EVOLUTIONARY PROGRAMMING

Guenter Rudolph, chair

Evolutionary Programming Based Stratified Design Space Sampling

Brian K. Beachkofski [*]
Air Force Research Laboratory
1950 5th St., Building 18D
Wright–Patterson Air Force Base
Dayton, OH 45433
(brian.beachkofski@wpafb.af.mil)

Dr. Gary B. Lamont [†]
Air Force Institute of Technology
2950 P St., Bldg. 640 Room 212
Wright–Patterson Air Force Base
Dayton, OH 45433
(gary.lamont@afit.edu)

Abstract

Recently there have been advances in stratified sampling techniques that attempt to enforce equal distributions not only across the design variables, but also onto the design space itself. This requires a numerically intensive optimization routine. Until now, no optimization strategy was able to distribute sample points evenly in the design space, but Evolutionary Algorithms (EA) act as an enabling technology to spread in such a way that the minimum distance between points is near ideal for the design space. The proposed technique is applied to a standard probabilistic analysis problem, a one-degree of freedom oscillator simulating a blade near harmonic resonance. The Evolutionary Programming results compared to Latin Hypercube Sampling indicate a better estimate with a smaller confidence interval.

1 Introduction

A general trend in stratified sampling methods is to use more intelligence in selecting sample locations rather than resorting to brute force methods such as Monte Carlo (MC). This approach is seen in the development of Latin Hypercube Sampling (LHS) [1] and more recently Distributed Hypercube Sampling (DHS)[2]. This paper outlines another attempt to intelligently choose the sample points rather than leaving the process completely random.

[*] Research Engineer, Propulsion Directorate, Turbine Engine Components Branch

[†] Professor of Computer Engineering, Department of Computer and Electrical Engineering

The algorithm will then be applied to a standard probabilistic analysis problem in the turbine engine community. The problem is a Single Degree of Freedom (SDOF) Oscillator representing a turbine engine blade under harmonic resonance.

A review of several sampling methods puts the current work into perspective. The theory behind sampling method evaluation of the integration of the probability density function is also explained, with emphasis on how the proposed method leads to better validity of the equal area assumption.

The justification for employing Evolutionary Programming as opposed to other optimization routines follows the sampling method review. The overall flow of the algorithm is explained, as is each independent operator in the routine. Sufficient information to reproduce the results is presented as well.

The results of the algorithm are sets of sample points for sampling the response space. The quality measure is the confidence interval of the predicted probability of failure and the Coefficient of Variation of minimum distances between any two sample points. Finally, an Analysis of Variance is conducted to show that the differences between methods are likely real and not due to a random variation of points selected from the same distribution.

2 Problem Definition

A standard problem in probabilistic analysis for turbine engines is the single degree of freedom oscillator first defined by Griffiths et al[4], developed as part of their work on probabilistic turbine engine blade design. An elementary damped spring oscillator, Figure (1), under cyclic loading combines the stiffness, K, and the mass, M, into a single parameter, ω_n, the blade natural frequency at nominal rotational loading.

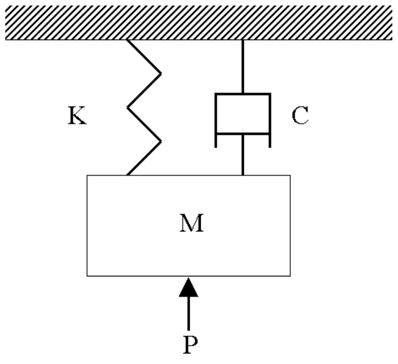

Figure 1: One-Degree of Freedom Oscillator.

The loading, P, is periodic with amplitude, P_0, and frequency of excitation, ω.

$$P(t) = P_0 \sin \omega t \tag{1}$$

The derivation of the limit state function that follows is the notation first used by Griffiths, but could be derived using any basic dynamics reference such as one by Rao [5]. An aerodynamic load generates the forcing function at an M per revolution disturbance typical of blade excitation due to inlet distortion or upstream (downstream) stators. With N as the rotor speed in revolutions per minute, the forcing frequency, in Hertz, can be written as:

$$\omega = \frac{M \cdot N}{60} \tag{2}$$

Finite Element Analysis (FEA) predicts the frequency of the blades under the centrifugal loading as a function of rotational speed. Using the slope of this function at the nominal speed, N_{nom}, a local approximation is made by a first order Taylor's Series expansion.

$$\omega_n = \omega_n(N_{nom}) + Slope(N - n_{nom}) \tag{3}$$

The critical damping ratio, ζ, is the damping value, C, divided by the value needed for critical damping, C_{crit}, and the frequency ratio, β, is the forcing frequency divided by the natural frequency. With this information, the steady state deflection amplitude is of the form:

$$\rho = \frac{\frac{P_0}{K}}{\sqrt{(1-(\frac{\omega}{\omega_n})^2)^2 + (2\cdot\zeta\cdot\frac{\omega}{\omega_n})^2}} = \frac{\frac{P_0}{K}}{\sqrt{(1-\beta^2)^2 + (2\cdot\zeta\cdot\beta)^2}} \tag{4}$$

Dividing by the numerator creates a non-dimensional dynamic amplification factor, D. The design limit of the blade is some percent, Y, of the nominal allowable steady state displacement.

$$\left(\frac{P_0}{K}\right)_{nom} \cdot D_{nom} = Y \cdot \rho_{allow_{nom}} \tag{5}$$

The variations of the forcing field as determined by Computational Fluid Dynamics change the load factor, LF, just as changes in the mode shape are determined by Finite Element Analysis are accounted for in the mode shape factor, MSF. These factors multiply the values of P_0 and K respectively:

$$\frac{P_0}{K} = \left(\frac{P_0}{K}\right)_{nom} \cdot LF \cdot MSF \tag{6}$$

The limit state function, ρ, is the difference between ρ and ρ allowable, or non-dimensionally as g:

$$Y \cdot LF \cdot MSF \cdot \frac{D(N, \omega_n, Slope, \zeta)}{D_{nom}} \cdot \frac{\rho_{allow_{nom}}}{\rho_{allow}} - 1 = g(\omega_n, \zeta, \rho_{allow}, N, Slope, LF, MSF) \tag{7}$$

When ρ exceeds ρ allowable, the function is greater than zero and the blade fails. Equation (7), the limit state function, is dependent on the seven variables in the problem: $\omega_n, \zeta, \rho_{allow}, N, Slope, LF, MSF$.

Table 1: Design Variable Distributions

Variable	Type	μ	σ	Nominal
ω_n	Normal	1975	25	1950
ζ	Log Norm	0.0025	0.0002	0.0025
ρ_{allow}	Normal	15	1.5	15
N	Normal	9200	100	9200
$Slope$	Normal	0.1	0.01	
LF	Normal	1	0.13	
MSF	Normal	1	0.13	
m	None	13		
Y	None	20%		

The distributions of the design variables are shown in Table 1. The "Type" column refers to the distribution type with mean μ and standard deviation σ. The nominal value is shown for those variables where it is needed and is usually the mean value. However the nominal blade natural frequency is not the mean due to the manufactured blades having a frequency biased from the design frequency. When the system variables are described as this, the problem has a probability of failure of 17.06%, as determined by a Monte Carlo analysis with one million realizations.

3 Stratified Sampling

Stratified sampling methods are a way to get a smaller confidence interval with a similar number of samples. If these methods can be optimized an even smaller confidence interval is produced for the same number of computations. The proposed method, Optimized Stratified Sampling (OSS), outlines an algorithm to accomplish this.

Probabilistic analysis predicts the probability of failure, p_f, hopefully including an estimate of the error, through integration of the joint probability distribution, $f(\underline{x})$, across the failure region, Ω:

$$p_f = \int_{\Omega} f(\underline{x}) d\underline{x} \tag{8}$$

Although there are several classes of methods to determine this probability, such as Fast Probability Integration FPI methods [8] or Mean-Based Reliability Methods [10], Fox shows that there is not an established method of estimating the error for these classes [3]. This is a major detriment since an analysis designed to a small margin may be an unsafe design if there is no error estimation. Sampling methods are better since they are an unbiased failure probability estimator, converge to the answer, and provide error estimate as well.

3.1 Stratified Sampling Methods

Monte Carlo techniques require a large number of samples in order to converge to the correct probability. This precludes any analysis that has a long limit state function evaluation time. Stratified sampling methods converge much more quickly [9] by subdividing each design variable into n equal probability bins. Further the sample set is constrained to one sample per bin enforcing the exact distribution over each design variable. Samples appear proportional to the distribution of a converged Monte Carlo analysis. Therefore, these methods are also known as Quasi-Monte Carlo (QMC) methods [13]. QMC methods maintain the benefits of sampling methods (e.g. unbiased estimate) while converging more quickly.

An equivalent integration of Equation (8) is to integrate over the entire space, but multiplying by a step function, H, of the limit state equation, g, that defines failure. If the system fails the step function evaluates to one, otherwise the step function is zero. This transformation is in Equation (9).

$$\int f(\underline{x}) H\left(g(\underline{x})\right) d\underline{x} \tag{9}$$

Sample methods evaluate this integral by weighting individual sample points by their respective area representation, A, as shown Equation (10).

$$\sum_{i=1}^{n} H\left(g(\underline{x}_i)\right) A\left(\underline{x}_i\right) = \frac{n_f}{n} \tag{10}$$

Since the sample point locations are not known a priori, each point is assumed to have equal area representation. The entire probability area is equal to 1 with n sample points, so each weighting is assumed to be $\frac{1}{n}$. Since the weighting is no longer a function of the sample point it can be pulled out of the summation. The sum of the step function also collapses to the number of failure points, n_f, because all failures evaluate to one and non-failures are equal to zero. The overall estimate of the failure probability is therefore Equation (10).

All sampling methods use this same method to calculate the estimate of the failure probability regardless of the method to obtain the sample point locations. There are several methods to obtain those sample points such as Latin Hypercube Sampling, Distributed Hypercube Sampling, and the proposed Optimized Hypercube Sampling.

3.1.1 Latin Hypercube Sampling

Latin Hypercube Sampling (LHS), first outlined by McKay et al [1], uses only the constraint on the single sample point per bin. The failure probability estimate converges quickly with number of samples and is widely used for probabilistic analysis. The theory is similar to Gaussian integration in that the space is completely evaluated by weighted results from single points. The weight for each point is equal since the volume represented by each sample is randomly distributed. When the points are more equally distributed, the variation in hypervolume representation decreases and the equal weighting assumption becomes more valid.

3.1.2 Distributed Hypercube Sampling

Distributed Hypercube Sampling (DHS) [2] adds a second constraint, advancing the idea that the points must be well distributed by constraining the sample set when projected onto a two-dimensional face of the hypercube. The term well distributed was defined as having a low coefficient of variation, COV, of the minimum distance between sample points. The added constraint increases the quality of the response as can be seen in a decrease in the confidence interval size. However the distribution in the volume of the design space is still not addressed, leaving all optimization to the surface projection. Additionally the COV is reduced, but all points may be closely grouped together and the equal area representation assumed by all sampling methods is not valid possibly leading to poor estimates.

3.1.3 Optimal Stratified Sampling

Making the set evenly distributed on the edges and surfaces of the hypercube shrinks the uncertainty as shown by DHS. It follows that if the same constraint is applied to the hypercube volume rather than edges the benefit would increase. Just as LHS ignores hypercube surface distributions, DHS ignores hypercube volume distributions, leaving the gap that Optimized Stratified Sampling (OSS) fills.

Additionally, the proposed method requires a constraint that the minimum distance between points should be equal to an optimum distance based on perfect equal hypervolume representation by each sample point. The optimum is determined by Equation (11), where n is the number of bins and m is the number of dimensions.

$$\begin{aligned} volume &= n^m \\ \frac{volume}{point} &= \frac{n^m}{n} \\ d_{opt} &= \frac{n}{\sqrt[m]{n}} \end{aligned} \tag{11}$$

The jump from volume per point to the distance is a mathematically imprecise leap, taking the m root of the volume per point to find the edge length a hypercube of that volume. However imprecise, the resulting set is well distributed and spans the design space.

One method of measuring the distribution fitness established by [2] is the covariance of the minimum distance between points. The fitness of sample sets generated by the proposed method is also evaluated by this method.

4 Evolutionary Programming Algorithm

The proposed method uses Evolutionary Programming to optimize the sample set since alternative methods fall short in at least one area.

The primary alternative optimization routine is a gradient based scheme. Although the landscape is smooth for small changes, larger shifts are highly nonlinear and the gradient method fails. Additionally, the computation time for the gradients is high and since gradient is only valid for small changes, many gradients must be calculated in the course of optimization. EP uses the smooth behavior to make small changes without the expense of gradient calculations.

Another option is to use complete deterministic search. No deterministic method could search the whole space, which for our sample problem of 100 bins and 7 dimensions has a huge number of combination to search. Specifically the number of possible solutions is n^m choose n, which grows quickly with respect to increases in either bins or dimensions. The number of evaluations for the EP method is dependent only on the population size and generation count. These can be held to acceptable levels to prevent excessive computation costs.

A third option is a partial deterministic search. However this does not use the information from previous evaluations to affect the following samples. Without that intelligence the method is more inefficient and would presumably produce less optimal results in general. EP uses the best previous set to build the next set from, not losing the benefit of previous calculations.

