Evolutionary Optimization of Dynamic Control Problems Accelerated by Progressive Step Reduction

Q. Tuan Pham
School of Chemical Engineering
University of New South Wales
Sydney 2052, Australia
+612-9385 5267
tuan.pham@unsw.edu.au

ABSTRACT
In this paper, we describe the use of an evolutionary algorithm (EA) to solve dynamic control optimization problems in engineering. In this class of problems, a set of control variables must be manipulated over time to optimize the outcome, which is obtained by solving a set of differential equations for the state variables. A new problem-specific technique, progressive step reduction (PSR), is shown to considerably improve the efficiency of the algorithm for this application. Factorial experimentation and rigorous statistical analysis are used to determine the effects of PSR and tune the parameters of the algorithm.

Categories and Subject Descriptors
G.1.6 **Optimization**: Stochastic programming.

General Terms: Algorithms.

Keywords: Evolutionary algorithm, evolutionary strategy, evolutionary optimization, dynamic control, factorial experiment, progressive step reduction.

1. INTRODUCTION
Engineers are becoming more and more interested in evolutionary optimization methods. Their job is to optimize processes and equipment to get the best out of whatever resources are available to them, but conventional deterministic optimization methods often fail in real life due to the complex behaviour of the numerical models used to represent reality. In a great number – perhaps the majority - of practical engineering problems, the system to be optimized is described by a system of differential equations (such as those governing the speed of reactions) whose coefficients depend on a number of control variables (such as temperature, pressure, catalyst concentration, heat or power input, degree of steering, etc.). The engineer has to manipulate these control variables over time to get the best outcome at the end of the process: best product quality, highest yield, lowest cost, etc. Such dynamic control problems are of great practical importance in engineering, and a number of optimization methods have been tested on them: iterative dynamic programming [1], evolutionary algorithm [13], ant colony algorithm [14].

GECCO'05, June 25-29, 2005, Washington, DC, USA.

To solve this class of optimization problem, Pham [13] introduced several specialized operators: swap, creep, shift and smooth. This paper will introduce another technique: progressive step reduction (PSR), where the control variables are first held constant then gradually allowed to vary with time. A two-level factorial experiment will be used to estimate the effectiveness of the various operators.

2. THEORY
2.1 The objective function
Dynamic control problems in engineering have the form

$$\underset{\mathbf{u}(t)}{Maximize}\ y\left(\mathbf{x}\left(t_f\right)\right)$$

subject to

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}\left(t, \mathbf{x}, \mathbf{u}\right)$$

$$\mathbf{x}\left(0\right) = \mathbf{x}_0$$

$$\mathbf{u}_L \le \mathbf{u}\left(t\right) \le \mathbf{u}_H$$

where t is the independent variable (henceforth called "time" although it may also be another coordinate), $\mathbf{x}$ the state vector (size m), $\mathbf{u}$ the control vector (size n), and t_f the final time. The equations are almost always integrated numerically. The objective is to manipulate $\mathbf{u}(t)$ to optimize y. In practice, the time interval $[0, t_f]$ is divided into a finite number p of steps (20 to 40 in this work) and each u_i is specified for each of these steps. Previous researchers kept $\mathbf{u}$ constant within each step but this work allows the values to ramp linearly between the specified values. Thus, each u_i is itself a vector of p elements. The gene is formed by stacking the u-vectors into a vector of length np.

We will consider three such problems:

Problem 1: Control of stirred reactor [1]. Due to mulitiple reactions, $m = 8$, $n = 4$. The previous maximum of the objective function, given by [1], is 339.10 for 20 time steps.

Problem 2: Control of batch reactor [3][15]. Due to multiple reactions, $m = 7$, $n = 4$. The previous maximum of the objective function is 0.610775 [14] for 40 time steps.

Problem 3: Control of plug flow reactor [3][8][14]. This system has $m = 2$, $n = 1$. The previous maximum of the objective function is 0.476946 for 40 time steps [14].

2.2 The Evolutionary Algorithm

The evolutionary algorithm used in this work can best be described as a modified (λ+μ) evolution strategy with some features of evolution programs [1]. It uses real coding of the variables and the following features:

- Parent selection: random (fitness independent).
- Generation gap: none (members are allowed to survive indefinitely).
- Reproduction operators: mutation, crossover, extrapolation, interpolation, creep, swap, shift, smooth (to be explained later).
- Selection for next generation: N offspring are created, where N is the population size, mixed with the parent population, and binary tournament is used repeatedly on the mixed population until the total population is reduced back to N.

