Technical Report CSM-401 (March, 2004)
Population Based Incremental Learning Versus Genetic Algorithms: Iterated Prisoners Dilemma
Timothy Gosling, Nanlin Jin and Edward Tsang
University of Essex, England

Population Based Incremental Learning Versus Genetic Algorithms: Iterated Prisoners Dilemma

Timothy Gosling
Dept. of Computer Science
University Of Essex
C04 3SQ, England
Email: tdbgos@essex.ac.uk

Nanlin Jin
Dept. of Computer Science
University Of Essex
C04 3SQ, England
Email: njin@essex.ac.uk

Prof. Edward Tsang
Dept. of Computer Science
University Of Essex
C04 3SQ, England
Email: edward@essex.ac.uk

Abstract- Axelrod's originally experiments for evolving IPD player strategies involved the use of a basic GA. In this paper we examine how well a simple GA performs against the more recent Population Based Incremental Learning system under similar conditions. We find that while PBIL performs well, GA in general does slightly better although more experiments should be conducted.

I. INTRODUCTION

Experiments in to evolving strategies to play Iterated Prisoners Dilemma (IPD) were initially carried by Axelrod in 1987 [1]. These experiments found that from an evolutionary stand point Tit-For-Tat was a dominant strategy. In the years since these early results others have attempted to evolve IPD strategies and made claims about the dominance of various other approaches to playing the game [2][3][4]. This paper is not directly concerned with the arguments for or against various strategies but is rather concerned with how the more recent statistical approaches to evolutionary computation compare with traditional GA approaches in evolving those strategies.

To this end a series of experiments have been run comparing the effectiveness of IPD strategies evolved independently by straightforward GA and PBIL implementations.

The paper introduces the strategy representation scheme used, the idea behind PBIL, the way in which PBIL and GA were compared and the results of that comparison.

II. REPRESENTATION

IPD strategies may be represented in a number of different ways partially dependent upon the particular variation of IPD being investigated [5][6]. For the GA/PBIL comparison being undertaken the original Axelrod representation for a three-round memory based game was used [1].

Under this system a 0 represents co-operation and 1 represents defect. A player's memory contains information on the last three rounds of play each of these rounds being represented by a bit pair. The bit pair consists of a record of both the player's move and its opponent's move in one of the previous rounds. Organising the pairs in time order provides a six-bit string that can be interpreted as a number between 0 and 63. Assuming a simple cooperate or defect reaction to any memory there are therefore a maximum of 64 possibly responses to a 3-round memory and the IPD strategy can thus be represented as a 64 bit string, each position in the string providing the response co-operate or defect to a specific memory. The below diagram illustrates how the player memory and strategy work together to produce a players move in the current round:

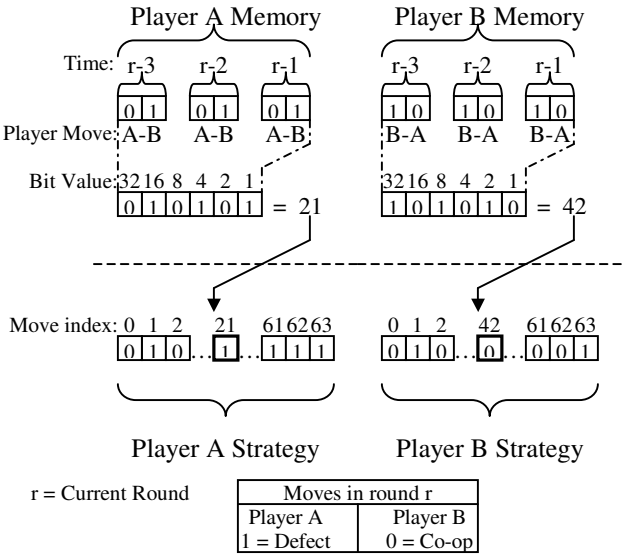

Figure 1. Player Memory and Strategy Representation

Since at the beginning of a game players start with no memory of previous rounds additional information is required as part of the representation. To this end an additional 6 bits is used to provide the player with an initial starting memory and so its first index into the strategy. As play progresses the starting memory is eventually forgotten

and play continues purely based upon true memories of the current interaction. The starting information could therefore be considered as a predisposition of a player towards its opponent. The full IPD representation is shown below:

