

# Matching Algorithms to Problems: An Experimental Test of the Particle Swarm and Some Genetic Algorithms on the Multimodal Problem Generator

James Kennedy  
*Bureau of Labor Statistics  
2 Massachusetts Ave., NE  
Washington, DC 20212  
kennedy\_jim@bls.gov*

William M. Spears  
*Code 5510 - AI Center  
Naval Research Laboratory  
Washington, DC 20375-5337  
spears@aic.nrl.navy.mil*

## Abstract

A multimodal problem generator was used to test three versions of genetic algorithm and the binary particle swarm algorithm in a factorial time-series experiment. Specific strengths and weaknesses of the various algorithms were identified.

## 1. Introduction

This paper will compare the performance of the binary particle swarm and several varieties of genetic algorithm on sets of problems produced by a multimodality problem generator. The study will be constructed in the form of a repeated-measures factorial experiment, reporting results from multivariate analysis of variance. Research questions involve effects of various aspects of problems on performance of the particle swarm and genetic algorithms with mutation or crossover, or both.

One difficulty with empirical comparisons of search algorithms is that results may not generalize beyond the test problems used. For instance, a new algorithm may be carefully tuned so that it outperforms some existing algorithms on a few problems. Unfortunately, the results of such studies typically have only weak predictive value regarding performance on new problems. We would like to be able to characterize problems in a way that allows us to predict the performance of an algorithm on new, previously unseen problems.

There are several ways to strengthen the results obtained from empirical studies. The first is to remove the opportunity to hand-tune algorithms to a particular problem or sets of problems. Problem generators allow us to report results over a randomly generated set of problems from particular classes of problems rather than a few hand-chosen examples. Thus we increase the predictive power of the results by describing performance on the problem class as a whole.

An advantage of problem generators is that they can be parameterized, allowing researchers to design controlled experiments in which one or more properties

of a class of problems can be varied systematically to study the effects on particular search algorithms. This provides a second method for improving empirical results, by allowing us to conduct well-designed experiments which look at main effects as well as interactions of independent variables. While it is understood that all possible combinations of all conceivable variables cannot be manipulated in a single study, a small number of representative levels of variables which are considered important can be studied systematically; when the effects of these variables are understood, then other variables can be introduced, until all relevant effects have been charted.

One of the most important variables to consider is time. Many empirical studies report the performance of algorithms at the end of a run, say at 20,000 evaluations or after 10 minutes. However, as we will show, conclusions can often turn out to be surprisingly dependent on the time cutoff, often reversing if a smaller (or larger) cutoff is used. Investigation of the processes over time is important for a scientific understanding of the behaviors of these algorithms, as well as giving important information to engineers who may wish to develop applications using them. Thus a third method for improving empirical results is to sample the performance of algorithms many times during a run.

## 2. Particle Swarm

The particle swarm algorithm is an adaptive algorithm based on a social-psychological metaphor; a population of individuals adapt by returning stochastically toward previously successful regions in the search space, and are influenced by the successes of their topological neighbors. In particle swarm, individuals (particles) are represented as vectors  $\vec{x}_i(t)$  (i.e., the vector for particle  $i$  at time  $t$ ). Particle swarm adaptation, originally presented as a method for searching continuous spaces [3, 4], has recently been adapted to binary spaces through a simple modification [5]. Instead of moving particles as  $\vec{x}_i(t) = \vec{x}_i(t-1) + \Delta\vec{x}_i(t)$ , particles exist as

vectors of probabilities, defined as a logistic function  $s(\Delta \bar{x}_i(t))$ . For each vector  $\bar{x}_i(t)$ , a random vector  $\bar{p}_i(t)$  is chosen from a uniform distribution in  $[0.0, 1.0]$ ; if  $p_{id}(t) < s(\Delta x_{id}(t))$  then  $x_{id}(t) = 1$ , else  $x_{id}(t) = 0$ . Preliminary tests have found this algorithm to perform satisfactorily on standard test functions [5]; the present experiment was designed in part as a rigorous investigation of the properties of the discrete particle swarm algorithm.

