A Comparison of Cohort Genetic Algorithms with Canonical Serial and Island-Model Distributed GA's

Huafeng Pei **Erik Goodman**

Genetic Algorithms Research and Applications Group (GARAGe)
Case Center, Michigan State University, 2857 W. Jolly Road, Okemos, MI 48864
Email: goodman@egr.msu.edu

Abstract

This work considers the cohort genetic algorithm, a new type of genetic algorithm introduced by Holland. The cohort GA differs in several ways from the traditional canonical serial GA and island-model distributed GA. A key motivation for its development was to reduce "hitchhiking" – premature convergence of currently low-significance loci located near loci at which good building blocks are found early in the search process. This work compares one version of the cohort GA with canonical serial and island-model distributed GA's on the basis of their abilities to reduce hitchhiking. The comparison is done using two types of test functions: the "royal road with potholes" function and hyperplane-defined functions ("HDF's"). It is experimentally shown that even though theoretically the cohort GA can reduce hitchhiking, the particular version of the cohort GA tested is prone to another form of premature convergence, and it performed worse than the other GA's. It is also shown that a small change in the placement of offspring among cohorts in the cohort GA may dramatically improve its performance. This suggests that further work on the cohort GA may well be fruitful.

1 INTRODUCTION

The genetic algorithm (GA) is a family of search methods introduced by Holland [1975]. Much research has been done in order to understand how the GA works and how to improve its performance.

The cohort GA is a new type of GA designed more recently by Holland [1998] [2000]. It is aimed at reducing the "hitchhiking" effect that occurs in the process of a GA's search. Hitchhiking is a form of premature convergence that can hinder the GA or even make it unlikely for the GA to find a good solution for a given problem. Hitchhiking is most severe when the maximum reproduction rate is relatively high – for example, if the expected number of offspring of the best individual in the population is on the order of two or more. Hitchhiking is reduced when fitness is scaled so that the expected number of copies of the most fit individual produced in the next generation is 1.2 or fewer, but then, as Holland points out, other problems arise: 1) exploitation of the fitness difference is slowed, and 2) there is higher variance in the sampling of the fitness distribution (many times, individual with better-than-average fitness will be lost from the population). This higher variance occurs because GA's typically use any fractional fitness excess above 1.0 as a probability of creating a second copy of an individual in the next generation. Thus the best individual's gain becomes uncertain. The cohort GA is designed to allow a reduction in reproduction rates without introducing this stochastic sampling problem. Holland conjectured that the cohort GA's mechanism, with relatively low maximum scaled fitnesses, will reduce hitchhiking, thus improving the performance of GA on classes of problems in which hierarchical assembly of building blocks is important to the solution trajectory.

In this work, we tested the hypothesis that a cohort GA can reduce the hitchhiking effect and therefore improve the performance of GA's by comparing it with a canonical serial GA and an island-model distributed GA on two types of seemingly appropriate test functions. We used a version of Holland's cohort GA provided by Belding [Holland, 1998], a student of Holland. We note that Holland's most recent publication on the cohort GA [Holland, 2000] includes some new mechanisms he has introduced to fight the convergence issues we (later) found in our work with his earlier version of the cohort GA; we have not yet experimented with his newer formulation.

Section 1 introduces the hitchhiking effect and the cohort GA. Section 2 presents the experimental design. Results are given in Section 3, and conclusions in Section 4.

2 HITCHHIKING AND THE COHORT GA

Hitchhiking is the effect that when a building block (short, high-fitness schema) is discovered, the alleles at loci *near but outside* the building block spread through the population almost as rapidly as the building block itself [Holland, 1998] [Mitchell, *et al.*, 1992] [Forrest & Mitchell, 1992]. When the particular alleles at these nearby loci make little or no contribution to the fitness, their hitchhiking results in reducing exploration in the parts next to the building block (due to premature convergence at those loci). This often severely slows discovery of building blocks near one already found. Holland [1998] argues convincingly that hitchhiking is made severe by high maximum reproduction rates. In order to enable the building blocks that are presented in the best individuals to spread through the population quickly, exploiting the knowledge they bear, fitnesses are often scaled to allow the best individual in the population to produce 2 or more offspring in each generation. This high maximum reproduction rate makes the hitchhiking problem severe.

Of course, increasing the mutation rate while keeping the maximum reproduction rate high can reduce hitchhiking. However, a high mutation rate can reduce the ability of the population to retain information about building blocks that have already been discovered.

If the maximum reproduction rate is set to a low value, such as 1.1 or 1.2, the mutation rate can be set low enough to retain building blocks that have already been discovered (with few crossovers and mutations in each generation, most individuals survive unchanged). But the number of generations until the second offspring of the best individual occurs becomes uncertain, because the second offspring is generated with probability much less than 1 in each generation. Such a low reproduction rate becomes a source of extreme variance. To control this variance and reduce hitchhiking, Holland proposed the cohort genetic algorithm.