A piecewise partial deterministic search would find consecutive points by a partial deterministic search. This has a large drawback of removing degrees of freedom from the design until there is only a single location left for the last sample point regardless of that points fitness. EP allows movement in all points at each iteration, avoiding that problem.

The decision to use EP over other EA methods is based on the fact that they do not fail where these alternatives do. Although the simplicity can cause EP to become trapped in a local minimum, the search landscape is relatively smooth and monotonically increasing in fitness so that hindrance should not be in issue in this problem.

The basis of the EP method is that the child generation is generated using only mutation, leaving out recombination and repair operators. Additionally, it would seem that recombination would be detrimental to the sample set. The fitness function is dependent equally on all points in the set. Recombination would destroy these sets while mutation only effects a small percentage of the data.

4.1 Data Structure

Each chromosome is a two-dimensional array, $a_{n,m}$, with length n, the number of bins, and width m, the dimensionality of the design space. Each sample point is a row with the integer bin number for that dimension in the corresponding column. In the example shown in Table 2, the first sample point would be found in bin 41 for the first dimension, bin 34 for the second, etc.

Table 2: Data Structure

	1	2	...	m
1	41	34	...	86
2	62	12	...	70
3	89	42	...	31
⋮	05	59	...	97
n	22	83	...	15

4.2 Operators

The algorithm structure is shown in Figure (2) as a flowchart beginning with initialization of the population by random creation. These members are evaluated by the objective function then each member produces several children by means of mutation. Note that no encoding or decoding functions are needed for fitness function evaluation in the algorithm since the genotype and phenotype are integer values of the same form. The children are evaluated with the best surviving to reproduce again. That loop continues until some generation count is reached and a final optimized solution exists. Each of these parts will be introduced and described in the following sections.

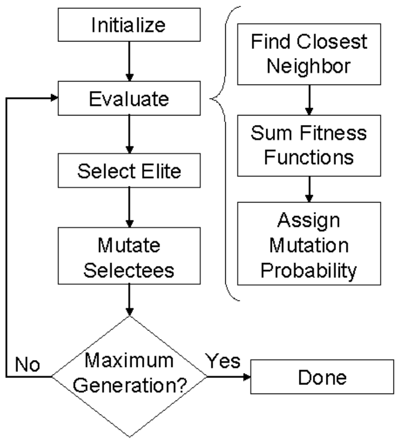

Figure 2: Operator Flowchart

The Evolutionary Programming parameter values used in the analysis can be found in Table 3. The small population size combined with elitism is an effort to best utilize the smooth response by continuing any better children. The fitness values are believed to converge after 500 generations, which is validated in the convergence subsection (5.2). The mutation rate is set to an expected single mutation per dimension. These parameters are applied to the one degree of freedom oscillator example so there are 10 bins and 7 dimensions.

Table 3: Evolutionary Programming Parameter Values

Parameter	Symbol	Value
Population size	μ	2
Children	λ	10
Generations	t_{max}	500
Mutation rate	p_m	1/n
Bin count	n	10
Dimensionality	m	7

4.2.1 Initialization operator

The initialization operator generates sample populations by randomly choosing the elements of each dimension of the array without repetition until that column in the data array is filled. The result is a feasible set of sample points that undergoes fitness evaluation and becomes the parents for all future members. Since the next steps in the algorithm are evaluation and selection, the number of initial population members is actually λ rather than μ. This allows for greater diversity in the initial population, hopefully meaning better initial fitness values.

4.2.2 Evaluation operator

The evaluation operation assesses the sample set using the objective function, Equation (12), which both minimizes the variance of the minimum distance between points. This is accomplished by minimizing the square difference between the optimal distance, Equation (11), and the minimum distance. Therefore a single objective function works to fulfill both objectives since making all the distances optimal also reduces the covariance of the distances as well.

$$\Phi(a(t)) = \sum_{i=1}^{\mu} \left(\min|a(t)_{i,:} - a(t)_{j,:}| - \frac{n}{\sqrt[m]{n}} \right)^2 \qquad (12)$$

$$j = 1, \ldots, \mu \neq i$$

There are three main elements of the evaluation operation. The first aspect is finding the minimum distance to another sample location. The second takes utilizes that distance information and finds the fitness value for each population member. The third step is assigning a mutation probability to each point.

The first step, finding the closest point, is the single most computationally expensive aspect of the algorithm, so efforts are taken to reduce this cost. For example, when the distance from one point to another is calculated, that information is stored for use calculating the reverse distance, roughly reducing the computation time by half.

The second step is evaluating the fitness value, a function of the distances found in the first step. The objective function, Equation (12), is the sum of the square of the difference of the minimum distance and the optimum distance.

The third step is using the minimum distance information to define a probability of mutation to each sample location. The goal is to have a more fit function, so points are assigned a mutation probability based on the amount it adds to the fitness function. The probability comes from quadratic dynamic scaling. Each point is fit to a function such that the least fit function has a unit value and the most fit has zero probability. Further, the slope at the most fit point is zero so all points with values close to the most fit have near zero probabilities as well.

4.2.3 Mutation Operator

The mutation operation, a simple transposition operation defined by Simões in [7], varies the parent population creating the children though asexual reproduction. Each surviving parent produces the same number of children implying λ, number of children, must be a multiple of μ, number of parents. The overall probability of mutation is the inverse of the cell number, making the expected value of the binomial distribution to be one mutation per dimensions. Once the number of discrete swapping events is defined by selection from a binomial distribution, the points at which the swapping occurs is chosen probabilistically based on the quadratic scaling done in the evaluation step and the values are exchanged.

4.2.4 Selection Operator

The selection operation is of $(\mu + \lambda)$ formation, grouping the parent and child generations together for selection. They are ranked based on the fitness function and the μ fittest members are deterministically selected for survival. Other selection methods such as weighted random selection, but forcing the solution through elitism seemed to have few detriments for the faster convergence.

5 Results and Analysis

The heart of the matter is whether the proposed method produces better results than previous methods in an acceptable runtime [1]. Convergence is a very important aspect of the algorithm, for if the EA did not converge or converged to an unacceptable fitness value the results gained from that sample set are of limited value. The benefit is shown by hypothesis testing and analysis of variance.

5.1 Experimental Design

The experiment is designed to show the benefit of the new technique over other alternatives as it would be applied in an actual design problem. The procedure begins with the calculation of the sample points in the Optimized Stratified Sampling. Individual sets of n samples points define a sample design. Each design is evaluated several times to account for the random locations in the selected sample bins. The standard deviation of the probability estimates provides the information for the confidence interval. The experiment will compare the Coefficient of Variation of the minimum neighbor distances and the standard deviations of failure probability estimates for the optimized designs versus Latin Hypercube Sampling.

5.2 Convergence

Convergence, visualized by plotting the fitness versus generation, generally slows as the fitness improves since it takes longer to find a better solution. Figure (3) presents the fitness for the minimum maximum and mean convergence for the ten designs versus the logarithm of the generation. The logarithm is used because it allows changes in the convergence rate to be seen more easily.

Figure (3) shows that towards the end of the analysis the slopes generally decrease showing that the solutions have converged. Convergence is not sufficient for a good solution because an algorithm can converge to a suboptimal solution. For example, if the mutation rate is too high, the better sample sets might not be found because too many mutations occur.

To check if the converged solution has reached a value near that which was hoped, the COV values are compared to the COV from DHS. They range from 0.003 to 0.016 as compared with 0.083 for DHS [2], so the convergence value seems appropriate. The minimum,

[1]The algorithm was run on a Hewlett-Packard Pavilion desktop computer running Windows98 with a 550MHz Pentium II processor with MMX and 64MB RAM.

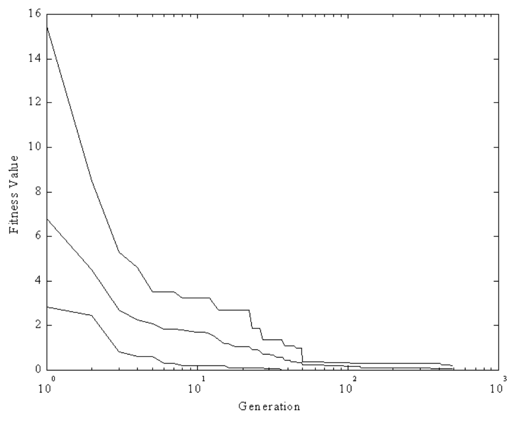

Figure 3: Fitness Convergence

maximum, and mean value plot of the COV is shown in Figure (4).

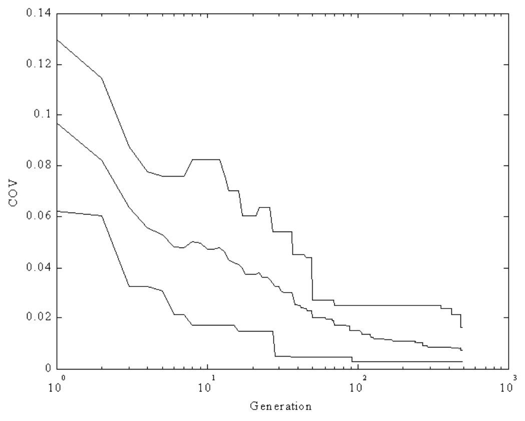

Figure 4: COV Convergence

5.3 Confidence Interval

The example problem used two methods for prediction of failure probability: Latin Hypercube and Optimized Stratified Sampling. DHS was not used because the algorithm was not available to the author, although results are compared to metrics published previously. All methods use 1000 function evaluations, 100 runs (10 designs, 10 times each) of 10 evaluations each. The goal is to determine if the new method reduces the size of the confidence interval.

Equation (13) generates a confidence interval based on repeated estimates of the mean as can be found in any standard basic statistics book [12]. The standard deviation of estimates of the mean is σ and n is the number of estimates.

$$\text{Conf} = 1.96\frac{\sigma}{\sqrt{n}} \tag{13}$$

Table 4 shows the final estimate of the mean and the associated confidence interval for the different methods after 1000 limit state function evaluations.

Table 4: Result Data

Statistic	LHS	OSS
Mean (μ)	17.5%	18.0%
Conf. Int.	2.60%	2.27%

The change from LHS to OSS lead to a 13% decrease in Confidence Interval. To determine if this decrease is meaningful or likely due to random variations an Analysis of Variance must be applied.

5.4 ANOVA Results

The Analysis of Variance (ANOVA) shows that there is a significant difference in the OSS over the LHS method. The ratio of the variances for the two methods is compared to the F tables to see if the difference is significant. Both the method variances have ninety-nine degrees of freedom based on the one-hundred statistics minus one for the mean probability of failure. The F-value, Table 5, corresponds to a confidence level for the null hypothesis that the variances are different of 91.1% confidence.

Table 5: ANOVA Values

Source	σ^2	F-Value
OSS	1.345E-2	1.312
LHS	1.765E-2	

The significant reduction in variance is due to the good distribution of sampling points in the design space through use of EP. The better distribution improves the assumption of equal weighting of sample points. The COV of the OSS method is significantly lower than published values for the DHS, even with the optimization applied to the volume rather than the surface of the design space.

6 Conclusions

The proposal of Optimized Stratified Sampling for choice of sample points follows the current trend in stratified sampling methods of imposing more constraints on the sample set so the points are more evenly distributed. The goal of even distribution is to make the equal weighting assumption of the integration method more valid. Previously there was no method of optimizing the sample set within a design space that is too big to search deterministically and has no gradient information. Evolutionary Programming acts as an enabling technology that allows optimization where it could not be before.

The method implemented on the standard probabilistic analysis problem of a one-degree of freedom model of harmonic resonance converged to a distribution that is much better than alternate methods. This results in a better estimate of the probability of failure and s smaller confidence interval about that estimate.

It seems that the landscape assumption is not invalidated by the data. Elitism helps convergence and does not seem to produce traps at local minima.

Many times the cost of performing function evaluations in a probabilistic analysis is extremely high, almost prohibitive. The method described in this paper makes an effort to get the most information out of the evaluations that are performed. This method uses a small amount of computation before the probabilistic evaluation to achieve an improved confidence interval with acceptable computation times.

The algorithm is written in MATLAB and runs very quickly on moderate to high-speed desktop computers making it only a minimal cost to the developer. The recommendation for future analysis would be to run the algorithm on a fast desktop system. Current processors often outperform many SGI systems for single processor calculations. A moderate speed (1GHz) Pentium III or better processor often outperforms 300MHz SGI R12000 processors for single processor computation.

There is still an opportunity for improvement on the algorithm. The proposed method minimizes the variance of minimum distances in the volume of the cube and enforces uniform sampling in each dimension. It should be possible to enforce uniform distribution on each surface of the hypercube as well. This would attempt to insure that the projections onto each lower dimension are distributed as well as possible instead of just the highest and lower level.

References

[1] M. D. McKay, R. J. Beckman, & W. J. Conover, "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," in *Technometrics*, 21(2), pp. 239-245, 1979.

[2] R. D. Manteufel, "Distributed Hypercube Sampling Algorithm," in *Proc. of 42nd AIAA Structures, Structural Dynamics, and Materials Conference*, 16-19 April, 2001.

