Computer Science and Systems Analysis

Computer Science and Systems Analysis Technical Reports

Miami University *Year* 1999

A Probabilistic Solution Generator of Good Enough Designs for Simulation

Mufit Ozden
Miami University

Yu-Chi Ho
Harvard University

This paper is posted at Scholarly Commons at Miami University.
http://sc.lib.muohio.edu/csa_techreports/1

MIAMI UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ANALYSIS

TECHNICAL REPORT: MU-SEAS-CSA-2000-001

A Probabilistic Solution Generator of Good Enough Designs for Simulation

Mufit Ozden and Yu-Chi Ho

School of Engineering & Applied Science | Oxford, Ohio 45056 | 513-529-5928

A Probabilistic Solution Generator of Good Enough Designs for Simulation

by
Mufit Ozden
Computer Science & Systems Analysis Department
Miami University, Oxford, Ohio 45056
Yu-Chi Ho
Division of Engineering and Applied Sciences
Harvard University, Cambridge, MA 02138

Working Paper #00-001 (January/2000)

A Probabilistic Solution Generator of Good Enough Designs for Simulation

Mufit Ozden
Department of Computer Science and Systems Analysis
Miami University
Oxford, OH 45056

Yu-Chi Ho
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138

Working Paper # 2000-001 **January 2000**

A Probabilistic Solution Generator of Good Enough Designs for Simulation

Mufit Ozden*
Department of Systems Analysis
Miami University
Oxford, OH 45056

Yu-Chi Ho**
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138

December, 1999

Abstract: We build a probabilistic solution generator using the learning automata theory, which can generate a small set of "good enough" designs with a predetermined high probability. The main goal of our work is to reduce a large design population to a much smaller subset of good designs that can be analyzed thoroughly in a subsequent simulation study to identify the best design among them. In the process of building the solution generator, a rough-cut design evaluation method with a high noise error is employed in order to screen designs very rapidly_ may it be an approximate method, a heuristic approach, or short simulation runs. The solution generator has been applied successfully to several serious test problems with noisy objectives.
Keywords: Simulation, Stochastic Optimization, Ordinal Optimization, Learning Automata Theory

1. Introduction

The main purpose of a simulation study is usually to analyze and improve the performance of a system under different scenarios defined by various operational and design decisions. In this paper, when we use the term "design" or "solution", we imply a feasible combination of decisions under study without distinguishing the type of decisions in question. But, the decisions that we consider here are discrete in nature. If we are willing to discretize continuous decisions however, they can also be handled within

---

* Corresponding author: e-mail: ozdenm@muohio.edu
** Yu-Chi Ho is supported in part by EPRI/ARO Contract WO833-03,NSF grant EEC 95-27422, Army contract DAAH04-94-0148, AFOSR contract 49620-98-1-0387, and ONR contract N00014-98-1-0720.

the context of this paper. In general, even for a simple system, the decision scenarios are too numerous to evaluate individually. A simulation model that represents all the important aspects of a real system is different from an optimization approach that guides the search for a better solution using a powerful optimization algorithm. Usually in a simulation study, the analyst starts cutting down the number of alternative designs to a handful promising ones using heuristic procedures, approximate analytical methods, or most likely than not, a pure human judgment. The analyst then estimates the performance of those designs by the simulation model in order to determine the best with a statistical analysis. One point that needs to be emphasized is that each simulation run is usually a slow, computationally intensive process if one is after a good estimate of the performance in the long run and needs to be repeated several times for each alternative design in order to produce a precise enough, overall average statistics.

In this paper, our goal is to produce "good enough" (or in short, "good") solutions rather than an optimal one quickly for a subsequent, more sophisticated simulation analysis. Solutions are considered to be good designs if they perform better than a predefined threshold value of the objective function. In formal mathematical terms, the problem is stated as follows:

$$\begin{aligned} &\min_{\theta \in \Theta} J(\theta), \quad \text{where} \\ &J(\theta) = E[\, L(\theta,\omega) \,], \quad \forall \theta \in \Theta, \end{aligned} \tag{1}$$

and Θ is a large discrete solution set of a finite size, L is a real-valued response function, which depends on the design $\theta \in \Theta$ and a random variable ω. Typically, the nature of $J(\theta)$ is not known explicitly by the problem solver. We can evaluate the function $L(\theta,\omega)$ for a given design by running a simulation model. Then, an estimate of $J(\theta)$ is computed from several independent replications of the simulation runs. We do not assume anything about the form of the objective function or the discrete solution set. Therefore, in general the problem (1) is extremely difficult to solve except for very small design sets.