### 3. The Multimodal Problem Generator

The multimodality of (i.e., number of peaks in) a search space is an important characteristic of that search space. This section outlines a simple problem generator that produces random problems with a controllable degree of multimodality. The motivation for this generator stemmed from an interest in the differences between mutation and crossover in genetic algorithms, and its implementation below will allow insights into characteristics of the performance of the particle swarm as well. Consider a simple two peak problem, with optima at 000...000 and 111...111. Individuals (strings) with roughly 50% 1's and 0's are the lowest fitness strings, while individuals with mostly 1's or mostly 0's have high fitness. Mutation of any high fitness individual on either peak will tend to keep the individual on that peak, driving it up or down the peak to a small degree. Crossover, however, produces quite different results, depending on the location of the parents. If the two parents are on the same peak, the offspring are also highly likely to be on that peak. However, if the two parents are on the two different peaks, the offspring are highly likely to be in the valley between the two peaks, where the fitness is low.

One can hypothesize, then, that crossover may hurt GA performance on the two peak problem. What if there are more than two peaks? It appears reasonable to hypothesize that crossover could be even more deleterious, since the crossover of individuals on different peaks is even more likely to produce poor offspring, until the population has converged to one peak. To explore these hypotheses a multimodality problem generator was created, in which the number of peaks (the degree of multimodality) can be controlled easily and methodically by the experimenter. The description of the generator is as follows.

The idea is to generate a set of  $P$  random  $L$ -bit strings, which represent the location of the  $P$  peaks in the space. To evaluate an arbitrary bit string, first locate the nearest peak (in Hamming space). Then the fitness of the bit string  $c$  is the number of bits the string has in common

with that nearest peak, divided by  $L$ . The optimum fitness for an individual is 1.0.

$$f(c) = \frac{1}{L} \max_{i=1}^P \{L - \text{Hamming}(c, \text{Peak}_i)\}$$

This particular problem generator is a generalization of the  $P$ -peak problems introduced in [1]. De Jong et al. [2] compared the performance of various GAs on problems with 1 peak and problems with 500 peaks. What was most noticeable in that study was the severe drop in performance of GA's including crossover for the 500-peak problems, while the performance curves for the GA with mutation only (i.e., no crossover) are almost identical for the two classes of problems. This provides strong confirmation of the increasing initial advantage of mutation as multimodality increases.

### 4. Design of the experiment

The experiment reported here used the multimodality generator to investigate the effects of number of peaks ( $P$ ) and length of bit vector ( $L$ ) on the performance of four algorithms, measured as a vector of best-so-far performance values sampled over time:

- A GA using crossover and selection only (GA\_c)
- A GA using mutation and selection only (GA\_m)
- A "traditional" GA with both crossover and mutation, plus selection (GA), and
- The binary particle swarm (PS).

Thus the study is conceived as a  $4 \times 2 \times 2$  factorial experiment, with three between-groups factors:

- Type of algorithm
- Number of peaks (20 vs. 100 peaks), and
- Longer and shorter bit vectors (20 vs. 100 bits)

and one within-trial factor, which is called simply "time."

It was hypothesized that characteristics of the algorithms would interact with problem dimensions, and that the conditions under which an algorithm might excel or fail could eventually be identified. With this knowledge, researchers can choose an algorithm which is appropriate for their particular case.

Dependent variables in this experiment were the best-so-far performance evaluations of each trial at the 20th (first generation) evaluation, the 1,000th, 2,000th, and so on to the 20,000 evaluation -- a total of 21 measurements per trial. Multivariate analysis of variance (MANOVA) comprised tests of the effects of the three independent variables, including main effects and interactions, on the vector of values, that is, the series of changes in performance over time.

## 5. Method

Programs written in C were compiled on a Sun Solaris machine running Unix. For each setting of  $L$  and  $P$  twenty random problems were generated. Each algorithm was run once on each of those problems, with each trial extending for 20,000 evaluations. Random seeds specified for each trial determined that all algorithms operated on the same problems. An “evaluation” comprised actual evaluation of the objective function; this was not done when a bit vector had not changed from the previous evaluation due to algorithmic operations. The mutation rate for GA and GA\_m was 0.001, the crossover rate was 0.60 for GA and GA\_c; for the particle swarm,  $V_{\max}$  was 2.0 and  $\phi$  (the “acceleration constant”) was set at 2.0. Two levels of  $L$  and  $P$  were administered: levels for both variables were 20, in the low condition, and 100 in the high. Populations for all algorithms comprised 100 individuals.