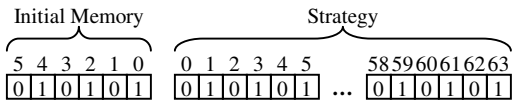

Figure 2. Complete IPD Representation

III. INTRODUCTION TO PBIL

Population based incremental learning (PBIL [7]) is a statistical approach to evolutionary computation that combines elements of GAs and Reinforcement Learning.

Under a simple PBIL scenario the basic representation of a solution can be the same as in a GA but instead of storing each possibility explicitly the population is replaced by a probability distribution. To elaborate further. If we consider a single member of a GA population it consists of a chromosome with a number of alleles. Each allele often represents some single variable in the solution and may take on a number of possible values, for the given population member though the value of each alleles is fixed. Within the population of solutions values for particular alleles exist with differing frequencies, if each population member was identical then only one value for each alleles would exist in the population as a whole, at the other extreme a wide variety of values would exist for each alleles within the population with little variation in frequency among them. PBIL essentially represents these frequencies directly and dispenses with the population itself. Thus under PBIL, each value of each alleles has a frequency or probability of existing within a hypothetical population associated with itself - the probability of each value within an allele must add up to 1. To generate a real solution string it is possible to select allele values probabilistically from the PBILs probability distribution. A diagram illustrating the difference between GA and PBIL representation can be found towards the end of this section (Figure 3).

To update a GAs population, population members are first evaluated and then recombine in some way to generate a new population. Members with a higher fitness have a greater probability of either finding their way in to the new population or helping generate new population members. Mutation is usually used to help increase diversity and reintroduce information that may have been lost at an earlier stage.

PBIL updates in a rather different manner. What needs to be updated is the probability distribution rather than a fixed population. The simplest way to perform the update is to find a good candidate solution and then increase the probability of each of the values of its alleles in the distribution (positive learning). The reverse can be done with a bad candidate solution with probabilities of values being reduced (negative learning). The rules for updating the probability of values can be quite simple and are usually tied to a learning rate (LR). The learning rate determines by how much the probability of a value under a given allele should increase and thus by how much the remaining value probabilities should be reduced. Fixed or variable learning rates can be used; if the LR is variable it may be tied to the relative fitness of the candidate solution being used to update the distribution.

Since a PBIL system will often have no real population of solutions to draw candidates from a temporary pool of solutions maybe generated from the distribution. The solutions in this pool can then be evaluated and the best and worst used to update the distribution.

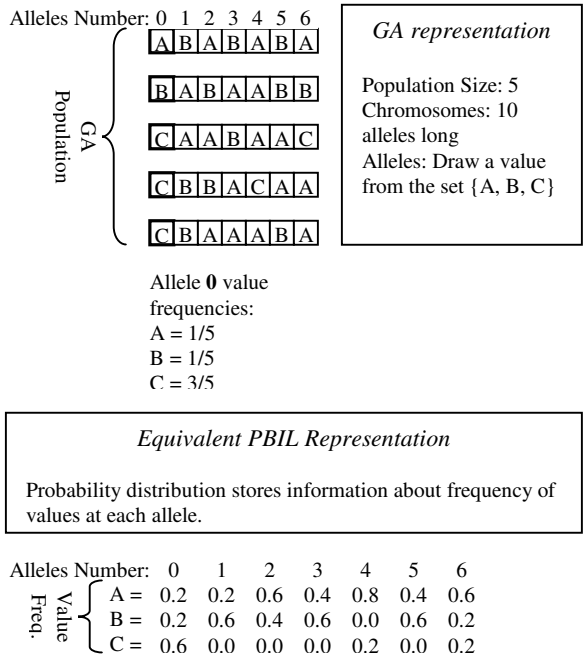

Figure 3.Differences between GA and PBIL representation

Mutation is often used with PBIL to help increase the search space much as with GA. Various schemes to implement mutation exist however two approaches are either to vary the value frequencies by some amount with low probability or, alternatively, apply mutation with a low probability to generated population members before they are evaluated.

