Removing the Genetics from the Standard Genetic Algorithm

Shumeet Baluja
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
baluja@cs.cmu.edu

Rich Caruana
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
caruana@cs.cmu.edu

Abstract

We present an abstraction of the genetic algorithm (GA), termed population-based incremental learning (PBIL), that explicitly maintains the statistics contained in a GA's population, but which abstracts away the crossover operator and redefines the role of the population. This results in PBIL being simpler, both computationally and theoretically, than the GA. Empirical results reported elsewhere show that PBIL is faster and more effective than the GA on a large set of commonly used benchmark problems. Here we present results on a problem custom designed to benefit both from the GA's crossover operator and from its use of a population. The results show that PBIL performs as well as, or better than, GAs carefully tuned to do well on this problem. This suggests that even on problems custom designed for GAs, much of the power of the GA may derive from the statistics maintained implicitly in its population, and not from the population itself nor from the crossover operator.

1. THE GENETIC ALGORITHM (GA)

Genetic algorithms (GAs) are biologically motivated adaptive systems based on natural selection and genetic recombination. In the standard GA, candidate solutions are encoded as fixed length vectors. The initial population of solutions is chosen randomly. These candidate solutions, called "chromosomes," are allowed to evolve over a number of generations. At each generation, the fitness of each chromosome is calculated; this is a measure of how well the chromosome optimizes the objective function. Subsequent generations are created through a process of selection, recombination, and mutation. Chromosome fitness is used to probabilistically select which individuals will recombine. Recombination (crossover) operators merge the information contained within pairs of selected "parents" by placing random subsets of the information from both parents into their respective positions in a member of the subsequent generation. Due to random factors involved in producing "children" chromosomes, the children may, or may not, have higher fitness values than their parents. Nevertheless, because of the selective pressure applied through a number of generations, the overall trend is towards higher fitness chromosomes. Mutations are used to help preserve diversity in the population. Mutations introduce random changes into the chromosomes. A good overview of GAs can be found in [Goldberg, 1989] [De Jong, 1975].

Although there has recently been some controversy in the GA community as to whether GAs should be used for static function optimization, a large amount of research has been, and continues to be, conducted in this direction. [De Jong, 1992] claims that the GA is not a function optimizer, and that typical GAs which are used for function optimization often use different, specially customized mechanisms which are not suited for GAs used for "adaptation" in dynamic environments. Nonetheless, as many of the more successful applications and current trends in GA research focus on optimization (most often in static environments), this study also concentrates on this domain.

The GAs used in this study are characterized by 5 parameters: population size, crossover type, crossover rate, mutation rate, and elitist selection. The population size is the number of chromosomes present in every generation. The crossover type determines how the information is recombined (Figure 1). Three crossover types were examined: *One-point crossover*: Given two parent chromosomes, select a randomly chosen crossover point and swap contents of the chromosomes beyond the chosen point. *Two Point crossover* is similar to one point except that two crossover points are randomly selected, and the contents of the chromosomes between those points are swapped. In *Uniform crossover*, the parent is chosen randomly for each bit position. The crossover rate is the percentage of the time that crossover of information occurs when two chromosomes are selected to recombine. (If crossover does not occur, the two chromosomes are copied directly into the next generation's population.) The mutation rate is the probability of randomly flipping the value in each bit position of each chromosome at every generation. Elitist selection can either be on or off. If it is on, the best chromosome from generation *G* is automatically carried to generation *G+1*. With elitist selection, the quality of the best solution in each generation monotonically increases over time. Without elitist selection, it is possible to lose the best chromosome due to stochastic errors. Techniques somewhat similar to elitist selection have also been studied outside of the domain of genetic algorithms. "Best-so-far" techniques are explored, in the context of simulated annealing methods, in [Boese & Kahng, 1994].

```
                    Parent A 00000000 00000
One Point           Parent B 11111111 11111
Crossover
                    Child A  00000000 11111
                    Child B  11111111 00000
                                     ↑

                    Parent A 000 0000000 000
                    Parent B 111 1111111 111
Two Point
Crossover           Child A  000 1111111 000
                    Child B  111 0000000 111
                                ↑       ↑

                    Parent A 0000000000000
Uniform             Parent B 1111111111111
Crossover
                    Child A  0101010111000
                    Child B  1010101000111
```

Figure 1: Samples of Crossover. One Point, Two Point, and Uniform Crossover.

2. FOUR PEAKS: A PROBLEM DESIGNED TO BE GA-FRIENDLY

Consider the following class of fitness functions defined on bit strings containing 100 bits and parameterized by the value T:

$$z(x) = \text{Number of contiguous Zeros ending in Position 100}$$

$$o(x) = \text{Number of contiguous Ones starting in Position 1}$$

$$REWARD = \begin{cases} 100 & \text{if } o(x) > T \wedge z(x) > T \\ 0 & \text{else} \end{cases}$$

$$f(x) = MAX(o(x), z(x)) + REWARD$$

Suppose T=10. Fitness is maximized if a string is able to get both the REWARD of 100 and if the length of one of O(X) or Z(X) is as large as possible. The optimal fitness of 189 (when T=10) is obtained by strings containing either eighty-nine 1's followed by eleven 0's or eleven 1's followed by eighty-nine 0's. Note that strings with O(X) and Z(X) larger than T, but with suboptimal lengths of O(X) and Z(X), can hillclimb to one of the two global maxima by repeatedly flipping bits at the end of the run of 0's or 1's that is largest. For example, if O(X)=20 and Z(X)=40, hillclimbing can reach the peak at Z(X)=89, O(X)=11 by flipping the 41st bit to 0, then the 42nd bit (if it is not already 0), etc.