[3] E. P. Fox, "On the Accuracy of various Probabilistic Methods," in *Proc. of 41st AIAA Structures, Structural Dynamics, and Materials Conference*, 3-6 April, 2001.

[4] J. Griffiths, C. Annis, Blair, B. Morris, K. Cornet, D. Ghiocel, and Nelson, "Probabilistic HCF Blade Design Progress Report," in *USAF Contract F33615-98-C-2928*, February, 2001.

[5] S. Rao, *Mechanical Vibrations, 2nd Ed.*, Reading, Massachusetts: Addison-Wesley, 1990.

[6] T. Bäck, *Evolutionary Algorithms in Theory and Practice*, New York, NY: Oxford University Press, 1996.

[7] A. Simões, "Using Genetic Algorithms with Sexual or Asexual Transposition: a Comparative Study," in *Proceedings of the Congress on Evolutionary Computation (CEC 2000)*, July 2000.

[8] B. H. Thacker, D. S. Riha, H. R. Millwater, M. P. Enright, "Errors and Uncertainties in Probabilistic Engineering Analysis," in *Proc. of 42nd AIAA Structures, Structural Dynamics, and Materials Conference*, 16-19 April, 2001.

[9] M. Stein, "Large Sample Properties of Simulations Using Latin Hypercube Sampling," in *Technometrics* Vol. 29(2), pp. 143-151, May 1987.

[10] B. M. Kwak & T. W. Lee, "Sensitivity Analysis for Reliability-Based Optimization using an AFOSM Method," in *Computers and Structures* Vol. 27, No. 3, pp.399-406,1987

[11] M. L. Shooman, *Probabilistic Reliability : An Engineering Approach*, Melbourne, FL: Krieger Publishing Company, 1990.

[12] P. G. Hoel, S. C. Port, & C. J. Stone, *Introduction to Statistical Thoery*, Boston, MA: Houghton Mifflin Company.

[13] D. Robinson & C. Atcitty, "Comparison of Quasi- and Pseudo-Monte Carlo Sampling for Reliability and Uncertainty Analysis," *AIAA–99–1589.*

[14] C. Annis, "Bayesian Network Analysis of the 1-D Oscillator Problem," *Independent Contractor Agreement 01-S441-48-01-C4*, http://www.StatisticalEngineering.com, 2001.

Adding Knowledge and Efficient Data Structures to Evolutionary Programming: A Cultural Algorithm for Constrained Optimization

Carlos A. Coello Coello and Ricardo Landa Becerra
CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Ingeniería Eléctrica/Sección de Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco, México, D. F. 07300
ccoello@cs.cinvestav.mx, rlanda@computacion.cs.cinvestav.mx

Abstract

This paper proposes a technique in which domain knowledge obtained from the search is used to improve the performance of evolutionary programming. Our approach is based on the concept of a cultural algorithm and is applied to constrained optimization problems in which a map of the feasible region is used to guide the search more efficiently. Our implementation uses 2^n-trees to store more efficiently this map of the feasible region. Our results indicate that the approach is able to produce very competitive results (we compared our results with respect to the homomorphous maps) at a considerably low computational cost.

1 INTRODUCTION

Evolutionary Computation (EC) techniques have become increasingly popular in the last few years. This popularity is mainly due to the fact that EC techniques have been able to successfully solve a wide variety of complex problems [8].

However, EC techniques are normally used as "blind heuristics" in the sense that no specific domain knowledge is used or required. Nevertheless, several researchers have proposed different mechanisms to extract knowledge (or certain design patterns) from an evolutionary algorithm to improve convergence of another one (e.g., [23, 14]).

In this paper, we propose the use of a biological metaphor called "cultural algorithm" as a global optimization technique. Cultural algorithms are based on the following notion: in advanced societies, the improvement of individuals occurs beyond natural selection; besides the information that an individual possesses within his genetic code (inherited from his ancestors) there is another component called "culture". Culture can be seen as a sort of repository where individuals place the information acquired after years of experience. When a new individual has access to this library of information, it can learn things even when it has not experienced them directly. Humankind as a whole has reached its current degree of progress mainly due to culture.

In this paper, we propose an approach in which domain knowledge (using the concept of cultural algorithm) extracted during a run of an evolutionary algorithm is used to guide the search more efficiently in constrained optimization problems [18, 3].

2 NOTIONS OF CULTURAL ALGORITHMS

Some social researchers have suggested that culture might be symbolically encoded and transmitted within and between populations, as another inheritance mechanism [6, 20]. Using this idea, Reynolds [21] developed a computational model in which cultural evolution is seen as an inheritance process that operates at two levels: the micro-evolutionary and the macro-evolutionary levels.

At the micro-evolutionary level, individuals are described in terms of "behavioral traits" (which could be socially acceptable or unacceptable). These behavioral traits are passed from generation to generation using several socially motivated operators. At the macro-evolutionary level, individuals are able to generate "mappa" [20], or generalized descriptions of their experiences. Individual mappa can be merged and modified to form "group mappa" using a set of generic or problem specific operators. Both levels share a communication link. Reynolds [21] proposed the use of genetic algorithms to model the micro-evolutionary process, and Version Spaces [19] to model the macro-evolutionary process of a cultural algorithm.

The main idea behind this approach is to preserve beliefs that are socially accepted and discard (or prune) unacceptable beliefs. Therefore, if we apply a cultural algorithm for global optimization, then acceptable beliefs can be seen as constraints that direct the population at the micro-evolutionary level [16].

3 PREVIOUS WORK

Reynolds et al. [22] and Chung & Reynolds [2] have explored the use of cultural algorithms for global optimization with very encouraging results. Chung and Reynolds [2] use a hybrid of evolutionary programming and GENOCOP [17] in which they incorporate an interval constraint-network [4] to represent the constraints of the problem at hand. An individual is considered as "acceptable" when it satisfies all the constraints of the problem. When that does not happen, then the belief space, *i.e.*, the intervals associated with the constraints, is adjusted. This approach is really a more sophisticated version of a repair algorithm in which an infeasible solution is made feasible by replacing its genes by a different value between its lower and upper bounds. Since GENOCOP assumes a convex search space, it is relatively easy to design operators that can exploit a search direction towards the boundary between the feasible and infeasible regions.

In more recent work, Jin and Reynolds [11] proposed an n-dimensional regional-based schema, called *belief-cell*, as an explicit mechanism that supports the acquisition, storage and integration of knowledge about non-linear constraints in a cultural algorithm. This *belief-cell* can be used to guide the search of an EC technique (evolutionary programming in this case) by pruning the instances of infeasible individuals and promoting the exploration of promising regions of the search space. The key aspect of this work is precisely how to represent and save the knowledge about the problem constraints in the belief space of the cultural algorithm.

The idea of Jin and Reynolds' approach is to build a map of the search space similar to the "Divide-and-Label" approaches used for robot motion planning [13]. This map is built using information derived from evaluating the constraints of each individual in the population of the EC technique. The map is formed by dividing the search space in sub-areas called *cells*. Each cell can be classified as: feasible (if it lies completely on a feasible region), infeasible (if it lies completely on an infeasible region), semi-feasible (if it occupies part of the feasible and part of the infeasible regions), or unknown (if that region has not been explored yet). This map is used to derive rules about how to guide the search of the EA (avoiding infeasible regions and promoting the exploration of feasible regions).

4 CONSTRAINED OPTIMIZATION

In this paper, we use cultural algorithms with evolutionary programming (CAEP) [2]. The basic idea is to "influence" the mutation operator (the only operator in evolutionary programming) so that current knowledge about the properties of the search space can be properly exploited.

In a cultural algorithm there are two main spaces: the normal population adopted with evolutionary programming and the belief space. The shared acquired knowledge is stored in the belief space during the evolution of the population. The interactions between these two spaces are described as follows [2]:

1. Select an initial population of p candidate solutions, from a uniform distribution within the given domain for each parameter from 1 to n.
2. Assess the performance score of each parent solution by a given objective function f.
3. Initialize the belief space with the given problem domain and candidate solutions.
4. Generate p new offspring solutions by applying a variation operator, V, as modified by the influence function, *Influence*. Now there are $2p$ solutions in the population.
5. Assess the performance score of each offspring solutions by the given objective function f.
6. For each individual, select c competitors at random from the population of size $2p$. Conduct pairwise competitions between the individual and the competitors.
7. Select the p solutions that have the greatest number of wins to be parents for the next generation.
8. Update the belief space by accepting individuals using the acceptance function.
9. Go back to step 4 unless the available execution time is exhausted or an acceptable solution has been discovered.

Most of the steps previously described are the same as in evolutionary programming [7]. The acceptance function accepts those individuals that can contribute with their knowledge to the belief space. The update function creates the new belief space with the beliefs of the accepted individuals. The idea is to add to the current knowledge the new knowledge acquired by the accepted individuals.

The function to generate offspring used in evolutionary programming is modified so that it includes the influence of the belief space in the generation of offspring. Evolutionary programming uses only mutation and the influence function indicates the most promising mutation direction. The remaining steps are the same used in evolutionary programming.

For unconstrained problems, Chung [1] proposes the use of two types of knowledge: (1) situational, which provides the exact point where the best individual of each generation was found; and (2) normative, which stores intervals for the decision variables of the problem that correspond to the regions where good results were found.

5 BELIEFS AS CONSTRAINTS

As we mentioned before, Jin and Reynolds [11] modified Chung's proposal as to include in the belief space information about feasibility of the solutions. We will explain next the changes performed in more detail, since our current proposal is an extension of Jin & Reynolds' algorithm.

First, Jin and Reynolds eliminated the situational knowledge and added constraints knowledge. Taking advantage of the intervals of good solutions that are stored in the normative portion of the belief space, they created what they called "belief cells". These belief cells are a subdivision of the search space within the intervals of good solutions, such that feasibility of the cells can be determined. When the intervals of the variables are modified, the cells are also modified. As indicated before, there are 4 types of cells (see Figure 1)[1]: (1) feasible, (2) infeasible, (3) semi-feasible (contain part of both areas) and (4) unknown.

The influence that the belief space has on the generation of offspring consists of moving individuals that lie on infeasible cells towards feasible cells. Actually, in this process, semi-feasible cells are given preference because in most difficult constrained problems, the optimum lies on the boundary between the feasible and infeasible regions. However, Jin & Reynolds [11] do not modify the rules used to update the normative

[1]Other authors have also proposed the use of a map of the feasible region. See for example [15].

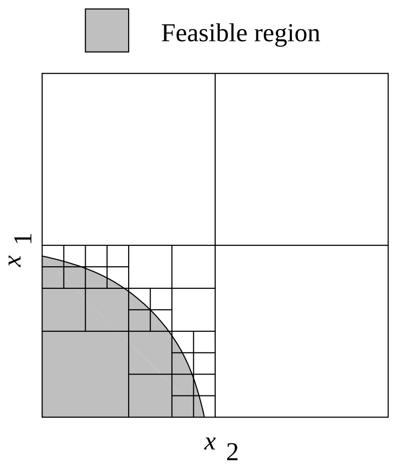

Figure 2: Graphical representation of the division of the semi-feasible cells.

part of the belief space proposed by Chung [1]: the intervals are expanded if the accepted individuals do not fit within them; conversely, they are tightened only if the accepted individuals have a better fitness. This may reduce the intervals towards infeasible regions in which the objective function values are higher.

6 PROPOSED APPROACH

The approach proposed here is a variation of Jin & Reynolds' technique [11]. However, in our case, we incorporate spatial data structures (2^n-trees) in order to store the map of the feasible region more efficiently. Next we will describe the main differences between traditional evolutionary programming and our approach.

6.1 INITIALIZATION OF THE BELIEF SPACE

The lower and upper boundaries of the promising intervals for each variable are stored in the normative part of the belief space, together with the fitness for each extreme of the interval. This part is initialized putting in the boundaries of the variables the values given in the input data of the problem. The initial fitnesses in all cases are set to $+\infty$[2].

Regarding the constraints of the problem, the interval given in the normative part is subdivided into s subintervals such that a portion of the search space is divided in hypercubes (see Figure 2). The following information of each hypercube is stored: number of feasible individuals (within that cell), number of infeasible individuals (within that cell), and the type of

[2]This is assuming a minimization problem.

