Generating Arachnid Robot Gaits with Cyclic Genetic Algorithms

Gary B. Parker
Department of Computer Science
Indiana University
Bloomington, IN 47405
gaparker@cs.indiana.edu
http://www.cs.indiana.edu/hyplan/gaparker.html

ABSTRACT

Learning gaits for walking robots is difficult because the elements of control require repetition of a sequence of integrated steps. Controllers for the six-legged robots, such as the Stiquito, and the eight-legged spider robots, in development, exemplify the difficulties of evolving gaits. The controllers for these robots store the actuator commands in a field-programmable gate array as a list of integers. This list must be ordered in the proper sequence of activations that, when continually repeated, produces a gait. This paper discusses the Cyclic Genetic Algorithm (CGA) used to evolve gait actuation lists, and discusses the extension of the CGA from use on the six-legged ant robot to the eight-legged spider robot. Optimal gaits for the six-legged were evolved in previous work, expanding the operators used in the CGA produced optimal gaits for the eight-legged robot.

1. Introduction

Robotic control presents an interesting problem for learning algorithms since it usually requires sequential solutions where a series of actions is continually repeated. Although learning to actuate a single leg for a single step is not difficult, learning to walk requires all legs to repeat steps in the correct sequence, and transition smoothly from one step to the next, which increases in difficulty when more legs are added and an optimal gait is required. Additionally, unpredictable changes in robot capability make adaptivity a requirement. This dynamic system suggests that some form of evolutionary computation should serve well as the learning algorithm. This can be especially true when the robot has a simple control interface where its accepted inputs are just a list of primitive activations. This is the case with the gait controllers of some primitive robots that possess hardware that can only store a list of integers which the controller can interpret and use to activate actuators. The coordinated pattern of the gait must be stored in this input integer list. In addition, it must contain the activations required to manipulate the state of the robot in such a way that allows smooth transitions while continually looping.

Gait generation for six-legged robots, has been addressed with several approaches. Donner (1986) developed control strategies for the SSA six-legged robot using decomposition of the task's elements and locality of control. He also developed a programming language, OWL, to provide real-time performance. Brooks (1989) used subsumption architecture in the development of his robots. Beer and Gallagher (1992) used a neural net to control a simulated cockroach and an actual hexapod robot (1994). These solutions proved to be successful for the robots used but we wanted a solution that could work on the most primitive of robots and that required less *a priori* knowledge of how to walk.

Spencer (1994) used genetic programming to develop gaits for a simulated robot using minimal knowledge about the intricacies of walking. Although his results were promising in that the system consistently learned gaits which generated sustained forward movement, we wanted a system that would learn the optimal gait for our model which had a very primitive control interface and no means of feedback. In addition, we wanted it to be continually adaptive to robot capabilities and be a quickly converging algorithm suitable for anytime learning as described by Grefenstette and Ramsey (1992). To this end, cyclic genetic algorithms were developed (Parker and Rawlins, 1996). They are capable of generating the proper sequence of primitive instructions that can be continually repeated to produce a gait. Tests showed that these quickly converging algorithms could produce optimal gaits on six-legged robot simulations.

Further tests done by Parker, Braun, and Cyliax (1997) showed that the gaits produced on the simulation could effectively be transferred to the actual robot. This was accomplished by training on a model loaded with specific information taken from an individual robot through careful measurement. Tests were done to see if the CGA could generate reasonable gaits from this model. Gaits were generated by running a CGA for 2000 generations on populations made up of 64 individuals (all sections of the starting chromosomes were initialized with a random number within the appropriate range). The results of training were verified by experimentation on the robot. Success was judged by measuring the distance traveled by the robot during physical tests. The results showed that the CGA was capable of generating useful gaits that were comparable to those developed by the robotics hardware technician.

Eight-legged robots offer advantages over six-legged. They have added stability during full speed forward movement since they can maintain four legs on the ground whereas six-legged can have a maximum of three. In addition, they are less susceptible to disablement caused by failing actuators (they can maintain static stability and near full speed forward movement with a leg or two missing).

Eight-legged robots also have disadvantages. They have more moving parts so they are harder to build and maintain. In addition, the additional legs add more degrees of freedom to control. This greatly increases the complexity of the learning problem. In this paper, we test the CGA with this increase in complexity by employing it to generate gaits for primitive eight-legged robots. We could not locate previous work where evolutionary computation was used for this purpose. The problem was found to be significantly harder than six-legged gait generation.