The four peaks problems also have two suboptimal local optima with fitnesses of 100 (independent of T). One of these is at O(X)=100, Z(X)=0 and the other is at O(X)=0, Z(X)=100. Hillclimbing will quickly get trapped in these local optima. For example, if O(X)=5 and Z(X)=20, hillclimbing will continue to increase the value of Z(X) until Z(X)=100. The only way for a string that has both O(X)≤T and Z(X)≤T to find the global optimum by single-bit hillclimbing is if it continues to add bits to the shorter of the two ends, despite never receiving better fitness in doing so. This entails repeatedly making "correct" decisions while searching large plateaus; this is extremely unlikely in practice. In fact, steepest ascent hillclimbing is unable to do this because the hillclimber must be able to accept moves to equally performing states, instead of only moving to better states. By increasing T, the basins of attraction surrounding the inferior local optima increase in size exponentially while the basins around the global optima decrease at the same rate. Figure 2 represents one very simplified view of the four peak's search space where fitness is plotted as a function of the number of contiguous 1's and contiguous 0's.

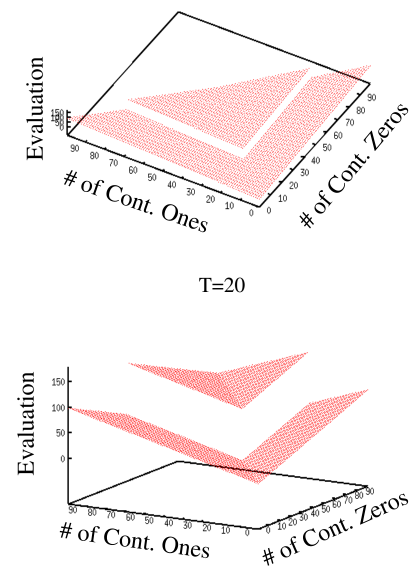

Figure 2: Two Views of the same four peaks problem. As T increases, the area in the upper triangle decrease.

A traditional GA is not restricted to single-bit hillclimbing. Crossover on a population of strings, some of which have O(X)>>Z(X), and others which have O(X)<<Z(X), but none of which have both O(X)>T and Z(X)>T, will occasionally create individuals with O(X)>T and Z(X)>T. When this happens, the string will receive the extra REWARD of 100 and will have higher fitness than its parents. A GA can discover these high fitness individuals by recombining useful building blocks present in different lower-fitness members of the population. The four peaks problems are custom designed to benefit from the GA's crossover operator, assuming the population is able to maintain the important building blocks. The four peaks problems are designed to work best with single point crossover because this crossover operator maximizes the chance that the O(X) and Z(X) ends of the string will be recombined without modification.

3. SELECTING THE GA'S PARAMETERS

One difficulty in using a GA on a new problem is that there are GA control parameters (e.g., population size, mutation rate,...) that affect how well the algorithm performs. To avoid the potential problems of not correctly setting the parameters of the GA, GAs with 108 different parameter settings were run on the four peaks function with T=11. Each GA was run 60 times with different initial random populations. In these runs, five parameters were varied:

- Population Size - 100, 200, 500
- Crossover Type - One Point, Two Point, Uniform
- Crossover Rate - 60%, 80%, 100%
- Mutation Rate - 0.001, 0.01
- Elitist Selection - On/Off