## 6. Results

All multivariate effects reported below were significant with  $\alpha=0.0001$ . There is reason to be concerned that heteroscedasticity of data, especially small or zero variance in cells that rapidly and unanimously converged on the global optimum, may have resulted in underestimation of error variance and subsequent inflation of  $F$  ratios. Thus, this report will not dwell on  $p$ -values, but only suggest that  $F$  is a good indication of the relative amount of variance explained by each factor. In the interest of saving space, descriptions of analyses will refer to the graphs that follow, rather than reporting the multitude of means.

### 6.1. Time main effect

All algorithms improved over time, with most trials ascending from a mediocre random start to the global optimum or near it, resulting in a very large statistical effect of time,  $F(20, 285)=2143.40$ . The following sections report how the experimental factors interacted with time.

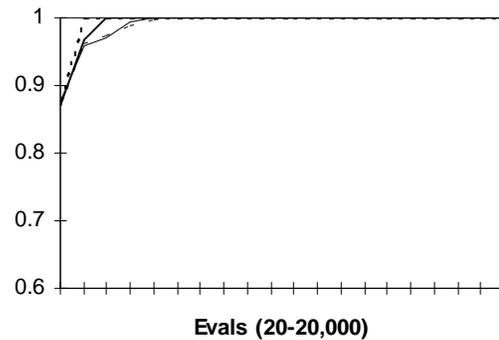
### 6.2. Algorithm $\times$ Time

The interaction of algorithm with time was moderately strong,  $F(60, 851.11)=16.67$ . Across the four

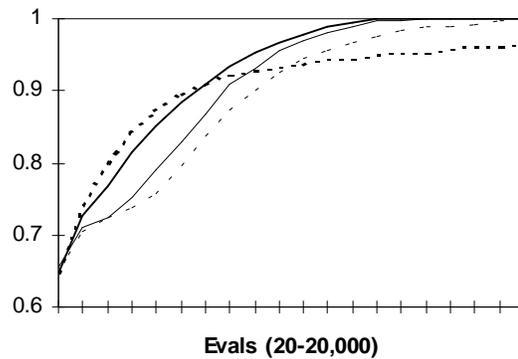
**Figures 1-4. Mean best-so-far performance of the four algorithms compared within each of the four conditions: high/low  $L \times$  high/low  $P$ .**

conditions, it is seen that GA\_m generally tended to perform well in early evaluations and fade later in the time series, while GA and GA\_c tended to start with a rush followed by slowed improvement, then picked up until they outperformed GA\_m. PS on most trials started somewhat slower than GA\_m but faster than the other

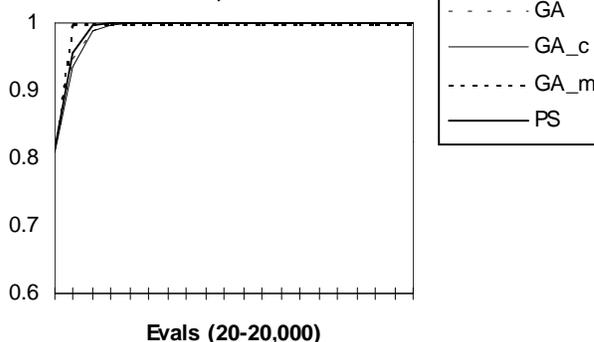
**P=100, L=20**



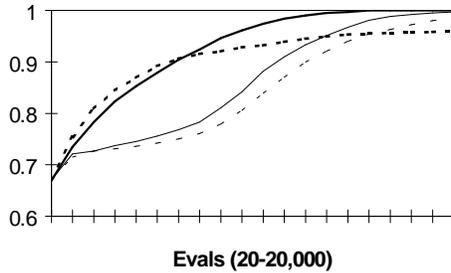
**P=20, L=100**



**P=20, L=20**



**P=100, L=100**



two GA's, and was in all cells the first algorithm to overtake GA\_m; in three of the cells it was the first, and sometimes the only, algorithm to attain a mean performance of 1.0. In the cell where it did not attain the global optimum first, it was second, following GA\_m in the  $L=20, P=20$  condition.