The cohort GA is intended to reduce hitchhiking by lowering the reproduction rate without using random numbers and probabilities to control the number of offspring produced [Holland, 1998]. In the cohort GA, the population is divided into an ordered set of subpopulations. These subpopulations are called cohorts. The initial population is generated randomly, and distributed evenly among cohorts. Individuals in cohort 1 will produce offspring first. Then the individuals in the successive cohorts will have chances to produce. When the last cohort is reached, the process begins again with cohort 1. Reproduction is thus carried out by cycling through the cohorts. When it is time for an individual in a cohort to reproduce, the individual crosses over with another individual in the same cohort, then the offspring undergo mutation probabilistically. The fitness of an offspring determines the cohort into which the offspring is put, which determines when its turn is to produce offspring. In this way, a string with high fitness will produce offspring sooner than a string with low fitness. Over an extended interval, the string with higher fitness will produce more offspring than the string with lower fitness, and in a deterministic manner.

All strings in the population can produce a fixed number of offspring when it is their turn to reproduce, no matter what their fitness values are. This method lowers the reproduction rate. Theoretically, this method can reduce hitchhiking. In order to verify this idea, we conducted a comparison study of a cohort GA with a canonical serial GA and an island-model distributed GA, which we will describe in Sections 3 and 4.

3 EXPERIMENTAL DESIGN

The main purpose of this work is to test whether or not a cohort GA can reduce the hitchhiking effect. By comparing a cohort GA with a canonical serial GA and an island model distributed GA, we also tested whether a cohort GA is more efficient than other GA's. Another objective of this work is to test how some factors affect a cohort GA's performance.

3.1 TESTBEDS – ROYAL ROAD AND HDF

3.1.1 Royal Road Function

The hitchhiking phenomenon was noted by Holland while using his original RR functions to test the building block hypothesis. Thus it is natural to use the RR function as a testbed to compare the cohort GA's performance with other GA's. The Royal Road (RR) function used in this work is Holland's revised version of the RR function.[1] We also took Holland's default setting for the parameters of the RR. The characteristics of the RR function, such as known building blocks, known fitness and a hierarchical structure of building blocks, should enable us to track the GA's performance over time. The revised RR function is especially good for testing because it incorporates the idea of "deception" or a fitness valley that the GA must cross to find a global optimal solution.

3.1.2 HDF's

Hyperplane-defined functions (HDFs) are a set of randomly generated functions. They are designed to allow a large number of building blocks to be combined in a variety of ways. They also incorporate the idea of "pot holes", which refers to the fitness valleys that must be crossed in order to reach the global optimum. The interactions among the building blocks are more dramatic in HDFs than in RR. The HDF used in this work is generated by the code provided by Holland in [1998] with

[1] Holland presented this version of RR at the Fifth International Conference on Genetic Algorithms in 1993, then posted it to the Internet "GA Digest" mailing list. We take Jones's [1995] description as a reference.

the following parameter settings: *chrl* = 80, *nelt* = 8, *minl* = 6, *maxl* = 12, *npr* = 2 and *nocom* = 5.

The advantages of HDFs are they are easy to generate, hard to reverse-engineer (so not easy for the GA-designer to "cheat" on), and easy to analyze after the fact – i.e., very suitable for testing GA's [Holland, 1998].

3.2 PERFORMANCE CRITERIA

Based on the different nature of the two test functions, we chose different criteria to measure the GA's performance on them. The RR function has explicit levels. Therefore, for the RR, we measured the number of function evaluations required to achieve a certain RR level. For the HDF, in order to measure the degree of convergence in the intron part of the chromosome, we generated a relatively short chromosome (length 80). For each intron locus, we measured the difference between the total number of ones and the total number of zeros of all chromosomes in the population and used the absolute value of their difference as a criterion. We also measured the maximum fitness value that the various GA's achieved given a fixed number of function evaluations.

3.3 COHORT GA IMPLEMENTATION DETAILS

There are many ways to implement the cohort GA's central idea. We used Holland's [1998] version as a starting point and also made some modifications for some of our tests.

Several factors may affect a cohort GA's performance, such as population size, offspring placement strategy, deletion strategy and crossover candidates chosen. These factors are considered during the tests, corresponding changes are made to the original implementation, and repeated tests are done with different settings and implementations.

3.3.1 Population Size

The cohort GA must be given the relationship between the number of cohorts, *nocoh*, and the size of each (lencoh). Holland's default setting is *nocoh* = 20 and lencoh = 20; thus population size is 400. We chose various values for *nocoh* and lencoh and different combinations of these values to perform the tests in an attempt to see how these two parameters affect the cohort GA's performance.

3.3.2 Offspring Placement Strategy

The offspring placement strategy specifies which cohort an offspring with a certain fitness will be placed in. It will affect selection pressure as well as interactions between cohorts. In Holland's original implementation, an offspring with fitness *u* is placed in cohort *d*, where *d* is determined by the equation:

$$d = \mathrm{mod}(t + doub, nocoh),$$

where *t* is the current cohort number and

$$doub = \lceil 2 \times umax / u \rceil,$$

where *umax* is maximum fitness value found so far. (Holland has used more sophisticated strategies in his later-reported work.)