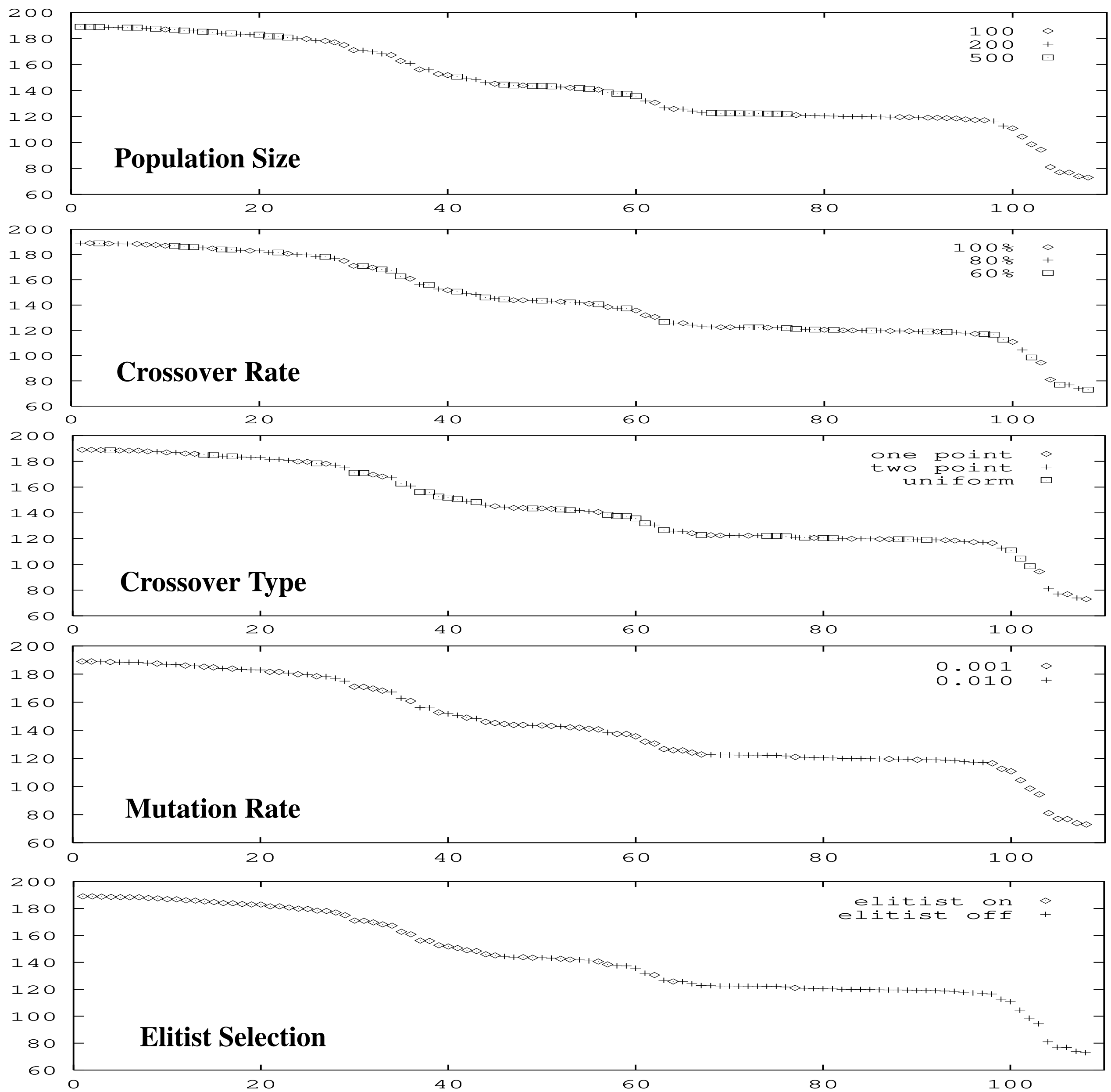

Figure 3: The 108 GA runs. Each point represents the average best evaluation (over 60 runs) of a GA. The parameters of the GA are shown in the graphs. The runs are sorted from best (left) to worst (right). The Y-Axis is the performance of the algorithm, the X-Axis is the test number.

The average best scores of the runs are presented in Figure 3. The five graphs present the performance of the algorithms, while varying the five parameters listed above. The data has been sorted by performance (the best performers on the left) to allow rapid visual identification of the better settings for each parameter. Several general results can be seen. The most apparent effect is that of elitist selection; GAs which employ elitist selection do better than the ones which do not. Second, as expected, the GAs which use one point crossover perform the best. As mentioned before, the four peaks problem is designed to do well with one point crossover. Third, larger populations did, in general, better than smaller ones. Again, due to the requirement in the four peaks problems for maintaining at least two classes of diverse individuals (one with many contiguous zeros, and one with many contiguous ones), this result is also expected. Performance was less sensitive to the mutation rate and crossover rate settings we tried.

For simplicity, in the remainder of the paper only the five best GAs will be compared. These GAs have the following parameter settings:

GA 1: Pop.: 500, One Point Crossover, Crossover Rate = 80%, Mut. Rate = 0.001, Elitist On

GA 2: Pop.: 500, One Point Crossover, Crossover Rate=100%, Mut. Rate = 0.001, Elitist On

GA 3: Pop.: 500, One Point Crossover, Crossover Rate = 60%, Mut. Rate = 0.010, Elitist On

GA 4: Pop.: 200, Uniform Crossover, Crossover Rate = 100%, Mut. Rate = 0.001, Elitist On

GA 5: Pop.: 200, One Point Crossover, Crossover Rate = 80%, Mut. Rate = 0.010, Elitist On

4. POPULATION-BASED INCREMENTAL LEARNING

Population-based incremental learning (PBIL) is a combination of evolutionary optimization and hillclimbing [Baluja, 1994]. The object of the algorithm is to create a real valued probability vector which, when sampled, reveals high evaluation solution vectors with high probability. For example, if a good solution to a problem can be encoded as a string of alternating 0's and 1's, a suitable final probability vector would be 0.01, 0.99, 0.01, 0.99, etc.