In addition to the well-known mutation, crossover, interpolation and extrapolation operators, the following operators were introduced by Pham [13] to accelerate the search for this type of problems: Creep is a particular type of mutation which causes a small change in all of the u-values, used to explore the immediate neighbourhood. Shift causes the value of one control variable u_i at time t to spread to earlier or later times. Smooth consists of performing a rolling average over a random section of u. Swap interchanges the value of u_i at a random time and that at the neighbouring time.

The diversity-conserving mutation technique of Pham [12] is also used here: the specified mutation frequency is augmented by a term $2^{-d/\delta}$ where d is the cartesian distance between the parents and δ is a specified small distance (0.001 of the largest possible distance, in this work). Thus, when two parents are very similar, d tends to 0 and the mutation frequency approaches 1.

2.3 Progressive step reduction (PSR)

A technique that is specifically applicable to dynamic control problems is introduced in this work: the search is started off with a small number (4 in this work) of time steps for the control variables **u**, i.e., **u** is allowed to ramp between four specified values over t_f. Every certain number of generations, the number of steps is doubled, until a prespecified number of steps is reached or exceed. The idea is to allow the broad features, or low-frequency components, of the u-profiles to be identified early on before making the fine adjustments to these profiles, similarly to the way a painter makes a broad sketch of a subject before refining the details (Figure 1). In this work, the subdivisions are performed at a frequency which ensures that each number of time steps is given roughly an equal number of generations.

This works use a variable length chromosome of real-valued "genes" to implement the idea. Every time the number of time steps p is doubled, the length of the chromosome (np) is doubled and the values of the control variables **u** at each new (intermediate) time point is obtained by interpolation, to give an identical profile. Since the doubling is carried out only 4 or 5 times during the whole search, the computational cost is negligible.

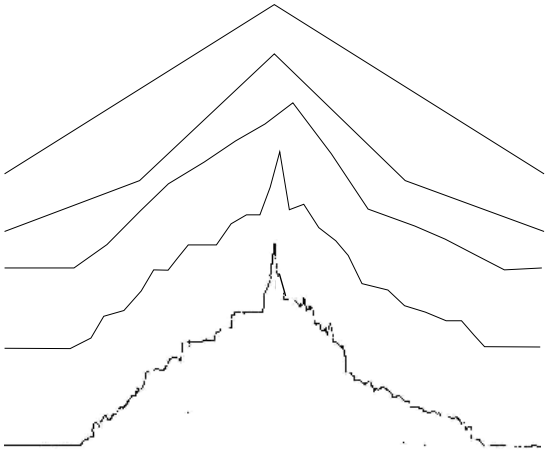

Figure 1. Illustration of Progressive Step Reduction: profile of Mt St Michel progressively refined from 2 to *N* steps.

2.4 Tuning of algorithm

Although evolutionary algorithms are robust, they are notoriously slow and there have been numerous researches on tuning the parameters such as population size and operator frequencies, as reviewed by Michalewicz *et. al.* [10]. A manual or "brute force" approach has sometimes been followed, for example by De Jong [5] and Schaffer *et. al.* [16] who tested various combinations of population size, mutation rates, crossover probability, elitism/non-elitism and other parameters. Grefenstette [6] used a meta-genetic algorithm to optimize genetic algorithms; however, meta-optimization methods are very time consuming. Davis [4] proposed an adaptive method, where the relative merit of various operators (creep, mutation, crossover, etc.) is evaluated, depending on the frequency with which they produce an improvement. Pham [12] proposed a competitive evolution method, where different instances of EA with different parameters compete with each other to produce the fittest offspringz.

Adaptive methods have their merit. However, for a class of problems that is very important and have many recurring instances in real life, it would be desirable to explicitly optimize the search parameters "once and for all". Of course, each problem is different and no one optimization method can be guaranteed to be best for all possible problems, as has been famously shown by Wolpert and Macready [17]. However, hopefully the class of dynamic control problems considered here have sufficient common features that the evolutionary algorithm can be tuned to work reasonably efficiently for most of them.