### 6.3. P (Number of peaks) $\times$ Time

Varying the number of peaks affected the dynamics of the algorithms rather strongly,  $F(20, 285)=22.70$ . The effect is seen in the pronounced dip in the curves of GA and GA\_c when  $P=100$ , much less when  $P=20$ . The two GA's with crossover showed fast improvement in the very earliest evaluations, then leveled out, with gradual but persistent improvement for the last half of the trials. Thus, the effect of  $P$  is seen in overall diminished mean performance, though much of this effect will be explained below by interactions.

### 6.4. L (Bit string length) $\times$ Time

The variable which interacted the most, by far, with time, was  $L$ , the length of the bit string being optimized,  $F(20, 285) = 985.78$ . In the graphs, this can be seen as an overall depression of performance in the two  $L=100$  cells, compared to the ease with which all algorithms found the global optimum when  $L=20$ . In fact, when  $L=20$ , all trials settled on the global optimum before 5,000 evaluations; when  $L=100$ , a great many trials failed to find the optimal bit string after 20,000 evaluations. Thus this variable made the difference between an easy and a hard situation.

### 6.5. Interaction of Algorithm $\times$ P $\times$ Time

Number of peaks interacted weakly with algorithm,  $F(60, 851.12) = 3.34$ , suggesting that the effect of  $P$  was not importantly different between the algorithms. In comparing the graphs, it can be seen that the two graphs with  $P=20$  differ from the two with  $P=100$  mainly in an increase in difficulty for GA and GA\_c with the higher number of peaks, contrasted with almost identical performance by GA\_m and PS under both levels of  $P$ .

### 6.6. Interaction of Algorithm $\times$ L $\times$ Time

The interaction of algorithm with  $L$  over time was moderate,  $F(60, 851.12) = 15.04$ . When  $L=20$ , GA\_m improved rapidly from the start of the trial directly to the global optimum, followed by PS, with GA and GA\_c lagging behind, and all algorithms converging quickly on the optimum. On the other hand, when  $L=100$ , the initial rush by GA\_m faltered short of the optimum, to be overtaken by GA and GA\_c; these algorithms with crossover started slowly but outperformed GA\_m in the long run. PS found the global optimum on all trials, somewhat more slowly than GA\_m but faster than GA and GA\_c when  $L=20$ , and faster than all other algorithms when  $L=100$ . Hence, GA\_m was most affected, and PS least affected, by changes in the length of the bit string.

### 6.7. Interaction of L $\times$ P $\times$ Time

Number of peaks interacted moderately with length of the bit string,  $F(20, 285)=14.69$ . Looking at the graphs, we see that the effect of  $L$  was greater when  $P$  was high. For each level of  $L$ , performance was lower when  $P$  was high than when  $P$  was low, but the effect was greater, and occurred later in the sequence, when  $L$  was high.

### 6.8. Interaction of Algorithm $\times$ L $\times$ P $\times$ Time

The interaction of the three independent variables with time was slight,  $F(60, 851.12) = 3.68$ . The "dip" in performance by GA and GA\_c was most pronounced when both  $L$  and  $P$  were high, while GA\_m and PS performed approximately the same in both conditions where  $L$  was high, and the same in the conditions where  $L$  was low. Most of this variance however is explained by simpler interactions.

### 6.9. Univariate effects

The patterns of univariate effects over time, shown in the Appendix, can help explain the pattern of multivariate effects reported above. The main effect of algorithm, that is, the difference in performance between the various algorithms, is nonsignificant when the trials begin, as it should be with random initialization, but then increases steadily until about the 4,000th evaluation. This spreading of algorithms in the statistics is seen in the graphs as the "dip" in the crossover algorithms, versus the steep improvement of GA\_m and PS. Differences decrease for approximately the next 10,000 evaluations,

until the point that GA and GA\_c catch up with GA\_m, then increase again as crossover overtakes mutation.

The main effect of  $P$ , number of peaks, starts out high, apparently because with more peaks it is more likely that a random start will be near one: high- $P$  conditions started with somewhat better performance evaluations. The effect then rapidly decreases to nonsignificance, with a nadir at 2,000 evaluations, then slowly increases until about 8,000 evaluations, where the effect of  $P$  on the crossover algorithms begins to diminish simultaneously with the tapering off of improvement by GA\_m and the effect decreases, once again, nearly to nonsignificance.