Initially, the values of the probability vector are initialized to 0.5. Sampling from this vector reveals random solution vectors because the probability of generating a 1 or 0 is equal. As search progresses, the values in the probability vector gradually shift to represent high evaluation solution vectors. This is accomplished as follows: A number of solution vectors are generated based upon the probabilities specified in the probability vector. The probability vector is pushed towards the generated solution vector(s) with the highest evaluation. The distance the probability vector is pushed depends upon the learning rate parameter. After the probability vector is updated, a new set of solution vectors is produced by sampling from the updated probability vector, and the cycle is continued. As the search progresses, entries in the probability vector move away from their initial settings of 0.5 towards either 0.0 or 1.0. The probability vector can be viewed as a prototype vector for generating solution vectors which have high evaluations with respect to the available knowledge of the search space.

PBIL is characterized by 3 parameters. The first is the *number of samples* to generate based upon each probability vector before an update (analogous to the population size of GAs). This was kept constant at 200 (the smallest size used by the best GAs). The second is the *Learning Rate*, which specifies how large the steps towards good solutions are. This was kept at a constant 0.005. The third is the *Number of Vectors to Update From*. In these experiments, only the best 2 vectors were used to update the probability vector in each generation (the other 198 are ignored). The PBIL parameters used in this study were determined by informal testing using several different parameter settings.[1] The PBIL algorithm is shown in Figure 4.

This algorithm is an extension of the Equilibrium Genetic Algorithm developed in conjunction with [Juels, 1993, 1994]. Another algorithm related to EGA/PBIL is Bit-Based Simulated Crossover (BSC) [Syswerda, 1992][Eshelman & Schaffer, 1993]. BSC regenerates the probability vector at each generation; it also uses selection probabilities (as do standard GAs) to generate the probability vector. In contrast, PBIL does not regenerate the probability vector at each generation, rather, the probability vector is updated through the search procedure. Additionally, PBIL does not use selection probabilities. Instead, it updates the probability vector using a few (in these experiments 2) of the best performing individuals.

The manner in which the updates to the probability vector occur is similar to the weight update rule in supervised competitive learning networks, or the update rules used in Learning Vector Quantization (LVQ) [Hertz, Krogh & Palmer, 1993]. Many of the heuristics used to make learning more effective in supervised competitive learning networks (or LVQ), or to increase the speed of learning, can be used with the PBIL algorithm. This relationship is discussed in greater detail in [Baluja, 1994].

4.1. PBIL's Relation to Genetic Algorithms

One key feature of the *early* portions of genetic optimization is the parallelism in the search; many diverse points are represented in the population of early generations. As the search progresses, the population of the GA tends to converge around a good solution vector in the function

1. One interesting difference between the parameter settings used here and those used in previous studies is that PBIL performed better on four peaks if the update was based on two vectors instead of just one.

```
****** Initialize Probability Vector ******
    for i :=1 to LENGTH do P[i] = 0.5;

    while (NOT termination condition)
            ***** Generate Samples *****
            for i :=1 to NUMBER_SAMPLES do
                    solution_vectors[i] := generate_sample_vector_according_to_probabilities (P);
                    evaluations[i] :=Evaluate_Solution (solution_vectors[i]);

            solution_vectors = sort_vectors_from_best_to_worst_according_to_evaluations ();

            **** Update Probability Vector towards best solutions****
            for j :=1 to NUMBER_OF_VECTORS_TO_UPDATE_FROM
                    for i :=1 to LENGTH do P[i] := P[i] * (1.0 - LR) + solution_vectors[j][i]* (LR);
```

PBIL CONSTANTS:

NUMBER_SAMPLES: the number of vectors generated before update of the probability vector (200).

LR: the learning rate, how fast to exploit the search performed (0.005).

NUMER_OF_VECTORS_TO_UPDATE_FROM: the number of vectors in the current population which are used to update the probability vector (2)

LENGTH: number of bits in the solution (determined by the problem encoding).

Figure 4: The PBIL/EGA algorithm for a binary alphabet.

space (the respective bit positions in the majority of the solution strings converge to the same value). PBIL attempts to create a probability vector that is a prototype for high evaluation vectors for the function space being explored. As search progresses in PBIL, the values in the probability vector move away from 0.5, towards either 0.0 or 1.0. Analogously to genetic search, PBIL converges from initial diversity to a single point where the probabilities are close to either 0.0 or 1.0. At this point, there is a high degree of similarity in the vectors generated.

Because PBIL uses a single probability vector, it may seem to have less expressive power than a GA using a full population that can represent a large number of points simultaneously. For example, in Figure 5, the vector representations for populations #1 and #2 are the same although the members of the two populations are quite different. This appears to be a fundamental limitation of PBIL; a GA would not treat these two populations the same. A traditional single population GA, however, would not be able to *maintain* either of these populations. Because of sampling errors, the population will converge to one point; it will not be able to maintain multiple dissimilar points. This phenomenon is summarized below:

> "... the theorem [Fundamental Theorem of Genetic Algorithms [Goldberg, 1989]], assumes an infinitely large population size. In a finite size population, even when there is no selective advantage for either of two competing alternatives... the population will converge to one alternative or the other in finite time (De Jong, 1975; [Goldberg & Segrest, 1987]). This problem of finite populations is so important that geneticists have given it a special name, genetic drift. Stochastic errors tend to accumulate, ultimately causing the population to converge to one alternative or another" [Goldberg & Richardson, 1987].

