Partial Recombination for the Co-Evolution of Model Parameters

Gary B. Parker
Computer Science
Connecticut College
New London, CT 06320 USA
Email: parker@conncoll.edu

***Abstract* - Partial recombination is a type of crossover for genetic algorithms that focuses on a subset of the chromosome. It provides a means of doing crossover were only the genes involved in producing the fitness are affected. In this paper, we use it to evolve the parameters for a model that represents the capabilities of a robot. The values of these parameters are evolved as the robot periodically performs an action that is also being performed by the model. Each action performed by the robot does not include every possible turn command so using partial recombination allows the system to only change the parameters involved in the action. Tests show that partial recombination makes a significant difference in the co-evolution of model parameters.**

I. Introduction

One type of learning in evolutionary robotics uses evolutionary computation operating on a simulation (model) of the robot to learn the control of robot. In some cases, this evolutionary computation takes place in an offline learning system. The model has several measured or estimated parameters that are specific to the robot being represented. These parameters are initially made to be as accurate as possible, but due to changes in the robot and/or environment can be off significantly. The inadequacy of the model directly affects the outcome of the learning by the evolutionary computation. The Co-evolution of model parameters is a type of *punctuated anytime learning* used in evolutionary robotics to assist in offline learning. While the robot control is being evolved, the model parameters are also being involved to constantly provide the most accurate model.

A standard genetic algorithm [1] can usually be used to evolve the model parameters, but in some cases there is a difficulty. These cases are when not all of the parameters are used in any particular case. Some of the parameters may be of importance in this particular case whereas others are not used at all. The difficulty with this is that the genetic algorithm will evolve all of the parameters as if they all have the same importance. The parameters that should be evolving will be learned correctly, but the parameters that should not be evolved, which are referred to as hitchhikers, garbage bits, or freeloaders, will attain values that restrict them in later evolution. This is because they will no longer have the randomness that they had before evolution began. Another issue with the standard genetic algorithm, in this case, has to do with the schema within the chromosome. When two different genes within the same schema are separated by a large distance on the chromosome, the length of the schema is said to be long and the probability that crossover will disrupt this schema is high [1]. Often two different parameters of a model are closely related in that sense they should be within the same schema. Without prior knowledge of which of these parameters are closely related it's difficult to design the standard GA chromosome in such a way that these genes are close together.

Messy genetic algorithms [2] can help to take care of this linkage issue. Each gene in the messy genetic algorithm chromosome has a location associated with it. This allows genes within a single schema to become close to each other within the chromosome. Through this method, schema that would be long using a standard genetic algorithm can be short because the associated genes can be closer together. Messy genetic algorithms can assist in the parameter evolution by keeping associated genes close together avoiding disruption through crossover. However, one of their strengths in this case, is a weakness. The location of the genes, in addition to the value of the genes, is learned. This is a great benefit if there is no other information as to how the genes are related. In the case of parameters, however, there is often some runtime knowledge of how they are related. This is the situation with the parameters learning that we present in this paper. Although their relationship is not known before the evolution begins it can be guessed as the learning proceeds.

Partial recombination is a means of ensuring that genes related to each other are close together in the chromosome. In addition, it deals with the problem of losing the randomness of genes not used in previous evolution. In this paper we describe how partial recombination was used to greatly improve the results of the co-evolution of model parameters method employed to learn robot control in a simulated *area coverage problem*. Partial recombination was used with a standard genetic algorithm to evolve the parameters used by the model of a hexapod robot. Tests were done to show the progress of learning the desired control program with and without partial recombination employed in the genetic algorithm used to evolve the model parameters.

II. The Area Coverage Problem

The area coverage problem has been discussed in previous papers [3,4] and since it is not the primary topic of this paper it will only receive a cursory explanation. Area coverage is a type of path planning that is concerned with the coverage of an area. Some applications are mine sweeping, search and rescue, haul inspection, painting, and vacuuming. For the area coverage problem used in this research, the robot was to fully search, starting from a specific point (Figure 1), an area of specific width (180 cm). The simulated search was for mines that would be fully contained in the area. In order to detect a mine, the robot had to have the entire width of its body (excluding the legs), at its mid point, within the same 60x60 cm square as the mine. For test purposes, every 60x60 block had a mine placed in it. The robot's task was to find as many mines as possible while ensuring that no mines had been missed. The robot's movement was not restrained in any way by the environment. There was no physical constraint requiring it to stay within the mine area.

