Tristan Shores

ECE478

Thursday, October 28, 2012

Evolution of Simulated Millipede Gait

Introduction

The relative performance of crossover, mutation, double crossover, and double mutation genetic algorithms in generating an optimal cyclic forward gait for a simulated millipede was explored. A graphical software program written in C# simulated the motion of a millipede of varying physical characteristics (e.g. spine length, leg number, leg size, leg spacing). Matlab was used to create 3D plots that represent the performance of the tested genetic algorithms.

Millipede Leg Position Genome

A snapshot of a millipede's legs placement provides an angle (θ) for each leg. A 0° leg angle describes a leg perpendicular to the spine. A rotational limit of $\pm$ 20° was selected to avoid crossed legs. Forward leg rotation is represented by a positive angle. Each leg angle was expressed as a 5-bit binary string, termed a leg position gene. The concatenation of all leg position genes in fixed leg order results in a binary string, termed a genome. For example, if a millipede has 6 legs, a 30-bit binary string captures the millipede's leg positions. The initial/reset leg-state of a millipede is a 30-bit binary string of zeros (all legs are perpendicular to the spine i.e. straight out).

Millipede Motion

Millipede motion occurs by cyclic execution of the following two steps:

1. Rotational movements of each leg of the millipede (i.e. represented by a genome). The spine does not move, and is grounded.
2. A body motion (forward/backward/rotation), which resets each leg to perpendicular.

Step 1 represents the millipede reaching with its legs, and step 2 represents the resulting motion of the millipede’s body as it straightens out its legs.

The transformation of the legs rotations into various millipede body motions for a millipede with N pairs of legs of length L is determined as follows:

The forward motion of each leg (δf) is given by: $Lsin\theta$

Linear motion of left side of millipede (ΔF_{left}) is given by: $\frac{\sum_1^N \delta f_{left}}{N}$

Forward motion of right side of millipede (ΔF_{right}) is given by: $\frac{\sum_1^N \delta f_{right}}{N}$

Forward motion of the millipede occurs if: $\Delta F_{left} > 0 \; and \; \Delta F_{right} > 0$

Backward motion of the millipede occurs if: $\Delta F_{left} < 0 \; and \; \Delta F_{right} < 0$

Clockwise motion of the millipede occurs if: $\Delta F_{left} > 0 \; and \; \Delta F_{right} < 0$

Counter-clockwise motion of the millipede occurs if: $\Delta F_{left} < 0 \; and \; \Delta F_{right} > 0$

If the conditions for backward, forward, clockwise, or counter-clockwise motion are satisfied:

The magnitude of forward motion is given by: $\text{Minimum}(\Delta F_{left}, \; \Delta F_{right})$

The magnitude of backward motion is given by: $\text{Maximum}(\Delta F_{left}, \; \Delta F_{right})$

The magnitude of clockwise motion is given by: $\Delta F_{left} - \Delta F_{right}$

The magnitude of counter-clockwise motion is given by: $\Delta F_{right} - \Delta F_{left}$

Millipede Fitness

Millipede fitness was arbitrarily evaluated as the magnitude of forward motion. The fittest leg position genome has a +20° rotation for each leg, since a millipede that reaches forward with its legs to the greatest extent will move forward the furthest when its legs are reset. E.g. for a 6 legged millipede, the optimal gait was represented by the genome: "10100 10100 10100 10100 10100 10100" (gaps for visual clarity).

Note that a different fitness criteria could have just as readily been selected (e.g. backwards or rotational motion).

Evolutionary Process

The following steps describe the generalized evolutionary process:

1. Random rotational leg motions (genes) were generated for a millipede and the resulting genome placed in a genome pool. The process was repeated until a genome pool of N_{pool} genomes was created. The genome pool was then evolved over N_{gens} generations using a particular genetic

algorithm. The genome from the resulting genome pool with the highest genome fitness was recorded. This entire step was repeated 100 times and the average of the recorded highest genome fitness values was determined for the given N_{pool} and N_{gens}.