While the above explanation of PBILs operation is sufficient to explain how alleles with a discrete set of values or symbols maybe represented it does not provide explain how continuous ranges maybe dealt with. Since continuous ranges are not required in the formation of the IPD strategies used here, no explanation of how this is accomplished will be provided.

IV. PBIL SPECIFICS

The PBIL implementation used for running the IPD experiments represents each allele as a cell with a specific numeric range and number of symbols. For the purposes of IPD each cell is ranged 0 to 1 with two symbols, i.e. 0 and 1. At the beginning of an experiment each cell was set such that 0 and 1 had an equal probability.

The positive reinforcement rule used in the experiments was simple (taken from [7]). The mechanism used is described below:

$$prob_c = prob_c \times (1.0 - LR) + (cand_c - LR)$$

$prob_c$ - Probability of a 1 in cell c of the distribution

LR - The learning rate (between 0 and 1)

$cand_c$ - Bit value at position c in the candidate solution

No negative learning was used in the experiments. Learning rates were varied between experiments but were constant within a give experiment. The range of learning rates used was between 10% and 0.5%.

Mutation was not always used within the experiments reported on here. When used, it was applied by changing generated population members with a probability of 0.7% per allele. When changed a value would simply flip from 1 to 0 or vice versa.

A generation in the sense of PBIL consists of the creation of a population, evaluation of that population and an update of the distribution by the fittest population member. In all the experiments described here PBIL was used with a greater number of generations than the GA for reasons described later.

To perform an update of the probability distribution two sets of IPD strategies were generated. The first 'update' or 'population' set was relatively small and used for updating the distribution; the second 'test' set was used purely for evaluating the first set. An example would be a population set of 10 coupled with a test set of 99. In this case each of the 10 population members would be tested against each of the 99 testing members to find its fitness. The population member with the highest mean score against each of its 99 opponents would be used to update the distribution.

This mechanism is necessary to even out the disparity in information use between the GA and PBIL when updating. While the GA (in a sense) makes use of its entire population to create a new population the PBIL system only uses one population member to update the distribution. Different ways of resolving this disparity might be used but the one above was selected for its ease of understanding. By selecting a smaller population size played against a larger testing pool the quality of individual evaluations may be maintained. By repeating the process for a larger number of generations than the GA a fairer use of information by both is maintained. In each case the total number of games played and solution evaluations must be maintained or bias will be introduced. To help prevent bias the following must hold:

$$GA_{evals} = GApop \times (GApop - 1) \times GAgens$$

$$PBIL_{evals} = PBILpop \times PBILtestpop \times PBILgens$$

$$GA_{evals} = PBIL_{evals}$$

GAevals/PBILevals - Total number of solution evaluations

GApop - The GA population size

GAgens - The number of generations the GA runs for

PBILpop - The population size of the PBIL system

PBILtestpop - The testing pool size of the PBILsystem

PBILgens - The number of generation the PBILsystem runs

With the above in mind a population size 100 GA running for a 100 generations could be played against a PBIL system using a population size of 10 and a testing pool of 99 running for 1000 generations. If however the GAs population was reduced a corresponding reduction in the population size, test pool size or number of generations would be required by the PBIL system. Some combinations evidently will be more effective than others and in the course of these experiments different variations were considered.

For the purposes of comparison with the GA ten IPD strategies are generated at various points through out the experimental run (the GA records corresponding data after every generation). The ten strategies are generated in exactly the same way as for evaluation or testing. The interval between points at which comparison strategies are recorded can be determined by:

$$ComparisonOutputInterval = PBILgens / GAgens$$

The experimental random number generator seeds were select from the system clock, the random number generator itself can be found in [8].