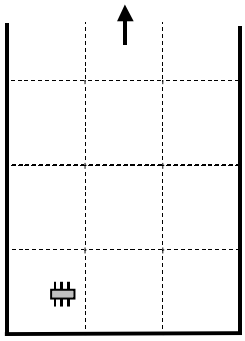

Figure 1. Search area for coverage. The area is unbounded in one direction. The robot is to execute a pattern of movement where it will not miss any squares as it checks each for simulated mines.

A. The Robot's Turns

Although this test is simulated, the robot is based on an actual robot and its capabilities. The robot is the ServoBot, a small (25x12cm; 25x24cm including legs), inexpensive hexapod robot with a BASIC Stamp II controller. The controller is capable of holding the program that can produce a cycle of activations required for 12 servo motors to move the robot's legs in a normal gait. In addition, it can hold a sequence of commands that result in a sequence of turns and straights performed by the robot.

A gait is produced by the controller sending signals (pulses) to the robot's actuators (servo motors). The control program includes a sequence of activations that the on-board controller will continually repeat. Each activation controls the instantaneous movement of the 12 servo actuators. A repeated sequence of these activations can be produced by an evolved neural network [5,6,7], a genetic program [8], or an evolved cyclic control program. In this work, we evolved a cyclic control program using a cyclic genetic algorithm [9] to produce an optimal gait for a specific ServoBot [10]. The gait generated for area coverage tests was a tripod gait. The tripod gait is where legs 0, 3, & 4 alternate with legs 1, 2, & 5 in providing the thrust for forward movement. While one set of legs is providing thrust, the other set is repositioning for its next thrust. In the case of the ServoBot used, the entire cycle (a gait cycle) lasted for 58 activations with each set providing 29 activations of thrust.

Differing degrees of turn were provided in the gait cycles through the use of *affecters*. These affecters could interrupt activations to the thrust actuators for either the left or right side of the robot. Since the normal gait consisted of a sequence of 29 pulses of thrust to move the leg from the full front to full back position, anything less than 29 would result in some dragging of the legs on that side. For example: a right side affecter of 7 would allow only 14 (2 x 7) thrusts on the right side while keeping 29 on the left. The result would be that the left side would move further than the right resulting in a right turn. Affecters from 0 to 15 (4 bits) were possible. 0 meant that side would get no thrust producing a maximum turn. 15 would not affect the normal gait so the result should be a straight track. A one bit indicator specified if the affecter was for the right or left.

Each gait cycle, made up of 58 activations, was assigned an affecter, which resulted in a turn throughout that cycle. For consistency, each gait cycle started with legs 0, 3, & 4 full forward and legs 1, 2, & 5 full back; all the legs were on the ground. As the gait cycle started legs 0, 3, & 4 would provide the thrust as legs 1, 2, & 5 would start to lift and move forward to reposition for their thrust after 29 activations. A single gait cycle was defined as being complete when the legs returned to their starting positions (in this case, after 58 activations).

The capabilities of the actual robot were measured and stored in a list of 32 gait cycles (Figure 2). Each element of the list could be identified by its gait cycle number (five bits). The high order bit described whether the turn would be left (1) or right (0) and the remaining four bits indicated the level of turn. The list (F T ΔH) of three numbers after that indicated the resultant turn after applying that gait cycle for one cycle. F was the distance in centimeters that the robot moved forward. T was the distance traveled left or right. Left movement resulted in a negative T, right in a positive T. ΔH was a measurement (in degrees) of the change in heading from the start heading the heading after execution of the gait cycles. Left was negative, right was positive.

```
( 0 (3.7 4.0 24.3))  (16 (5.0 -3.7 -26.7))
( 1 (3.7 4.0 22.2))  (17 (5.7 -3.7 -24.7))
( 2 (3.8 4.3 20.2))  (18 (6.0 -5.0 -22.2))
( 3 (4.8 4.3 18.8))  (19 (6.3 -4.8 -20.3))
( 4 (5.3 4.0 16.7))  (20 (7.3 -4.3 -18.8))
( 5 (6.5 4.0 14.7))  (21 (8.4 -4.3 -15.8))
( 6 (7.3 3.8 13.2))  (22 (9.6 -3.8 -13.5))
( 7 (8.1 3.5 12.2))  (23 (10.4 -2.8 -10.3))
( 8 (8.4 3.5 11.0))  (24 (11.1 -2.3 -7.0))
( 9 (9.4 2.8 8.3))   (25 (11.9 -1.5 -5.0))
(10 (10.1 2.3 6.2))  (26 (12.1 -1.5 -3.7))
(11 (11.4 0.8 2.7))  (27 (12.1 -1.0 -2.7))
(12 (12.1 -0.1 0.0)) (28 (12.4 -1.0 -2.3))
(13 (12.1 -0.5 -1.3))(29 (12.1 -1.0 -2.7))
(14 (12.4 -0.5 -1.8))(30 (12.1 -1.3 -2.3))
(15 (11.6 -0.8 -1.3))(31 (11.6 -0.8 -1.3))
```