In this work, a two-level factorial experiment is carried out on each test problem, using the software MINITAB [11]. Each of the following factors is varied between two levels: population size (2 or 20), crossover, extrapolation, interpolation, creep, swap, shift, smooth, PSR (on or off for all the previous). When a reproduction operator is active, it has equal probability to all other active operators. For each combination of factors, the optimization is run 5 or 10 times, each time terminating at 1000 function evaluations. When PSR is not active, the number of time divisions is the same as the final number of time division when PSR is active, i.e., the degree of refinements is the same.

Since the algorithm cannot run without at least one reproductive operator, the mutation operator is always kept active. The reason for choosing this operator is that it ensures that the whole search domain can be covered and thus the global optimum can always be found, given enough iterations. The other operators can thus be considered as performance-enhancing factors. When at least one other operator is active, mutation is given a base probability (i.e., without the diversity-conserving term) one tenth of each of the other operators, in accordance with the low mutation probabilities used in common genetic algorithm practice.

3. RESULTS

3.1 Problem 1

3.1.1 Optimal profile

Figure 2 shows the optimal *u*-profiles for Problem 1. The spikes appear to be "real" and occur in all the results. This profile yields a maximum y of 339.706 which is significantly better than the previous best, but this may be due to the use of ramping rather than constant u-value in each time step.

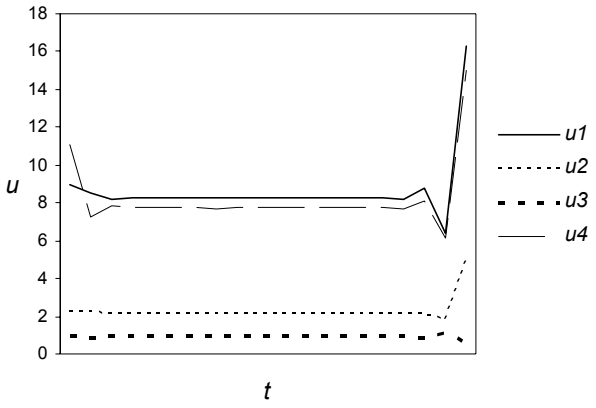

Figure 2. Optimal profile for u in Problem 1.

3.1.2 Effects of all factors

The results of all the runs for Problem 1 were subjected to a factorial analysis to estimate the effect of each parameter and their first-level interactions. The parameters have been coded according to normal factorial design convention: -1 for inactive, +1 for active. For population size, -1 corresponds to 2 members and +1 corresponds to 20 members. The results are shown in Table 1 and Figure 3.

Table 1. Estimated Effects for Problem 1

Term	Effect	T	P
PSR	12.441	81.25	<0.001
Pop. size	-10.783	-70.42	<0.001
Shift	7.567	49.42	<0.001
Smooth	5.267	34.40	<0.001
Pop. size*PSR	4.823	31.49	<0.001
Shift*PSR	-4.588	-29.96	<0.001
Creep	4.537	29.63	<0.001
Pop. size*Smooth	3.724	24.32	<0.001
Shift*Smooth	-3.537	-23.10	<0.001
Smooth*PSR	-3.509	-22.92	<0.001

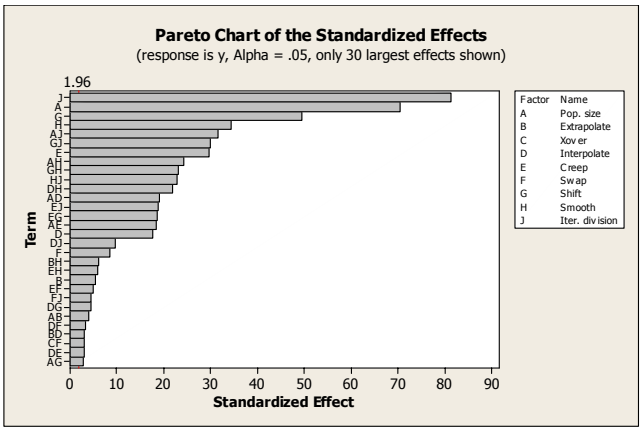

Figure 3. Pareto chart illustrating the effects of the biggest factors for Problem 1 ("Iter.Division" = PSR).