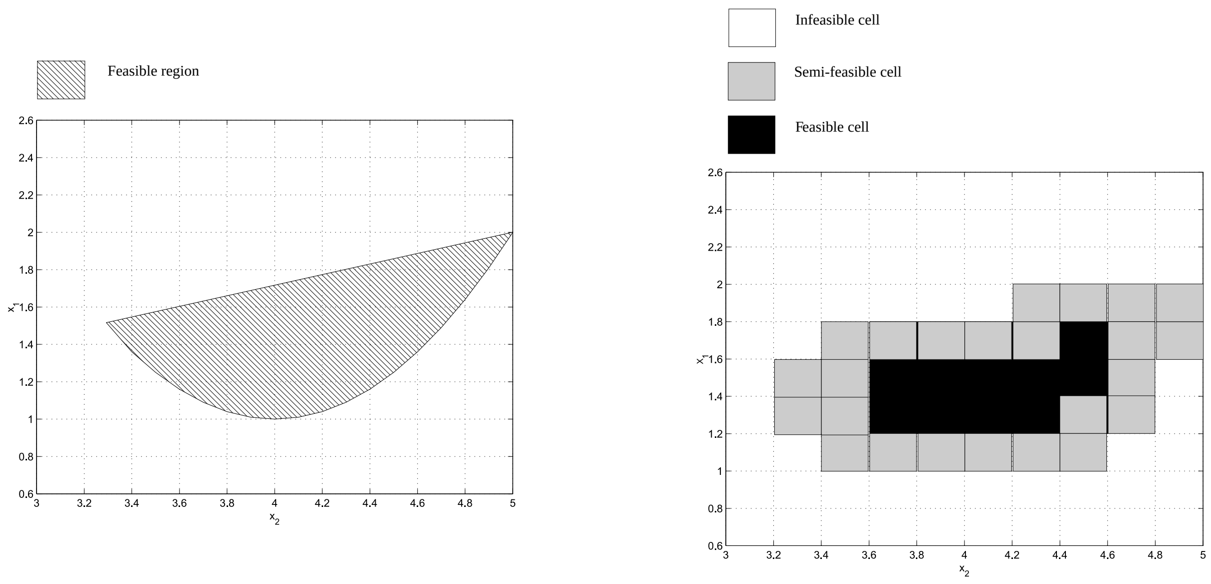

Figure 1: The figure at the left illustrates the feasible region of a problem. The figure at the right illustrates the representation of the constraints part of the belief space for the search space of the same problem. In this example, the intervals stored in the normative part must be [0.6, 2.6] for x_1, and [3, 5] for x_2.

region. The type of region depends on the feasibility of the individuals within. Four types are defined:

- **if** $feasible\ individuals = 0$ **and** $infeasible\ individuals = 0$, **then** $cell\ type = unknown$
- **if** $feasible\ individuals > 0$ **and** $infeasible\ individuals = 0$, **then** $cell\ type = feasible$
- **if** $feasible\ individuals = 0$ **and** $infeasible\ individuals > 0$, **then** $cell\ type = infeasible$
- **if** $feasible\ individuals > 0$ **and** $infeasible\ individuals > 0$, **then** $cell\ type = semi-feasible$

To initialize this part, all counters are set to zero and the cell type is initialized to "unknown" (other values could be used in this case, but that would obviously affect the performance of the algorithm).

6.2 UPDATING THE BELIEF SPACE

The constraints part of the belief space is updated at each generation, whereas the normative part is updated every k generations. The update of the constraints part consists only of adding any new individuals that fall into each region to the counter of feasible individuals. The update of the normative part is more complex (that is the reason why it is not performed at every generation). When the interval of each variable is updated, the cells or hypercubes of the restrictions part are changed and the counters of feasible and infeasible individuals are reinitialized. Furthermore, this update is done taking into consideration only a portion of the population. Such a portion is selected by the function *accept()*, taking as a parameter (given by the user) the percentage of the total population size to be used.

In the approach proposed in this paper, the conditions to reduce the intervals are stronger: an interval is reduced only if the accepted individual has a better fitness AND it is feasible. In order to make this mechanism work, it is necessary to modify the acceptance function so that feasible individuals are preferred and fitness is adopted as a secondary criterion. If this is not done, then the condition for interval reduction will not hold most of the time because the accepted individuals are more likely to be infeasible.

6.3 INFLUENCE OF BELIEFS IN THE MUTATION OPERATOR

Mutation takes place for each variable of each individual, with the influence of the belief space and in accordance with the following rules:

- If the variable j of the parent is outside the interval given by the normative part of the constraints, then we attempt to move within such interval through the use of a random variable.

- If the variable is within a feasible, a semi-feasible or an unknown hypercube, the perturbation is done trying to place it within the same hypercube or very close to it.

- Finally, if the variable is in an infeasible cell, we try to move it first to the closest semi-feasible cell. However, if none is found, we try to move it to the feasible or unknown closest cell. If that does not work either, then we move it to a random position within the interval defined by the normative part.

6.4 TOURNAMENT SELECTION

The rules for updating the belief space may produce that knowledge becomes specialized at a slower rate. To improve the speed of the algorithm, we take advantage of the rules for performing tournament selection. After performing mutation, we will have a population of size $2p$ (p parents generate p children). Tournament is performed considering the entire population (i.e., we use $(\mu + \lambda)$ selection). Tournaments consist of c confrontations per individual, with the c opponents randomly chosen from the entire population. When the tournaments finish, the p individuals with the largest number of victories are selected to form the following generation. The tournament rules adopted for the current proposal are very similar to those adopted by Deb in his penalty approach based on feasibility [5]. However, unlike Deb's approach, in our case, we never add violated constraints (as normally done with penalty-based approaches).

The new tournament rules adopted by our approach are the following:

1. If both individuals are feasible, or both are infeasible, then the individual with the best fitness value wins.

2. Otherwise, the feasible individual always wins.

6.5 USE OF 2^n-TREES

One of the main drawbacks of Jin & Reynolds' approach [11] is its intense memory usage. Since the belief maps of each decision variable has to be stored, the approach runs out of memory very fast and cannot possible handle problems with more than a few decision variables (memory requirements grow exponentially with the number of decision variables of the problem). This led us to develop a scheme in which 2^n-trees are used to partition the feasible region in cells so that with higher-dimensionality problems the memory usage is not exponentially increased. The idea was inspired by the popularity of spatial data structures to store efficiently navigation maps in robotics [13] and to represent efficiently 3D objects in computer graphics [10].

In order to be able to use 2^n-trees within our implementation, we have to partition only the projection of the search space in some dimensions, since 2^n-trees have practical use only when $n \leq 4$, where n corresponds to the number of decision variables of our problem [13]. An example of how to partition a 2D space using a quadtree with a depth of 2 is shown in Figure 3.

Note that the decision of how to partition decision variable space as to comply with this restriction is very important since the number of nodes used may be incremented rather than reduced! For example, if an octree is adopted, using a node division we will divide three dimensions and our tree will have $2^3 + 1 = 9$ nodes in total. However, if we use a tree that divides only one dimension and through 3 successive divisions we partition a 3D space, the leaf nodes will have the same result that the octree but using 15 nodes.

From the previous discussion we can infer that we should use a 2^n-tree with the largest possible n, but being careful of not using too much memory. Our conjecture is that $n = 3$ is the largest number with which the problem remains manageable.

Once the number of dimensions to be partitioned has been decided, we have to decide which are the dimensions to be partitioned. The idea is to choose the 3D projection that best divides the search space into a feasible and an infeasible region (or regions). However, since the number of possible combinations of three dimensions grows exponentially with the number of variables, it soon becomes impossible to try them all. Therefore, we can choose a group of combinations to be tried such that the size of this group grows linearly with the number of variables of the problem. In order to determine the "goodness" of a certain partition, we have to count the number of feasible and infeasible individuals in each leaf node. A node will be considered good as long as one of these two values (i.e., feasible and infeasible individuals) tends to zero. For example, let's assume that n_f is the number of feasible individuals in a certain node and that n_i is the number of infeasible individuals in that same node. Thus, a small number $\min(n_f, n_i)$ in each node will indicate a good partition. From the previous discussion, we can say

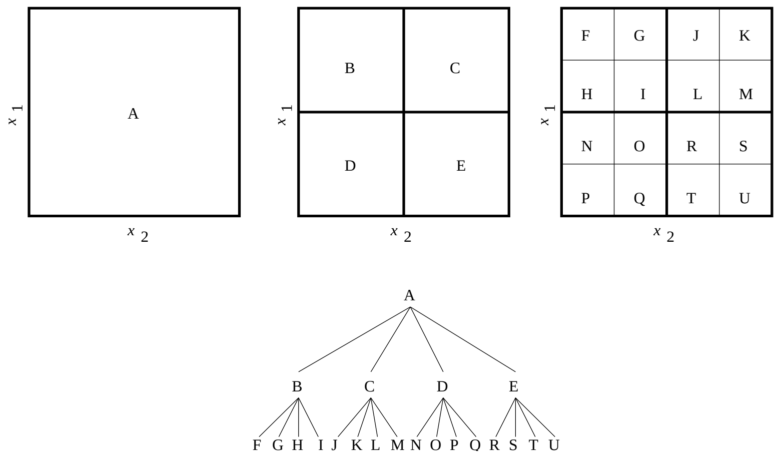

Figure 3: Example of the partition of a 2D space using a quadtree of depth two.

that we are looking for a partition that minimizes:

$$\lambda = \sum_{leaf\ nodes} \min(n_f, n_i)$$

Having this partition, we can continue partitioning with the same method until reaching the maximum allowable depth. Since we have tried several partitioning methods for a single node division, it is a better choice to choose a small depth limit so that no much time is spent in the creation of the tree.

The method described to expand nodes is only done for nodes corresponding to semi-feasible cells, and it stops when reaches the maximum depth of the tree. The tree is rebuilt every time the normative part is updated.

7 COMPARISON OF RESULTS

To validate our approach, we have used some test functions from the well-known benchmark proposed in [18] which has been often used in the literature to validate new constraint-handling techniques. The specific test functions used are the following:

1. **g01**:

   Minimize:

$$f(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

subject to:

$$\begin{aligned}
g_1(\vec{x}) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \le 0\\
g_2(\vec{x}) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \le 0\\
g_3(\vec{x}) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \le 0\\
g_4(\vec{x}) &= -8x_1 + x_{10} \le 0\\
g_5(\vec{x}) &= -8x_2 + x_{11} \le 0\\
g_6(\vec{x}) &= -8x_3 + x_{12} \le 0\\
g_7(\vec{x}) &= -2x_4 - x_5 + x_{10} \le 0\\
g_8(\vec{x}) &= -2x_6 - x_7 + x_{11} \le 0\\
g_9(\vec{x}) &= -2x_8 - x_9 + x_{12} \le 0
\end{aligned}$$

where the bounds are $0 \le x_i \le 1$ $(i = 1, \ldots, 9)$, $0 \le x_i \le 100$ $(i = 10, 11, 12)$ and $0 \le x_{13} \le 1$.

2. **g04**:

   Minimize:

$$f(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

subject to:

$$g_1(\vec{x}) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0$$
$$g_2(\vec{x}) = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0$$
$$g_3(\vec{x}) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0$$
$$g_4(\vec{x}) = -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0$$
$$g_5(\vec{x}) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0$$
$$g_6(\vec{x}) = -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0$$

where: $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$, $27 \leq x_i \leq 45$ $(i = 3, 4, 5)$.

3. **g08**

   Minimize:

$$f(\vec{x}) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$

   subject to:

$$\begin{aligned} g_1(\vec{x}) &= x_1^2 - x_2 + 1 \leq 0 \\ g_2(\vec{x}) &= 1 - x_1 + (x_2 - 4)^2 \leq 0 \end{aligned}$$

   where $0 \leq x_1 \leq 10$, $0 \leq x_2 \leq 10$.

4. **g12**

   Maximize:

$$f(\vec{x}) = (100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2)/100$$

   subject to:

$$g(\vec{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0$$

   where: $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$, and $p, q, r = 1, 2, \ldots, 9$. The feasible region of the search space consists of 9^3 disjointed spheres. A point (x_1, x_2, x_3) is feasible if and only if there exist p, q, r such that the above inequality holds.

The parameters used by our approach are the following: population size = 20, maximum number of generations = 2500, the normative part is updated at every 20 generations with the 25 % of the population (acceptance %), tournaments consist of 10 encounters by individual (half the population size), the maximum depth of the octree is equal to the number of decision variables of the problem. These parameters were derived empirically after numerous experiments.

Our results are compared to the homomorphous maps of Koziel & Michalewicz [12] (one of the best current constraint-handling techniques for evolutionary algorithms known to date) in Table 1. The results of Koziel and Michalewicz were obtained with 1,400,000 fitness function evaluations, whereas our approach required only 50,020 fitness function evaluations.

As can be seen in Table 1, our approach produces very good results with respect to the homomorphous maps (which is considerably more difficult to implement) at a fraction of its computational cost. The main reason for this cost reduction is that the belief cells are used to guide the search of the evolutionary algorithm very efficiently, avoiding that it moves to unpromising regions of the search space.

8 CONCLUSIONS

We have presented an approach based on cultural algorithms and evolutionary programming for constrained optimization. The approach has provided good results at a relatively low computational cost. This suggests that the proper use of domain knowledge may certainly improve the performance of an evolutionary algorithm when this is properly done. Also, it suggests that such domain knowledge may be extracted during the evolutionary process in which we aim to reach the global optimum of a problem. This contrasts with the more conventional approach of using domain knowledge extracted from previous runs of an evolutionary algorithm (see for example [9]).

One of the main drawbacks of cultural algorithms in constrained search spaces (i.e., memory usage) is attacked using spatial data structures that can efficiently store the belief space. Our preliminary results reported in this paper indicate that the approach is a viable alternative, although it is obviously necessary to perform more experiments, particularly in problems with a higher number of decision variables. Such work is currently under way.

Acknowledgements

The first author acknowledges support from CONACyT through project No. 32999-A. The second author acknowledges support from CONACyT through a scholarship to pursue graduate studies at the Com-

Table 1: Comparison of the results for the test functions selected from [18] using 2^n-trees. Our approach is called CAEP (Cultural Algorithms with Evolutionary Programming) and Koziel & Michalewicz's approach [12] is denoted by KM.