Figure 2. The robot's capabilities stored in 32 gait cycles. The first number in each element of this list is the gait cycle number. When looked at as a five bit number, the first bit designates whether the turn is left or right and the remaining four bits designate the strength of turn. A strength of 0 is a maximum turn; a strength of 15 is no turn. The left column shows the right turn gait cycles and the right column shows the left turn gait cycles. The three numbers listed after each gait cycle number represent the robot's capabilities. They are the measured results of running that gait for one cycle.

B. Simulated Environment

The test area (Figure 1) was simulated by an *xy* grid where point (0,0) was the lower left corner. The lower right corner of the area was the point (180,0). The lower boundary was at $y = 0$, the left boundary was at $x = 0$, the right boundary was at $x = 180$, and there was no upper boundary. Mines were considered to be in 60×60 square blocks. The first row had centers at (30,30), (90,30), and (150,30). The second row started at (90,30), etc. The robot's start position was placed at (45,30) with an initial heading of 090 (Figure 2). This location assured acquisition of the first mine and put it in a good starting place to acquire the first row of mines. Motion was determined by applying each gait cycle from the chromosome one at a time. Using the current *xy* position and heading of the robot, a new position was calculated by applying the forward (*F*) and left/right (*T*) movements stored for that gait cycle as described in the previous section. The new heading was an addition of the current heading and the gait cycle heading change (ΔH).

C. The Complication in the Problem

The cyclic genetic algorithm (CGA) was developed to allow for the learning of a single loop control program for robot control [9]. They differ from the standard GA in that the genes of the chromosome are tasks that are to be completed in a set amount of time as opposed to traits of the solution. A CGA is capable of learning cycle of gait cycles needed for the controller of a simulated robot to produce a pattern of movement that will cover the area, but its answer is only as good as the model used for training. Although the model was produced through accurate measurements of the actual robot, natural changes in the robot's capabilities will quickly render this model ineffective. Only through continual adjustments to the model's parameters can the CGA continue to find an optimal solution. To simulate this complication and force the use of adaptive learning, the CGA was to learn the controller for a robot to do area coverage with an inaccurate model. A "correct" gait cycle list that differed from the one created through capability measurements was generated. It had each turn rate shifted to be off by one so that each turn strength was changed to be less than expected. Gait cycle *n* of the new list would be equivalent to gait cycle $n + 1$ of the old list. The max turns (strength 0 for both left and right) were thrown out. This new gait cycle list was used to simulate the actual robot. This simulated actual robot did not turn as sharply as the training model indicated so all the path planning solutions generated using the measured model would be slightly off. This was to simulate an actual situation where the robot lost some of its turn capability in all turns.

III. The Co-Evolution of Model Parameters

The co-evolution of model parameters [11] (a type of punctuated anytime learning [12]) dynamically links the model to the actual robot for the accurate use of evolutionary computation in learning robot control. It involves doing periodic tests of evolved solutions on the actual robot to co-evolve the accuracy of the robot's model being used by the CGA. This type of anytime learning allows for an adaptive real-time learning system that needs only global observation to make corrections in the robot model.

The form of evolutionary computation used to co-evolve the model parameters in past experimentation [11] is usually the standard genetic algorithm. A population of individuals is generated before training begins. This population can start out either as randomly generated individuals or as a combination of perturbations (to varying degrees) of the original model parameters. Each individual is made up of a set number of genes. Each gene represents a corresponding field in the robot's model. These genes evolve to produce models that correspond in performance to the actual robot. After each *n* generations the best and two other area coverage solutions are tested on the actual robot. These measurements are used to judge the accuracy of a population of model parameters by comparing the performance of the area coverage solutions on the actual robot with their performance on each model. The most accurate individual in the population of model parameters is used for continued controller evolution. Fitnesses for each individual in the population of model parameters are used as they co-evolve with the controller solutions. The population of model parameters will continue to evolve

until interrupted by updated actual test information. This solution requires three actual tests every *n* generations.