The *Effect* column shows the mean difference in *y* between the low and high level of each variable. A positive effect means that the presence or high level of the factor is beneficial. The *T* column gives the effects divided by twice the pooled standard error, which is more statistically meaningful than the unscaled effect. The *P* column indicates the significance level: a *P*-value of 0.005 indicates that the effect is significant to a level of 99.5%. Only the ten factors and interactions with strongest effects are shown here, in decreasing order of effect.

It can be seen that PSR and population size have the strongest effects. Their effects are illustrated in Figure 4. The left half shows the *y*-values for a population size of 2, the right half for a population size of 20. It is obvious that the behaviours of the two population sizes are quite different, so from this point on they will be treated separately. However, it can be seen at a glance that on the whole the smaller population size yields better results. Within each population size, the points in the left quarter are without PSR, those in the right quarter are when PSR is used. It is clear that PSR leads to a very significant improvement for both population sizes, in terms of avoiding poor performance (low value of *y*).

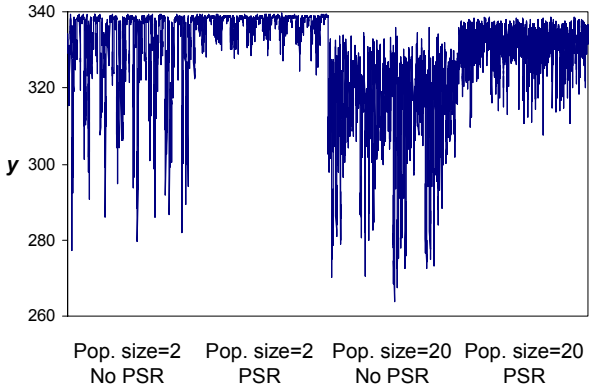

Figure 4. Objective function values for Problem 1.

Figure 5 summarizes the data in the form of a box plot. Each box has three horizontal lines denoting the 25, 50 (median) and 75 percentile values (Q1, Q2, Q3). Means are shown as circles. The vertical lines or "whiskers" indicate the data range, excluding outliers. An outlier is a point outside the range [Q1-1.5(Q3-Q1), Q3

+ 1.5(Q3-Q1)]. Outliers are shown as asterisks. The boxes are, from left to right: population of 2 without PSR, population of 2 with PSR, population of 20 without PSR, population of 20 with PSR.

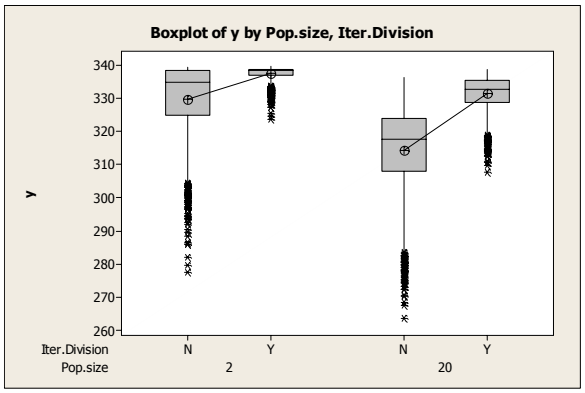

Figure 5. Box plot of objective values for different combinations of population size and PSR ("Iter.Division" = PSR).

The effect of PSR is so clearly beneficial that from this point onwards we discard all the runs without it. We will also consider the runs with population size 2 separately from those with population 20, as they seem to behave quite differently. Thus, we will consider separately the two subsets: P2PSR (Popuplation size 2, with PSR) and P20PSr (Pop. size 20, with PSR).

3.1.3 Effect of Operators, Subset P2PSR

The effects of the ten strongest factors and interaction in subset P2PSR are shown in Table 2. All levels of interaction have been considered. Among the reproductive operators, creep and shift have by far the strongest effects and are beneficial (positive) but their interaction is somewhat detrimental (negative). These are specialized operators introduced by Pham [13]. The effect of creep and shift are positive (beneficial). It may appear surprising that some operators may have a negative effect, but that is because they reduce the operating frequency of the more beneficial operators.

Table 2. Estimated Effects for Problem 1, subset P2PSR.