V. GA SPECIFICS

Individual strings (strategies) in the initial population are generated randomly with 50% possibility of choosing "Defection" and 50% possibility of choosing "Cooperation" at every bit of every 70-bits string.

Performance (fitness) of a string is evaluated by the average score that it earns from playing Iterated Prisoners' Dilemma with every other string in the same population.

Like natural selection, individuals having higher fitness are selected with higher probability. First of all, the fittest string is ensured to be selected as a parent, which is called "Elitism". Each of rest parents is chosen using the "Roulette-Wheel Algorithm". A random number $r \in [0, f_1 + f_2 + \ldots + f_{popSize})$ is created, then the string i whose fitness notated f_i is selected, where $f_1 + f_2 + \ldots + f_{i-1} \le r < f_1 + f_2 + \ldots + f_i$.[9]

Strings are selected pair-wise and undergo one-point crossover, exchanging portions of strings of each other. Newly created intermediate strings mutate with very low rate (0.7%) by randomly alternating one bit of "cooperation" to "defection", or vice verse.

The offspring of the parent strings go on to form a completely new population for the next generation.

Strings used for comparison with PBIL are the first 10 strings chosen of every generation by Roulette-Wheel selection without Elitism.

VI. GAMES

Part of the evaluation process of an IPD strategy involves playing against other IPD strategies. To this end each game used for evaluation consisted of 150 moves being played and the standard score grid below being used:

Score A / Score B		Player A: Co-op	Player A: Defect
Player B	Co-op	3 / 3	5 / 0
Player B	Defect	0 / 5	1 / 1

Figure 4. Standard Pay-off table

The same game parameters were used for the comparison discussed below.

VII. GA AND PBIL COMPARISON

Providing a comparison between GA and PBIL systems in a way that provides neither with an advantage is difficult, however as much bias as possible has been removed. The following describes how the comparison was eventually realized.

To provide a comparison between the GA and PBIL systems for evolving IPD strategies, both systems were run independently and their resulting strategies tested against one another.

In running the comparison between GA and PBIL it was important to not provide a significant advantage to either, it was also critical that the comparison mechanism itself not be unfair or subject to too much uncertainty.

To counter the first problem the PBIL and GA systems were run for a differing number of generations with differing population sizes, the total number of evaluations and individual evaluation quality was maintained however. See 'PBIL Specifics' above for more details. To sensibly compare the GA and PBIL strategies at comparable intervals in the runs, ten strategies were generated by the PBIL and recorded for comparison and ten strategies were selected probabilistically and recorded for comparison by the GA. The recording process began at initialization and was performed at regular intervals up until the end of the run in each system. In the experiments reported here GA runs lasted 100 or 300 generations, PBIL runs lasted between 1000 and 6000 generations.

To provide a comparison between a single GA and PBIL experiment for a given time, each of the ten strategies from each was played against all the strategies from the opposition. The results of these games, the mean scores and standard deviations were recorded. The mean strategy score for the GA and PBIL at the specified time can then be found easily. A single comparison such as this results in 100 games being played. When comparing GA and PBIL runs in total this process is repeated for all compatible comparison points in both the PBIL and GA systems (either 100 or 300 points).

To improve the validity of the results each PBIL and GA experiment was repeated ten times (unless otherwise stated). Each of the PBIL and GA experiments could then be compared to one another and the results averaged. This results in 100 comparisons being done for a single time instance and so 10000 games being played between 200 strategies.