To solve the problem described in Section 2.3, the co-evolution of model parameters was employed. The cyclic genetic algorithm (CGA) was used to learn the cycle of gait cycles needed by the controller to move the robot in a pattern that would result in successful area coverage while a standard GA with partial recombination was used to evolve the model parameters.

IV. Partial Recombination

Partial recombination is a type of recombination that focuses on a part of the chromosome. It is a means of doing crossover were only the genes involved in producing the fitness are affected. In addition, genes that have some fitness relationship to each other (of the same schema) tend to have more probability of being adjacent to each other. Partial recombination is not proposed as a new means of general recombination, but it is presented as a means of doing recombination in problems where all the genes of the chromosome are not used in the solution of every variance of the problem. In this paper, we use it to evolve the parameters of a model that represents the capabilities of a robot. The parameters help define the results of several possible turn commands. The values of these parameters are tested and altered as the robot performs an action performed by the model. Each action performed by the robot does not include every possible turn command so using partial recombination allows the system to only change the parameters involved in the action.

To describe how partial recombination works, we will use a simple example (Figure 3). The figure shows two chromosomes that each have 8 genes (each represented by a variable). We can assume that each gene is a number that is helping to define a part of the solution (such as a parameter) and is made up of some set number of bits. Although these chromosomes will be used in the solution of a problem, not all the genes of the chromosomes will be required to help define the solution for each particular instance of the problem. Let us assume that an instance of the problem is being used at this time to determine the fitness of the chromosome. Assume that only the genes represented by variables *a*, *b*, *d*, and *g* are required in the solution of this instance of the problem. In order to perform partial recombination, the chromosomes are reduced to contain only the needed genes. These new chromosomes are shown in Figure 3. All of the individuals (chromosomes) in the population will be reduced in this way. Any one of a variety of normal crossover methods can be performed at this point. In our example we show a one-point crossover that is restricted to be done between the genes. After the offspring are produced, the chromosomes are reformed back to their original makeup except for the changes from crossover.

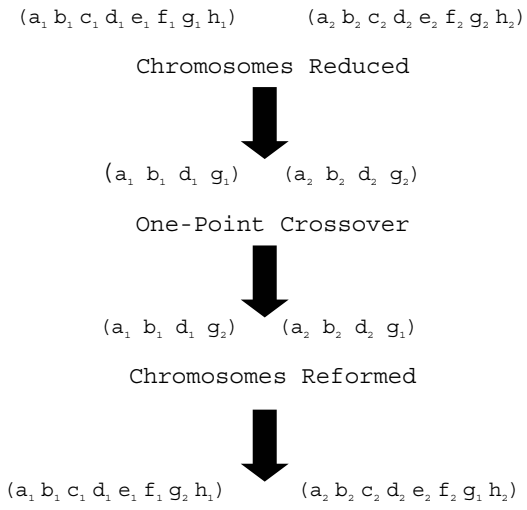

Figure 3: Sample chromosome for discussion of partial recombination.

A. Benefits of Partial Recombination

There are three benefits to partial recombination: the shortening of chromosome lengths reduces the complexity, genes belonging to the same schema are more likely to be adjacent during crossover, and genes not involved in the current solution to the problem will not be affected by crossover.

1) Chromosome Length: Chromosome length is one of many parameters that affect the computation time that is required for a genetic algorithm to solve a problem. Although the actual length of the chromosome is not reduced by partial recombination, the length of the chromosome involved in crossover is. This factor should help reduce complexity and reduce the over all computation time required for the genetic algorithm to solve the problem.

2) Linkage Problem: Holland defined schemata to provide a basis for associating combinations of attributes with potential for improving current performance [1]. He said that schema with short length (defining positions being close together) have less likelihood of disruption during crossover (this is sometimes referred to as the linkage problem). His discussion was pertinent to one-point crossover. De Jong [13] went on to discuss the effects of schema length in relationship to *n*-point crossover and noted that two-point (even-point) crossover was less likely to cause a disruption than one-point (odd-point) crossover. Goldberg, Deb, and Korb introduced messy genetic algorithms [2] that, among other things, deal with the linkage problem. In messy GAs, each bit has associated with it a location in the chromosome. We could have used a derivation of messy GAs by assigning each gene a location indicator. However, it was

determined that partial recombination would be a less complicated solution. In addition, although the related building blocks in the model parameter learning problem cannot be predetermined, they become better known during runtime. There is no need for the power of the messy GA which learns the best building block positions; they can be assigned by partial recombination during learning. They do change, however, so re-assignments must be possible.