Term	Effect	T	P
Creep	2.39	28.57	<0.001
Shift	1.527	18.25	<0.001
Creep*Shift	-0.968	-11.57	<0.001
Shift*Smooth	-0.944	-11.28	<0.001
Creep*Shift*Smooth	0.529	6.32	<0.001
Xover	-0.487	-5.82	<0.001
Xover*Creep	0.449	5.36	<0.001
Swap	-0.314	-3.75	<0.001
Creep*Swap	0.239	2.86	0.004
Xover*Shift	0.231	2.76	0.006

There are significant effects arising from the interaction between operators. A full analysis of variance (ANOVA) considering all the possible interactions show that the operators themselves account for 40.9% of the variance, the interactions account for 15.24% and random variations for the rest.

3.1.4 Effect of Operators, Subset P20PSR

We now consider the runs with population size 20 and PSR (Table 3). Again the factors are sorted according to effect. The shift, smooth and interpolation operators now have much stronger effects than the others and they are all beneficial.

Table 3. Estimated Effects for Problem 1, subset P20PSR.

Term	Effect	T	P
Shift	4.433	20.57	<0.001
Smooth	3.382	15.69	<0.001
Interpolate	2.514	11.67	<0.001
Shift*Smooth	-2.354	-10.92	<0.001
Interpolate*Smooth	-2.201	-10.21	<0.001
Extrapolate	-1.044	-4.84	<0.001
Creep	0.937	4.35	<0.001
Swap	-0.933	-4.33	<0.001
Interpolate*Shift*Smooth	0.737	3.42	0.001
Extrapolate*Creep*Smooth	-0.712	-3.3	0.001

3.2 Problem 2

3.2.1 Effects of all factors

The *y*-plot in Figure 6 and box plots in Figure 7 show that, again a population size of 2 with PSR is again the best combination.

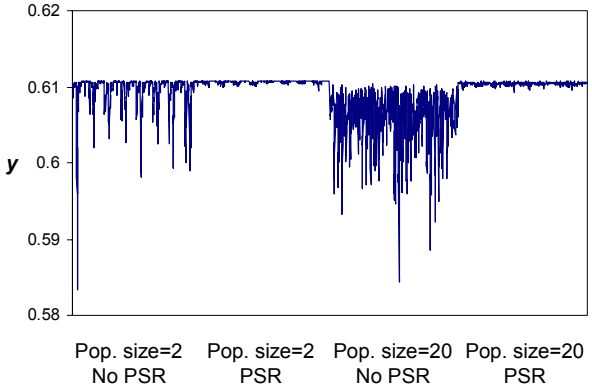

Figure 6. Objective function values for Problem 2.

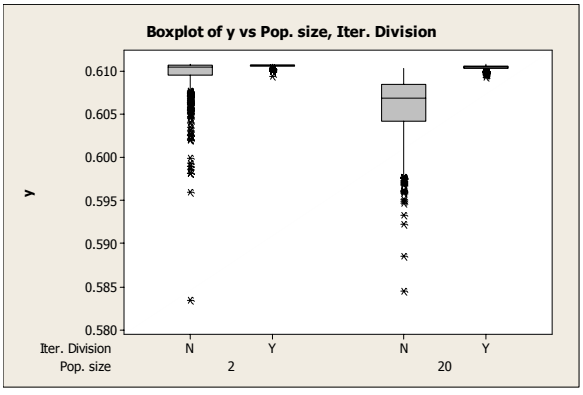

Figure 7. Box plot of objective function values for Problem 2 for different combinations of population size and PSR ("Iter.Division" = PSR).

Tables 4 and 5 show that Creep has the biggest beneficial effects for both subsets P2PSR and P20PSR.

Table 4. Estimated Effects for Problem 2, subset P2PSR.

Term	Effect	T	P
Creep	0.000152	17.36	<0.001
Extrapolate	0.000058	6.63	<0.001
Extrapolate*Creep	-0.000039	-4.48	<0.001
Extrapolate*Smooth	0.000036	4.05	<0.001
Extrapolate*Creep*Smooth	-0.000031	-3.58	<0.001
Interpolate	-0.000029	-3.36	0.001
Swap	-0.000028	-3.15	0.002
Extrapol*Xover*Interpol *Shift	0.000027	3.04	0.002
Interpolate*Creep	0.000025	2.83	0.005
Xover*Creep	0.000024	2.69	0.007

Table 5. Estimated Effects for Problem 2, subset P20PSR.