In this way, an individual with fitness *umax* is placed in the cohort next to the current cohort. Another individual is placed in a cohort based on the ratio of *umax* and its fitness value. An individual with higher fitness value will be put nearer the current cohort, and vice-versa.

During the experiments in RR, we found out that with this implementation, the individuals tend to accumulate in a small number of cohorts. In order to spread the individuals among all the cohorts and keep the cohort GA working as intended, we tried a new placement strategy with

$$doub = \lceil (nocoh - 1) + (u - umin) \times (2 - nocoh + 1) / (umax - umin) \rceil$$

and *umin* is the minimum fitness found so far. In this way, a chromosome with the fitness *umin* will be placed in the cohort *nocoh* –1 from the current cohort (which is the farthest cohort from the current cohort) and a chromosome with the fitness *umax* will be placed in the cohort next to the current cohort. Other individuals will be placed in cohorts between 2 and *nocoh* – 1 steps removed from the current cohort, where the *doub* value is proportional to the individual's fitness value.

The above strategy is deterministic. We also tried putting the offspring in a randomly selected cohort subject to some probability distribution, in order to produce more migration effect by providing the opportunity for inter-cohort mating. But, of course, this introduces just the sort of stochastic variability that Holland is seeking to minimize with the cohort GA.

3.3.3 Deletion Strategy

Each pair of individuals produces four offspring. To keep the population size constant, the parents are deleted and two other chromosomes randomly selected from two other cohorts are also deleted. The cohorts that are the source of chromosomes for deletion are also randomly selected. The deletion of random chromosomes from random cohorts affects the cohort GA's ability to keep the good individuals found so far; it also affects selection pressure. (In fact, one might look at it as re-introducing stochastic variation in effective fitness-proportional reproduction that the cohort GA was trying to reduce.) In the original implementation, the source cohorts from which to delete chromosomes are randomly selected from cohort positions *nocoh*/2 to *nocoh*-1 relative to the current cohort; that is, the distant half of the cohorts. We also tried deleting chromosomes randomly from 2 to *nocoh* –1

relative to the current cohort to lower selection pressure.

3.3.4 Crossover Candidates

In the original implementation, both parents are selected from the current cohort. In addition to this, we also tested having the one parent selected from the current cohort and with some probability, another parent randomly selected from another cohort. This strategy was also an attempt to provide inter-cohort mating and avert the "clustering" found in the original cohort GA.

3.4 DETAILS OF CANONICAL SERIAL GA AND ISLAND-MODEL DISTRIBUTED GA

Though the architecture of a typical GA is well known, implementation details vary from system to system. Even very small differences in implementation may result in significant changes. The canonical serial GA and island-model distributed GA software used was "GALOPPS" (The "Genetic Algorithm Optimized for Portability and Parallelism" System) [Goodman, 1996].

3.4.1 Implementation of Canonical Serial GA

1) Make initial population with uniform random bits.
2) Evaluate fitness of each new individual in current generation, fitness statistics.
3) Terminate the program if stopping criterion met.
4) Select survivors and parents for next generation, using "stochastic universal sampling" method to pick a list of chromosomes that will be the parents or the survivors for next generation, sampling with replacement; list size = population size.
5) Reproduce in one of two ways: "standard" canonical serial GA, or allowing niching of the population by using crowding and incest reduction. Niching is used to try to reduce premature population convergence. Use of this method helped us to see where the cohort GA stands in comparison to other ways of reducing premature convergence.

   In a "standard" serial GA, both parents are uniform randomly selected from the list generated in step 4. One-point crossover and single-bit mutation or multi-bit mutation are performed according to the crossover rate and mutation rate. The offspring then replace the parents.

   The niching technique included two mechanisms: DeJong crowding and incest reduction [Goodman, 1996]. With incest reduction, pairs for crossover are picked by choosing the first parent at uniform random from the above list, then uniform randomly choosing several candidates for the other parent (here, 3 candidates). Among these candidates, the one with the greatest Hamming distance from the first parent is picked as the second parent, helping to reduce, for example, crossover between individuals which are identical or nearly so. After crossover (and any mutations) are done, for each child, "crowding-factor" (here, three) members of the above list are selected (at uniform random). Among the three candidates, the one with smallest Hamming distance from the child is replaced.

   The list generated by step 4 is not altered in this process. All individuals in this list are used in some crossover and/or mutation operation, or else survive unaltered into the next generation.
6) Go to step 2.

3.4.2 Implementation of Island-Model Distributed GA

The island-model distributed GA divides the whole population into several subpopulations. It provides a chance for parallel execution by allowing use of several processors or computers. In our experiments, we used one workstation to serially simulate parallel execution. In that approach, we take advantage only of the distributed GA's ability to reduce premature convergence, but not its capability for parallel execution.