The effect of  $L$  also begins high, as random functions are affected by the higher dimensionality of the high- $L$  condition. The central limit theorem suggests that evaluations of higher dimension will have smaller variance, which means that the best value found after 20 evaluations will be less extreme, i.e., nearer to 0.5. The effect of  $L$  increases rapidly until about 2,000 evaluations, then begins to decrease again; this pattern results from the fact that all algorithms had reached the global optimum in all trials of the  $L=20$  conditions by the 5,000th evaluation, while the  $L=100$  trials continued to improve, closing the gap. Note also that after the 5,000th evaluation, interactions of other factors with  $L$  are equal to the main effect of that factor, as the low level of  $L$  contributes no variance to the total.

The interaction of algorithm  $\times L$  increases from nonsignificance to a rather high  $F$  by the 4,000th evaluation, as the GA's with crossover (GA and GA\_c) flatten out, diverging from the rapid-starting algorithms, before finally beginning to ascend. The effect of algorithm  $\times P$  increases to about 6-12,000 evaluations, reflecting the relatively flatter early improvement of GA and GA\_c when  $P$  is higher. The increase of the  $P \times L$  interaction at about 7,000 evaluations again reflects the relatively greater difference between the crossover algorithms and the others when both  $P$  and  $L$  were high.

Finally, the three-way interaction simply follows the trend of the algorithm  $\times P$  interaction after about 5,000 evaluations; as was mentioned earlier, the low- $L$  conditions contribute nothing to the variance after this point, as all trials have converged unanimously.

## 7. Conclusions

Our method of using a multivariate analysis of variance to analyze the performance of different algorithms on random problems created by a parameterized problem generator has yielded some interesting insights. The multivariate effects represent differences in the dynamic processes of several algorithms over time and thus per-

mit no small number of simple descriptive statistics, such as means, to report. In general, the largest effect by far, next to the main effect of time itself, was the effect of increasing the length of the bit string. This should not be surprising, as the size of the combinatorial problem expands exponentially with  $L$ . What might be surprising is the fact that increasing the problem from  $2^{20}$  to  $2^{100}$  did not prevent these robust algorithms from succeeding, but only slowed them down slightly. The exception there of course is GA\_m, the genetic algorithm with mutation only, which never succeeded in finding the global optimum in 20,000 evaluations when both  $L$  and  $P$  were high, and found it only once when  $L$  was high and  $P$  low. The poor performance of GA\_m when  $L$  is high apparently arises from our use of a constant mutation rate of 0.001. As  $L$  increases the proportion of individuals that are mutated also increases, creating too much disruption.

De Jong et al. [2] had noted that crossover might not perform especially well on functions featuring high modality. In their study it was seen that GA and GA\_c started out slowly, and performed relatively poorly, for a great number of evaluations, until finally a "critical mass" was attained as a majority of chromosomes converged on a single peak; after that happened, performance improved rapidly, usually until a global optimum was found. This was contrasted with the performance of GA\_m, which began with a rush but quickly ceased improving. These effects were clearly evident in the current data; in fact, we can elaborate somewhat beyond the previous findings. GA and GA\_c, the two algorithms with crossover, started out, in all conditions, almost as fast as GA\_m, then flattened out. The flattening of performance is least evident in the  $L=20, P=20$  cell, a condition which all algorithms found easy, but even here the two algorithms with crossover are seen to slow after a strong start.

The flattening of progress is seen in the current data, to some degree, in all the crossover algorithms in all conditions. High dimensionality and especially high multimodality exaggerate the tendency for crossover to flounder before progressing. The opposite effect was seen with GA\_m; the mutation algorithm always started faster than any of the others, but leveled out usually at about the same point the crossover algorithms began to improve.