Term	Effect	T	P
Creep	0.000124	9.05	<0.001
Shift	0.000047	3.46	0.001
Swap	0.000044	-3.2	0.001
Extrapolate*Smooth	0.000042	3.09	0.002
Extrapol*Xover*Interpol*Swap	0.000035	2.55	0.011
Extrapolate*Creep	0.000032	-2.34	0.020
Interpolate*Creep	0.000032	2.31	0.021
Smooth	0.000031	2.26	0.024
Xover*Creep*Swap*Smooth	0.000029	2.12	0.034
Extrapolate*Xover*Interpolate	-.000028	-2.02	0.044

3.3 Problem 3

3.3.1 Effects of all factors

Figures 8 and 9 show that the effects of PSR and population size are similar to the previous problems. Tables 6 and 7 show that Creep has the biggest beneficial effects for both subsets P2PSR and P20PSR for Problem 3.

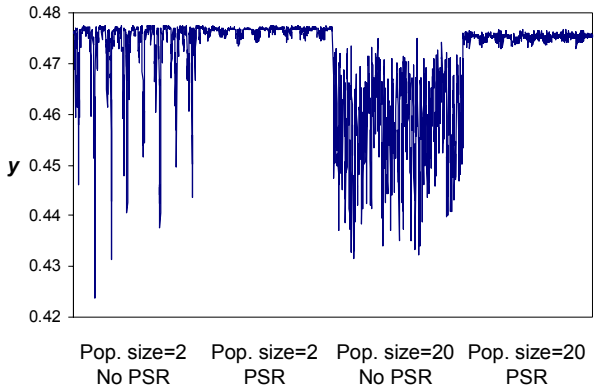

Figure 8. Objective function values for Problem 3.

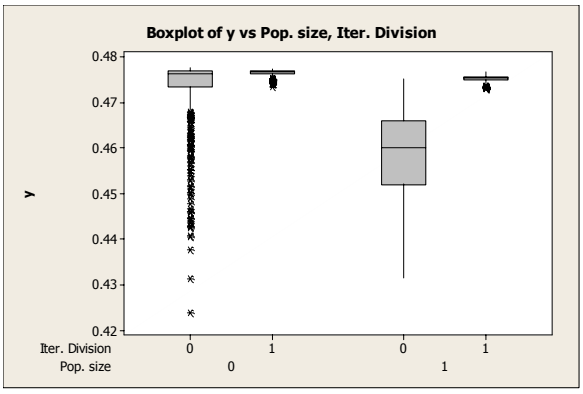

Figure 9. Box plot of objective values for Problem 3 for different combinations of population size and PSR ("Iter.Division" = PSR).

Table 6. Estimated Effects for Problem 3, subset P2PSR.

Term	Effect	T	P
Creep	0.000727	19.45	<0.001
Xover	-0.00022	-5.92	<0.001
Extrapolate	0.000195	5.23	<0.001
Xover*Creep	0.000194	5.2	<0.001
Extrapol*Xover*Creep*Shift	-0.00015	-3.95	<0.001
Extrapolate*Smooth	0.000136	3.64	<0.001
Extrapolate*Creep	-0.00013	-3.59	<0.001
Extrapolate*Xover*Shift	0.000127	3.4	0.001
Extrapolate*Interpolate	-0.00012	-3.28	0.001
Extrapol*Interpolate*Creep	0.000123	3.28	0.001

Table 7. Estimated Effects for Problem 3, subset P20PSR.

Term	Effect	T	P
Creep	0.000581	13.14	0.000
Extrapolate*Smooth	0.000189	4.28	0.000
Extrapol*Creep*Smooth	-0.000190	-4.21	0.000
Extrapolate*Xover	0.000176	3.98	0.000
Interpolate	-0.00018	-3.96	0.000
Interpolate*Creep	0.000161	3.65	0.000
Creep*Swap	-0.00014	-3.07	0.002
Extrapolate*Xover *Creep	-0.00013	-2.92	0.004
Extrapolate	0.000108	2.43	0.015
Swap*Shift	-0.0001	-2.32	0.021

3.4 Summary for all problems

Table 8 shows the *T*-values (effects scaled by standard error) for all the reproduction operators for P2PSR (population size = 2, PSR applied), for all three problems.