VIII. EXPERIMENTAL RESULTS

While a large number of PBIL configurations and somewhat smaller number of GA configurations were tried, the most effective GA and PBIL configuration tried are shown below:

Type	GA	PBIL
Population Size	100	5
Learning Pool	NA	99
Mutation Rate	0.007	Not Used
Learning Rate	NA	0.025
Generations	300	6000
Data Points	300	300

Table 1: Comparison of most effective GA and PBIL configurations

These two configurations form the basis for comparison within the rest of this section. The relative mean fitness used in the graphs below is determined by the mean fitness of all of player A's strategies divided by the mean fitness of all of player B's strategies. Player A and B are determined by what the graph is attempting to demonstrate. When comparing various PBIL configurations against GA, player A will always be the best GA configuration while player B is each of the tested PBIL configurations. As a result any score above 1 shows greater effectiveness for player A strategies and anything below 1 shows greater effectiveness for player B strategies.

The following diagram shows the relative performance of different PBIL configurations against GA. In this case the GA parameters are identical to those above and the PBIL parameters only vary in terms of the learning rate (Figure 5):

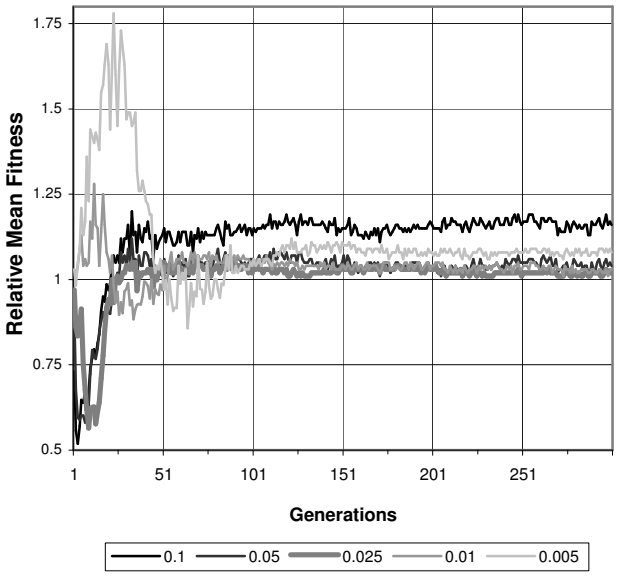

Figure 5: Best GA v Population 5, no mutation PBIL with varying learning rates. PBIL initial fares well or poorly depending on LR but ultimately does worse than GA.

If mutation is applied to PBIL its performance is generally reduced. At low learning rates this effect is minimal but at higher learning rates the negative effect becomes more evident The diagrams below show the reduction in effectiveness caused by mutation at different learning rates (Figure 6, 7):

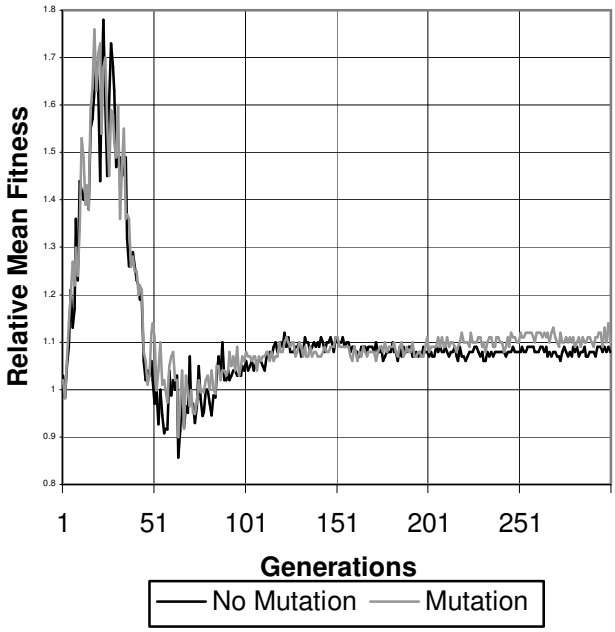

Figure 6: PBIL performance against GA with and without GA at low learning rate (0.005). Generation 1- 51, GA does very well compared to PBIL. Generation 51-101, PBIL recovers somewhat. Generation 101 onwards, PBIL performance stabilizes but is less effective than GA.