In this case, each subpopulation is simulated for some number of generations (a "cycle"). Each population receives one turn per cycle. At the beginning of each population's turn, it reads one or more individuals from each of its declared neighboring subpopulations according to a migration table. Migrants are duplicated, not removed from the source population. In addition to defining neighbors, the migration table also says how many individuals are to migrate in from each neighbor each cycle, whether these migrants include the best individual and/or some number of randomly selected individuals, and which strategy is to be used for replacing existing individuals by migrants. The migration incest reduction and migration crowding parameters are used to direct the donating and receiving process. When a migrant is to be selected randomly and migration incest reduction is specified (non-zero), that number of candidates is first randomly chosen. The one with the farthest Hamming distance from the best individual of the receiving subpopulation is selected. Migration crowding means the random choice of k candidates for replacement in the receiving subpopulation and picking for replacement the one that is closest in Hamming distance to the migrant. Then the run of each subpopulation follows the canonical serial GA with crowding and incest reduction.

4 RESULTS

These experiments investigate the cohort GA by comparing it with a canonical serial GA and island-model distributed GA, and by studying issues involved in implementing the cohort GA. Each of these results on the RR function and the HDF was an average of 20 runs.

4.1 COMPARISON OF RESULTS ON RR

4.1.1 Initial Experiments

The first set of experiments investigated the performance of five different GA's on the RR function. The GA's included: original cohort GA, cohort GA with new placement implementation, the island-model distributed GA, the canonical serial GA and the canonical serial GA with niching.

The original cohort GA settings were as follows:

Number of cohorts: 20
Initial size of each cohort: 20 (population size: 400)
Offspring placement strategy:
$doub = \lceil 2 \times umax / u \rceil$
(note that d's calculation is always the same)
Deletion strategy: $nocoh/2$ to $nocoh - 1$
Crossover: within the same cohort
Stopping criterion: function evaluations > 300,000

We also ran the cohort GA with a new offspring placement strategy:

$$doub = \lceil (nocoh - 1) + (u - umin) \times (2 - nocoh + 1) / (umax - umin) \rceil$$

The tests on canonical serial GA's and island-model distributed GA were done with population size 400 and:

Canonical serial GA:

Crossover rate: 0.15 (one point crossover)
Mutation rate: 0.0002/bit (0.048/chromosome)
Linear scaling, with best fitness/mean fitness = 1.25
Stopping criterion: generation when function evaluations > 300,000

When niching was used:

Parameters are the same as canonical serial GA, plus:
Crowding factor: 3; Incest reduction: 3

Island-model distributed GA: 8 subpopulations, 50 each

The settings of each subpopulation were the same as for the canonical GA with crowding and incest reduction.

Number of cycles: 10
Neighbors of each subpopulation: 2 adjacent in a ring
Number of migrants: 2 (one is the best, one is random)
Migration incest reduction: 3
Migration crowding factor: 4

The parameter settings above are the defaults. In the following experiments we report only the exceptions.

Table 1 lists the results on the RR function, of five GA's with population size of 400. (In Tables 1-7, an average number shown without a parenthesized number means 100% of runs achieved that level; blank means that the level was never achieved in 300,000 function evaluations.) Surprisingly, the cohort GA performed the worst. In a total of 20 runs at this setting, the cohort GA achieved only level 1 and only in 5 runs. The results with the new placement implementation were much better. All runs reached level 1 and 70% of the runs reached level 2. The other GA's achieved levels 1 and 2 with fewer function evaluations, but only in a smaller percentage of the runs. They usually did not reach level 2 within the number of function evaluations allowed.

Table 1: Function evaluations until a RR level is achieved and (% of runs in which GA achieved that level – 100% when not specified, unless cell is blank, which means that level was not achieved in any run).

		level 1	level 2	level 3
original cohort GA	average	61326 (25%)		
	std. dev	60606		
cohort GA with new placement implementation	average	1786	28779 (7%)	
	std. dev	923	41898	
Island-model distributed GA	average	11344	59046 (65%)	
	std. dev	8975	24390	
Canonical Serial GA	average	1927	5552 (45%)	
	std. dev	853	1572	
Canonical Serial GA with niching	average	1885	5306 (35%)	
	std. dev	720	678	

Traditional GA's have been found to work better on RR when the population size is relatively large. In that case, the initial population typically contains most or all of the basic building blocks needed to get to further levels through crossover. Otherwise, it can take a GA a long time to make the basic building blocks through mutation, especially because of the "potholes" (deception in the fitness function). Therefore, larger population sizes were used in the rest of the experiments on RR.

4.1.2 Varying Cohort Numbers and Initial Sizes

To test the effect of population size on the cohort GA, we ran the cohort GA's with different numbers of cohorts (20, 35, and 50) and different initial sizes of each cohort (20, 40, and 80). Thus, the population sizes varied from 400 to 4000. The results are listed in Tables 2 to 5.