2. Step 1 repeated for every combination of N_{pool} (odd value) andN_{gens} (even value) in the domain: $3 \leq N_{pool} \leq 80$ and $2 \leq N_{gens} \leq 80$. The average highest genome fitness value for each unique combination of N_{pool} and N_{gens} was then plotted using Matlab.

3. Steps 1 & 2 were performed for each of the genetic algorithms:
    a. Random (no genetic algorithm applied to each generation).
    b. Crossover
    c. Double crossover
    d. Mutation
    e. Double mutation

Description of Genetic Algorithms

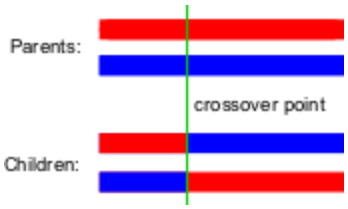
The application of each genetic algorithm followed the form:

1. Remove the parent genome with the lowest fitness from the pool. This results is a pool size of: $N_{pool} - 1$. This is an even number due to the odd values for pool sizes that were used.
2. Make a copy of the parent genome with the highest fitness.
3. Evolve each parent genome (in the case of mutation or double-mutation), or pair of genomes (in the case of crossover or double-crossover), using the given genetic algorithm. This does not change the pool size, which is: $N_{pool} - 1$.
4. Add in the parent genome with the highest fitness. This restores the pool size to: N_{pool}.

The particular evolution action on each genome is described below. In all evolutionary strategies, if the decimal value of any evolved gene in a child genome was outside the range $\pm$ 20, it was truncated to fall within that range.

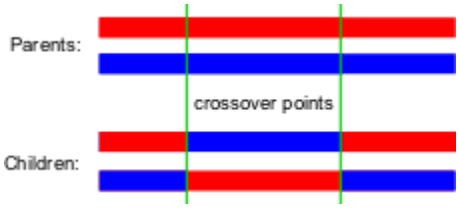
<u>Crossover</u>:

Two parent genomes were combined to create two child genomes as shown below.



Double crossover:

Two parent genomes were combined to create two child genomes as shown below:



Mutation:

A bit in the genome bit string was inverted at a random position in the bit string.

Double Mutation:

Two bits in the genome bit string were inverted at two random positions in the bit string.

Results:

The fitness value for each pool size and number of evolution generations are plotted below for each evolutionary strategy. The highest fitness value possible (fastest forward motion) was normalized to a value of 10.