Similarly, PBIL will converge to a probability vector that represents one of the two solutions in each of the populations in Figure 5; the probability vector can only represent one of the dissimilar points. Methods designed to address this problem are discussed later.

Population #1	**Population #2**
0 0 1 1	1 0 1 0
1 1 0 0	0 1 0 1
1 1 0 0	1 0 1 0
0 0 1 1	0 1 0 1
Representation	**Representation**
0.5, 0.5, 0.5, 0.5	0.5 0.5 0.5 0.5

Figure 5: The probability representation of 2 small populations of 4-bit solution vectors; population size is 4. Notice that both representations for the populations are the same, although the solution vectors each represent are entirely different.

5. EMPIRICAL ANALYSIS ON THE FOUR PEAKS PROBLEM

We compared the effectiveness of the GA and PBIL on four peaks for different settings for T. Each algorithm was allowed 1500 generations per run. The total number of evaluations per run were: 300,000 for PBIL (1500x200), 750,000 for GA1-3: (1500x500), and 300,000 for GA4,5 (1500x200). In order to put the global maximum at 200 for all of the problems, the function was slightly modified to make the REWARD = 100 +T. Each algorithm was run twenty five times for each value of T.

Figure 6 shows the performance of the best 5 GAs and PBIL on four peaks as a function of T. As expected, as T gets larger, the problems get harder and the quality of the solutions deteriorates. The performance of PBIL, however, is comparable to, or better than, that of the GAs for all values of T. Thus PBIL, which explicitly maintains statistics which a GA holds in its population, but which does not cross solutions from different regions of the search space, performs at least as well as GAs that use crossover and that are optimized for this problem. Table I shows for each algorithm, the number of runs (out of 25 total) in which the algorithm achieved an evaluation greater than 100. An evaluation greater than 100 means that the algorithm found a solution with at least T ones and T zeros. See the Appendix for a typical run of PBIL on the four-peaks problem.

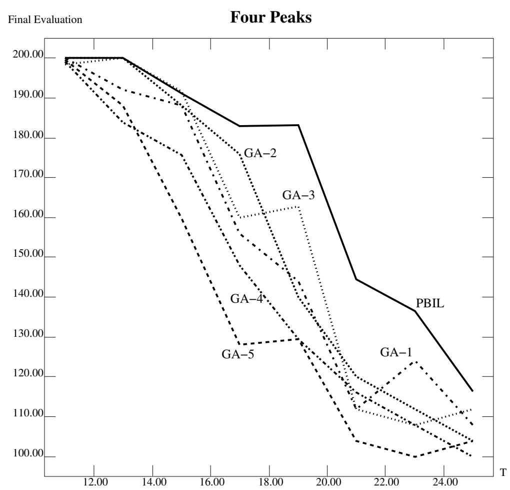

Figure 6: A comparison of the five best GAs, with PBIL. The X-Axis is the ‘T’ parameter in the four peaks problem. As T increases, the problem becomes more difficult. The Y-Axis is the average best evaluation of each algorithm, averaged over 25 runs. The optimal solution to each problem has an evaluations of 200.

Table I: Number of Runs out of 25 in which final evaluation was greater than 100.

	T							
Algorithm	**11**	**13**	**15**	**17**	**19**	**21**	**23**	**25**
GA-1	25	23	22	14	11	3	6	2
GA-2	25	25	22	19	10	5	3	1
GA-3	25	25	23	15	13	3	2	3
GA-4	25	21	19	12	2	4	2	0
GA-5	25	22	15	7	4	1	0	1
PBIL	25	25	25	24	23	15	12	5

5.1. Why Does PBIL Do as Well as GAs?

For crossover to discover individuals in the small basins of attraction surrounding the global optima, it must mate individuals from the basins of two different local minima. By maintaining a population of solutions, the GA is able—in theory at least—to maintain samples in different basins. Unfortunately, as mentioned before, most genetic algorithms are not good at maintaining this diversity. Premature convergence to solutions which sample few regions of the search space is a common problem. This deprives crossover of the diversity it needs to be an effective search operator on this problem. When this happens, crossover begins to behave like a mutation operator sensitive to the estimated reliability of the value of each bit [Eshelman, 1991]. If all individuals in the population converge at some bit position, crossover leaves those bits unaltered. At bit positions where individuals have not converged, crossover will effectively mutate values in those positions. Therefore, crossover creates new individuals that differ from the individuals it mates only at the bit positions where the mated individuals disagree. This is analogous to PBIL which creates new trial solutions that differ mainly in bit positions where prior good performers have disagreed.

On the four peaks problems, PBIL (which does not use crossover) performs comparably to the best GAs. Therefore, it is unlikely that the GA is benefiting from crossover’s ability to recombine building blocks from different local minima. Perhaps the main value of the GA’s population is as a means of maintaining statistics about the value of each bit position, as modeled in PBIL. PBIL works similarly to single population GAs because these cannot maintain diverse points in their populations. [1]

PBIL will not necessarily outperform GAs at all population sizes. As the population size increases, the observed behavior of a GA more closely approximates the ideal behavior predicted by theory [Holland, 1975]. For example, on the four peaks problems, for large enough population sizes, the population may contain sufficient samples from the two local minima for crossover to effectively exchange building blocks and find the global optima. Unfortunately, the desire to minimize the total number of function evaluations prohibits the use of large enough populations to make crossover behave ideally. PBIL will not benefit the same way from larger populations as GAs, since it only uses a few individuals from the population to update the probability vector. The main advantage of larger populations in PBIL is the potential for better individuals to update the probability vector.