Table 2 shows that with the original cohort GA, the average number of function evaluations used to achieve level 1 was relatively small with the number of cohorts equal to 20 and the cohort sizes at 40 or 80. With a cohort size of 80 and the number of cohorts at 35 or 50, most of

the runs reached level 1, but needed many more function evaluations. This could be because the population sizes were much bigger in these two cases. The initial population is more likely to contain instances of level 1 building blocks or schemata similar to them. In the latter case, after a long period of search, mutation led the RR to reach level 1. This confirmed that we need a bigger population size for RR. However, even though the number of cohorts, the cohort size and their ratio had some impact on the original cohort GA, it never achieved RR level 2 in any of the runs we tried, due to the rapid accumulation of individuals in only few cohorts.

Table 2. For original cohort GA, average number of function evaluations to achieve level 1

cohort size		# cohorts 20	# cohorts 35	# cohorts 50
20	average	61327 (25%)	38282 (40%)	60356 (45%)
	std. dev	60606	33704	49227
40	average	6093 (45%)	9124 (60%)	34577 (60%)
	std. dev	15062	19969	62927
80	average	16136 (50%)	139569 (80%)	151198
	std. dev	45328	157056	171798

With the new offspring placement implementation, most cohort GA runs reached RR level 2, and some reached RR level 3, as shown in Tables 4 and 5. This shows that the new offspring placement implementation improved the performance of the cohort GA. The new offspring placement implementation was used for all remaining runs reported. The cohort GA's performance didn't change linearly with changing of the number of cohorts, cohort size or population size. But with number of cohorts at 20, the cohort GA performed better, especially with initial cohort size of 80.

Table 3: For cohort GA with new placement implementation – function evaluations to reach level 1

cohort size		# cohorts 20	# cohorts 35	# cohorts 50
20	average	1786	2854	3215
	std. dev	923	1063	1970
40	average	3352	3697	4392
	std. dev	1412	2295	2962
80	average	5151	4727	6541
	std. dev	2093	3288	4430

Based on this observation, we proceeded with tests with the number of cohorts equal to 20 and 35 and the cohort size varying from 20 to 200, in an attempt to see whether the ratio of the cohort size and the number of cohorts really has some effect on the cohort GA's performance. The results of 20 cohorts are listed in Table 6.

Table 4: For cohort GA with new placement implementation -- function evaluations to reach level 2

cohort size		# cohorts 20	# cohorts 35	# cohorts 50
20	average	28779 (70%)	37431 (80%)	44014 (55%)
	std. dev	41898	17337	27720
40	average	49627 (80%)	52819 (75%)	46973 (75%)
	std. dev	43897	26108	36142
80	average	35433 (95%)	59598 (80%)	64613 (85%)
	std. dev	18686	32518	41379

While setting the number of cohorts at 20 still gave overall better performance, the performance did not improve linearly with an increase in the initial cohort size. The ratio of the cohort size and the number of cohorts does not seem to determine the cohort GA's performance.

The island-model distributed GA and canonical serial GA's were tested on population sizes of 1600 and 4000. These population sizes are those that gave better performance using the cohort GA. Table 7 lists the comparison results for population size of 4000.

Table 5: For cohort GA with new placement implementation, function evaluations to achieve level 3

cohort size		# cohorts 20	# cohorts 35	# cohorts 50
20	average		65647 (15%)	73216 (10%)
	std. dev		14330	11677
40	average	148545 (10%)	165175 (15%)	
	std. dev	14332	61756	
80	average	107102 (30%)		316671 (5%)
	std. dev	7142		

The results show that among the four GA's, the island-model distributed GA and the two canonical serial GA's

gave significantly better results than the cohort GA. This may indicate a defect in the implementation of the cohort GA or weaknesses in relation to the RR's challenges.

Table 6: Function evaluations to reach each level, with 20 cohorts. Stopping criterion = 500,000 evaluations.

cohort size		level 1	level 2	level 3	level 4
20	average	1786	28779 (70%)		
	std. dev	923	41898		
40	average	3352	49627 (80%)	148545 (10%)	
	std. dev	1412	43897	14332	
80	average	5151	35433 (95%)	107102 (30%)	
	std. dev	2093	18686	7142	
120	average	5586	43111 (85%)	165575 (40%)	
	std. dev	2981	20938	61669	
160	average	7800	56257 (90%)	172955 (35%)	
	std. dev	4351	26904	50264	
200	average	6577	49952 (80%)	193042 (20%)	420068 (5%)
	std. dev	4412	25705	71593	

The new offspring placement implementation obviously improves the cohort GA performance, but during the experiments we can still see that the individuals tend to accumulate in a small number of cohorts instead of spreading among all the cohorts, after a certain number of cycles. Tables 8 and 9 illustrate the degree of accumulation using the original and new offspring placement implementations, respectively. The number of cohorts here is 20 and initial cohort sizes are 20. With the original offspring placement, after only 20 cycles and about 1000 function evaluations, the population has prematurely converged with the maximum fitness 1.86, RR level 0. The individuals have accumulated in 10 cohorts instead of being spread among 20 cohorts. With the new offspring placement, after 160 cycles and 8000 function evaluations, the population has converged with the maximum fitness 4.3, RR level 1.