Partial recombination applied to the parameter learning problem offers two benefits in regards to the linkage problem. One is that the genes that represent the parameters are natural schema. They are a set of bits that define one specific parameter and in that way are associated with each other in affecting the fitness of a chromosome. The other way that partial recombination can help contribute to a solution to the linkage problem is though gene adjacency. Genes needed in the solution to the problem are identified in the order that they are needed. In the case of parameter evolution, where the parameters represent the effects of turns, the robot control solution uses which ever turns are specified in the solution currently being tested. This control solution requires a sequence of turns that will produce the desired pattern over the ground. Each turn is related to the adjacent turn in the sequence in respect to the resultant pattern of movement. Since the parameters for the turns are put into the reduced chromosome in the order that they are required by the control program, there will be a greater chance that the adjacent turns will be adjacent in the chromosome. For example, if the control program called for turns that were being represented by parameters in the following order: d, h, a, e, h, d, h, a, e, h …, the reduced chromosome would look like: (d h a e). Each gene is a schema with the gene that is adjacent to it. This ordering, which assists with the linkage problem, comes naturally through knowledge of what the control program requires.

3) Hitchhiker Genes: Genes that are part of the chromosome, but not involved in the determination of fitness, are referred to as hitchhiker genes. If hitchhiker genes were always hitchhikers this would not be a problem, but in our parameter evolution problem the hitchhiker genes of one control solution may be the most important genes of another control solution. Since the parameters represent the effects of turns in control solutions which will not use all the turns for every attempt at a solution, it is important to preserve the value of the parameters that represent turns not currently being used.

Figure 4 will help to explain the problem. Assume that this is a population of parameter solutions made up of 8 individuals with it each individual having 8 genes. The genes are each made up of 3 bits. We will not be concerned with the genes represented by *b*, *d*, *f*, *g*, or *h*. The *a* and *c* genes will be parameters used by our problem and will therefore affect the fitness. The *e* gene, at this point, is a hitchhiker. As can be seen, the values of *a*, *c*, and *e* are distributed evenly throughout the possible values. Now assume that genes *a*, *b*, *c*, and *d* are used for fitness evaluation and a relatively high fitness is attained when both *a* and *c* have 1 as the value of their first bit. Individuals 4 and 5 will have relatively high fitness and dominate as parents. We can expect that the distribution will not be as equal as it is now and that in the next generation individuals with both of their *a* and *c* genes beginning with 1 will have a greater representation. This is desirable since these will produce better solutions. Unfortunately, the *e* gene will also lose its even distribution. We can expect that there will be a predominance of *e* genes that are either 101 or 100. These may not be the best values for *e*. In fact, without the even distribution, it may be difficult for the genetic algorithm to attain the optimal value for the *e* gene. This is especially true and compounded when the evolution is repeated several times without the *e* gene involved in the calculation of fitness. The partial recombination solves this problem by leaving the values of the *e* genes in tact while evolving the *a*, *b*, *c*, and *d* genes.

```
0: (000 b0 011 d0 111 f0 g0 h0)
1: (001 b1 110 d1 010 f1 g1 h1)
2: (010 b2 101 d2 000 f2 g2 h2)
3: (011 b3 000 d3 011 f3 g3 h3)
4: (100 b4 100 d4 101 f4 g4 h4)
5: (101 b5 111 d5 100 f5 g5 h5)
6: (110 b6 001 d6 110 f6 g6 h6)
7: (111 b7 010 d7 001 f7 g7 h7)
```

Figure 4: Sample population of individuals for the explanation of the preservation of hitchhiker genes.

B. Partial Recombination Applied to the Area Coverage Problem

Partial recombination was originated to deal with the co-evolution of model parameters for the area coverage problem. Although the standard GA with normal recombination worked well in co-evolving the model parameters for hexapod gait generation [11], it was deemed inappropriate for evolving the model parameters for area coverage. Since a turn cycle used up to nine gait cycles, 23 of the 32 gait cycles would have no bearing on the fitness. They would be altered as a side effect while the nine or less being evolved tended toward an optimal. This could result in the loss of vital building blocks required to evolve the gait cycles destined for future use.