2. Method of Gait Generation

The general approach used to generate a gait for the simulated robot was to develop a model capable of representing all states of the robot and use a cyclic genetic algorithm to train this model to walk forward.

2.1. The Robot

The robot used in the formation of our simulated six-legged robot and model was the Stiquito II (Figure 1). Developed by Mills (1994) as an alternative to the larger, more complex and expensive six-legged robots; this robot has a very simple controller that can only store a sequence of numbers that it can cycle through producing activations for the 12 actuators. It is made up of a plastic body, steel wire legs, Nitinol wire actuators, and control lines that connect to the parallel port of an IBM compatible personal computer. Alternatively, control can be supplied by an infrared transmitter when the robot is in a colony cage which supplies its power while letting it maintain autonomous movement. The Nitinol actuators react to voltage applied by contracting; this bends the wire legs. When the voltage is 0 the steel wire legs act as a spring and extend the Nitinol actuators back to their extended (relaxed) position. There are two actuators per leg; one to lift the leg, one to pull it back. The activation of the 12 actuators is controlled by varying the signal passed through the parallel port. Its only inputs are activations, which the control interface uses to tell what Nitinol wires to contract. Due to the nature of Nitinol, the legs are always in motion unless the Nitinol is fully contracted or relaxed.

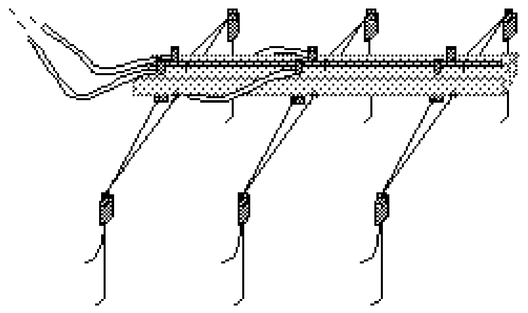

Figure 1: Stiquito II. Used for six-legged robot tests. An extension of the Stiquito is in development as an eight-legged robot.

An input to the Stiquito II is a 12 bit number where each bit represents an actuator (Nitinol wire). A signal of 1 contracts the wire and a signal of 0 relaxes it. Figure 2 shows an example of an activation and its result on the robot. The activation can be thought of as 6 pairs of actuations. Each pair is for a single leg with the first bit of the pair being that leg's vertical activation and the second being that leg's horizontal activation. The legs are numbered 0 to 5 with 0,2,4 being on the right from front to back and 1,3,5 being the left legs from front to back. The activation 100101101001 results, as shown (a solid bold leg means it is providing forward thrust by being on the ground and moving back and a dashed leg means it is moving up), in one phase of the classic tripod gait which is considered to be the optimal gait for speed in this simple rigid robot when all its actuators are fully functioning.

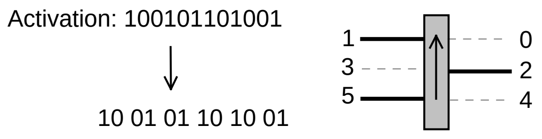

Figure 2: Six-Legged Robot Activation

The eight-legged robot is conceptual at this point, but is intended to be an extension of the Stiquito with a slightly longer body and two additional legs. Although the leg motion of this robot will not correspond closely to that of actual spiders, future arachnid robots, in development, will have the more spider-like pulling and pushing motion of the front and back legs. The CGA will be used in a similar manner to develop gaits for these arachnid robots.

The input to the arachnid robots is a 16 bit number (two per leg). The activations work in a similar manner as demonstrated in Figure 2. Figure 3 shows an example of an activation and its result on the robot.