		BEST RESULT		**MEAN RESULT**		**WORST RESULT**	
TF	**OPTIMAL**	**CAEP**	**KM**	**CAEP**	**KM**	**CAEP**	**KM**
g01	-15	-15.0	-14.7864	-13.7574	-14.7082	-12.0	-14.6154
g04	-30665.539	-30665.539	-30664.5	-30665.539	-30655.3	-30665.537	-30645.9
g08	-0.095825	-0.095825	-0.095825	-0.095825	-0.0891568	-0.095825	-0.0291438
g12	1.000	1.000	0.999999857	0.9994375	0.999134613	0.994375	0.991950498

puter Science Section of the Electrical Engineering Department of CINVESTAV-IPN.

References

[1] Chan-Jin Chung. *Knowledge-Based Approaches to Self-Adaptation in Cultural Algorithms.* PhD thesis, Wayne State University, Detroit, Michigan, 1997.

[2] Chan-Jin Chung and Robert G. Reynolds. A Testbed for Solving Optimization Problems using Cultural Algorithms. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, Cambridge, Massachusetts, 1996. MIT Press.

[3] Carlos A. Coello Coello. Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, January 2002.

[4] Ernest Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.

[5] Kalyanmoy Deb. An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2/4):311–338, 2000.

[6] W. H. Durham. *Co-evolution: Genes, Culture, and Human Diversity.* Stanford University Press, Stanford, California, 1994.

[7] Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution. Forty Years of Evolutionary Programming.* John Wiley & Sons, Inc., New York, 1999.

[8] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[9] Eduardo Islas Pérez, Carlos A. Coello Coello, and Arturo Hernández Aguirre. Extraction of Design Patterns from Evolutionary Algorithms using Case-Based Reasoning. In Yong Liu, Kiyoshi Tanaka, Masaya Iwata, Tetsuya Higuchi, and Moritoshi Yasunaga, editors, *Evolvable Systems: From Biology to Hardware (ICES'2001)*, pages 244–255. Springer-Verlag. Lecture Notes in Computer Science No. 2210, October 2001.

[10] C.L. Jakson and S.L. Tanimoto. Octrees and Their Use in Representing Three-Dimensional Objects. *Computer Graphics and Image Processing*, 14(3):249–270, 1980.

[11] Xidong Jin and Robert G. Reynolds. Using Knowledge-Based Evolutionary Computation to Solve Nonlinear Constraint Optimization Problems: a Cultural Algorithm Approach. In *1999 Congress on Evolutionary Computation*, pages 1672–1678, Washington, D.C., July 1999. IEEE Service Center.

[12] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[13] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Norwell, Massachusetts, 1993.

[14] Sushil J. Louis and Judy Johnson. Solving Similar Problems using Genetic Algorithms Case-Based Memory. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 283–290, San Francisco, California, 1997. Morgan Kaufmann Publishers.

[15] Carlos E. Mariano and Eduardo F. Morales. Distributed Reinforcement Learning for Multiple Objective Optimization Problems. In *2000 Congress on Evolutionary Computation*, volume 1, pages 188–195, Piscataway, New Jersey, July 2000. IEEE Service Center.

[16] Zbigniew Michalewicz. A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155. The MIT Press, Cambridge, Massachusetts, 1995.

[17] Zbigniew Michalewicz and Cezary Z. Janikow. Handling Constraints in Genetic Algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 151–157, San Mateo, California, 1991. Morgan Kaufmann Publishers.

[18] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[19] Tom Mitchell. *Version Spaces: An Approach to Concept Learning.* PhD thesis, Computer Science Department, Stanford University, Stanford, California, 1978.

[20] A. C. Renfrew. Dynamic Modeling in Archaeology: What, When, and Where? In S. E. van der Leeuw, editor, *Dynamical Modeling and the Study of Change in Archaelogy.* Edinburgh University Press, Edinburgh, Scotland, 1994.

[21] Robert G. Reynolds. An Introduction to Cultural Algorithms. In A. V. Sebald and L. J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific, River Edge, New Jersey, 1994.

[22] Robert G. Reynolds, Zbigniew Michalewicz, and M. Cavaretta. Using cultural algorithms for constraint handling in GENOCOP. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 298–305. MIT Press, Cambridge, Massachusetts, 1995.

[23] Zhiming Zhang and T. Warren Liao. Combining Case-Based Reasoning with Genetic Algorithms. In Scott Brave and Annie S. Wu, editors, *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, pages 305–310, Orlando, Florida, 1999.

Convergence velocity of an evolutionary algorithm with self-adaptation

Mikhail A. Semenov

IACR-Rothamsted
Harpenden, Hertfordshire
AL5 2JQ, United Kingdom

Abstract

A stochastic Lyapunov function was used to assess the convergence velocity of a simple evolutionary algorithm with self-adaptation, which searches for a maximum of a "fitness" function. This algorithm uses two types of parameters: "fitness" parameters belonging to the domain of the function, and strategy parameters, which control changes of fitness parameters. It was shown that the convergence velocity of the evolutionary algorithm with self-adaptation is exponential, similar to the convergence velocity of the optimal deterministic algorithm, the Fibonacci search, on the class of unimodal functions.

1 OPTIMAL DETERMINISTIC SEARCH ALGORITHM

Let $K[a,b]$ be a class of unimodal functions $f:[a,b]\to\mathbb{R}$. Let P^n be a set of *n*-point sequential deterministic algorithms $\{p_n\}$. A *n*-point algorithm p_n searches for maximum of a function $f\in K[a,b]$ by sequentially selecting x_k, based on calculation of values $f(x_1),\dots,f(x_{k-1})$, where $k\le n$. For any $f\in K[a,b]$ and $\forall\, p_n$ let the error of an algorithm p_n on a function f be defined as

$$\delta(p_n,f)=\left|x_n-x_f\right|$$

where x_f is a value where the function f has maximum $f(x_f)=\max\limits_{y\in[a,b]} f(y)$. A guaranteed error of the algorithm p_n on the class of functions $K[a,b]$ is defined as

$$\Lambda(p_n)=\sup_{f\in K[a,b]}\delta(p_n,f)$$

An algorithm p_{opt} is an optimal *n*-point sequential deterministic algorithm, if it has the minimum guaranteed error compared with all other algorithms from P^n

$$\Lambda(p_{opt})=\inf_{p_n\in P^n}\Lambda(p_n)$$

We denote $\gamma(n)=\inf\limits_{p_n\in P^n}\Lambda(p_n)$

THEOREM (Vasilev, 1980) *Fibonacci search* Φ_n *is the only optimal algorithm on the class* $K[a,b]$ *with*

$$\gamma(n)=\Lambda(\Phi_n)=\frac{(b-a)}{F_{n+2}}$$

where $F_n=\left[\left(\left(1+\sqrt{5}\right)/2\right)^n-\left(\left(1-\sqrt{5}\right)/2\right)^n\right]\Big/\sqrt{5}$

is a Fibonacci number.

2 CONVERGENCE VELOCITY OF SUPERMARTINGALES

We use the stochastic Lyapunov function to assess the convergence velocity of the evolutionary algorithm with self-adaptation (Kushner, 1967; Semenov and Terkel, 1984; Semenov, 2001). If X_t is a stochastic process with values from an arbitrary state space X, then its stochastic Lyapunov function is a numerical function $V(X_t)$ decreasing on average along the trajectories of the process X_t, i.e. it is a super-martingale (Doob, 1990; Neveu, 1964; Williams, 2000). The convergence velocity of a super-martingale allows the assessment of the convergence velocity of the stochastic Lyapunov function, which in turn can be used to assess a convergence velocity of a stochastic process itself. The aim of this section is the Proposition 2.1 , which estimates the convergence velocity of a super-martingale

Let us fix a probability space (Ω, A, P), where Ω denotes the sample space, A is a σ-algebra of measurable sets and P is a probability measure, provided with an increasing family of σ-algebras A_t (t is the discrete time). Let (V_t) be a supermartingale adapted to the family (A_t). In Proposition 2.1 we analyse the asymptotic behaviour of (V_t) under the following restrictions: (1) V_t decreases on average each time by a fixed constant $a > 0$, (2) the variation of V_t does not exceed on average $b > 0$.

PROPOSITION 2.1. *Let* (V_t) *be a supermartingale,* $V_0 = 0$. *If the following conditions hold*

1. $E^{A_t}(V_{t+1}) \le V_t - a$
2. $E^{A_t}\left(\left(V_{t+1} - E^{A_t}(V_{t+1})\right)^2\right) \le b$

where a>0, b>0, then $\forall \varepsilon > 0$ *the inequality holds asymptotically almost everywhere*

$$V_t \le -at + \overline{o}(t^{1/2+\varepsilon})$$

where $E^{A_t}(V_{t+1})$ *is a conditional expectation with respect to a* σ*-algebra* A_t.

PROOF: Let us decompose a supermartingale V_t into a sum of martingale Y_t and an increasing process H_t

(2.1) $\quad V_t = Y_t - H_t$

where

$$Y_0 = V_0,\ Y_{t+1} - Y_t = V_{t+1} - E^{A_t}(V_{t+1})$$
$$H_0 = 0,\ H_{t+1} - H_t = V_t - E^{A_t}(V_{t+1})$$

Using inequality 1 from Proposition 2.1, H_t can be assessed as

(2.2) $\quad H_t = \sum_{i=1}^{t} (H_i - H_{i-1}) + H_0 \ge at$

Let show that a martingale $Y_t \in L^2$. Indeed

$$Y_t = \sum_{i=1}^{t} \left(V_i - E^{A_{i-1}}(V_i)\right) + V_0$$

and according to Inequality 2

$$E\left(\left(V_{t+1} - E^{A_t}(V_{t+1})\right)^2\right) = E\left(E^{A_t}\left(\left(V_{t+1} - E^{A_t}(V_{t+1})\right)^2\right)\right) \le b$$

Let K_t be an increasing process in Doob's decomposition of a submartingale Y_t^2 (Doob, 1990). Let us evaluate K_t using inequality 2

$$K_{t+1} - K_t = E^{A_t}(Y_{t+1}^2) - Y_t^2 = E^{A_t}(Y_{t+1} - Y_t)^2 =$$
$$E^{A_t}\left(V_{t+1} - E^{A_t}(V_{t+1})\right)^2 \le b$$

therefore

(2.3) $\quad K_t \le \sum_{i=1}^{t} (K_i - K_{i-1}) + K_0 \le bt$

According to (Neveu, 1964)

(2.4) $\quad Y_t = \overline{o}\left(K_t^{1/2+\varepsilon}\right)$ a.e. on $\{K_\infty = \infty\}$

where $K_\infty = \lim_{t\to\infty} K_t$ and by definition

$g(t) = \overline{o}(h(t))$, if $\lim_{t\to\infty} \frac{g(t)}{h(t)} \to 0$. Using inequality (2.3) and (2.4) $\forall \varepsilon > 0$

(2.5) $\quad Y_t = \overline{o}\left(t^{1/2+\varepsilon}\right)$

Replacing Y_t and H_t in (2.1) by (2.2) and (2.5) we obtain

$$V_t \le -at + \overline{o}\left(t^{1/2+\varepsilon}\right)$$

□

3 CONVERGENCE VELOCITY OF EVOLUTIONARY ALGORITHM WITH SELF-ADAPTATION

We now apply Proposition 2.1 to prove the convergence of a simple evolutionary algorithm with self-adaptation (Semenov, 2001; Semenov and Terkel, 1985) (Back et al., 2000b; Beyer, 1995). The population consists of only one individual (x_t, x_t^*) at time t, which produces M offspring according to the following formulae:

(3.1) $$\begin{cases} x_{t,i}^* = x_t^* \exp(\vartheta_{t,i}) \\ x_{t,i} = x_t + x_{t,i}^* \xi_{t,i} \end{cases}$$

where the random variables $\vartheta_{t,i}$ are independent and uniformly distributed on the interval [-2,2] and $\xi_{t,i}$ are independent and uniformly distributed on [-1,1]. From the M generated rivals $\{(x_{t.i}, x_{t.i}^*), i = 1,..M\}$ only one is selected, which has a maximum $f(x_{t,i})$ and it becomes the next state (x_{t+1}, x_{t+1}^*), where $f(x) = -|x|$.

Let us estimate the convergence velocity of the evolutionary algorithm with self-adaptation $\left(x_t, x_t^*\right)$. Corollary 3.4 shows that the evolutionary algorithm with self-adaptation has an exponential convergence velocity. The convergence velocity will be estimated using Proposition 3.1. Let us construct a stochastic Lyapunov function for the process $\left(x_t, x_t^*\right)$ as

(3.2) $$V\left(x, x^*\right) = \ln\left(E_{x,x^*}\left(\left|x_+\right|\right)\right) - k\ln\left(x^*\right)$$

where $\left(x_+, x_+^*\right)$ is the state generated from $\left(x, x^*\right)$.