---

1. We suspect that it is the need to maintain diversity that caused PBIL to perform better when updating the probability vector from the best 2 solutions than from just the best solution. This is currently under study. We have not experimented with updating from more than the best 2 solutions to avoid scaling issues such as having to scale the magnitude of the update by the relative quality of the solution. By using as few solutions as possible to update the vector we can safely avoid the problems associated with updates from poor performing samples.

6. DISCUSSION

6.1. Other Variations of PBIL

The PBIL algorithm described in this paper is very simple. There are variations of the algorithm that can improve search effectiveness. Two variations which have been tried include mutations and learning from negative examples. Mutations in PBIL serve a purpose analogous to mutations in GAs— to inhibit premature convergence. In GAs, when the population converges to similar solutions, the ability to explore diverse portions of the function space diminishes. Similarly, when the probability vector in PBIL converges towards 0s and 1s, exploration also is reduced. Mutations perturb the probability vector with a small probability in a random direction. The amount of the perturbation is generally kept small in relation to the learning rate.

A second variation is to learn from negative examples instead of only positive ones. In the PBIL algorithm described in this paper, the probability vector is updated towards the M best vectors in the population. However, the probability vector can also be shifted away from the worst vectors. In implementations attempted in [Baluja, 1995], the probability vector was moved towards the single best vector, and away from the single worst vector. Learning from negative examples improved the algorithm's performance in the problems attempted.

Another update method is to incrementally update the probability vector as each new trial is generated rather than updating it from only a few solutions in the new population. This is somewhat analogous to "steady-state GAs" that replace individuals in the population one or two at a time rather than replacing the entire population (as "generational" GAs do) [Syswerda, 1990][De Jong & Sarma, 1992]. These GAs have the potential of keeping more diverse members in their population for longer periods of time than generational GAs; this can aid population-based crossover operators in finding regions of high performance. In an incremental version of PBIL, the probability vector will be influenced by many more vectors than in the version of PBIL used here. This may have the effect of preserving more diversity in the generated solutions by making the probability vector more sensitive to differences in solution vectors. To ensure that more emphasis is placed on better solution vectors, the strength of the update to the probability vector would be moderated by the fitness of the individual relative to individuals seen in the past.

6.2. Experiments on Other Test Problems

PBIL's performance on the four peaks problem suggests it should compare favorably with traditional GAs. We might expect PBIL to do especially well, in comparison to GAs, on problems which are not custom designed to be GA-friendly. The results of a large scale empirical comparison of seven iterative and evolutionary based optimization heuristics support this [Baluja, 1995]. Because of space restrictions, only a brief overview of the experiments and results is reproduced here. Twenty-six optimization problems, spanning six sets of problem classes which are commonly attempted in genetic algorithm literature, were examined. The problem sets include job-shop scheduling, traveling salesman, knapsack, binpacking, neural network weight optimization and standard numerical optimization. These problems were chosen because much of the GA optimization literature has concentrated on exactly these, or very similar, types of scheduling, packing, routing, and optimization problems. Unlike the four peaks problem, these were typical benchmark problems, and were not custom designed to be GA-friendly. The parameters of all the algorithms were not tuned for each problem, rather the parameters were held constant for all runs. The settings of the parameters were chosen to give good performance on all of the problems, without biasing the parameters to any one specific problem.

The algorithms examined in the study were: two variations of PBIL, one which moved only towards the single best solution in each generation, and the other which also moved away from the worst generated solution. Both PBIL algorithms also employed a small mutation, which randomly perturbed the probability vector. Two variations of GAs were also examined. The first is very similar to the ones explored in this paper: elitist selection, two point crossover, 100% crossover rate, 100 population size, and mutation rate 0.001. The second used the same parameters except: uniform crossover and an 80% crossover rate. The second also scaled the evaluation of every solution in each generation by the evaluation of the worst generated solution (in the generation). Finally, three variations of next-step stochastic hillclimbing techniques were examined. These varied in how often restarts in random positions occurred, and whether moves to regions of equal evaluation were allowed. Each algorithm tested was given 200,000 evaluations of the goal function, and was run 20 times.

The results from the study indicated that using GAs for the optimization of static functions does not yield a benefit, in terms of either the final answer obtained, or speed, over simpler optimization heuristics such as PBIL or Stochastic Hill-Climbing. In the 26 problems attempted, PBIL with moves away from the worst solution, performed the best, in terms of final solutions obtained, on 21 problems. Learning from negative examples helped in 25 out of the 26 problems. Overall, PBIL with only moves towards good solutions performed next best. Hill-Climbing did the best on 3 problems, and the GA did the best on 2 problems. Details can be found in [Baluja, 1995].

