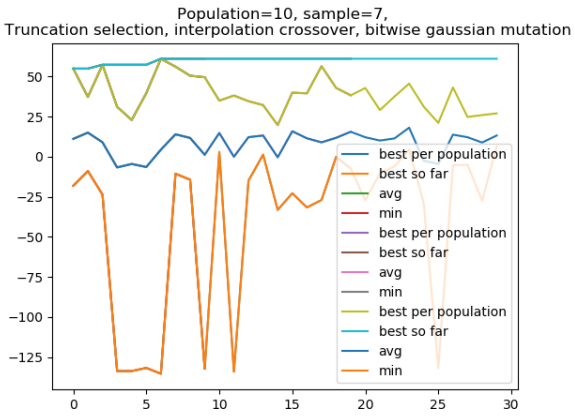


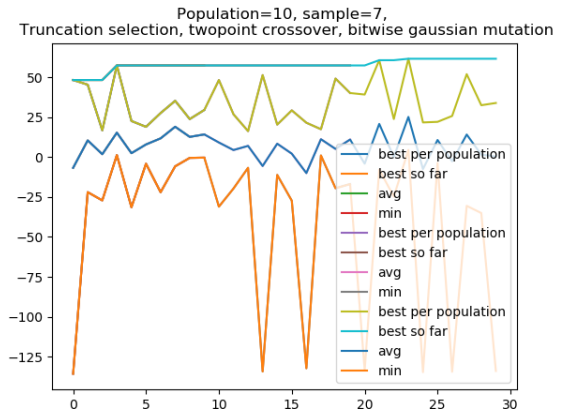
[2] Kim, J.-Y., Park, I.-W., and Oh, J.-H., "Walking Control Algorithm of Biped Humanoid Robot on Uneven and Inclined Floor," *Journal of Intelligent and Robotic Systems*, Vol. 48, 2007, pp. 457–484. https://doi.org/10.1007/s10846-006-9107-8.

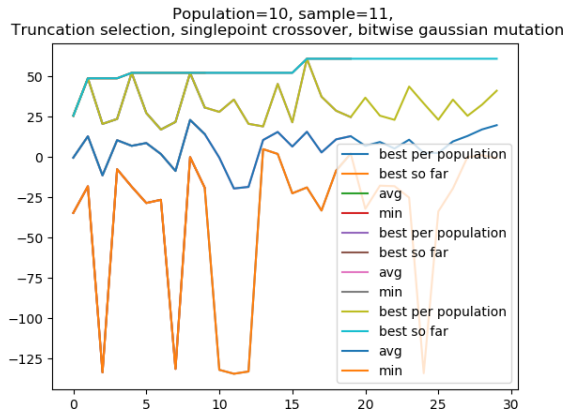
[3] Sheppard, C., *Genetic Algorithms with Python*, 1st ed., 2017.

[4] Kochenderfer, M. J., and Wheeler, T. A., *Algorithms for Optimization*, 2019.


Population=30, sample=15,
Truncation selection, interpolation crossover, bitwise gaussian mutation
best per population
best so far
avg
min
best per population
best so far
avg
min


Population=10, sample=7,
Truncation selection, interpolation crossover, bitwise gaussian mutation
best per population
best so far
avg
min
best per population
best so far
avg
min
best per population
best so far
avg
min


Population=10, sample=7,
Truncation selection, twopoint crossover, bitwise gaussian mutation
best per population
best so far
avg
min
best per population
best so far
avg
min
best per population
best so far
avg
min


Population=10, sample=11,
Truncation selection, singlepoint crossover, bitwise gaussian mutation
best per population
best so far
avg
min
best per population
best so far
avg
min
best per population
best so far
avg
min