The result of this accumulating is a kind of premature convergence. A large number of individuals tend to gather within a few cohorts, indicating that their fitness values are similar to the degree that they could not be separated by the current offspring placement strategy.

4.1.3 Varying Crossover Candidate and Offspring Placement

To give a better chance for the individuals with less similar structures to mate, we tried two strategies that will enable inter-cohort crossover: choosing different crossover candidates and non-deterministic placement of the offspring.

Table 7: Function evaluations to reach a level, with population size 4000. Island models used 8 subpopulations of 200 each.

GA type		level 1	level 2	level 3
cohort GA with new placement implementation	average	6577	49952 (80%)	193042 (20%)
	std. dev	4412	25705	71593
Island-model distributed GA	average	3353	36117	76048
	std. dev	185	5782	16013
Canonical Serial GA	average	8210	38561	79776
	std. dev	5458	8578	13681
Canonical Serial GA with niching	average	8125	39345	84292
	std. dev	5279	7854	17292

To change the crossover candidate, one candidate is still selected from the current cohort, but with probability 0.1, the other candidate is chosen from another randomly chosen cohort. In another words, one-tenth of the second candidates do not come from the current cohort.

In non-deterministic offspring placement, the offspring may be placed in a randomly selected cohort, rather than using the calculation of *d*. The probability of this random placement was also set to 0.1.

The results showed that, with these settings, the cohort GA needs more function evaluations to achieve a certain level. They showed that employing a form of inter-cohort crossover did not improve the cohort GA's performance. This might be due to a potential defect of the implementation of inter-cohort crossover. It might also be due to the fact that the RR's saturation effect is too strong to be overcome by the cohort GA.

In an attempt to alleviate the premature convergence, we also tried to reduce the selection pressure by changing the implementation of deletion from being *delete from second half of the cohorts* to being *delete from all cohorts except the current cohort*. With this change, the performance of the cohort GA went down. This showed that the selection pressure alone is not a big factor in causing premature convergence of the cohort GA on RR.

4.1.4 Summary

The experiments described in this section investigate the

cohort GA's performance on the RR function, Several parameters and implementations were changed in order to test their effects on the cohort GA's performance.

The results of these experiments indicated that the number of cohorts and the offspring placement have the most effect on cohort GA performance on RR. Twenty cohorts gave the best overall performance, especially when the initial cohort size was 80. But this ratio of cohort size and the number of cohorts was not found to be generalizable. A new implementation of offspring placement in an attempt to spread all the individuals among all the cohorts yielded a great improvement in the cohort GA. But compared with an island-model distributed GA and two canonical serial GA's, the best cohort GA's performance remained worse. The RR function's drawbacks might cause this version of the cohort GA to fail. But there might be some practical problems that also have the characteristics of the RR function. So there is a need to find another way to implement the cohort GA's central idea and avoid the problems found here (a candidate perhaps being [Holland, 2000])

Table 8: Maximum fitness values and cohort sizes recorded in a cohort GA at various numbers of cycles -- original offspring placement implementation.

Number of Cycles	Max. Fitness Value	Cohort Sizes
0 (Initial cohort sizes)		{all at 20}
5	1.84	{0, 0, 0, 0, 0, 43, 35, 42, 53, 45, 44, 30, 22, 16, 22, 10, 5, 8, 6, 19}
10	1.84	{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 70, 61, 57, 59, 58, 51, 22, 15, 5, 1}
20	1.86	{80, 56, 60, 56, 51, 41, 28, 19, 7, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}
30	1.86	{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 61, 58, 62, 57, 51, 44, 44, 17, 3, 3}
40	1.86	{65, 50, 65, 57, 64, 43, 34, 14, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
80	1.86	{48, 67, 66, 61, 48, 40, 28, 20, 15, 6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}
160	1.86	{45, 53, 63, 56, 64, 44, 39, 29, 6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

Table 9: Maximum fitness values and cohort sizes, cohort GA run, vs. number of cycles through cohort, using new offspring placement method.