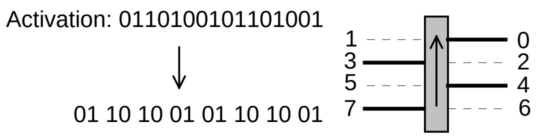

Figure 3: Eight-Legged Robot Activation

Some sequence of these activations will result in a pattern of leg movement that will produce a gait. An example of such a sequence, which would produce a viable gait for the eight-legged robot, is shown in Figure 4. The activations before the –1 are activated by the controller only once; the activations after are continually activated in a cycle until a new gait is loaded in the controller or the power is terminated.

```
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
                              -1
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0
0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1
0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
```

Figure 4: Sequence of Activations to Control Robot

2.2. The Model

Each of these robots was represented as a model that was intended to capture the importance of the robot's structure in regards to walking. The model was a simple data structure that held each leg's capabilities and current state (Figure 5).

```
Fields specific for each leg:
   current up -- current vertical position of the leg.
   max up -- position off the ground when completely up.
   max down -- posit off the ground when completely down.
   current back -- current horizontal position of the leg.
   max back -- posit relative to completely forward when
               completely back.
Fields applicable to all legs:
   rate up/down -- rate of vertical movement when actuator
               activated.
   rate back/forward -- rate of horizontal movement when
               actuator activated.
```

Figure 5: Simulation Model Data Structure

The current up and down fields hold the leg's state. The combination of all the leg's states defines the robot's state. The remaining fields in the data structure define the leg's capabilities, which were measured before training and remained static throughout. Input activations used the capabilities to determine how much to change the current state.

2.3. Robot Stability

In these tests, we were assuming a requirement for static stability. This means that the legs on the ground are sufficient to support the body even if it is in a resting state. This is not possible for bipeds and quadrupeds during normal gaits. Static equilibrium can be measured by drawing a convex polygon around the insect connecting consecutive feet on the ground (Figure 6). The stability margin is the distance from the insect's center of gravity to the nearest line in the polygon (Ting, Blickhan, and Full, 1994). As long as the stability margin is above 0 the animal is statically stable.

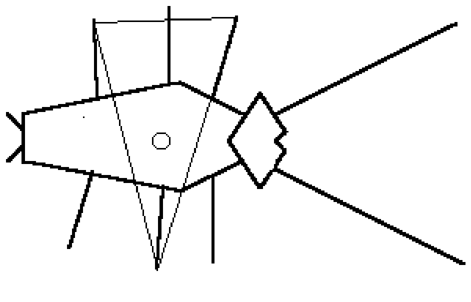

Figure 6: Hexapod Static Stability

Our calculations were not made to this degree of accuracy. Stability was generalized to be determined by which legs had their feet on the ground instead of by each leg's position when its foot was on the ground. For the six-legged robot, the possibilities could be easily delineated within a few cases. Determination of robot stability became a bigger issue in attempting to balance a spider-like robot. Some method was needed to determine the robot's balance depending on the feet on the ground. With eight legs, there are many possible configurations that result in static stability, but some offer greater balance than others. In addition, we wanted to distribute the support as much as possible around the center of gravity. Having the widest spread of legs on the ground would be the most stable, but no gait can keep all the extreme legs (0,1,6&7) continuously on the ground. The method developed to determine stability was to use pitch and bank indicators for each leg (Figure 7).

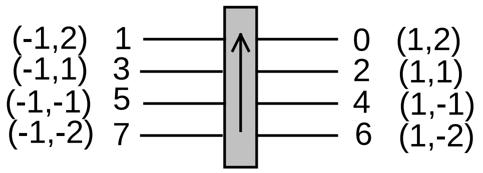

Figure 7: Robot Stability Bank and Pitch

The first number in each pair assigned to each leg is the bank (side to side balance) whereas the second number in each pair is the pitch (front to back balance). Addition of these numbers will give an indication of the robots balance. For example; if legs 0, 3, 4, & 7 are on the ground as shown in Figure 3, the addition of the banks would be 0 since two legs are down on each side and the addition of the pitches will also be 0, indicating even distribution left to right and forward and back. This method, in addition to other indicators such as if a balanced pair (examples: 0 & 7 or 3 & 4) existed, was used to determine robot stability.

3. Cyclic Genetic Algorithms

The CGA is a variation of the classic GA introduced by John Holland (1975). The chromosome of a classic GA is made up of genes; each gene or a combination of them usually represents some characteristic of the individual. The CGA incorporates time into the chromosome structure by making each gene a task that is to be accomplished in a set amount of time. Figure 8 shows such a chromosome. It represents a sequential program that can be executed in order: task 1, then 2, etc. In addition, we can take some portion of the chromosome and repeat the tasks; creating a cycle.

This allows the chromosome to represent a program that has a start section and an iterative section. The start section can contain tasks required to initialize while the iterative section can contain the tasks that are required in a continuous cycle. A trailing stop section can also be added to effect a smooth transition back to the at rest state.