When the design set is defined by continuous variables only, approximation methods and direct search techniques using calculus and gradient have been proposed for the solution of these problems. But in the discrete design domain, only a handful of global stochastic optimization methods have been proposed in the literature, for example, [Andradottir, 1996; Gutjahr and Pflug, 1996; Yan and Mukai, 1992; and Narendra and

Thathachar, 1989]. These methods guarantee the optimality of the searched solution only in an infinite number of steps and therefore, they usually require a long simulation run for convergence. Recently, we have seen marriage of the global optimization heuristics, such as Genetic Algorithms (Michalewicz, 1994), Simulation Annealing (Haddock and Mittenthal, 1992) , and Tabu Search (Glover et al. 1996) with the commercial simulation programs. These methods have basically been designed for hard-to-solve combinatorial problems and they handle the randomness in the response function by running a full simulation program for a given design several times. In practice, these approaches are more appropriate only for a small-scale problem. The ordinal optimization [Ho, Sreenivas, and Vakili, 1992; Ho, 1999], on the other hand, approaches the problem by relaxing the requirement of optimality with the "good enough" solutions and achieves such solutions with a crude performance evaluator at an exponential convergence rate through an ordinal comparison of designs [Dai 1996; and Lee, Lau, and Ho 1999].

In this paper, our preprocessing method borrows the concept "good enough" solutions from the ordinal optimization and uses an adaptive learning method of the learning automata theory to build a probability distribution concentrated over a small set of good designs within the entire population. In the following sections, after covering the background methods, we will describe the nature of the generator and then show theoretically how the probability of good designs increases in successive iterations. Then, on the basis of this theory, we will develop the formulas to determine the number of iterations required to achieve a predefined probability for the good designs. Capabilities of the generator will be demonstrated through some hard test problems.

2. Two Stochastic Optimization Methods

In this section, we will introduce two stochastic optimization approaches, which form the important ingredients of our method. We will first describe briefly the general approach and philosophy of the ordinal optimization. Then, we will delineate the adaptive learning procedure of the learning automata theory, which is used to build our probabilistic generator.

2.1 The Ordinal Optimization

As alluded before, the difficulty of problem (1) stems from two determinants. Optimization over a large set of discrete designs is typically a combinatorial type of problem, where each design is constructed as a feasible combination of many decisions. Furthermore, the random nature of the problem requires multiple evaluations through some form of experimentation for each design, which undermine the optimization algorithm to a greater extent.

The Ordinal Optimization approach confronts the first problem in two fronts. First, it softens the requirement of a global optimization with a subset of "good enough" solutions, which are defined as the top n% solutions. Thereby, it enlarges the target of optimization from a single point to a set of points. In practice, most people would be content with a better solution than the current practice if the system is already in existence, or with a design that falls anywhere in the top 1 or 5% of the solution population for a difficult design problem. Secondly, instead of searching the entire design population, it reduces design choices to a much smaller subset of representative ones by uniformly sampling the population. Once we settle for the top n% designs, this representative subset would also have the good enough designs at the same proportionality as in the original design population on the average.

The Ordinal Optimization approach confronts the second difficulty by ordering the alternative designs through a rough-cut performance evaluation procedure rather than determining the cardinal performance value for each and every design. It is a well known fact in the Statistics that if we are after constructing an interval on a mean performance value, the width of the confidence interval shrinks only at the rate of $1 / t^{0.5}$, where t is the number of average observations. However, if we are after ordering the alternative designs, it has been shown that the relative order of two designs can be determined at an exponential rate of samples, [Dai 1996].

Suppose that we designate a set of good designs with S_G and the reduced set of the selected designs from the population with S_S after a crude ordinal comparison of the alternative designs possibly in a high noise. Ho, et al. 1992 showed that the alignment probability of these two sets, defined as $P\{| S_G \cap S_S | > k \}$, can be made several orders of

magnitude larger than if we use only a random selection in forming S_S. In addition, the alignment probability can be quantified for different shapes of cost functions and it increases exponentially with respect to the size of S_S, [Lau and Ho, 1997; Lee, Lau, and Ho, 1999]. The general approach of the Ordinal Optimization can be summarized as in Figure 1.