Number of Cycles	Max. Fitness Value	Cohort Sizes
0 (Initial cohort sizes)		{all at 20}
5	1.84	{6, 2, 5, 1, 0, 19, 17, 17, 20, 15, 11, 13, 18, 28, 31, 29, 39, 41, 52, 35}
10	1.96	{23, 17, 27, 15, 16, 12, 3, 3, 2, 0, 8, 12, 15, 27, 30, 26, 35, 41, 51, 36}
20	2.02	{14, 9, 23, 6, 35, 10, 18, 15, 8, 15, 32, 33, 49, 44, 33, 25, 15, 10, 5, 0}
30	2.26	{17, 13, 7, 5, 2, 5, 3, 0, 0, 0, 57, 35, 59, 56, 44, 34, 21, 17, 14, 10}
40	2.34	{30, 44, 24, 30, 20, 12, 46, 27, 43, 39, 42, 8, 10, 16, 6, 1, 1, 0, 0, 0}
80	3.52	{1, 39, 52, 57, 30, 54, 69, 33, 37, 16, 9, 0, 1, 0, 0, 1, 0, 0, 0, 0}
160	4.3	{6, 62, 82, 74, 77, 47, 43, 7, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0}
180	4.3	{21, 0, 0, 0, 0, 0, 0, 0, 0, 119, 111, 78, 48, 20, 2, 0, 0, 0, 0, 0}

4.2 COMPARISON RESULTS ON HDF

To use HDF as a direct tool to verify the cohort GA's effect on hitchhiking, we generated a small HDF with chromosome size equal to 80. It is easier to look directly at the intron loci when the chromosome size is small. The measurement we used was to calculate the total number of zeroes and the total number of ones at each intron locus in the whole population. Then the sum of their absolute differences should indicate the degree of convergence of intron loci.

Because HDF's do not have an explicit concept of level, we measured maximum fitness values that the different GA's achieved with the number of function evaluations at 2500 and 6000. We chose the maximum number of 6000 because the GA's likely either have already found the optimum solution or have prematurely converged by that time. In these experiments, we only compared cohort GA, canonical serial GA and canonical serial GA with niching.

4.2.1 Varying population size

In the initial experiments we used Holland's original implementation for the cohort GA. The population sizes were 200, 400 and 800. The number of cohorts was 20 and the cohort sizes were 10, 20 and 40. Table 10 lists the results for population size 200. Since the results were quite similar, we did a t-test for significance of the differences in the data from the 20 runs of each type.

The results after 6000 function evaluations are similar to those after 2500. This showed that at both times, the canonical serial GA with niching yielded better results. Especially for the sum of differences of the intron parts, the canonical serial GA with niching did significantly better than without niching. (when niching was used, the convergence of the intron part was greatly reduced). Even though the convergence of the intron part of the cohort GA was at about the same level as that of the canonical serial GA with niching, its maximum fitness level was significantly lower than those of the other two GA's, so its lower convergence should not be given much weight.

The results for population sizes 400 and 800 are similar to those for 200, except that they show little difference when the number of function evaluations was only 2500, because when population sizes are larger, it takes more evaluations to initialize the starting populations. But later, the same pattern appeared as with population size 200.

Table 10: Max fitnesses and sum of differences in intron parts after 2500 evaluations. Population size = 200. Below: t-tests of significance. (1, 2) compares canonical serial GA and canonical serial GA with niching. Bolded numbers indicate difference was significant at $p < 0.05$.

	(1) Canonical Serial GA		(2) Canon. Serial GA with Niching		(3) Cohort GA	
	max. fitness	sum of diff.	max. fitness	sum of diff.	max. fitness	sum of diff.
avg.	13.90	3896.60	14.30	2404.80	8.90	1982.50
std. dev	3.34	759.90	2.74	486.19	1.71	327.59

	t-test on maximum fitness	t-test on sum of differences
(1,2)	0.597465326	**2.02091E-06**
(1,3)	**1.59564E-05**	**4.27796E-08**
(2,3)	**3.67275E-07**	**0.003991985**

In these HDF tests, the cohort GA always showed earlier convergence, and the phenomenon of individuals accumulating in a small number of cohorts existed, as with the RR function. Thus, below, we applied the new implementation of offspring placement for the HDF.

4.2.2 Varying Offspring Placement Implementation

The new offspring placement implementation (as used for RR) was tried for HDF. Thus, for calculation of the cohort into which the offspring would be placed, we used

$$doub = \lceil (nocoh - 1) + (u - umin) \times (2 - nocoh + 1) / (umax - umin) \rceil$$

instead of

$$doub = \lceil 2 \times umax / u \rceil.$$

Table 11 lists the comparison between the original cohort GA and the cohort GA with the new implementation. The population size was 400. The t-tests on the two paired data sets indicated no significant difference between the two implementations. From observing of the experiments we found that the new implementation of offspring placement in HDF did not alleviate the large amount of accumulation of individuals in a small number of cohorts. Population size 800 yielded similar results. These results indicate that we cannot necessarily generalize the effect of the new implementation on RR to other test functions or practical problems while using the cohort GA.

Table 11: Comparison of maximum fitness reached by two cohort GA's after 2500 function evaluations and the sum of the differences in intron part.

	original cohort GA		cohort GA with new placement	
	max. fitness	sum of the diff.	max. fitness	sum of the diff.
average	9.05	2974.4	8.45	2847.4
std. dev	1.57	286.94	1.36	831.20

In addition to the experiments above, we also tested the relationship between the number of cohorts and the cohort size by conducting paired experiments. The pairs included 10/20 versus 20/10, 10/40 versus 20/20, and 10/80 versus 20/40 (x/y, x representing the number of cohorts and y, the cohort size). Each pair had the same population size. The results of these experiments showed that the ratios did not have a significant impact on the performance of the cohort GA.