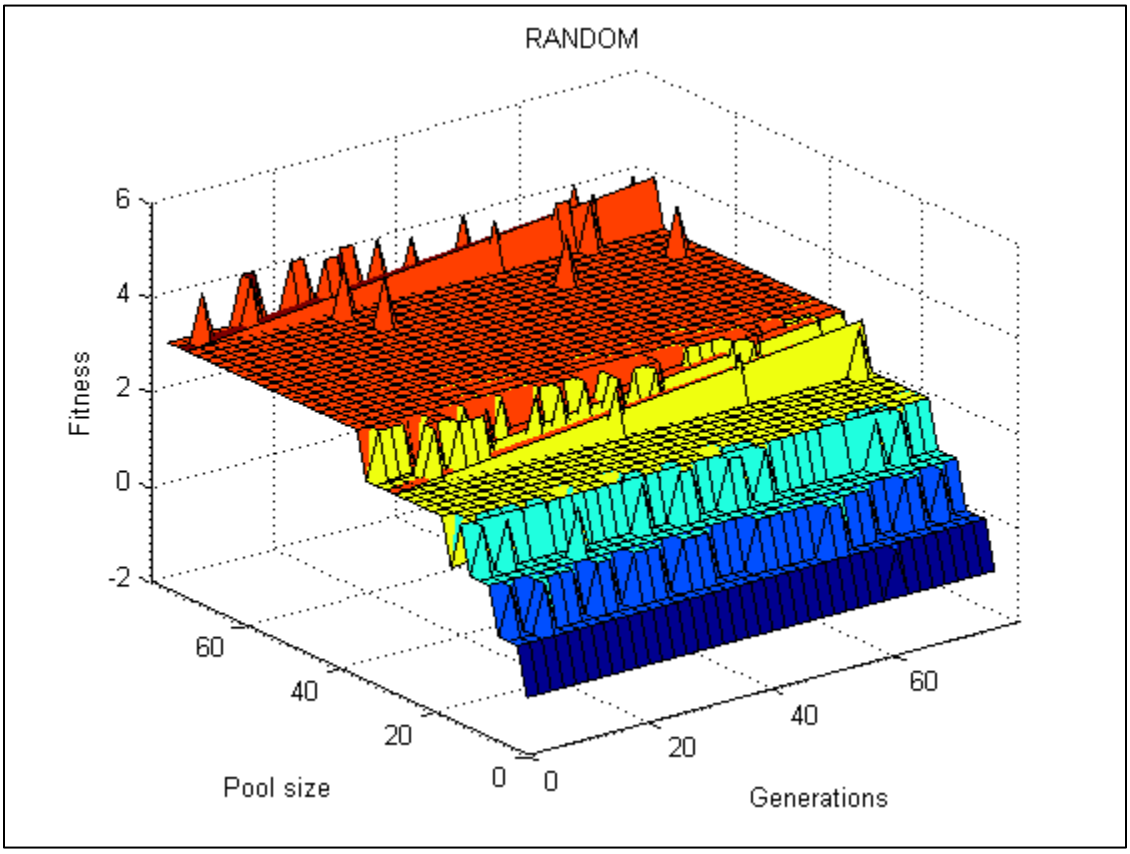
The minimum pool size and number of generations required to achieve the highest fitness value for each evolutionary strategy is summarized in the table below:

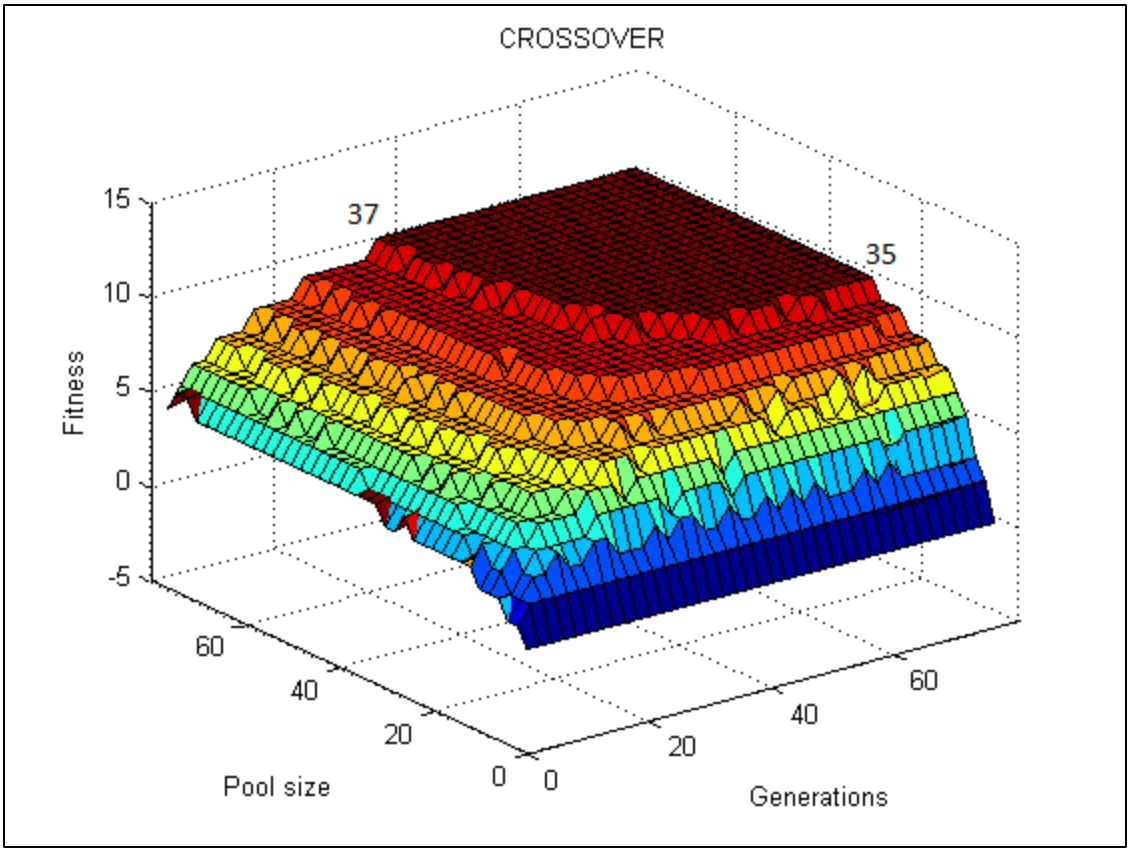
6-Legged Millipede	Random	Crossover	Mutation	Double Crossover	Double Mutation
Pool size	none	35	10	10	8
Generations	none	37	19	10	8