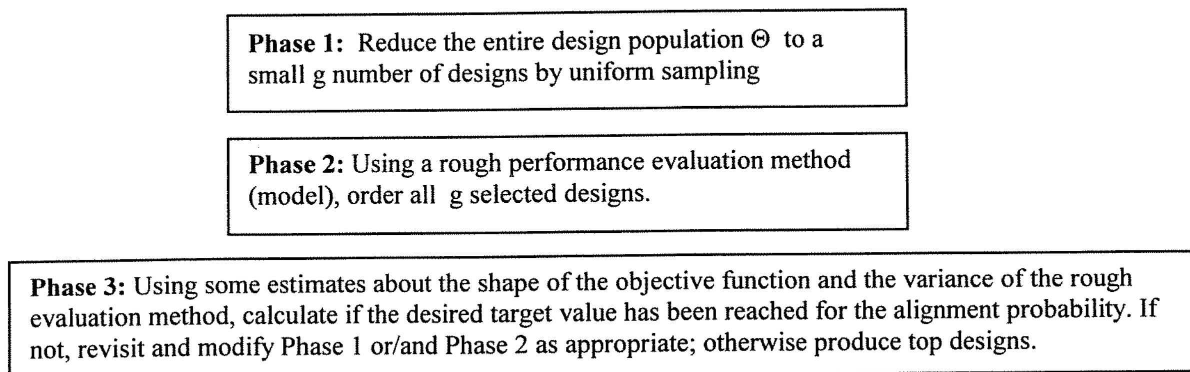


Figure 1: Phases of the Ordinal Optimization approach.

2.2 The Learning Automata Theory

The mathematical treatment of the famous "carrot and stick" learning paradigm in the control theory goes back to the early work by Tsetlin, 1962. The developments in learning automata theory over the intervening three decades is explained thoroughly in a book by Narendra and Thathacher, 1989.

Many different types of learning automata have been developed based on this principle of adaptive learning with different norms of behavior (convergence). Among these, the two most sought after ones are the "absolutely expedient" and "epsilon expedient" behavior. In common terms, the absolutely expedient automata achieve a strict improvement in its expected cost in any two successive iterations whereas the epsilon expedient one comes within an arbitrarily small distance of the global optimum w.p.1 in the long run. In all research related to this type of learning, the goal of the automata is to obtain the global optimum. The theory supports the convergence only in infinite steps. The speed of learning, adjusted by a constant, posses a serious challenge. Unfortunately, the dilemma exists between achieving the optimum expeditiously with greater chance of getting stuck in the local optima and going gradually toward the

optimum at the expense of a long computation time. The first author of this paper incorporated the learning automata as a general optimizer in a simulation environment in an earlier work, [Ozden, 1994].

More formally, a variable structure automaton is defined with the triple $\{\theta, \beta, A\}$ for designs θ, (normalized) performance evaluations β, and a learning algorithm A, as shown in Figure 2. When the automaton selects a design θ_i and applies it in a random environment in which the automaton works, it receives a performance evaluation (or, a feed-back) β_i from the environment. On the basis of this evaluation, the automaton modifies its design selection process according to its learning algorithm.

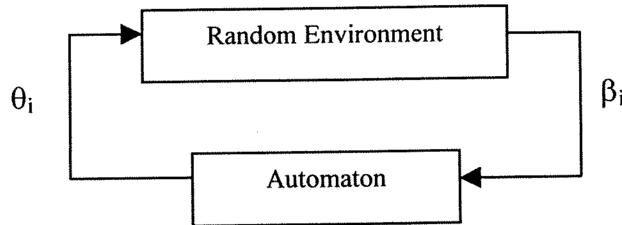


Figure 2. A learning automaton and its interaction with its random environment.