Partial recombination solved this problem by extracting the needed gait cycles for application of the genetic operators. These gait cycles (each of which could be considered a building block) were used to build a new chromosome by placing them in the order they were needed in the CGA generated turn cycle, which was probably the best guess for related order. Genetic operators were applied to this partial list of all the gait cycles for the designated number of generations. Upon

completion of training, they were re-inserted into the main model parameter population. This allowed for training on the appropriate gait cycles (building blocks) arranged in the proper order without disturbing the rest of the building blocks.

For the area coverage problem, the model parameters that were needed to be co-evolved were the components of the gait cycle table shown in Figure 2. The *F*, *T*, ΔH for each affecter needed to be learned. The chromosome for co-evolution is shown in Figure 5. Each gene was made up of affecter number (including the one bit indicating the turn direction) and a set of three numbers representing the robot's distance moved and change in orientation as described in Section 2.1.

((0 (F T ΔH)) (1 (F T ΔH)) (2 (F T ΔH)) . . . (31 (F T ΔH)))

Figure 5: Model parameter chromosome.

C. Partial Recombination as Part of the Co-Evolution of Model Parameters

Co-evolution starts when a best control solution is sent by the turn cycle (area coverage path) generating CGA to the Model parameter GA. Two more turn cycles are generated using this best solution. One is a perturbation of up to ± 1 on each of the non-zero repetitions in the turn cycle. The other uses a .25 probability starting from the first gene to find a gene that it sets the repetitions to 50. The partial recombination GA chromosome is built by extracting the needed gait cycles from each individual of the model parameter population. The three turn cycle solutions (the best found by the CGA, plus two perturbations) are each run on the actual robot and on the 64 partial robot models. The fitness of each is judged by comparing (finding absolute difference) its performance to the actual robot's performance. Two figures are compared for each gait cycle solution—the number of blocks covered and the number covered only once. This is done on the three turn cycle solutions and results in six total differences that are added together to get the fitness. This fitness is used to perform the standard GA operators of selection, crossover, and mutation. After 40 generations of evolution by the partial recombination GA, the partial chromosomes are re-inserted into the main chromosome with the best designated as the current model for further evolution of a control solution by the CGA.

V. Results

To test the usefulness of partial recombination two tests were run. One test was done using a standard GA with normal recombination while the other test used a standard GA with partial recombination. In each test the same crossover and mutation operators were used and the starting populations were the same. A population of 64 gait cycle lists (the model parameters) was generated by perturbing the original gait cycle list used for CGA training. The final (5000 generation) population of area coverage path plans generated by a CGA working on an accurate model was used as the initial population for the control programs. The punctuated anytime learning system tested all these solutions using the original gait cycle list (the best known model at the time). The best solution plus two perturbations, as described in Section 4.2, were used to find a fitness for each gait cycle list (the model parameters) in the population of models. The genetic algorithm, either with or without partial recombination (depending on which test is being run), was run for 40 generations. At the completion of this training, the best model was used to replace the original model. The CGA was run again for 40 generations using this new model to evolve a new best path planning solution. This process was continually repeated for a total of 1000 CGA generations. The best area coverage path planning solution (judged by its performance on the best know model at each 40 generation mark) replaced the simulated robot's operational solution if its performance was better on the actual robot.

The results are shown for 5 independent tests in Figures 6 and 7. Figure 6 shows the results without partial recombination, Figure 7 shows it with. Both tests used the same simulated robot, which was the "corrected" model, and both used the same 5 distinct sets of starting populations. The results show the number of blocks covered while using each method for model parameter evolution. As can be observed, the co-evolution of model parameters made improvements in all cases, but the improvements were faster and more consistent when using partial recombination. Only one out of the five populations reached an optimal solution without partial combination. All of the five populations reached an optimal solution with partial recombination.

Figures 8 shows the average of the five runs done without partial recombination compared to the average of the five runs down with partial recombination. It is clear that the use of partial recombination in the evolution of the model parameters significantly improved the outcome.