Each gene (which represents a task) can be as simple as a primitive command (machine instruction or robot actuation) or as complicated as a subprogram. The gene can also be a cycle in itself as long as there is some provision for when to

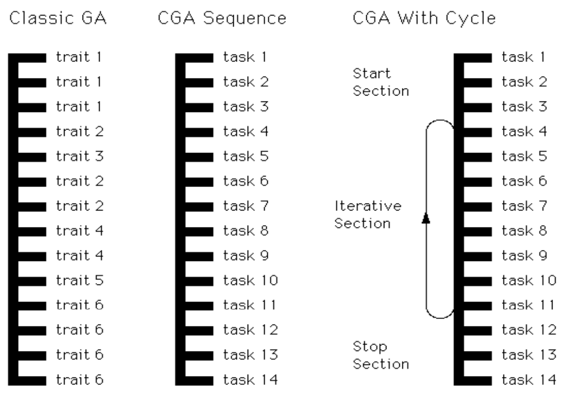

Figure 8: GA and CGA Chromosomes

stop that cycle and head to the next (which will be the next gene). Additional control genes can be added that globally effect the task genes by coordinating or inhibiting certain behaviors. The use of these will become more apparent as we discuss the specific application.

CGAs can have both fixed and variable length chromosomes. In either case, the system must be able to allot the proper number of tasks to each phase and be flexible enough to allow the CGA to form a complete cycle in the iterative section. When variable length are used, the genetic operators (usually crossover) provide the means for variation in the chromosome length. Some means of control is usually required because drastic changes in the chromosome length can severely impede convergence. When fixed length are used, the tasks at each gene can be repeated; the number of repetitions is encoded in the gene. In this way, fixed length chromosomes can take on the desirable characteristics of variable yet maintain the increased control of training fixed.

An example of the use of repetitions to form a variable length cycle in shown in Figure 9. Although the iterative section only has five possible distinct tasks, the number of times each is repeated can vary the total number of tasks in

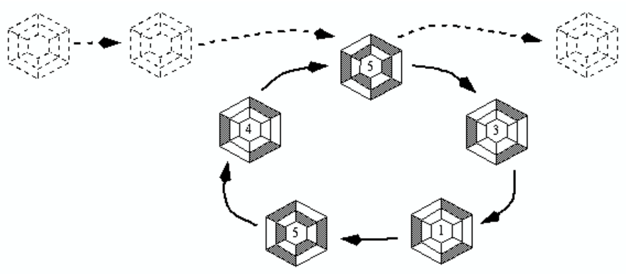

Figure 9: Fixed Length Cycle

the iteration. In this case, the first task is repeated 5 times, the second 3 times, etc. The result is a cycle of 18 tasks even though the iterations part of the chromosome is fixed at five genes. The figure also demonstrates that, although genes are defined, they may not be used. The two start section genes and one stop section gene have their task repeat numbers set to 0 making no tasks done before or after the cycle.

The genetic operators used by CGAs are generally variations on the classic GA operators and being chromosome specific will be described in detail in the next section.

4. CGAs Applied to Gait Generation

The CGA is particularly well suited for gait generation. For this application we used chromosomes with only a start and iterative section. The start section was executed once. This allowed the robot to move from its start stance, which is only used while at rest (no motion). The start section set up the robot to move into a continuous cycle where sustained fluid motion could exist. The iterative section was repeated until a total of 100 activations was reached. The trailing stop section was not required as the Nitinol will relax to the at-rest position when activation inputs stop.

4.1. CGA Chromosomes

Although initial experiments with the six-legged robot used both fixed and variable length chromosomes (Figure 10); the fixed were found to be the best and were the only kind used in the eight-legged robot experiments.

Variable Length

(A_1 A_2 -1 A_3 A_4 A_5 A_6 A_7 A_8 A_9)

A	Activation
C	Coordinators
I	Inhibitors
R	Repetitions

Fixed Length, Global Inhibitors & Coordinators

(C I ((A R)) ($(A\ R)_1$ $(A\ R)_2$ $(A\ R)_3$ $(A\ R)_4$ $(A\ R)_5$ $(A\ R)_6$
$(A\ R)_7$ $(A\ R)_8$ $(A\ R)_9$ $(A\ R)_{10}$ $(A\ R)_{11}$ $(A\ R)_{12}$))

Figure 10: CGA Chromosomes

The variable length chromosomes were the most primitive with each gene being an activation. The chromosome structure as shown in Figure 10 uses A_x to represent separate activations, the -1 separates the start and iterative sections.