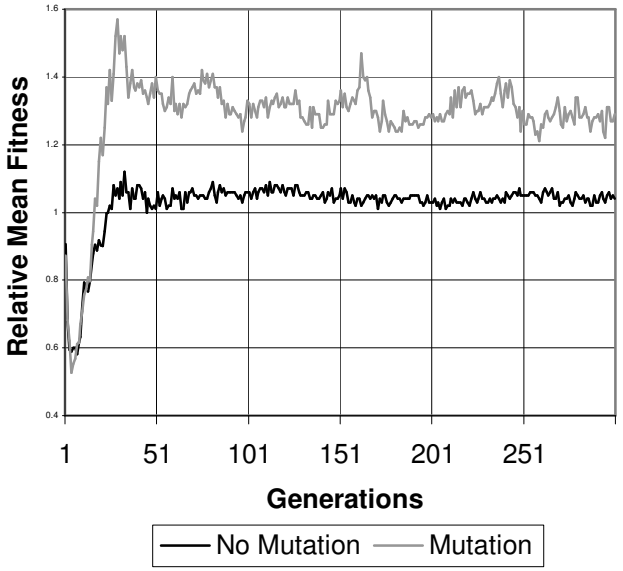

Figure 7: PBIL performance against GA with and without mutation at high learning rate (0.05). Generations 1-51, PBIL initial performs well but rapidly looses out to GA. Generation 51 onwards, PBIL stabalizes doing worse than PBIL. Mutated configuration PBIL does worse than non mutation equivelent

Increasing the number of tournaments played while reducing the number of generations of the PBIL yields results that are generally not quite as favorable although early performance does tend to be a little better (Figure 8).

In the results show below the PBIL population size has been increased to 10 while the number of generations has been reduced to three thousand, various learning rates are shown:

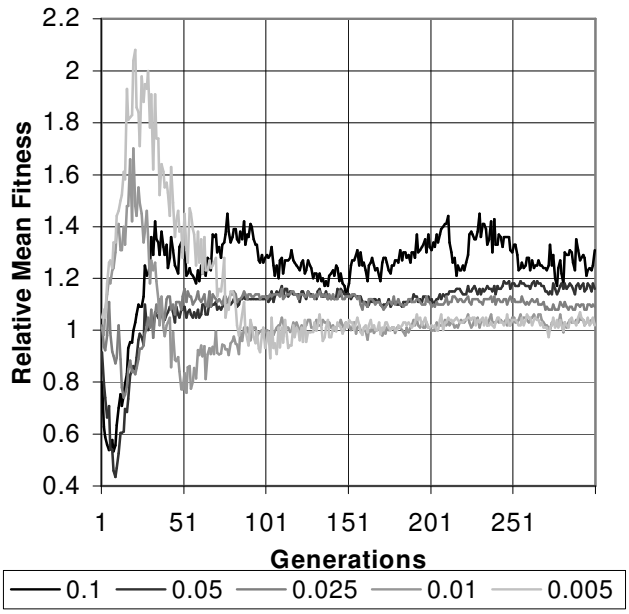

Figure 8: Best GA v Population 10, no mutation PBIL with varying learning rate. Generation 1-51, PBIL performs well or poorly depending on LR. Generation 51-101, PBIL performance begins to stabalize if it hasn't already. Generation 101 onwards, PBIL performance stabailizes but does worse than GA.

The population 10 PBIL experiments run with mutation enabled again perform somewhat worse (Figure 9):