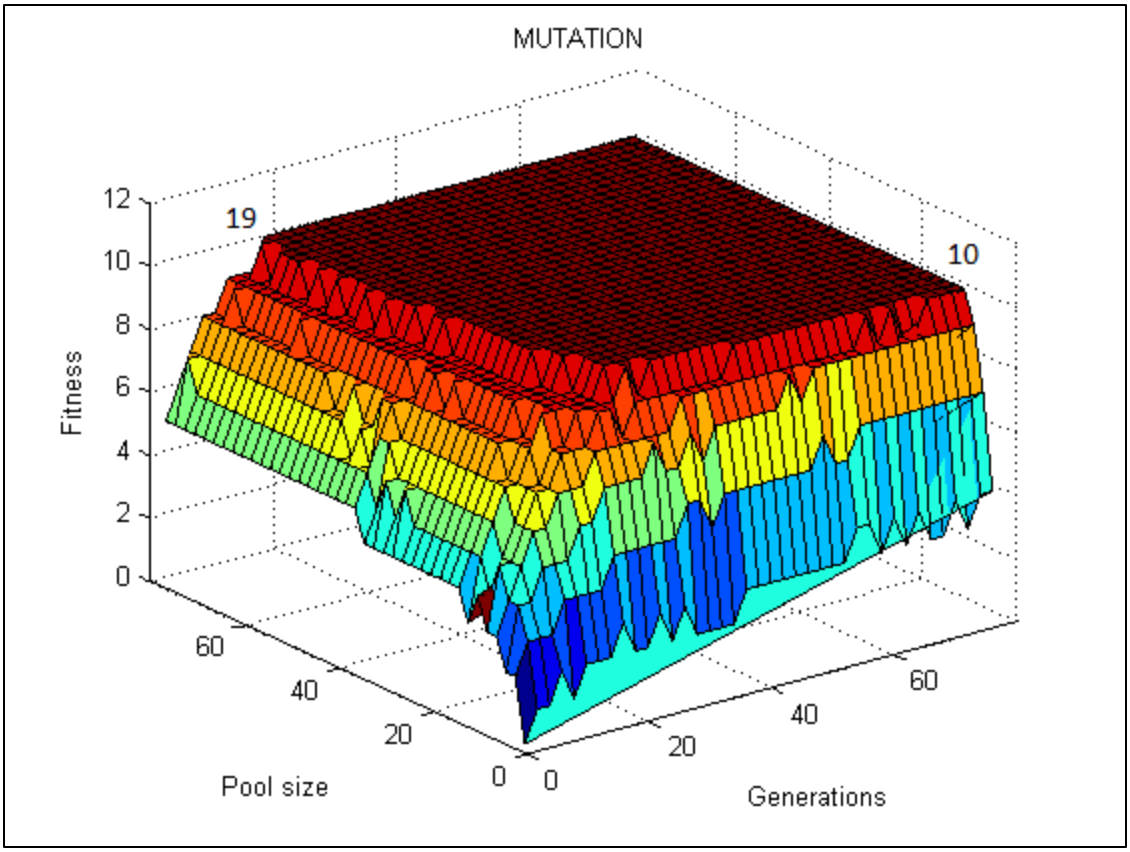
The most effective evolutionary strategy was double mutation, which required the smallest genome pool size (8) and least number of generations (8) to achieve optimal fitness.