The design selection process is directed with the action distribution p defined on the design set θ and the adaptive learning takes place in this distribution. Initially, the distribution is constructed uniformly giving equal probability to every design. In the case of an automaton using the specific learning algorithm known as the "reward and inaction" (L_{R-I} automaton), which we are going to integrate in our work, the probability associated with the current design is increased in a small amount if the performance evaluation received from the environment turns out to be positive (reward) while the probabilities of the other designs are reduced proportionally. No change takes place in the distribution in the case that the evaluation turns out to be negative (inaction). The next design is again selected by sampling the updated distribution for the next iteration. The iteration continues until one of the designs accumulates a very high probability close to 1. Hence, at iteration *n*, the learning process is a mapping defined as

$$p(n+1) = T(\theta_i,\ \beta_i,\ p(n)) \qquad (2)$$

If we assume that the performance values received from the random environment come only in two flavors, as "positive" (or 1) and "negative" (or 0) and assume a small positive so-called learning constant a, the mapping (2) for the $L_{R\text{-}I}$ automaton can be viewed in more explicit terms as

$$\text{If } \theta \text{ is } \theta_i \text{ at the iteration } n, \text{ then} \qquad (3)$$

$$p_i(n+1) = p_i(n) + a\,[1 - p_i(n)] \qquad \text{if } \beta_i \text{ is } 1$$

$$p_j(n+1) = (1-a)\,p_j(n) \qquad \forall j \text{ except } i$$

$$\text{and} \quad p_j(n+1) = p_j(n)\,, \qquad \text{otherwise } \forall j.$$

Given this probability vector $P(n)$ whose ith component is $p_i(n)$, the process $\{P(n) : n \geq 0\}$ describes a Markov process. The long-term convergence properties of the learning algorithm can be derived using this Markov property. The $L_{R\text{-}I}$ automaton using (3) as the learning algorithm has been proven to be absolutely expedient, (Narendra and Thathacher, 1989).

3. The Probabilistic Generator for Good Designs

In the Phase 1 of the original ordinal optimization of Figure 1, we reduce the entire design population into a small set of g number of designs through the uniform sampling. For a large problem, the entire design population is extremely large, in the order of billions or more while the size g of the reduced set of designs is usually in the order of 10^3. After uniform sampling, the top $n\%$ of the design population would be found at the same percentage level in the reduced set on the average. Hence in the entire population of say, 10^9 designs, if we want to select the top 1% of the designs, this means we will have 10^7 good designs to select from. After uniform 1000 sampling, we will have only10 such designs, on the average, in the reduced set of designs. We would like to increase this proportion of the good designs considerably in the reduced set if it all is possible.

In this paper, we will define the good designs on the basis of a threshold cost value of the objective function that the good designs must pass instead of a top percentage number. An advantage of this change is that in the practice, it is more meaningful for the managers to set a threshold cost value for the acceptable designs than specifying a percentage number. In this way, we know exactly what least performance we expect from the good designs. Hence,

$$S_G = \{\ \theta\ :\ J(\theta) < T,\ \ \forall\theta \in \Theta\} \tag{4}$$

,where T is a predefined threshold value. The Figure 2 shows good designs in an example of a linear cost function with the proportional error terms.

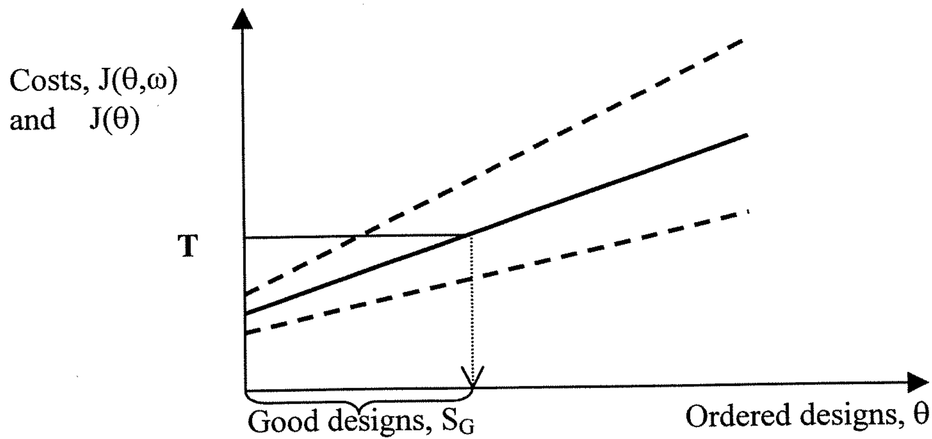


Figure 3. The set of the good designs for a linear cost function with proportional noisy error.