5 SUMMARY AND DISCUSSION

The results of the experiments on HDF showed that the canonical serial GA with niching could dramatically reduce the convergence of the intron part. This means that crowding and incest reduction did maintain the population diversity and reduce premature convergence. The results also indicated that the implementation of the cohort GA we used might have some defects in comparison to the implementation used by Holland, particularly with the improvements discussed in [Holland, 2000].

One particular parameter setting (mutation rate) may have

had a negative effect on the HDF runs reported here for the non-cohort GA runs. In contrast with the cohort GA, which did one or more mutations on each individual in the current cohort with probability ½, the non-cohort GA rates were set lower. We used the same mutation rate per bit (0.0002) as we used on RR. It gave a relatively low mutation rate (0.016) per chromosome. Another way of viewing the difference is that in the cohort GA, nearly all new individuals were generated by crossover, and half were also subject to mutation, whereas in these non-cohort GA runs on RR, about ¾ of the new individuals resulted from crossover and ¼ from mutation. However, for HDF, about 90% of new individuals resulted from crossover and only about 10% from mutation. With an increased rate, the non-cohort GA's might give even better performance, and this could be explored further.

6 CONCLUSIONS

The cohort genetic algorithm is designed as a means of reducing premature convergence -- specifically, hitchhiking. In this work, we investigated the performance of one version of the cohort GA on the RR function and the HDF and compared the cohort GA with a canonical serial GA and an island-model distributed GA in order to see how well the cohort GA works in comparison with other techniques for reducing hitchhiking. The experiments showed that even though theoretically the cohort GA should work well in dealing with hitchhiking and be more efficient, the implementation affects its performance very much. This version of the cohort GA didn't perform better in any of the comparison tests due to another form of premature convergence, in which the individuals tended to accumulate in a few cohorts instead of spreading among all the cohorts.

Besides using the original implementation, we also tested different settings and implementations in order to see how various factors affect a cohort GA's performance. The factors included population size, offspring placement strategy, deletion strategy, and inter-cohort crossover. Among these factors, population size, which also includes the relationship between the number of cohorts and cohort size and the offspring placement strategy, had the most significant effect on its performance. In particular, a new implementation of offspring placement in an attempt to spread all the individuals among all the cohorts yielded a great improvement in the cohort GA on RR.

The experiments also showed that crowding and incest reduction performed very well in preventing premature convergence. The degree of intron convergence was greatly reduced after using these niching techniques.

The comparison results indicated that there appear to be (possibly remediable) defects in this version of the cohort GA, and the fact that a small change in placement of offspring among cohorts greatly improved the cohort GA's performance also suggests that further work on the cohort GA may be fruitful.

7 FUTURE WORK

Here are two suggestions for future work:

1) Set an upper limit on the cohort size during the run according to the initial cohort size, For example, if the initial cohort size is 20, the maximum cohort size during the run could be set to 35. In this way, the individuals are forced to spread among the cohorts. The calculation of which cohort an offspring is to be placed in could be done as usual, but if its cohort size has already reach the upper limit, the offspring could be placed in another cohort that has fewer individuals. The new receiving cohort could be calculated deterministically or probabilistically.

2) Use the mean fitness value (*umean*) in the offspring placement strategy. Place the individuals with fitness values between *umin* and *umean* into the first half of the cohorts and place the individuals with fitness value between *umean* and *umax* into the second half of the cohorts. Also follow the principle that the individual with higher fitness value should be put nearer the current cohort.

The detailed implementation issues regarding those changes need to be considered carefully. However, further work seems likely to advance the cohort GA.

References

Forrest, S. and M. Mitchell, 1992. "Relative Building-Block Fitness and the Building-Block Hypothesis" In *Whitley, L. D. (Ed.) Foundations of Genetic Algorithms 2*, pp. 109-126, San Mateo.

Goodman, E., 1996. "An Intro. to GALOPPS, 'Genetic Algorithm Optimized for Portability and Parallelism' System," http://GARAGe.cse.msu.edu/software.

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

Holland, J. H., 2000, "Cohort GAs and Hyperplane-Defined Functions," *Evolutionary Computation*, **8**(4), pp. 372-391.

Holland, J. H., 1998. "Cohort Genetic Algorithms (CGA) and Hyperplane-Defined Functions (HDFs)," Version 1.0, Mathematica 3.0 package, edited and packaged by Theodore C. Belding, [Online] Available http://www-personal.umich.edu/~streak/software/jhh-hdf-1.0.tar.gz.

Jones, T., 1995. "A Description of Holland's Royal Road Function," *Evolutionary Computation*, 2(4), pp. 411-417.

Mitchell, M., S. Forrest, and J. H. Holland, 1992. "The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance," In *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pp. 245-254.