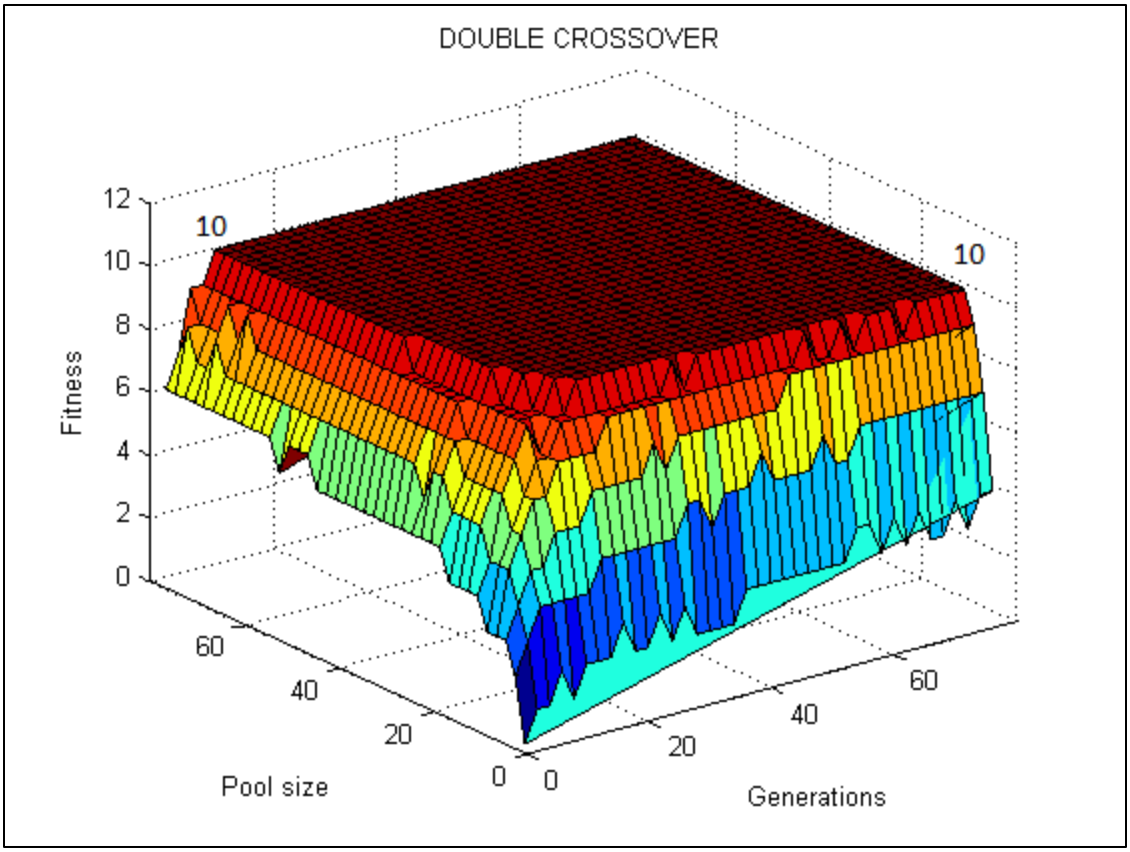
The least effective evolutionary strategy was random evolution, which never reached optimal fitness within the pool-size and generations domain.