Now, let us construct a linear automaton of type $L_{R\text{-}I}$ to extract a subset of good designs from the entire design population. We designate the performance evaluation β as an indicator function that gives only two possible values for a design $\theta_{i,}$, selected from the design distribution randomly by the automaton. If the observed cost value for the design θ_i is less than the threshold value T, the value β is set to *1* (a good design) and otherwise, to *0* (a bad design). At every iteration *n*, the automaton selects a design randomly using its current distribution *P(n)* and then gets this design evaluated by the environment. With the feedback from the environment, the automaton then updates the distribution according to the equations (3) and generates another design for the next

iteration $n+1$. After a predefined finite number of iterations, the final design distribution of the automaton becomes our design generator and can be used to generate any number of good designs. The formal steps of the learning process are as follows:

Algorithm:

Step 0: Set

a. The current stage value n to 1;
b. The final stage number as N;
c. The $p_i(n) = 1/r$ for all r number of possible designs;
d. The learning constant a to a small positive constant;
e. The threshold value as T.

Step 1: Generate a design θ_i using the design distribution $P(n)$ for the current stage n.

Step 2: Using a rough method, observe the performance measure for the selected design, θ_i. Say, the observed performance value (cost) is $L(\theta_i)$. Then,

if $L(\theta_i) < T$, set $\beta_i = 1$;
otherwise, set $\beta_i = 0$.

Step 3: Update the design distribution $P(n)$ according to (3). That is,

$$p_i(n+1) = p_i(n) + a\,[1 - p_i(n)] \qquad \text{if } \beta_i \text{ is } 1$$
$$p_j(n+1) = (1-a)\,p_j(n) \qquad \forall j \text{ except } i$$
$$\text{and} \quad p_j(n+1) = p_j(n)\,, \qquad \text{otherwise } \forall j.$$

Step 4: If n is equal to N value, then stop; the generator is $P(n)$. Otherwise, increment n by 1 and go to Step 1.

In the following section, we prove that the distribution of the automaton $L_{R\text{-}I}$ modified according to the above algorithm will increase the probability for the good designs strictly at every iteration. We will also determine how many iterations on the average it will take to obtain a generator with a capability of producing the good designs at a specified probability level.

4. Convergence of the Method in Finite Steps

The learning scheme described in the previous section is known to be absolutely expedient. Now, what we would like to know is how the expected total probability of the good designs changes from iteration to iteration. And how many iteration in total will it

take to get to a predefined probability for the good designs. We will show these results in terms of a theorem and a corollary that follows it.

Theorem: *The learning algorithm described in the above algorithm increases the expected probability of the good designs strictly in any two successive iterations.*

Proof: Suppose that the probability of a good design θ_i failing to pass the threshold value T is c_i and the probability of a bad design θ_j failing to pass the threshold value T is c_j. Then, by definition, $c_i < c_j$ for every pair $i \in S_G$ and $j \in S_B$. In general terms, define

$$c_k = P\{ J(\theta_k) < T \} \quad \text{for each } \theta_k.$$

The expected change in the probability of the good design θ_i is

$$\Delta p_i(n) = E[p_i(n+1) \mid p(n)] - p_i(n)$$

For the automata $L_{R\text{-}I}$ considered here, as Narendra and Thathachar (1989) shows, for any good design θ_i among total r number of designs, the change of expected probabiltiy will be

$$\Delta p_i(n) = a p_i(n) \sum_{j \neq i}^{r} p_j(n)(c_j - c_i) \qquad \text{(Appendix A)}$$

We separate the summation into two parts _ one over bad designs and the other over the good ones.

$$\Delta p_i(n) = a p_i(n)\left[\sum_{j \in S_B} p_j(n)(c_j - c_i) + \sum_{j \in S_G} p_j(n)(c_j - c_i)\right]$$
$$= a p_i(n) \sum_{j \in S_B} p_j(n)(c_j - c_i) + a p_i(n) \sum_{j \in S_G} p_j(n)(c_j - c_i)$$

Then, by adding up Δp_i 's for the good designs, we can write the total probability change for the entire good design set as

$$\Delta p_G(n) = \sum_{i \in S_G} a p_i(n) \sum_{j \in S_B} p_j(n)(c_j - c_i) + \sum_{i \in S_G} a p_i(n) \sum_{j \in S_G} p_j(n)(c_j - c_i)$$