PROPOSITION 3.1 *A stochastic process* $V_t = V\left(x_t, x_t^*\right)$*, defined by (3.2), is a supermartingale and* $\exists a > 0,\ b > 0$ *such that inequalities*

1. $E_{x,x^*}\left(V\left(x_+, x_+^*\right)\right) \le V\left(x, x^*\right) - a$
2. $E_{x,x^*}\left(\left[V\left(x_+, x_+^*\right) - E_{x,x^*}\left(V\left(x_+, x_+^*\right)\right)\right]^2\right) \le b$

hold. $E_{x,x^*}\left(V\left(x_+, x_+^*\right)\right)$ *is a conditional expectation.*

PROOF: 1). It can be shown that

$$E_{x,x^*}\left(V\left(x_+, x_+^*\right)\right) - V\left(x, x^*\right) =$$
$$E_{x/x^*,1}\left(V\left(x_+, x_+^*\right)\right) - V\left(x/x^*, 1\right)$$

The function $G(x) = E_{x,1}\left(V\left(x_+, x_+^*\right)\right) - V\left(x, 1\right)$ was investigated numerically (see Appendix) and it was shown that $\exists a > 0 : \forall x \in \mathbb{R}\ G(x) \le -a$.

2). The second inequality is a direct deduction from the fact that both

$$E_{x,x^*}\left(\left[\ln E_{x_+,x_+^*}\left(\left|x_{++}\right|\right) - E_{x,x^*}\left(\ln E_{x_+,x_+^*}\left(\left|x_+\right|\right)\right)\right]^2\right)$$

and $E_{x,x^*}\left(\left[\ln\left(x_+^*\right) - E_{x,x^*}\ln\left(x_+^*\right)\right]^2\right)$ are bounded and the Cauchy-Schwarz integral inequality

$$\left[E\left(fg\right)\right]^2 \le \left[E\left(f^2\right)\right]\left[E\left(g^2\right)\right].$$

□

COROLLARY 3.3 *The following inequality for the process* $V_t = V\left(x_t, x_t^*\right)$*, defined by (3.2)*

$$V_t \le -at + \overline{o}\left(t^{1/2+\varepsilon}\right)$$

holds asymptotically almost everywhere.

COROLLARY 3.4 *Evolutionary algorithm with self-adaptation* $\left(x_t, x_t^*\right)$ *converges to* $\left(0, 0\right)$ *almost everywhere; moreover the following inequalities*

$$\left|x_t\right| \le \exp\left(-at\right), \qquad x_t^* \le \exp\left(-at\right)$$

hold asymptotically almost everywhere.

PROOF: Using formula (3.2) and Corollary 3.3 we can conclude that the following inequality

(3.6) $$E_{x_t,x_t^*}\left(\left|x_{t+1}\right|\right) / x_t^{*k} \le \exp\left(-at + \overline{o}\left(t^{1/2+\varepsilon}\right)\right)$$

holds asymptotically almost everywhere. Taking into account that $\exists\gamma > 0$ $E_{x_t,x_t^*}\left(\left|x_{t+1}\right|\right) = x_t^* E_{x_t/x_t^*,1}\left(\left|x_{t+1}\right|\right) \ge \gamma x_t^*$, we can transform (3.6) to

$$x_t^* \le \exp\left(-at\right)\left[\gamma^{-1/1-k}\exp\left(-\frac{a}{1-k}t + \overline{o}\left(t^{1/2+\varepsilon}\right)\right)\right]$$

The expression in the square brackets is less then 1 for large *t,* therefore the inequality

$$x_t^* \le \exp\left(-at\right)$$

holds asymptotically almost everywhere. The second inequality can be proved similarly.

□

4 CONCLUDING REMARK

In Section 3 by assessing a convergence velocity of the evolutionary algorithm with self-adaptation, we automatically proved its convergence. This proof is independent from the technique described in (Semenov, 2001).

Evolutionary algorithms with self-adaptation can be considered as universal methods for optimum search and can be used for solving optimisation problems of high complexity where heuristic deterministic procedures are difficult to develop (Back et al., 2000a). Universality of evolutionary algorithms with self-adaptation is achieved by allowing control parameters evolve along with the "fitness" parameters by an indirect effect of selection (Semenov and Terkel, 1985). Although stochastic search algorithms in general are less effective than deterministic ones, an evolutionary algorithm with self-adaptation has an exponential convergence velocity, which results from the co-evolution of control and "fitness" parameters. The Fibonacci search, the optimal deterministic search algorithm (Section 1), has an exponential convergence velocity $\left| x_n - x_f \right| < \beta e^{-\alpha n}$ with $\alpha \approx 0.48$. The evolutionary algorithm with self-adaptation has $\alpha \approx 0.09$ (note that n is a number of times f was calculated, therefore, $n = t * M$ for the evolutionary algorithm).

Acknowledgements

I am eternally indebted to D.A. Terkel for his contribution. IACR receives grant-aided support from the Biotechnology and Biological Sciences Research Council of the United Kingdom.

APPENDIX

Let us show that $\exists a > 0 : G(x) \le -a$. Because of apparent difficulties in dealing with this function analytically , we assess it numerically using the Monte-Carlo method. Let us decompose the function $G(x) = E_{x,1}\left(V\left(x_+, x_+^*\right)\right) - V(x,1) = V_1(x) + V_2(x)$ where $V_1(x) = E_{x,1}\left(\ln\left(E_{x_+,x_t^*}\left|x_{++}\right|\right)\right) - \ln\left(E_{x,1}\left|x_+\right|\right)$ and $V_2(x) = -k\, E_{x,1} \ln\left(x_+^*\right)$**.** Note, that

$E_{x,x^*}\left|x_+\right| = x^* E_{x/x^*,1}\left|y_+\right|$ and

$E_{x,1}\left|x_+\right| = E_{e^2,1}\left|x_+\right| + x - e^2 \quad \forall x \ge e^2$

Also note, that $E_{x,1} \ln\left(x_+^*\right) = const > 0 \quad \forall x \ge e^2$.

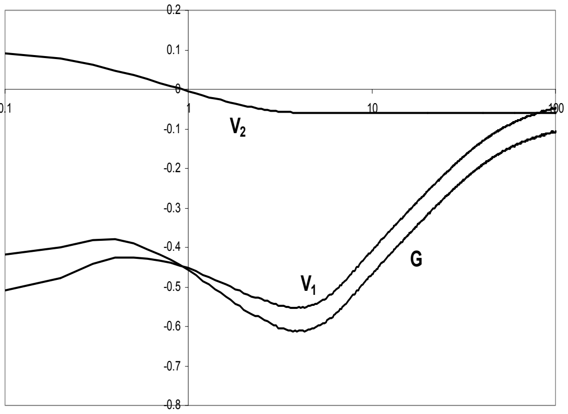

Figure 1. Functions V_1, V_2 and G were calculated by the Monte-Carlo methods for the number of offspring M=5. A logarithmic scale is used for the x-axis.

Results of Monte-Carlo calculations for the number of offspring M=5 are presented on Figure 1, $k = 0.1$

References

T. Back, D. B. Fogel, and Z. Michalewicz. (2000a). *Evolutionary Computation 1. Basic Algorithms and Operators*, IOP Publishing Ltd, Bristol.

T. Back, D. B. Fogel, and Z. Michalewicz. (2000b). *Evolutionary Computation 2. Advanced Algorithms and Operators*, IOP Publishing Ltd, Bristol.

H. G. Beyer (1995). Toward a theory of evolution strategies: self-adaptation. *Evolutionary Computation*, **3**(3):311-348.

J. L. Doob. (1990). *Stochastic processes*, John Wiley & Sons Inc, London.

J. H. Kushner. (1967). *Stochastic stability and control*, Academic Press.

J. Neveu. (1964). *Bases mathematiques du calcul des probabilities*, Masson, Paris.

M. A. Semenov (2001). Analysis of evolutionary search with mutators using a stochastic Lyapunov function. *GECCO 2001*, San Francisco, 372-376.

M. A. Semenov, and D. A. Terkel (1984). On the convergence set of stochastic Lyapunov function. *Am.Math.Society: Sov.Math.Dokl*, **30**(2):376-378.

M. A. Semenov, and D. A. Terkel (1985). On the evolution of hereditary variability mechanisms by means of indirect effect of selection. *General Biology*, **XLVI**(2):271-278.

F. P. Vasilev. (1980). *Computational methods for optimisation problems*, Nauka, Moscow.

D. Williams. (2000). *Probability with martingales*, Cambridge University Press, Cambridge.

A Hybrid Data Mining Approach to Discover Bayesian Networks Using Evolutionary Programming

Man Leung Wong
Department of Information Systems
Lingnan University
Tuen Mun, Hong Kong
mlwong@ln.edu.hk
(852)26168093

Shing Yan Lee
Department of Computer Science
and Engineering, CUHK,
Shatin, Hong Kong
sylee@cse.cuhk.edu.hk

Kwong Sak Leung
Department of Computer Science
and Engineering, CUHK,
Shatin, Hong Kong
ksleung@cse.cuhk.edu.hk

Abstract

Given the explosive growth of data collected from current business environment, data mining can potentially discover new knowledge to improve managerial decision making. We propose a novel data mining approach that employs evolutionary programming to discover knowledge represented in Bayesian networks and apply the approach to marketing data. There are two different approaches to the network learning problem. The first one uses dependency analysis, while the second approach searches good network structures according to a metric. Unfortunately, the two approaches both have their own drawbacks. Thus, we propose a novel hybrid of the two approaches. With this new idea, we endeavor to improve upon our previous work, MDLEP, which uses evolutionary programming for network learning. We also introduce a new operator to further enhance the search efficiency. We conduct a number of experiments and compare the hybrid approach with MDLEP. The empirical results illustrate that the approach improves over MDLEP.

1 INTRODUCTION

Conventional marketing research is a process in which data are analyzed manually to explore the relationships among various factors defined by the researcher. Even with powerful computers and versatile statistical software, many hidden and potentially useful relationships may not be recognized by the analyst. Nowadays, such problems are more acute as many businesses are capable of generating and collecting a huge amount of data in a relatively short period. The explosive growth of data requires a more efficient way to extract useful knowledge. Through data mining, marketing researchers can discover complex relationships among various factors and extract meaningful knowledge to improve the efficiency and quality of managerial decision making. In this paper, we propose a novel data mining approach that employs evolutionary programming to discover knowledge represented in Bayesian Networks and apply the approach to marketing data.

A Bayesian network is a graphical representation that depicts conditional independence among random variables in the domain and encodes the joint probability distribution [1]. With a network at hand, probabilistic inference can be performed to predict the outcome of some variables based on the observations of others. Therefore, Bayesian networks are often used in diagnostic systems [2].

Typically, a Bayesian network is constructed by eliciting knowledge from domain experts. To reduce imprecision due to subjective judgements, some researchers propose methods to construct Bayesian networks from collected data. In the literature, there are two main approaches to this network learning problem [3]. The first one is the dependency analysis approach [4, 3]. Since a Bayesian network describes conditional independence, we could make use of dependency test results to construct a Bayesian network that conforms to our findings. The second one, called the score-and-search approach [5, 6, 7], uses a metric to evaluate a candidate network structure. With the metric, a search algorithm is employed to find a network structure having the best score. Thus, the learning problem becomes a search problem. Unfortunately, the two approaches both have their own drawbacks. For the former approach, an exponential number of dependency tests should be performed. Moreover, some test results may be inaccurate [4]. For the latter approach, the search space is huge. Therefore, some algorithms [5] adopt greedy search heuristics which may easily make the algorithms get stuck in a local optimum [7].

In this work, a hybrid framework is developed for the network learning problem. Simply put, dependency analysis results are used to reduce the search space of the score-and-search process. With such reduction, the search process would take less time for finding the optimal solution.

Together with the introduction of a new operator and some modifications of our previous work, MDLEP, we call our new approach HEP (hybrid MDLEP). We have conducted a number of experiments and compared HEP with MDLEP. The empirical results illustrate that HEP improves over MDLEP. Moreover, it is found that HEP executes much faster than MDLEP which is very important for real-world applications.

This paper is organized as follows. In section 2, we present the backgrounds of Bayesian networks, the MDL metric, and MDLEP. In section 3, we describe our algorithm in detail. In sections 4 and 5, we report our experimental findings. We conclude the paper in section 6.

2 LEARNING BAYESIAN NETWORKS FROM DATA

2.1 BAYESIAN NETWORKS

A Bayesian network, G, has a directed acyclic graph (DAG) structure. Each node in the graph corresponds to a discrete random variable in the domain. An edge, $X \leftarrow Y$, on the graph, describes a parent and child relation in which X is the child and Y is the parent. All parents of X constitute the parent set of X which is denoted by Π_X. In addition to the graph, each node has a conditional probability tables (CPT) specifying the probability of each possible state of the node given each possible combination of states of its parent. If a node contains no parent, the table gives the marginal probabilities of the node [1].

Since Bayesian networks are founded on the idea of conditional independence, it is necessary to give a brief description here. Let U be the set of variables in the domain and let P be the joint probability distribution of U. Following Pearl's notation, a conditional independence (CI) relation is denoted by $I(X, Z, Y)$ where X, Y, and Z are disjoint subsets of variables in U. Such notation says that X and Y are conditionally independent given the *conditioning set*, Z. Formally, a CI relation is defined with:

$$P(x, y \mid z) = P(x \mid z) \quad \text{whenever} \quad P(y, z) > 0 \quad (1)$$

where x, y,and z are any value assignments to the set of variables X, Y, and Z respectively. A CI relation is characterized by its *order*, which is the number of variables in the conditioning set Z.