We also compared PBIL to the traditional GA on the "Trap Function" devised by [Eshelman and Schaffer, 1993] to be crossover-friendly. For the traditional GA we used two-point crossover instead of one-point crossover because this problem is custom designed to work better with it. Preliminary results again suggest that there is very little difference between PBIL and the traditional GA on a problem custom tailored to demonstrate the benefit of population-based crossover.

6.3. Avoiding Premature Convergence

One solution to the problem of premature convergence is the parallel GA (pGA) [Cohoon et *al.*, 1988][Whitley et *al.*, 1990]. In the pGA, a collection of independent genetic algorithms, each maintaining separate populations, communicate with each other via infrequent inter-population (as opposed to intra-population) matings. pGAs suffer less from premature convergence than single population GAs: although the separate populations typically converge to solutions in just one region of the search space, different populations converge to different regions, thus preserving diversity across the populations. Inter-population mating permits crossover to combine solutions found in different regions of the search space. A pGA should outperform a traditional single population GA on the four peaks problems for large T because the pGA should maintain a more diverse set of solutions for crossover to use. This does not mean, however, that pGAs are inherently more powerful than PBIL. If a single PBIL outperforms a single GA, a set of parallel intercommunicating PBILs (possibly using a crossover-like operator to merge probability vectors) will likely outperform a set of parallel intercommunicating GAs. Preliminary results support this hypothesis.

7. CONCLUSIONS

Previous empirical work showed that PBIL generally outperformed genetic algorithms on many of the test problems commonly used to evaluate GA performance. Those test problems, however, were designed to be hard, but not particularly GA-friendly. This left open the possibility that although PBIL performed better on these problems, there were other problems better suited to GAs where they would outperform PBIL. This paper compares the performance of PBIL to traditional GAs on a problem carefully devised to benefit from the traditional GA's purported mechanisms [Holland, 1975]. PBIL still does as well as, or outperforms the GA. The benefit of the traditional GA's population and crossover operator may be due to the mechanisms PBIL shares with GAs: generating new trials based on statistics from a population of prior trials.

Perhaps the most important contribution from this paper is a novel way of thinking about GAs. In many previous examinations of the GA, the GA was examined at a micro-level, analyzing the preservation of building blocks, and frequency of sampling hyperplanes [Holland, 1975][Mitchell, 1994]. In this study, the behavior of the GA was examined at a higher level. This led to an alternate model of the GA. In the standard GA, the population serves to *implicitly* maintain statistics about the search space. The selection and crossover mechanisms are ways of extracting and using these statistics from the population. Although PBIL also uses a population, the population's function is very different. PBIL's population does not maintain the information that is carried from one generation to the next; the probability vector does. The statistics of the search are *explicitly* kept. PBIL performs similarly to how GAs perform in practice, even on problems custom designed to benefit from the population and crossover operator of standard GAs.

ACKNOWLEDGEMENTS

The authors would like to thank Justin Boyan for his comments on the paper and algorithm. Thanks are also due to Ari Juels and Robert Holte for their discussions about the Four Peaks problem.

Shumeet Baluja is supported by a National Science Foundation Graduate Fellowship. This research was also partially sponsored by the Wright Laboratory, Aeronautical Systems Center and the Advanced Research Projects Agency under grant F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation, ARPA, or the U.S. Government.

REFERENCES

Ackley, D.H. (1987) "An Empirical Study of Bit Vector Function Optimization" in Davis, L. (ed) *Genetic Algorithms and Simulated Annealing*, 1987. Morgan Kaufmann Publishers, Los Altos, CA.

Baluja, S. (1995) "An Empirical Comparison of Seven Iterative and Evolutionary Optimization Heuristics". Carnegie Mellon University. Technical Report.

Baluja, S. (1994) "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning". Carnegie Mellon University. Technical Report. CMU-CS-94-163.

Boese, K.D. and Kahng, A.B. (1994) "Best-so-far vs. where-you-are: implications for optimal finite-time annealing." *Systems & Control Letters*, Jan. 1994, vol.22, (no.1).

Cohoon, J.P., Hedge, S.U., Martin, W.N., Richards, D. (1988) "Distributed Genetic Algorithms for the Floor Plan Design Problem". Technical Report TR-88-12. School of Engineering and Applied Science, Computer Science Department, University of Virginia.

De Jong, K. (1975) *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. Dissertation.

De Jong, K. (1992) "Genetic Algorithms are NOT Function Optimizers". In Whitley (ed.) *FOGA-2 Foundations of Genetic Algorithms-2*. 5-17. Morgan Kaufmann Publishers. San Mateo, CA

De Jong, K. & Sarma, J. (1992) "Generation Gaps Revisited". In Whitley (ed.) *FOGA-2 Foundations of Genetic Algorithms-2*. 19-28. Morgan Kaufmann Publishers. San Mateo, CA

Davis, L. (1991) "Bit-Climbing, Representational Bias, and Test Suite Design", *Proceedings of the Fourth International Conference on Genetic Algorithms*. (18-23). Morgan Kaufmann Publishers. San Mateo, CA

Eshelman, L.J. (1991) "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination". In G.J.E. Rawlins (editor), *Foundations of Genetic Algorithms*, 265-283.