Note that the second term is zero since the last double sum will have exactly two terms for each pair of good designs, say *k* and *l* as

$$a p_k(n) p_l(n)(c_l - c_k) + a p_l(n) p_k(n)(c_k - c_l) = 0.$$

Hence,
$$\Delta p_G(n) = \sum_{i \in S_G} a p_i(n) \sum_{j \in S_B} p_j(n)(c_j - c_i) > 0,$$
$$\text{since } (c_j - c_i) > 0 \;\; \forall i \in S_G \text{ and } \forall j \in S_B \qquad \otimes$$

Assumption 1: Assume that all good designs have the same failure probability, i.e., $c_G = P\{ J(\theta_i, \omega) < T : \omega \in \Omega\}$ for all $\theta_i \in S_G$ and the threshold value T. And all bad designs also have the same failure probability, $c_B = P\{ J(\theta_j, \omega) < T : \omega \in \Omega\}$ for all $\theta_j \in S_B$ and the threshold value T.

Corollary: *Under the assumption 1, suppose that we have the total probability of the good designs as P_0 in the design population. After n number of iterations the probability of the good designs will be*

$$p_G(n+1) = a(c_B - c_G)p_G(n)[1 - p_G(n)]. \tag{4}$$

For a given target probability P_G for the good designs, the average number of the iterations required in the above learning scheme is approximately

$$n = \ln(P_G/P_0) \Big/ \ln(1 + a\bar{c}) \qquad \text{where } \bar{c} = c_B - c_G. \tag{5}$$

Proof:

From the theorem, we know that the total probability change for the good designs

$$\Delta p_G(n) = \sum_{i \in S_G} a p_i(n) \sum_{j \in S_B} p_j(n)(c_j - c_i)$$

Substituting c_B for c_j $\forall j \in S_B$ and c_G for c_i $\forall i \in S_G$, we obtain

$$\Delta p_G(n) = a(c_B - c_G) \sum_{i \in S_G} p_i(n) \sum_{j \in S_B} p_j(n)$$

Hence, $$\Delta p_G(n) = a(c_B - c_G)p_G(1 - p_G). \qquad \otimes$$

If we ignore the second order terms, then

$$\Delta p_G(n) = a\bar{c}p_G(n), \qquad \text{where } \bar{c} = (c_B - c_G)$$

and $$p_G(n+1) = p_G(n) + a\bar{c}p_G(n) = (1 + a\bar{c}p_G(n)$$

$$p_G(1) = (1 + a\bar{c})p_0, \; p_G(2) = (1 + a\bar{c})^2 p_0, \ldots$$

Hence, in general, $p_G(n) = (1 + a\bar{c})^n p_0$.

If p_0, a, and $\bar{c}$ are known, then

$$n = \ln(p_G/p_0) \Big/ \ln(1 + a\bar{c}) \qquad \otimes$$

As we will see in the test problems, the approximation in the corollary is fairly accurate for the final target probabilities $p_G < 0.20$. It becomes more significant for larger probability values. Nevertheless, one can always use the recursive formula (4) to find out how many steps it will take to raise the probability to a target $p_G(n)$ starting from $p_G(0) = p_0$.

5. Application of the Solution Generator

We set up four different problems whose natures are known to us completely so that we can track the performance of our procedure. However, in the real world problems, we hardly know the complete nature of the environment that responds to designs generated by the automata. In all of these applications, we have used a spread-sheet programming that required very little computation time to solve.

The first three problems involve a population of 1000 feasible designs. In the first problem, the expected cost of designs is a linear function. But each observation of the design performance has a significant noise, which has a constant amplitude. In the second problem, we will assume a negative exponential expected cost with significant proportional noise values. This problem has a very few good designs relative to the design population. The third problem is a harder version of the second problem in that it has many deep local minima superimposed by a Sin-function on the exponential objective function of the problem 2. The fourth problem has a high polynomial expected cost function with proportional noise and much larger design population (about 4 million).

None of these examples comply with Assumption 1 strictly. But knowing the nature of the cost functions of the first three test problems, we still want to use the Corollary with the average failure rates to see how it predicts the total number of iterations approximately. As we will see, it turned out that the results of these predictions are very good. We used the areas above and below the threshold value between the maximum and minimum cost errors to estimate the failure probabilities of the good and bad designs. For the final probabilities of the good designs less than 0.25, the approximate formula (5) was sufficient, but for larger values closer to 0.50, the recursive formula (4) was much reliable

5.1 Linear Expected Cost Function with Constant Error

Here, we consider a design problem of 1000 alternative designs. As shown in Figure 4, the cost function for a design j is given by

$$L(\theta_j, \omega) = j + 100r,\ j = 1,2,...1000$$

,where r is a random number in the interval [0,1).