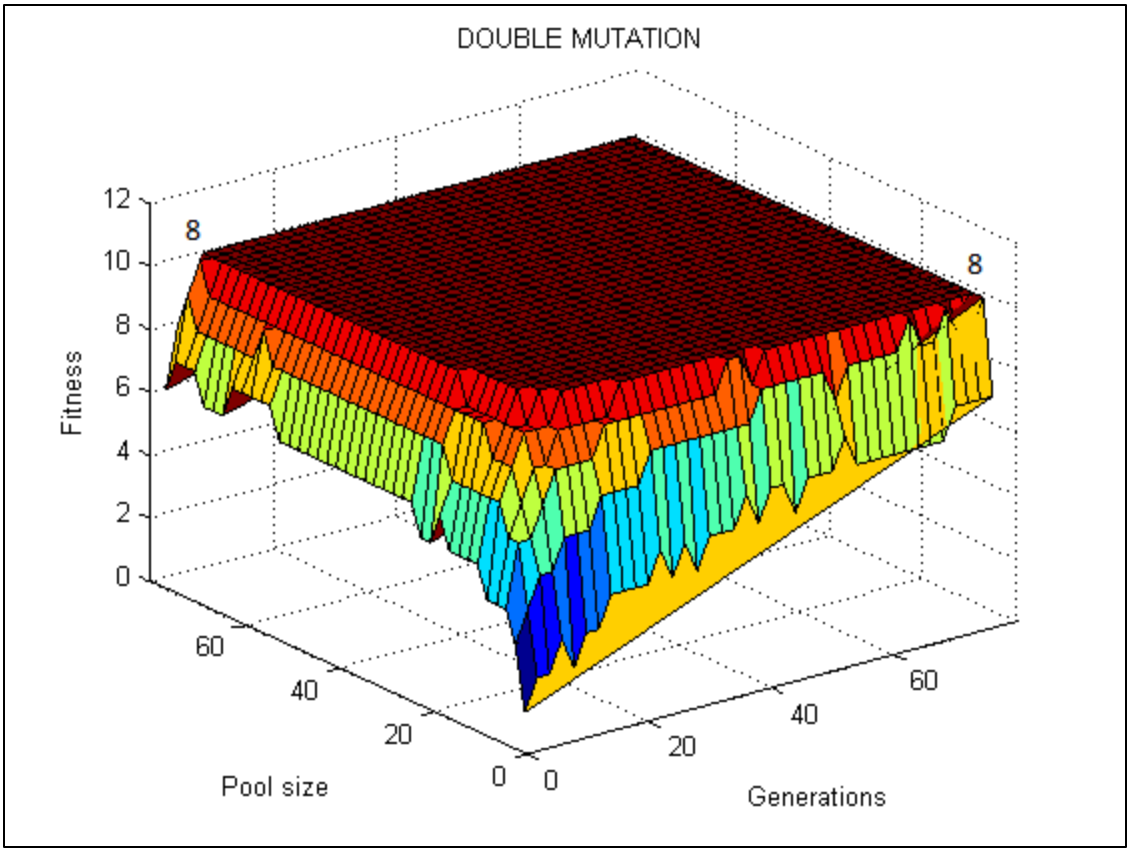
Matlab Plots For 6-Leg Millipede




CROSSOVER
37
35
15
10
5
0
-5
Fitness
60
40
20
0
Pool size
0
20
40
60
Generations


MUTATION
19
10
12
10
8
6
4
2
0
Fitness
60
40
20
0
Pool size
0
20
40
60
Generations


DOUBLE CROSSOVER
Fitness
Pool size
Generations
10
10
0
2
4
6
8
10
12
0
20
40
60


DOUBLE MUTATION
Fitness
Pool size
Generations
8
8
0
2
4
6
8
10
12
0
20
40
60

References

Braunl, T. (2008). *Embedded Robotics.* Berlin, Heidelberg: Springer-Verlag.

Crossover (genetic algorithm). (n.d.). In *Wikipedia*. Retrieved October 24, 2012, from http://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

Luger, G. (2009). *Artificial Intelligence.* Boston, MA: Pearson.