The fixed length chromosomes (Figures 10 & 11) were a list consisting of the individual's global coordinators and inhibitors, a one gene start section, and a 12 gene iterative section. The start and iterative genes were each made up of two integers: the activations integer (12 bit number for six-legged and 16 bit number for eight-legged) and the repetitions integer (8 bit number).

The start section was made up of one gene and the iterative section had twelve. The start section could be fixed at one because, with the repetitions, the robot had sufficient time to get into the proper position to enter a cycle. Twelve in the iterative section was originally used because it was thought to be enough move changes to handle every possibility (two per leg for the six-legged). It was found that more than eight were rarely used during six-legged training, so twelve was also used for the eight-legged robot.

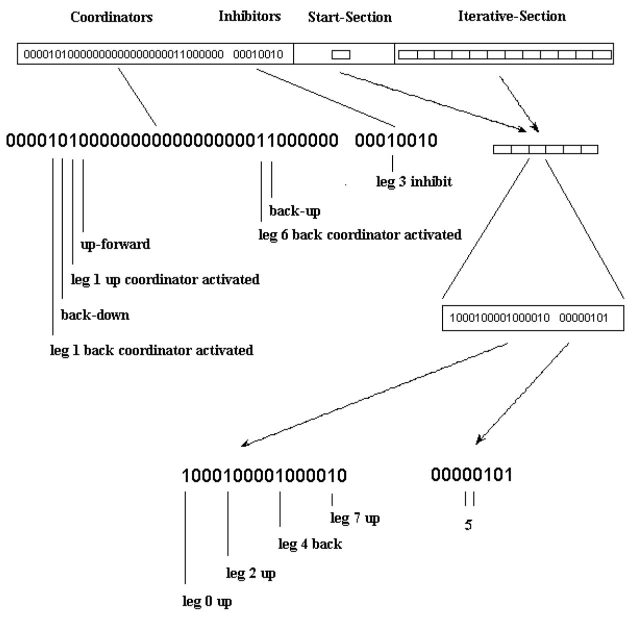

Figure 11: Fixed Length Chromosome, Arachnid Robot

4.2. Coordinators and Inhibitors

Coordinators and inhibitors were a part of the robot's coordination, which could evolve to increase leg control and proper movement. They were initiated as random numbers and were learned by the algorithm.

Coordinators for the six-legged robots were 12 bit numbers that directed the coordination of individual leg movement. This number could be looked at as six pairs of bits, one pair for each leg. The first being the back-down coordinator which, if activated, ensured that the leg would be down or moving in that direction if it was moving back. The second bit was the forward-up coordinator, which ensured that the leg would be moving forward if it was up.

Coordinators for the eight-legged robots were changed to 32 bit numbers (Figure 11). The possible coordinators were none, back-down, forward-down, up-forward, and up-back. Back-down ensured that the leg was going down if it was going back. Forward-down ensured that the leg was going down if it was going forward. Up-forward ensured that the leg was going forward if it was going up. Up-back ensured that the leg was going back if it was going up. Four bits for each leg were used: one determined if there would be a coordinator or not when the back actuator was activated or deactivated (leg going back or forward), a second determined if the back-down or forward-down coordinator would be used; the third bit determined if there would be a coordinator or not when the up actuator was activated (leg going up), the forth determined which up coordinator would be used. These new coordinators not only matched biological systems better, but also had the possibility of being more useful for other maneuvers such as backing up.

The inhibitors affected pairs of legs. They prevented pairs of legs from moving back at the same time. The 2,3 inhibitor prevented both legs 2 and 3 from going back at the same time. It allowed 2 to move back, but inhibited 3. The inhibitors for the six-legged robot were stored in a single 15 bit number (one bit per possible pair). This 15 bit number could be thought of as 5 groups. The first group made up of 5 bits indicated which legs would be inhibited from moving in the same direction as leg 0. Five bits were required to cover the remaining legs 1 through 5. The second group was made up of 4 bits showing what legs would be inhibited from moving in the same direction as leg 1. Since leg 0 had already been matched with all legs in the first group it does not appear in the second. This continues until all possible leg matchings have been addressed.