VI. Conclusions

Partial recombination significantly improves the outcome of robot learning during the co-evolution of model parameters. We believe that this is due to three factors. It shortens the length of chromosomes during crossover, which reduces the over all computation time required for the genetic algorithm to solve the problem. It allows genes belonging to the same schema to be adjacent to each other during crossover, reducing the

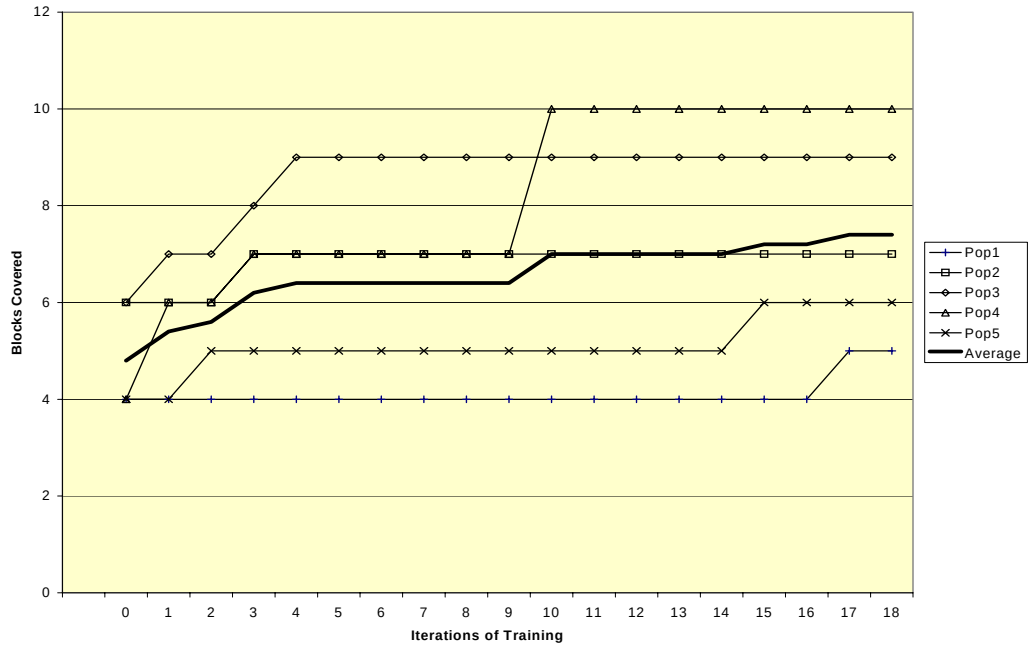

Figure 6: Result of area coverage learning without partial recombination.

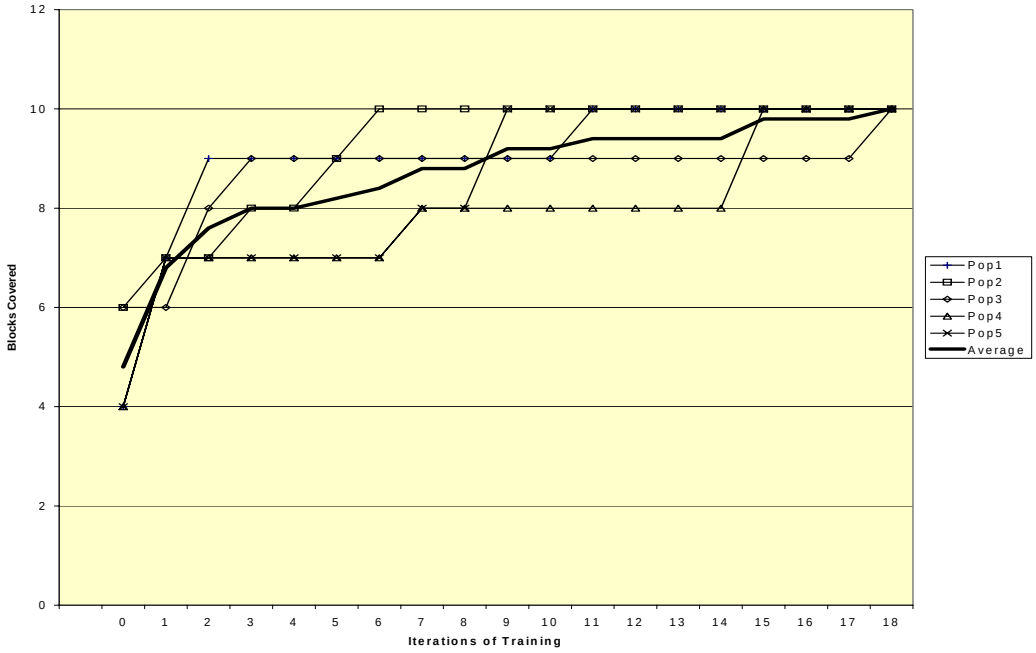

Figure 7: Result of area coverage learning with partial recombination.