Table 8. Estimated *T*-values of individual reproduction operators for all problems, subset P2PSR.

Term	Prob.1	Prob. 2	Prob. 3	Mean
Creep	24.62	16.10	17.73	19.48
Shift	15.73	-1.16	0.91	5.16
Extrapolate	1.31	6.15	4.76	4.07
Smooth	1.39	1.68	1.99	1.69
Interpolate	-0.70	-3.11	-2.23	-2.01
Swap	-3.23	-2.92	-2.16	-2.77
Xover	-5.02	-1.54	-5.40	-3.99

While the number of problems tested here is rather small, some tentative conclusions can be drawn. For this class of problems, it seems that a very small population size (with Pham's [12] diversity conserving mutation rate), combined with PSR of the "time" axis, are most efficient. Among the reproductive operators, the Creep operator is consistently and strongly beneficial. Crossover, interpolation and swap are either insignificant or detrimental. The smooth operator does not significantly contribute to numerical improvement but it is of practical importance, as it causes control function to vary smoothly with time. Observation of the evolving control curves shows that the optimum control curve is almost always smoother than non-optimal ones.

4. TESTING PERFORMANCE OF OPTIMIZED ALGORITHM

Based on the results of the above factorial experiments, the following "optimized" (although not necessarily "optimal") set of parameters were derived: population size = 2, probability of extrapolation = 0.2, of crossover = 0, of interpolation = 0, of mutation = 0.05, of creep = 0.5, of swap = 0, of shift = 0.2, of smooth = 0.05, PSR applied. This "optimized" algorithm was run 20 times, each for 1000 function evaluations, on each problems, and the mean objective value compared to the group means of each of the $2^9 = 512$ factorial combination tested earlier.

For Problem 1, the previous group mean objective values varied from 276.59 to 338.96. The optimized EA gave a mean of 338.84 (Figure 10). For Problem 2, the previous mean objective values varied from 0.59514 to 0.61076 while the optimized EA gives a mean of 0.61071 (Figure 11). For Problem 3, the previous mean objective values varied from 0.44173 to 0.47731 while the optimized EA gives a mean of 0.47697 (Figure 12). In all cases the performance of the "optimized" EA is quite near the top of the range. Taking into account the random variations, this is a very satisfactory result.

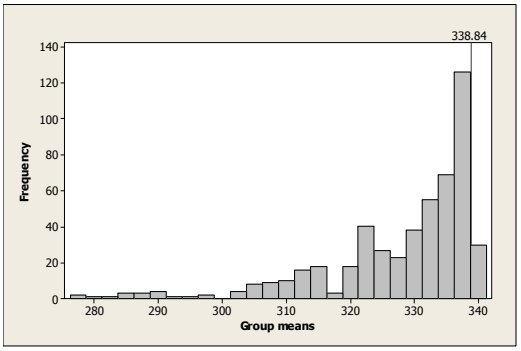

Figure 10. Histogram of mean objective function obtained by various combinations of factors for Problem 1.

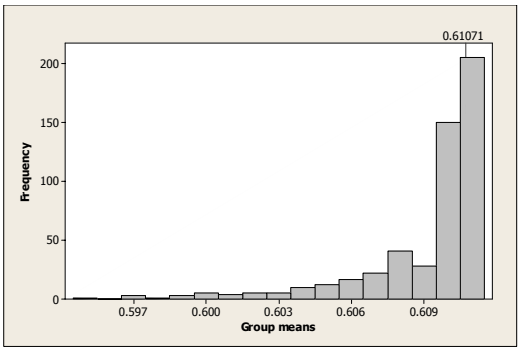

Figure 11. Histogram of mean objective function obtained by various combinations of factors for Problem 2.

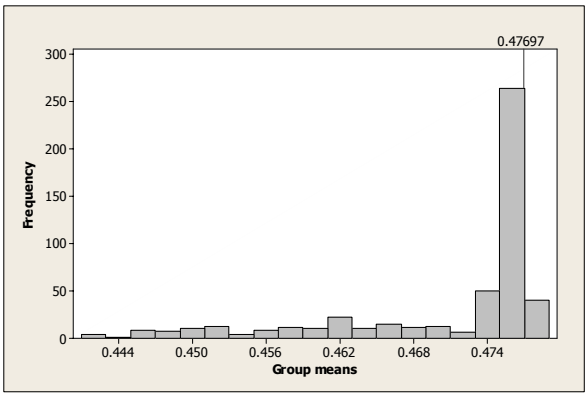

Figure 12. Histogram of mean objective function obtained by various combinations of factors for Problem 3.