Although the paired inhibitors worked sufficiently well for the six-legged robots, they were too cumbersome to work well with the eight-legged robot. In addition, they were being initiated by a comparison of the activations of the pair of legs. A more accurate match with legged arthropods and more useful for a robot is to have the inhibitor activate depending on the position (on ground or not) of its adjacent legs. The inhibitors for the eight-legged robot used a much simpler system where the leg was inhibited from lifting unless its neighbors were on the ground (Figure 11). Each leg needed only one bit to determine its action. If the bit for the leg to be lifted was 1, all adjacent legs had to be on the ground for it to lift; if 0, than it would lift without checking its neighbors. These inhibitors could be represented by an eight bit number.

4.3. Chromosome Conversion

The fixed length chromosomes needed to be converted to the simplest form (variable length shown in Figure 10) before they could be used by the robot or its model. When using the leg pair inhibitors (used on the six-legged), first the inhibitors then the coordinators were applied to the activation in each gene. The result was a new listing with each gene being the new activation and its number of repetitions. Then the repetitions were applied and the -1 was added to separate the start and iterative sections. When the eight bit inhibitors were used (eight-legged), the inhibition could not be initiated until movement was being calculated since they used information concerning current legs on the ground. This meant that only the coordinators were added during preprocessing.

4.4. Genetic Operators

The probability for selection was based on fitness, which was computed by averaging the fitness of the individual genes after chromosome conversion. The gene fitness equaled the forward motion produced by the gene's signal. This was done on the model by:

1) taking the current state of legs
2) applying the vertical movement
3) calculating the balance and probable legs on the ground from the model's current vertical position of each leg
4) applying the horizontal movement to alter the leg's state, but only counting legs on the ground in computation of the movement (fitness)
5) taking off some deduction for lack of balance and/or asymmetry of movement
6) repeat using next gene and the new legs' state.

This was sequentially done from the start to the end of the string and then repeated as many times as required in the iterative section. Using this fitness, the best individual was preserved; the rest of the new population was formed by stochastic selection of mates with the probability of selection proportional to the fitness.

```
(A1    A2   -1   A3    A4    A5    A6    A7    A8     A9 )
 -----^          --------------------^              ^-----

 (B1    B2  -1   B3    B4    B5     B6   B7    B8     B9 )
       ^-----                      ^--------------------^

(A1      B2  -1   A3    A4    A5     B6    B7    B8     A9 )
```

Figure 12: Fixed Length Chromosome Crossover

Crossover (Figure 12) was done by randomly picking corresponding spots. In the start section a single point equivalent in both chromosomes was picked. In the iterative section, since it could be considered a circle, crossover was performed at two points; again equivalent positions in both chromosomes. The effect was to swap sections within the circle. Crossover with variable length chromosomes differed in that the selected points were not corresponding. This allowed the crossover to vary the length of the chromosomes.

$(\underline{00}00\ \ \underline{000}0\ \ ((\underline{A}\ R)_0)\ \ ((\underline{0000}0\ \underline{00}01)\ (\underline{00}00\ \underline{0}011)\ (\underline{A\ R})_3\ )\ )$

$(11\underline{11}\ \ 111\underline{1}\ \ ((B\ \underline{T})_0)\ \ ((111\underline{1\ }01\underline{11})\ \ (11\underline{01}\ 0\underline{100})\ (B\ T)_3\ )\ )$

$(0011\ \ 0001\ \ ((A\ T)_0)\ \ ((0001\ 0011)\ \ (0001\ 0100)\ (A\ R)_3\ )\ )$

Figure 13: Gene-by-Gene Crossover

An alternate type of crossover was a gene-by-gene crossover (Figure 13), which would perform crossover in each of the corresponding genes of the two chromosomes. In the case where these genes were represented as lists, crosses could happen between the individual members of the list or within the bits of the specific numbers in the list.

Two types of mutation were used: 1) Gene replace -- each gene had a random chance of being replaced by a new completely random gene. 2) Gene mutate -- each part of the gene had a random chance of having one of its bits altered. The mutation rate was controlled by temperature. The higher the temperature, the greater the chance of mutation. The temperature decreased as the generations of training increased. It was also effected by the diversity in the population. As each individual's fitness was calculated, a count was made to see how many adjacent individuals had the same fitness. This was used to increase the temperature when several fitnesses were similar.