Goldberg, D. E. & Richardson, J. (1987) "Genetic Algorithms with Sharing for Multimodal Function Optimization". In Grefenstette (ed.) *Proceedings of the Second International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA.

Golberg, D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*.

Hertz, J., Krogh, A., & Palmer, G. (1993) *Introduction to the Theory of Neural Computation*. Addison-Wesley.

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

Mitchell, M, Holland, J.H. & Forrest, S. (1994) "When Will a Genetic Algorithm Outperform Hill Climbing?" in *Advances in Neural Information Processing Systems 6*. (ed) Cowan, J. Tesauro, G., Alspector, J., Morgan Kaufmann Publishers. San Francisco, CA.

Eshelman, L.J. & Schaffer, D. (1993) "Crossover's Niche". In Forrest (ed). *(ICGA-5) Proceedings of the Fifth International Conference on Genetic Algorithms*. 9-14. Morgan Kaufmann Publishers. San Mateo, CA.

Juels, Ari [1993, 1994] Personal Communication.

Syswerda, G. (1989) "Uniform Crossover in Genetic Algorithms". In *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, 2-9. J.D. Schaeffer, ed. Morgan Kaufmann.

Syswerda, G (1990) "A Study of Reproduction in Generational and Steady-State Genetic Algorithms. In *Proceedings of the Foundations of Genetic Algorithms Workshop*. Indiana, July 1990.

Syswerda, G. (1992) "Simulated Crossover in Genetic Algorithms". In Whitley (ed.) *FOGA-2 Foundations of Genetic Algorithms-2*. 239-255. Morgan Kaufmann Publishers. San Mateo, CA

Whitley, D. & Starkweather, T. (1990) "GENITOR II: A Distributed Genetic Algorithm". *Journal of Experimental and Theoretical Artificial Intelligence* 2: 189-214.

APPENDIX

A typical run of the PBIL algorithm is shown in Figure 7. The Four Peaks problem was set at T=15.

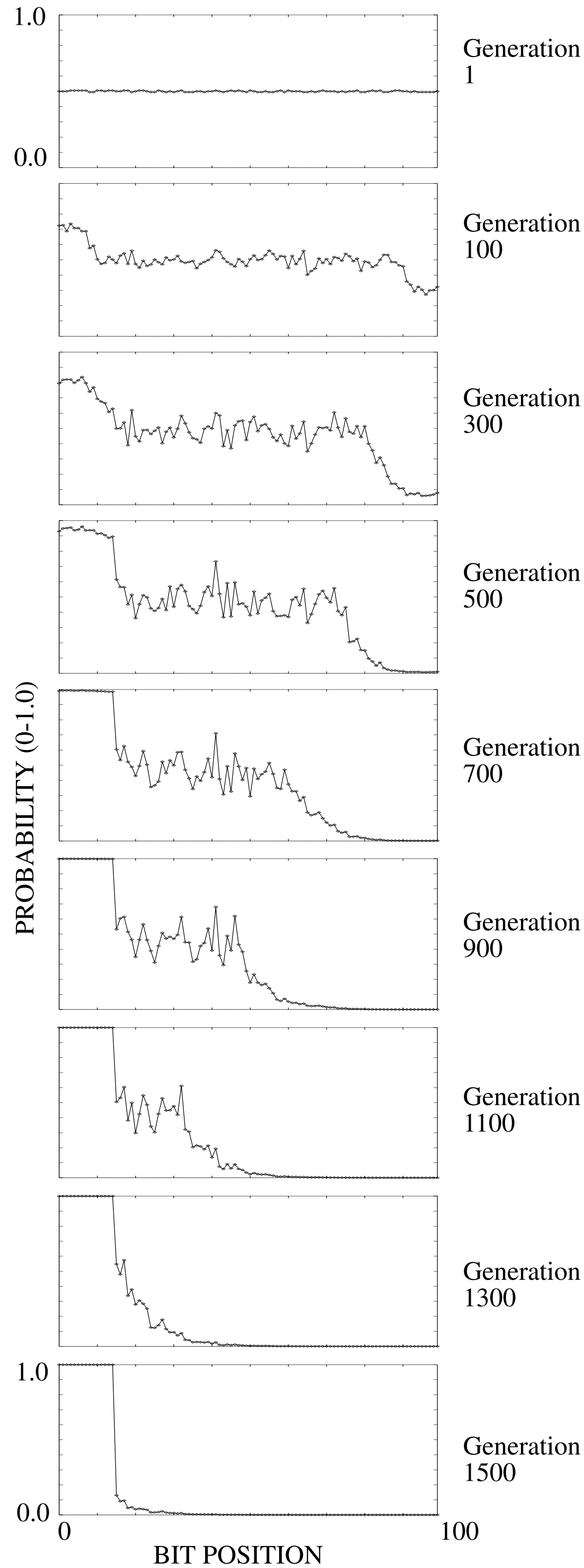

Figure 7: Evolution of the probability vector for a typical run of PBIL on the Four Peaks problem at T=15.