As mentioned before, researchers treat the network learning problem in two very different ways. The first approach tries to construct a Bayesian network using dependency information obtained from the data. By assuming that P is faithful to a Bayesian network G [4], we could add or remove edges from G according to the discovered conditional independence relations. Given the sets of variables, X, Y, and Z, we could check the validity of $I(X, Z, Y)$ by performing statistical test, called CI test. The major problem of this approach is that it is difficult to know if two nodes are conditionally dependent [4]. Furthermore, when a high-order CI relation is tested in a small data set, the test result may be unreliable [4]. The second approach makes use of a metric which evaluates the quality of a Bayesian network with respect to the given data. Such metric may be derived from information theory, Bayesian statistics, or Minimum Description Length principle (MDL). With the metric, the network learning problem becomes a search problem. Unfortunately, since the search space is huge, the search problem is difficult [7].

2.2 THE MDL METRIC

The MDL metric [6] is derived from information theory and incorporates the Minimum Description Length principle. With the composition of the description length for network structure and the description length for data, the MDL metric tries to balance between model accuracy and model complexity. Hence, the best network needs to be both accurate and simple. Using the metric, a better network would have a smaller score. Similar to other metrics, the MDL score for a Bayesian network, G, is *decomposable* [7] and could be written as in equation 2. Let $U = \{N_1, \ldots, N_n\}$ be the set of nodes and let Π_{N_i} denotes the parent set of node N_i. The MDL score of the network is simply the summation of the MDL score of Π_{N_i} of every node N_i in the network.

$$\mathrm{MDL}(G) = \sum_{N_i \in U} \mathrm{MDL}(N_i,\ \Pi_{N_i}) \quad (2)$$

2.3 MDLEP

Our previous work [8], called MDLEP, belongs to the score-and-search approach in which we use the MDL metric together with evolutionary programming (EP) for searching a good network structure. An individual in the search population is a candidate network structure. An outline of the algorithm is presented as follows:

1. Set t, the generation count, to 0.
2. Create an initial population, Pop(t) of m random DAGs (m is the population size.)
3. Each DAG in the population is evaluated using the MDL metric.
4. While t is less than the maximum number of generations,
    - Each DAG in Pop(t) produces one offspring by performing a number of mutation operations. Four different mutation operators, simple mutation, reverse mutation, move mutation, and knowledge-guided mutation are used. If there is cycle, a randomly picked edge in the cycle is removed.
    - The DAGs in Pop(t) and all new offspring are stored in the intermediate population Pop$'(t)$. The size of Pop$'(t)$ is 2*m.
    - Conduct a number of pairwise competitions over all DAGs in Pop$'(t)$. For each G_i in the population, q other individuals are selected. The fitness of G_i is compared against the q individuals. The score of G_i is the number of individuals (out of q) that are worse than G_i.
    - Select the m highest score individuals from Pop$'(t)$ with ties broken randomly. The individuals are stored in Pop($t+1$).
    - increment t by 1.
5. Return the individual that has the lowest MDL metric in any generation of the run as the output of the algorithm.

When comparing MDLEP against another approach using GA [9], it is found that MDLEP generally outperforms its opponent.

3 HYBRID MDLEP (HEP)

Although MDLEP outperforms its GA opponent, its efficiency can be enhanced by employing a number of strategies. First, a hybrid approach is introduced so that the knowledge from dependency tests is exploited during searching. Second, previous search results are reused through a new merge operator. Third, in contrast to MDLEP where repairing is needed, the formation of cycle is avoided altogether when producing new individuals. Since a hybrid approach is adopted in Bayesian network learning, this approach is called HEP (hybrid MDLEP). In the following subsections, the ideas will be discussed in detail.

3.1 A HYBRID APPROACH

In dependency analysis approach, CI test is typically used to check the validity of a conditional independence assertion $I(X,Z,Y)$ of any given two nodes X, Y and a conditioning set Z. Assume that the χ^2 test is employed, the assertion is modeled as the null hypothesis. A χ^2 test generates a p-value, ranges between 0 and 1, which shows the least level of significance for which the given data leads to the rejection of the null hypothesis. In effect, if the p-value is less than a predefined cutoff value, α, the hypothesis $I(X,Z,Y)$ is rejected. Otherwise, if the p-value is greater than or equal to α, the hypothesis could not be rejected and $I(X,Z,Y)$ is assumed to be valid. Consequently, this implies that the two nodes, X and Y, cannot have a direct edge between them. In other words, the edges $X \leftarrow Y$ and $X \rightarrow Y$ cannot exist in the resultant network.

With such observation, a hybrid framework for learning Bayesian networks is formulated which consists of two phases. In the first phase, low-order CI tests are conducted so that some edges could be removed. Only low-order CI tests are performed because their results are more reliable than higher order tests and the time complexity is bounded. In the second phase, a score-and-search approach is used together with the knowledge obtained previously. In particular, the search space is limited by excluding networks that contain the edges $X \leftarrow Y$ or $Y \rightarrow X$ for which $I(X,Z,Y)$ is assumed to be valid. Since the search space is reduced, the learning problem becomes easier and less time will be needed for finding the best network.

This idea could be applied readily in MDLEP. After obtaining the test results, all candidate networks having invalid edges are prevented from being generated.

Although such formulation can work fine, it must be emphasized that the choice of α has a critical impact. If improper α is used, in the worst case, either all edges are pruned away or all edges are retained. Hence, although it is possible to impose the restrictions from CI tests as *global* constraints, there is the risk of assuming our choice of α is appropriate.

As an alternative, a novel realization of the hybrid framework is developed in which a different α is used for each individual in the population. Thus, each individual has, besides the network structure, a cutoff value α which is also subjected to be evolved. As the evolutionary search proceeds, individual having an improper value of α will eventually be eliminated. In general, small value of α implies more constraints (less likely to reject an hypothesis) and results in a more restricted search space. Hence, if the value of α of an individual is too small which excludes some *important* edges, the individual will have a greater chance of being eliminated. On the other hand, if the value of α of an individual is too large, it is less likely to find the *right* edge (because there are many *wrong* alternatives) for its offspring. Consequently, the individual will also have a higher chance of being eliminated.

This idea is implemented in the first phase by storing the largest p-value returned by the CI tests for every possible conditioning set, Z (restricted to order-0 and all order-1 tests) in a matrix, Pv. In the second phase, for a given individual G_i in the population with associated cutoff value α_i, an edge $X \leftarrow Y$ cannot be added if Pv_{XY} is greater than α_i (i.e. $I(X, Z, Y)$ is assumed to be valid). The value of each α_i is randomly initialized in the beginning. In subsequent generations, an offspring will inherit the cutoff value from its parent with a possible increment or decrement by Δ_α.

3.2 THE MERGE OPERATOR

In addition to the four mutation operators, a new operator called merge is introduced. Taking a parent network G_a and another network G_b as input, the merge operator attempts to produce a better network structure (in terms of MDL score) by modifying G_a with G_b. If no modification can be done, G_a is returned.

Let M_i^x denotes the MDL score of the parent set $\Pi_{N_i}^x$ of node $N_i \in U$ in the network G_x. Recalling that the MDL score is decomposable and a network is an agglomeration of Π_{N_i} (for $i = 1, \ldots, n$). Thus, given two input networks G_a and G_b, a better network, G_c, could be generated by selecting $\Pi_{N_i}^c$ from $\Pi_{N_i}^a$ or $\Pi_{N_i}^b$ so that (1) there is no cycle in G_c and (2) the sum $\sum_{N_i \in U} M_i^c$ is less than $\sum_{N_i \in U} M_i^a$ or $\sum_{N_i \in U} M_i^b$. With such observation, the merge operator is devised which is an heuristic for finding a subset of nodes, $W \subset U$, with which $\Pi_{N_j}^a$ are replaced with $\Pi_{N_j}^b$ in G_a for every $N_j \in W$. Meanwhile, the replacement would not create cycles and has a MDL score smaller than that of G_a.

For the two input networks G_a and G_b, the merge procedure produces a node ordering by sorting $\delta_i = M_i^a - M_i^b$ in descending order. Since positive δ_i means that $\Pi_{N_i}^b$ is better than $\Pi_{N_i}^a$, the procedure follows the ordering in considering the replacement of $\Pi_{N_i}^a$ with $\Pi_{N_i}^b$. Beginning with the first node, N_i, in the ordering, the merge procedure invokes the procedure `findSubset(`N_i`)` to find a subset of nodes W' such that by replacing $\Pi_{N_j}^a$ with $\Pi_{N_j}^b$ for every $N_j \in W'$ in G_a, the resultant graph is still acyclic.

```
Procedure merge($G_a$, $G_b$)
  1. Find $\delta_i = M_i^a - M_i^b$ for every node $N_i \in U$.
  2. Produce a node ordering $L$ by sorting $\delta_i$ in descending order.
  3. While there are nodes in $L$ that have not been considered,
       • Get the next node, $N_i$, from $L$ which is unconsidered.
       • Invoke findSubset($N_i$) which returns $W'$.
       • Sum $\delta_j$ for every node $N_j \in W'$.
       • If the sum is positive, mark every node $N_j \in W'$ in $L$ as considered. Insert $W'$ to $W$.
  4. Replace $\Pi_{N_j}^a$ with $\Pi_{N_j}^b$ for every $N_j \in W$.
```

After obtaining W', the merge procedure calculates the sum $\sum_{N_j \in W'} \delta_j$. If the sum is positive, it inserts W' into W and removes W' from the ordering. The procedure repeatedly examines the next node in the ordering until all nodes are considered. Finally, the procedure replaces $\Pi_{N_j}^a$ with $\Pi_{N_j}^b$ in G_a for every $N_j \in W$.

Essentially, the merge operator increases the efficiency in several ways. Since the score of the composite network can be readily calculated, it is not necessary to invoke the procedure for MDL score evaluation which is time-consuming. Thus, the merge operator offers an economical way to create new structures. Furthermore, the operator improves the search efficiency by creating more good individuals in each generation. In our current implementation, the operator merges networks at the current population with dumped networks from the last generation. Thus, it reuses the search results obtained in previous generations.

3.3 PREVENTION OF CYCLE FORMATION

Since MDLEP consumes much time in repairing networks that contain cycles, HEP prevents cycle formation in all candidate networks to handle this problem. HEP maintains the *connectivity matrix* containing the count of directed paths between every pair of nodes. If $X \to \cdots \to Y$ exists in a network, HEP forbids adding the edge $X \leftarrow Y$ to the network. The matrix is updated when an edge is added or removed.

3.4 THE ALGORITHM

The algorithm of HEP is presented as follows:

CI test Phase

For every pair of nodes (X, Y),

- Perform order-0 and all order-1 CI tests.
- Store the highest p-value in the matrix Pv.

Evolutionary Programming Search Phase

1. Set t, the generation count, to 0.
2. Initialize the value of m, the population size.
3. For each individual in the population Pop(t),
    - initialize the α value randomly.
    - refine the search space by checking the α value against the Pv matrix.
    - Inside the reduced search space, create a DAG randomly.
4. Each DAG in the population is evaluated using the MDL metric.
5. While t is less than the maximum number of generations,
    - select $m/2$ individuals from Pop(t), the rest are marked "NS" (not selected)
    - For each of the selected ones,
        - merge with a random pick from the dumped half in Pop$'(t-1)$.
        - If merge does not produce a new structure, mark the individual with "NS"
        - otherwise, regard the new structure as an offspring.
    - For each individual marked "NS",
        - produce an offspring by cloning.
        - alter the α value of the offspring by a possible increment or decrement of Δ_α.
        - refine the search space by checking the α value against the Pv matrix.
        - change the structure by performing a number of mutation operations. Note that cycle formation is prohibited.
    - The DAGs in Pop(t) and all new offspring are stored in the intermediate population Pop$'(t)$. The size of Pop$'(t)$ is 2*m.
    - Conduct a number of pairwise competitions over all DAGs in Pop$'(t)$. For each DAG G_i in the population, q other individuals are selected. The fitness of G_i is compared against the q individuals. The score of G_i is the number of individuals (out of q) that are worse than G_i.
    - Select the m highest score individuals from Pop$'(t)$ with ties broken randomly. The individuals are stored in Pop$(t+1)$.
    - increment t by 1
6. Return the individual that has the lowest MDL metric in any generation of a run as the output of the algorithm.

4 COMPARISON WITH MDLEP

In our experiments, we compare HEP against MDLEP on a number of data sets. We use the data sets that are generated from two benchmark networks, ALARM and PRINTD which also appear in [8]. There are data sets of sizes 1,000, 2,000, 5,000, and 10,000 for ALARM and one data set of 5,000 cases for PRINTD. Since both algorithms are stochastic in nature, we have conducted 40 trials for each experiment. The programs are executed on the same Sun Ultra-5 workstation. For HEP, we set Δ_α to be 0.02. For both algorithms, the population size is 50 and the tournament size (q) is 7. We use 5000 generations as the common termination criterion and the maximum size of parent set is set to be five. We compare the performance under five different aspects:

- average MDL score obtained, the smaller the better (Score),
- average running time in seconds (Time),
- average score of the first generation solution (I-Score),
- average generation that the best-so-far is found (ANG),
- average number of edges added, omitted or reversed in compared to the original structure (ASD).