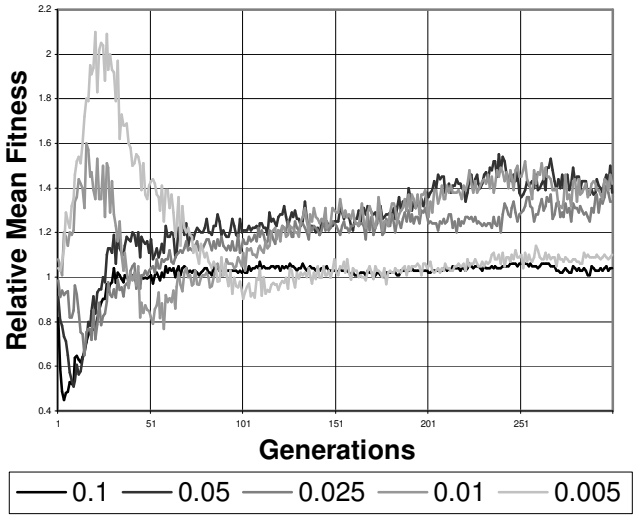

Figure 9: GA v Population size 10 PBIL with mutation and varying learning rates. Generation 1-51, PBIL performs according to learning rate well or poorly. Generation 51-101, PBIL performance stabalizes if it hasn't already. Generation 101 onwards, PBIL performance stabalizes and is worse than GA, often increasingly so.

Having seen various PBIL configurations performance in relation to the best GA configuration next is to show some variations in the GA configuration in relation to the PBIL.

In this comparison the 'best' PBIL configuration is used against the best GA configuration with a varying mutation rate (Figure 10):

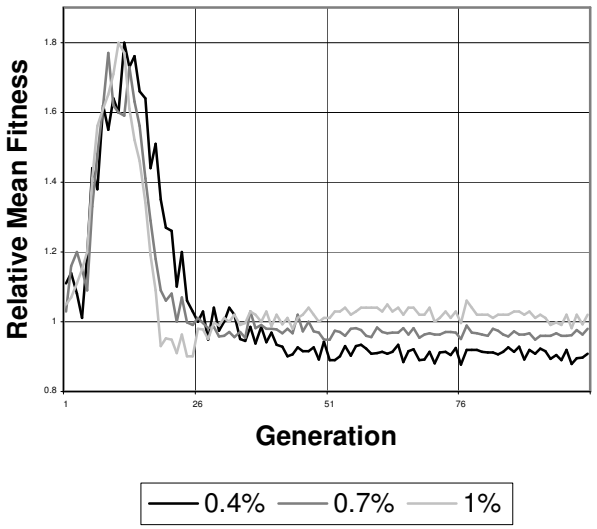

Figure 10: The effect of mutation rate on GA performance (against best PBIL configuration). Generation 1-25, PBIL does better than GA. Generation 25 onwards, GA generally does better than PBIL except at the higher mutation rate.

As can be seen (Figure 10) the selected 'best' GA (0.7%) provides an effective performance taking an early leading and continuing to do better than PBIL. The lower mutation rate further improves this result but takes longer to get there. Note that the 0.4% mutation rate result was achieved using only three experiment repetitions.

GA population size has also been experimented with to some limited degree. Reducing the population size casues GA considerable problems and in general PBIL will perform relatively better. The diagram below (Figure 11) shows a population size 20 GA running for 300 generations with a mutation rate of 0.7%. This appoximates the original experiments run by Axelrod [1], except that it runs for slightly longer. The opponent PBIL uses a population size of 5, training pool of 19 and runs for 1200 generations.