4.5. Special Operators

Gene-by-Gene Evaluation was a clean up operator that randomly picked one or two individuals from the population on each set of trails and examined each gene one at a time. Genes were evaluated on the whole and move-by-move by comparing the previous move fitness to the present. Genes that were worse than a preset minimum were eliminated. Genes that were good in the execution of their early repetitions and subsequently dropped below a threshold in the later repetitions were modified by reducing their repetitions. Genes that had zero repetitions were moved out so that only active genes were at the start of the iterative section. Following these eliminations, if the number of genes or the total number of gene repetitions fell below some threshold, additional random genes were added until the thresholds were met.

Inhibitor/Coordinator DFS randomly picked one or two individuals from the population on each set of trails and performed a bit by bit limited depth modified depth first search of the individuals inhibitors or coordinators. A bit would be randomly picked and switched, the new individual would be tested and the better of the two selected for another step. This was repeated for 16 iterations.

5. Tests

Training was done for 2000 generations using population sizes of 64 individuals. These populations were initiated with randomly generated individuals. The individual's fitnesses were compared by testing their performance on models after training. Fitnesses were a calculation of the robot model's average forward movement per move after 100 moves. Each move equated to an activation applied for 100 msec. The fittest of the population's individuals was saved at 0, 10, 100, 200, 500, 1000, and 2000 generations of CGA training.

6. Results

Comparison of the CGA performance on the six-legged simulation robot using different chromosome structures at 2000 generations revealed that the fixed length chromosomes with inhibitors, coordinators, and the gene-by-gene operator were most effective in producing rapid convergence to an optimal gait (Parker and Rawlins, 1996). A fitness of 9.5 (optimal for this model) was attained (Figure 14). This was the result of a tripod gait, which was learned after only 500 generations.

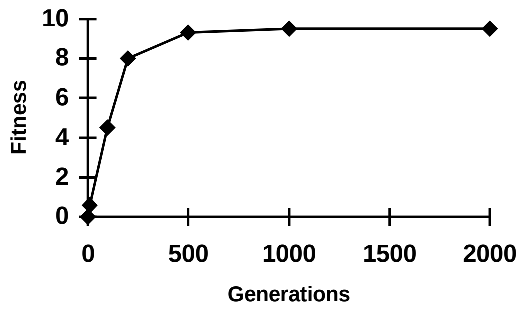

Figure 14: **Six-Legged Gait Learning**

The growth rate of the hexapod robot's fitness, as it learned, was very high in the initial stages. The CGA converged quickly to find a near optimal gait and it continued to refine the population's gaits until an optimal one (with a fitness of 9.5) was attained by generation 2000. Fitness improved as the system learned a gait that maintained stability while producing thrust with the legs on the ground. For a six-legged robot, static stability can best be maintained by keeping at least every other leg on the ground. A constant thrust can be attained, with this stability, by keeping exactly every other leg on the ground, producing thrust with these legs while repositioning the legs not on the ground, and then alternating the thrust and reposition legs; the tripod gait.

Convergence was slower for the eight-legged robot than it was for the six-legged. The CGA training on the eight-legged robot simulation resulted in average fitnesses of 9 (9.5 still optimal) by generation 2000. Figure 15 shows the progression of learning for this algorithm. The CGA initially learned quickly, then settled into a gradual climb to near optimal gaits by generation 2000.

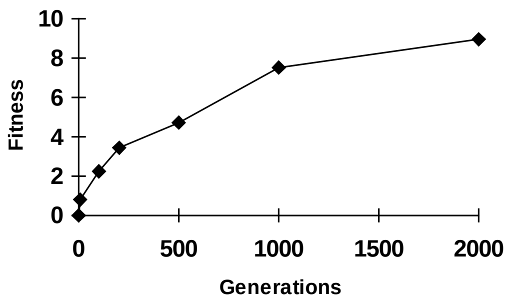

Figure 15: **Eight-Legged Gait Learning**

The difficulty with gait learning for the eight-legged robot is that there are several ways to maintain static stability while producing maximum thrust. Some of these alternative possibilities for an optimal gait have little in common when compared as they are represented in CGA chromosomes. This usually means more diversity in the population, yet a slower convergence. In addition, undefined gaits can be produced that meet both the stability and thrust requirements to be optimal.