These results justify two modifications to mutation that are sometimes performed. The first is to use a mutation rate proportional to  $1/L$ , in an attempt to maintain a constant level of disruption as  $L$  increases. The second is to reduce the mutation rate as the process continues, allowing mutation-only GAs to continue making progress in the latter stages of search. It should also be noted that we are using low mutation rates due to the lack of elitism

in our versions of GA. Higher mutation rates are useful when elitism is included, because the best individuals will not be lost due to disruption. Interestingly, our results also suggest that a novel strategy for multimodal functions might be to shift the emphasis from mutation to crossover during the course of a run. Finally, it is interesting to note that the traditional GA, with both crossover and mutation, is the worst performer and the algorithm most adversely affected by increasing multimodality. This raises serious concerns about the automatic use of both mutation and crossover in a GA, suggesting that it is often good to remove one of those operators.

In these trials the particle swarm algorithm reached the global optimum faster than any other except in the  $P=20, L=20$  condition. PS was hardly affected at all by increasing the modality of problems, and while it did not start progressing as rapidly as GA\_m in the  $L=20$  conditions, it quickly caught up and surpassed the mutation algorithm's performance long before the GA's with crossover did. PS was also the least affected by changes in problem dimensionality. The particle swarm found the global optimum on every trial, in every condition, and was the only algorithm for which that statement is true -- the most PS ever required was 14,620 evaluations to attain the global optimum. In sum, the particle swarm appears to be robust, given the variations presented here.

## References

- [1] De Jong, K. A. and Spears, W. M. (1990). An analysis of the interacting roles of population size and crossover in genetic algorithms. In H.-P. Schwefel and R. Männer, (Eds.), *Proceedings of the First International Conference on Parallel Problem Solving from Nature*, 38-47. Springer-Verlag.
- [2] De Jong, K., Potter, M., and Spears, W. (1997). Using problem generators to explore the effects of epistasis. In T. Bäck, (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, 338-345. Morgan Kaufmann.
- [3] Kennedy, J. (1997). The particle swarm: Social adaptation of knowledge. *Proceedings of the 1997 International Conference on Evolutionary Computation* (Indianapolis, Indiana), IEEE Service Center, Piscataway, NJ, 303-308.
- [4] Kennedy, J., and Eberhart, R. C. (1995). Particle swarm optimization. *Proceedings of the IEEE International Conference on Neural Networks* (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV: 1942-1948.
- [5] Kennedy and Eberhart, R. C. (1997, in press). A discrete binary version of the particle swarm algorithm. *Proceedings of the 1997 International Conference on Systems, Man, and Cybernetics*.

## Appendix. Univariate $F$ statistics over time (degrees of freedom =3 for Alg and its interactions, 1 otherwise).

Evals	Alg	P	L	Alg×L	Alg×P	P×L	Alg×P×L
20	0.18	165.07	3536.16	0.46	0.18	33.25	0.58
1,000	67.24	25.20	10112.36	5.00	1.04	0.22	1.40
2,000	165.02	0.18	14679.93	54.87	3.83	13.51	0.17
3,000	261.10	3.53	14515.03	210.61	3.52	0.06	1.43
4,000	282.39	23.30	10593.87	274.78	7.87	20.16	7.58
5,000	217.89	60.90	6090.26	217.89	16.77	60.90	16.77
6,000	181.59	121.67	4239.01	181.59	32.99	121.67	32.99
7,000	145.04	190.55	2905.38	145.04	49.22	190.55	49.22
8,000	101.20	173.05	1808.74	101.20	45.05	173.05	45.05
9,000	72.49	155.65	1189.71	72.49	41.57	155.65	41.57
10,000	47.11	103.29	739.47	47.11	29.18	103.29	29.18
11,000	41.81	80.21	529.33	41.81	21.10	80.21	21.10
12,000	45.00	72.92	484.22	45.00	49.73	72.92	19.73
13,000	50.44	64.51	410.93	50.44	18.64	64.51	18.64
14,000	53.14	43.03	340.84	53.14	14.30	43.03	14.30
15,000	77.61	30.55	334.90	77.61	13.64	30.55	13.64
16,000	105.81	27.97	377.71	105.81	10.80	27.97	10.80
17,000	142.12	27.94	400.91	142.12	9.11	27.94	9.11
18,000	188.60	22.25	417.82	188.60	6.88	22.25	6.88
19,000	164.57	10.20	301.36	164.57	3.80	10.20	3.80
20,000	152.23	6.50	253.34	152.23	2.07	6.50	2.07