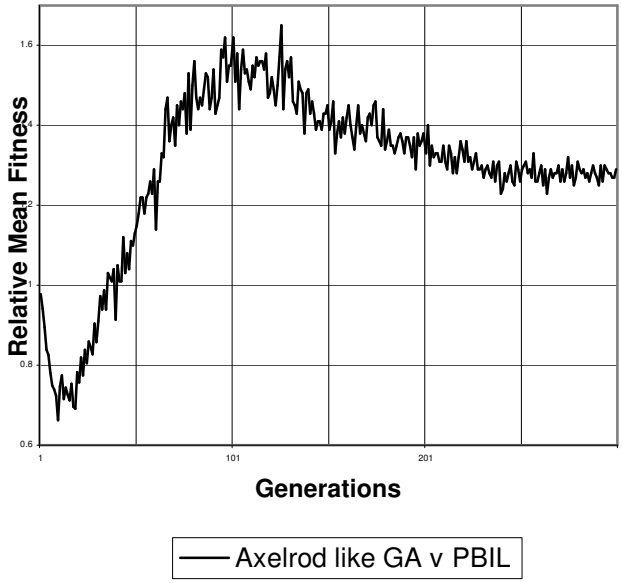

Figure 11: Axelrod like GA versus PBIL. Generation 1-51, PBIL initially does badly against the GA due to its low learning rate but is rapidly gaining ground after about 25 generations. Generation 51-101, PBIL continues to gain grownd and reaches a peak in realtive performance. Generation 101-201, PBILs relative performance is degraded but is still better than GA. Generation 201 onwards, PBILs performance stabilizes and does better than GA.

As can be seen from the above when the GAs population is quite small the use of a probability distribution by PBIL comes in to its own – it might not use any more evaluation but it can certainly generate more effective strategies even if it takes it a while.

IX. CONCLUSIONS

The results above would tend to suggest that a GA approach to evolving IPD strategies is very slightly superior to a PBIL approach. When both systems have stabilized the GA consistently scores slightly higher than PBIL system.

The initial relative performance of PBIL may be superior to GA if the learning rate is high enough as it is able to learn fairly effective strategies quickly, however this tends to be accompanied by later worse performance due to overly rapid convergence. Setting the learning rate of the PBIL system lower tends to allow the GA a better run in the beginning and reduces the negative effects of rapid convergence, under these condition PBIL performs favorably although GA arguably still fares very slightly better. Tuning the learning rate more precisely may ultimately provide a more comparable performance.

Increasing the PBIL population size seems to cause far greater variation in performance relative to the learning rate used but may help provide a way to close any perceived performance gap.

Introducing mutation into the PBIL system doesn't seem to remedy the effects of overly rapid convergence and indeed seems to generally exacerbate the problem. At higher learning rates and larger population sizes mutation would seem to help smooth performance out a little over time if not actually improve it.

When the GA population size is reduced PBIL comes into its own and is able to perform significantly better than GA. While the GA struggles under a lack of evaluations the PBIL is able to use what evaluations it has in the context of a hypothetically infinite population and respond accordingly. When the GA population is increased PBIL's advantage tends to be lost.

Any set of comparisons is difficult. The set of comparisons shown here only encompasses one particular problem using a limited number of configurations of both systems, as such its results shouldn't be considered definitive in arguing whether PBIL or GA is more appropriate under much wider conditions.

Further work on the effectiveness of comparable GA and PBIL systems should be undertaken to better understand when each is more applicable; some form of co evolutionary approach may yield interesting results for instance. It is important to establish when a technique is effective and when it is not.

REFERENCES

[1] Robert Axelrod, *The Evolution of Strategies in the Iterated Prisoner's Dilemma,* in Davis, L. (ed.), Genetic algorithms and simulated annealing, Research notes in AI, Pitman/Morgan Kaufmann, 1987, 32-41

[2] Ken Binmore, *Game Theory and the Social Contract I*, *playing fair*, MIT Press, 1994

[3] Ken Binmore, *Game Theory and the Social Contract II*, *just playing*, MIT Press, 1998

[4] Linster, B., *Evolutionary Stability in the Repeated Prisoners' Dilemma Played by Two-State Moore Machines*, Southern Economic Journal, 1992, pages 880-903

[5] Fogel, D.B. (1993). Evolving behaviors in the iterated prisoner's dilemma. Evolutionary Computation, 1(1), 77-97.

[6] Crowley, P.H. (1996). Evolving cooperation: strategies as hierarchies of rules. BioSystems, 37:67-80.

[7] Shumeet Baluja, Population Based Incremental Learning – A Method for Integrating Genetic Search Based Function Optimisation and Competitive Learning, *(Tech. Rep. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University (1994)*

[8] Numerical Recipes in C++: The Art of Scientific Computing – p? - William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery

[9] Course cc385, "Genetic Programming and its Applications", 2003 – University Of Essex, Prof Riccardo Poli