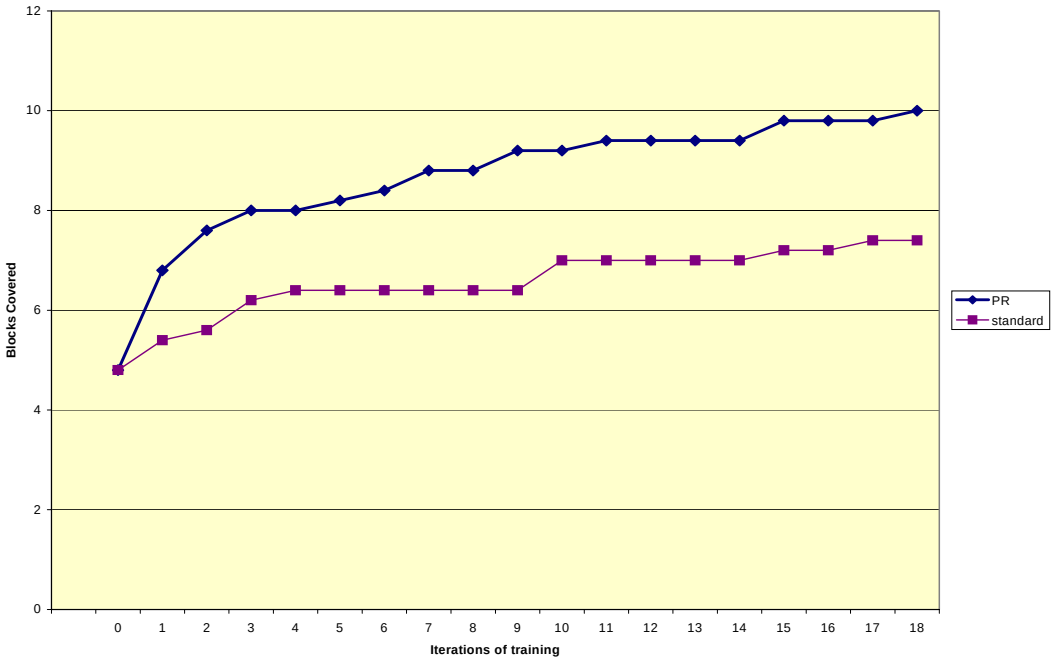

Figure 8: A comparison of area coverage learning using standard crossover verses partial recombination.

probability of schema disruption. It uses only the genes involved in the computation of the fitness for crossover, protecting the genes not involved in the current solution from being adversely affected. Although partial recombination was presented as a means of improving the co-evolution of model parameters for in the area coverage problem, it has general applicability to evolutionary computation problems where not all of the genes of the chromosome are involved in the solution for every instance of the problem.

REFERENCES

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

[2] D. E. Goldberg, K. Deb, and B. Korb, "Don't Worry, Be Messy," *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 24-30, 1991.

[3] G. B. Parker, "Evolving Cyclic Control for a Hexapod Robot Performing Area Coverage," *Proceedings of 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2001)*, pp. 561-566, 2001.

[4] G. B. Parker, "Learning Area Coverage Using the Co-Evolution of Model Parameters," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pp. 1286-1293, 2002.

[5] R. D. Beer and J. C. Gallagher, "Evolving Dynamical Neural Networks for Adaptive Behavior." *Adaptive Behavior*, 1, pp. 91-122, 1992.

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

[7] A. Lewis, A. Fagg, and G. Bekey, "Genetic Algorithms for Gait Synthesis in a Hexapod Robot." *Recent Trends in Mobile Robots*. pp 317-331, 1994.

[8] G. Spencer, "Automatic Generation of Programs for Crawling and Walking." *Advances in Genetic Programming*. pp. 335-353, 1994.

[9] G. Parker and G. Rawlins, "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, 1996.

[10] G. Parker, D. Braun, and I. Cyliax, "Evolving Hexapod Gaits Using a Cyclic Genetic Algorithm." *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing*. pp. 141-144, 1997.

[11] G. Parker, "Co-Evolving Model Parameters for Anytime Learning in Evolutionary Robotics," *Robotics and Autonomous Systems*, Volume 33, Issue 1, 31 October 2000, pp. 13-30, 2000.

[12] G. Parker, "Punctuated Anytime Learning for Hexapod Gait Generation," *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*. pp. 2664-2671, 2002.

[13] K. DeJong, *An Analysis of the Behavior of at Class of Genetic Adaptive Systems*. Doctoral dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor. 1975.