Table 1 provides a summary of the performance comparison between the two algorithms. The MDL score of the original network is also included (just below the name of each data set) as reference. Numbers in parentheses are the standard deviations.

For all data sets, HEP could always find better or equally good network structures in terms of both MDL

Table 1 Performance comparison between the two algorithms over different data sets

Data Set		Score	Time	I-Score	ANG	ASD
ALARM 1000 (18533.5)	HEP	17871.7 (37.2)	212.1 (12.1)	24542.5 (1103.6)	750.1 (1101.4)	12.0 (1.7)
	MDLEP	17974.4 (87.1)	1115.4 (70.2)	30833.6 (786.3)	4353.3 (666.6)	18.7 (4.0)
ALARM 2000 (34287.9)	HEP	33800.5 (86.8)	273.6 (33.5)	44188.5 (1179.2)	1012.7 (1409.2)	8.5 (1.5)
	MDLEP	34018.3 (205.6)	1608.5 (196.9)	57138.3 (1228.6)	4163.8 (762.9)	13.5 (3.9)
ALARM 5000 (81233.4)	HEP	81004.0 (0.0)	491.8 (116.7)	102002.0 (2254.3)	421.1 (737.6)	7.1 (0.4)
	MDLEP	81237.9 (432.4)	3224.3 (449.8)	133901.7 (1968.4)	4016.9 (677.6)	11.1 (4.1)
ALARM 10000 (158497.0)	HEP	158425.0 (30.2)	942.5 (358.8)	197090.0 (5398.9)	932.0 (995.9)	3.6 (0.8)
	MDLEP	158677.7 (394.8)	6443.0 (540.9)	257057.5 (4562.1)	3653.2 (800.5)	7.6 (3.7)
PRINTD 5000 (106541.6)	HEP	106541.6 (0.0)	224.5 (43.4)	111113.0 (447.0)	40.0 (8.8)	0.0 (0.0)
	MDLEP	106541.6 (0.0)	1692.2 (28.1)	116008.2 (509.1)	510.9 (87.4)	0.0 (0.0)

score and structural difference. Hence, under the same termination criterion, it suggests that HEP is more efficient than MDLEP. Moreover, our approach consumes much less time than MDLEP. In some case, the saving is so great that HEP has nearly an eight-fold speedup. Hence, HEP is more favorable to MDLEP for real world practice. Since we have adopted the hybrid framework for creating the initial population, HEP usually has a better starting point than MDLEP as reflected by I-score.

5 APPLICATION IN DATA MINING

In this section, HEP is applied to a data mining problem in direct marketing. The objective of the problem is to predict potential prospects from the buying records of previous customers. Advertising campaign, which includes mailing of catalogs or brochures, is then targeted on the group of potential prospects. Hence, if the prediction is accurate, it can help to enhance the response rate of the advertising campaign and increase the return of investment (ROI). The direct marketing problem is similar to the classification problem. However, rather than producing a clear cut between potential buyers from non-buyers, the direct marketing problem requires ranking the customer database according to the likelihood of purchase [10]. Since Bayesian networks can estimate the posterior probability of an instance (a customer) belonging to a particular class (buyer or non-buyer), they are particularly suitable for handling the direct marketing problem.

5.1 THE DIRECT MARKETING PROBLEM

Direct marketing concerns communication with prospects hoping to eliciting response from them. In contrast to the mass marketing approach, direct marketing is often targeted on a group of individuals that are potential buyers and are likely to give a response.

In a typical scenario, we often have a huge list of potential prospects. This list could be records of existing customers or data bought from *list brokers*. But among the huge list, there are usually few real buyers which amount to only one or two percents [10]. Since the budget of a campaign is limited, it is important to target the effort on potential buyers so that the response rate could be improved.

With the advancement of computing and database technology, people seek for computational approaches to assist in decision making. From the data set that contains demographic details of customers, the objective is to develop a *response model* and use the model to predict potential buyers. In certain sense, response models are similar to classifiers in the classification problem. However, unlike a classifier which makes a dichotomous decision (i.e. active or non-active), a response model needs to score each customer in the data set with the likelihood of purchase. The customers are then ranked according to the score. A ranked list is desired because it allows decision makers to select the portion of customers to roll out to [11]. For instance, out of the 200,000 customers on the list, we might wish to send out catalogs or brochures to the most promising 20% of customers so that the advertising campaign is cost-effective. Hence, one way to evaluate a response model is to look at its performance at different *depth-of-file*. In the literature, there are various approaches proposed for building response models [12, 13, 14].

5.2 EXPERIMENT

Because Bayesian networks can estimate the probability of belonging to certain class(es), they are also suitable to handle the direct marketing problem. By assuming the estimated probability to be equal to the likelihood of purchase, a Bayesian network is readily applicable to the direct marketing problem. Thus, it is interesting to evaluate the empirical performance of Bayesian network response models. Specifically, we compare the performance of the models evolved by HEP with those obtained by MDLEP.

5.2.1 Experiment Methodology

The response models are evaluated on a real-life data set. The data set contains records of customers of a specialty catalog company, which mails catalogs to good customers on a regular basis. There is a total of 106,284 customers in the data set and each entry is described by 278 attributes.

Typical in any data mining process, it is necessary to reduce the dimension of the data set by selecting the attributes that are considered relevant and necessary. Towards this feature selection process, there are many possible options. For instance, we could use either a *wrapper* selection process or a *filter* selection process [15]. In a wrapper selection process, different combinations are iteratively tried and evaluated by building an actual model out of the selected attributes. In a filter selection process, certain evaluation function, which is based on information theory or statistics, is defined to score a particular combination of attributes. Then, the final combination is obtained in a search process. In this experiment, we have manually selected nine attributes, that are relevant to prediction, out of the 278 attributes.

To compare the performance of different response models, we use decile analysis which estimates the enhancement of the response rate for marketing at different depth-of-file. Essentially, the ranked list is equally divided into ten deciles. Customers in the first decile are the top ranked customers that are most likely to give response. On the other hand, customers in the tenth decile are ranked lowest. Then, a *gains table* is constructed to describe the performance of the response model. In a gains table, we tabulate various statistics at each decile, including [10]:

Percentage of Active The actual percentage of active respondents in the decile.

Lift Lift is calculated by dividing the percentage of active respondents by the response rate of the file. Intuitively, it estimates the enhancement by the response model in discriminating active respondents over a random approach for the current decile.

Cumulative Lift The cumulative lift is calculated by dividing the cumulative percentage of active respondents by the response rate. Intuitively, this evaluates how good the response model is for a given depth-of-file over a random approach.

5.2.2 Cross-Validation Result

To make a comparison concerning the robustness of the response models, we adopt a cross-validation approach for performance estimation. Specifically, we employ a 10-fold cross-validation where the ten folds are partitioned randomly. In Tables 2 and 3, the experimental results for the models evolved by MDLEP and those obtained by HEP are shown respectively. We collect the statistics on the percentage of active response, lift, and cumulative lift at each decile averaged over the ten runs. The numbers after the "±" sign are the standard deviations. Table 2 indicates the first two deciles have cumulative lifts of 377.2 and 287.4 respectively, suggesting that by mailing to the top two deciles alone, the MDLEP models generate over twice as many respondents as a random mailing without a model. However, the lift in the third decile declines sharply to 86.1, which is lower than the next decile (139.4). The lifts in the fifth, seventh, and tenth deciles decrease sharply to 3.2, 0.7, and 0.4, respectively. This phenomenon suggests instability in the models. The results in Table 3 show that the HEP models have a cumulative lift of 392.4 in the top decile, higher than that of the MDLEP models. The HEP models are more stable because there is no sudden drop in lifts in all deciles.

Decile	Percent Actives	Lift	Cum. Lift
1	20.39% ± 1.14%	377.2 ± 20.04	377.2 ± 20.04
2	10.71% ± 1.64%	197.4 ± 26.17	287.4 ± 6.90
3	4.67% ± 0.75%	86.1 ± 15.04	220.4 ± 2.84
4	7.53% ± 0.60%	139.4 ± 14.80	200.0 ± 4.57
5	0.18% ± 0.38%	3.2 ± 6.89	160.6 ± 3.37
6	5.31% ± 0.75%	97.7 ± 13.33	150.1 ± 2.47
7	0.04% ± 0.12%	0.7 ± 2.21	128.7 ± 2.11
8	2.81% ± 0.51%	51.2 ± 7.48	118.7 ± 0.95
9	2.34% ± 0.47%	42.9 ± 7.55	110.9 ± 0.32
10	0.03% ± 0.09%	0.4 ± 1.26	100.0 ± 0.00
Total	5.40% ± 0.22%		

Table 2: Result of the MDLEP model.

Since an advertising campaign often involves huge investment, a response model which can categorize more potential prospects into the target list, which amount

Decile	Percent Actives	Lift	Cum. Lift
1	21.21% ± 1.34%	392.4 ± 19.36	392.4 ± 19.36
2	9.88% ± 0.87%	182.3 ± 12.48	287.3 ± 6.18
3	5.69% ± 0.64%	105.1 ± 12.60	226.7 ± 6.15
4	4.84% ± 0.63%	89.3 ± 13.37	192.2 ± 3.94
5	3.47% ± 0.78%	63.8 ± 13.80	166.6 ± 3.17
6	2.96% ± 0.72%	54.1 ± 12.91	147.7 ± 1.34
7	2.07% ± 0.51%	37.8 ± 8.69	132.2 ± 0.92
8	1.58% ± 0.27%	28.6 ± 4.65	119.2 ± 1.03
9	1.38% ± 0.36%	25.0 ± 7.06	108.8 ± 0.42
10	0.92% ± 0.28%	16.6 ± 5.13	100.0 ± 0.00
Total	5.40% ± 0.22%		

Table 3: Result of the HEP model.

to the first few deciles, is useful as it will enhance the response rate. From the experimental results, it seems that the HEP models are more desirable than the MDLEP models.

6 CONCLUSION

In this paper, we have reported a hybrid framework for learning Bayesian networks and have incorporated the ideas into HEP. In addition, we have also devised a new operator and modified a weakness of MDLEP. As experimental results suggest, our new approach (HEP) is much more efficient than MDLEP. Most importantly, we note that the new approach provides a tremendous speedup.

We apply HEP to a data set of marketing and compare the models obtained by HEP and those produced by MDLEP. From the experimental results, the HEP models predict more accurately than the MDLEP models.

In our current implementation, we change the cutoff value of an offspring by arbitrarily increasing or decreasing a fixed value of Δ_α from the parent's value. However, it is also possible to use an adaptive mutation strategy such that the Δ_α will become smaller as time proceeds. In effect, the search space is gradually stabilized which may lead to a further speed up. In future, we will explore this and other alternatives that are worth investigating.

Acknowledgments

This research was partially supported by the RGC Earmarked Grant LU 3012/01E.

References

[1] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[2] F. V. Jensen, *An Introduction to Bayesian Network.* University of College London Press, 1996.

[3] J. Cheng, D. A. Bell, and W. Liu, "Learning Bayesian networks from data: An efficient approach based on information theory," in *Proceedings of the Sixth ACM International Conference on Information and Knowledge Management*, (Las Vegas, Nevada), pp. 325–331, ACM, November 1997.

[4] P. Spirtes, C. Glymour, and R. Scheines, *Causation, Prediction, and Search.* MA: MIT Press, second ed., 2000.

[5] E. Herskovits and G. Cooper, "A Bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, no. 4, pp. 309–347, 1992.

[6] W. Lam and F. Bacchus, "Learning Bayesian belief networks-an approach based on the MDL principle," *Computational Intelligence*, vol. 10, no. 4, pp. 269–293, 1994.

[7] D. Heckerman, "A tutorial on learning Bayesian networks," tech. rep., Microsoft Research, Advanced Technology Division, March 1995.

[8] M. L. Wong, W. Lam, and K. S. Leung, "Using evolutionary programming and minimum description length principle for data mining of Bayesian networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, pp. 174–178, February 1999.

[9] P. Larrañaga, C. Kuijpers, R. Mura, and Y. Yurramendi, "Structural learning of Bayesian network by genetic algorithms: A performance analysis of control parameters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, pp. 912–926, September 1996.

[10] O. P. Rud, *Data Mining Cookbook: modeling data for marketing, risk and customer relationship management.* New York: Wiley, 2001.

[11] J. Zahavi and N. Levin, "Issues and problems in applying neural computing to target marketing," *Journal of Direct Marketing*, vol. 11, no. 4, pp. 63–75, 1997.

[12] L. A. Petrison, R. C. Blattberg, and P. Wang, "Database marketing: Past present, and future," *Journal of Direct Marketing*, vol. 11, no. 4, pp. 109–125, 1997.

[13] S. Bhattacharyya, "Evolutionary algorithms in data mining: Multi-objective performance modeling for direct marketing," in *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, pp. 465–473, August 2000.

[14] C. X. Ling and C. Li, "Data mining for direct marketing: Problems and solutions," in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 73–79, 1998.

[15] M. Singh, *Learning Bayesian Networks for Solving Real-World Problems.* PhD thesis, University of Pennsylvania, 1998.