5. CONCLUSIONS

The Progressive Step Reduction introduced in this paper has been shown to be beneficial in helping to accelerate the solution of dynamic optimization problems. A reliable and efficient evolutionary optimization algorithm for this class of problems seems to be one using a small population size, progressive step reduction, a large probability for creep and perhaps extrapolation and shift, and smaller probabilities for the others, especially swap and crossover. Although the average improvement in objective function for a given number of function evaluations may seem small, the improvement in reliability or reproducibility (in terms of the range from worst to best solutions for different runs) is quite considerable and hence of real practical usefuless.

A statistically rigorous method has been applied to evaluate the usefulness of various operators. This has rarely been done in previous works in evolutionary computation. It is suggested that statistical analysis be used more in this field, due to the stochastic nature of the results. The full factorial experiment approach used in this work requires a large number of runs. For the test problems in question it is not too time consuming (a few days on a Pentium-4 computer), but for larger problems or a multitude of problems, a more compact experimental design such as Latin Square may be appropriate.

6. REFERENCES

[1] Bäck, T. and Schwefel, H.P. An overview of evolutionary algorithms for parameter optimization. *Evol. Comp.,1* (1993), 1-23.

[2] Bojkov, B. and Luus, R. Use of random admissible values for control in iterative dynamic programming. *IEC Res*., *31* (1992), 1308-1314.

[3] Dadebo, S.A., McAuley, K.B. Dynamic optimization of constrained chemical engineering problems using dynamic programming. *Comput. Chem. Engng, 19* (1995), 513-525.

[4] Davis, L. Adapting operator probabilities in genetic algorithms. *Proc. 3rd Internat. Conf. on Genetic Algorithms*, George Mason University, June 4, pp.61-69. Morgan Kaufman Pub., San Mateo, California (1989).

[5] De Jong, K.A. *An analysis of the behavior of a class of genetic adaptive systems*. Doctoral thesis, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975.

[6] Goldberg, D.E. *Genetic Algorithms in Search,Optimization and Machine Learning*. Addison Wesley, Reading, MA, 1989.

[7] Grefenstette, J.J. Optimization of control parameters for genetic algorithms. *IEEE Trans. on Systems, Man and Cybernetics, SMC 16* (1986), 122-128.

[8] Gunn, D.J., Thomas, W.J. *Chem. Eng. Sci., 20* (1965), 89.

[9] Holland,J.H. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[10] Michalewicz A., Eiben A.E. and Hinterding R. Parameter Selection. In Sarker Ruhul, M Mohammadian, Xin Yao, ed., *Evolutionary Optimization*, Kluwer Ac Pub, Boston, 2002, p.279-306.

[11] MINITAB Release 14 for Windows. Minitab Inc. (2003), USA.

[12] Pham, Q.T. Competitive evolution: a natural approach to operator selection. In: *Progress in Evolutionary Computation*, Lecture Notes in Artificial Intelligence, Vol. 956, p.49-60. X. Yao (ed.), Springer-Verlag, Heidelberg, 1995.

[13] Pham, Q.T. Dynamic Optimization of Chemical Engineering Processes by an Evolutionary Method. *Comp. Chem. Eng., 22* (1998), 1089-1097.

[14] Rajesh, J., Gupta, K., Kusumakar, H.S., Jayaraman, V.K., Kulkarni, B.D. Dynamic optimization of chemical processes using ant colony framework. *Comput. Chem*. *25* (2001), 583-595.

[15] Ray, W.H., *Advanced Process Control*. McGraw-Hill, New York, 1981.

[16] Schaffer, J., David, R.A., Caruana, L.J. and Das, R.. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J.D. Schaffer (ed.), *Proc. 3rd Internat. Conf. on Genetic Algorithms*. Morgan Kaufman, San Mateo, California (1989).

[17] Wolpert, D.H. and Macready, W.G. No free lunch theorem for optimization. IEEE Trans. Evol. Computation, 1(1) (1997), 67-82.