As a result, these tests produced in an interesting variation of plausible gaits. Figure 16 shows 3 of these gaits. All showed near optimality in the robot simulation, but the far right one (what we call the quadripod gait) is most like the gaits employed by biological systems. In addition to these recognizable gaits, several tests resulted in nonstandard gaits that looked like random lifting and pulling of the legs, but resulted in near optimal forward speed. Tests on actual robots will reveal if these are viable gaits.

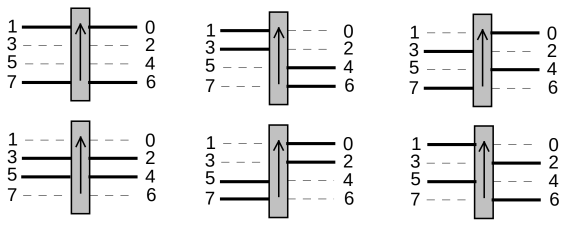

Figure 16: Eight-legged Gaits

7. Conclusions

The Cyclic Genetic Algorithm can, with virtually no knowledge of how to walk, produce gaits in models of six and eight-legged robots. With only minimal *a priori* knowledge; which went into the definition of the coordinators, inhibitors and the gene-by-gene evaluator; it is capable of producing the optimal tripod gait for six-legged robots and the optimal quadripod gait for eight-legged robots. In addition, the CGA produced output is in the primitive form required for direct input to the simplest of control interfaces.

The success we had with using CGAs to solve these specific applications gives us confidence that, through proper generalization, we can solve more complex repetitive tasks.

8. Future Research

Future research will apply CGA learning to generate gaits using a simulation of a spider like robot with tests conducted on the actual robot. Additional tests will be used to determine if the CGA is adaptive enough to alter the eight-legged robot gaits in response to leg disabilities. With eight legs, the CGA should be able to maintain almost optimal speed even after severe leg capability degradations.

Research is also planned in the incorporation of the anytime learning concept into actual six and eight legged robots by having the CGA constantly running in the background; recalculating the best gait as the robot's capabilities change.

Acknowledgments

This research was done at Indiana University's Adaptive Systems Laboratory directed by Jonathan Mills and was supported in part by NSF Graduate Research Traineeship Grant GER93-54898.

Bibliography

Beer, R. D., and Gallagher, J. C. (1992). "Evolving Dynamical Neural Networks for Adaptive Behavior." *Adaptive Behavior*, 1 (pp. 91-122). Cambridge: MIT Press.

Brooks, R. A. (1989). "A Robot That Walks: Emergent Behaviors from a Carefully Evolved Network." *Neural Computation* (pp. 254-262).

Donner, M. D. (1986). *Real-Time Control of Walking*. Boston; Basel; Stuttgart: Birkhauser.

Gallagher, J. C. and Beer, R. D. (1994). "Application of Evolved Locomotion Controllers to a Hexapod Robot." Technical Report CES-94-7, Department of Computer Engineering and Science, Case Western Reserve University.

Grefenstette, J. J. and Ramsey, C. L. (1992). "An Approach to Anytime Learning." *Proceedings of the Ninth International Conference on Machine Learning*, (pp. 189-195), D. Sleeman and P. Edwards (eds.), San Mateo, Ca: Morgan Kaufmann.

Holland, J. H. (1975). *Adaption in Natural and Artificial Systems*. Ann Arbor, Mi: The University of Michigan Press.

Mills, J. (1994). "Stiquito II and Tensipede: Two Easy-to-Build Nitinol-Propelled Robots." Technical Report #414, Computer Science Department, Indiana University.

Parker, G., Braun, D., and Cyliax, I. (1997). "Learning Gaits for the Stiquito." *Proceedings of the 8th International Conference on Advanced Robotics* (ICAR'97). (pp. 285-290).

Parker, G. and Rawlins, G. (1996). "Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots." *Proceedings of the World Automation Congress (WAC'96), Volume 3, Robotic and Manufacturing Systems*. (pp. 617-622).

Spencer, G. (1994). "Automatic Generation of Programs for Crawling and Walking." *Advances in Genetic Programming*. (pp. 335-353) K. Kinnear, Jr. (ed.), Cambridge, Ma: MIT Press.

Ting, L. H., Blickhan, R., & Full, R. J. (1994). "Dynamic and Static Stability in Hexapod Runners." *Journal of Experimental Biology*, 197. (pp. 251-269).