For the treatment of this problem with our learning scheme, we assume a threshold value T=150. The failure probabilities of the good designs are not constant in this problem, but we will still use the average value in the formulas of the Corollary in order to estimate the required number of iterations. Since we know exactly the behavior of the cost function and the noise, we can calculate the failure probability of the good designs approximately. Among the good designs, 100 of them may fail with the probabilities changing linearly between 0 and 1. Hence, on the average, the failure probability c_G= (100(1.0+0.0)/2+50(0.0)/150= 0.33. The failure probability of the bad designs for this threshold is 1. The initial probability of the 150 good designs has the value of 0.15. Using the approximate formula (5) and the recursive formula (4) of the Corollary, we can estimate the number of iterations that will take to increase this probability to various levels. Figure 5 shows the results. Note that for the target probabilities less than 0.25, the approximate and recursive results are fairly close and they separate dramatically for the large probabilities. Of course, the recursive formula is the more accurate one between the two and is not much harder to evaluate.

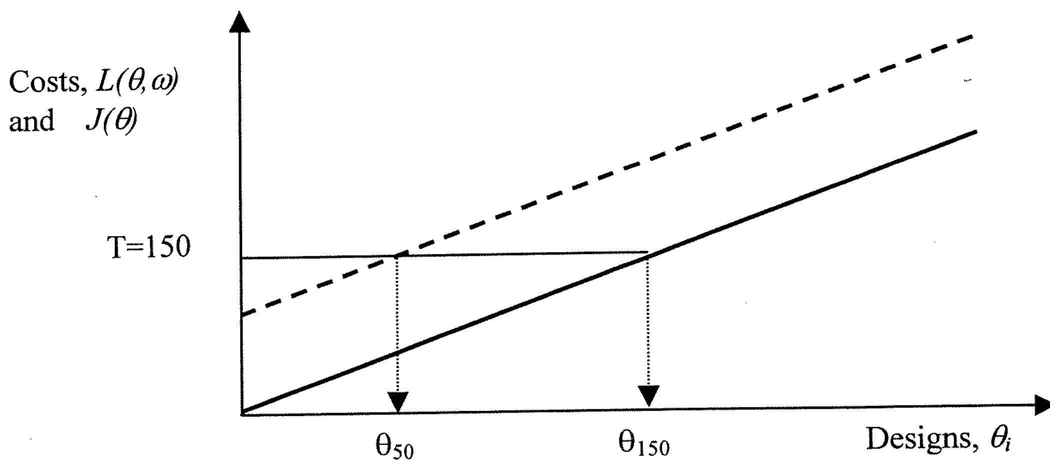


Figure 4. Cost function and threshold value for the test problem 1.

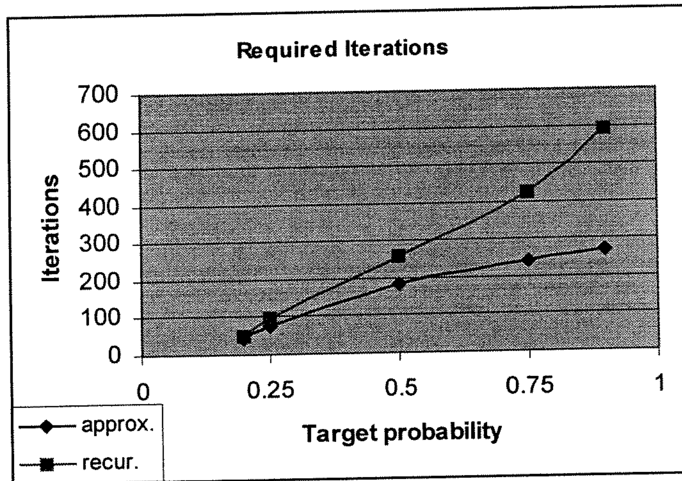


Figure 5. The required number of iterations for various target probabilities through the approximate and recursive formulas of the corollary.

We have performed two sets of 10 independent runs as follows: to increase the initial probability 0.15 of the good designs to the target probability of 0.50; and to increase the initial probability 0.15 of the good designs to 0.90. The number of iterations are calculated from formula (4) were 250 and 600, respectively. In all these cases, the learning constant a is set to 0.01. The results in terms of the cumulative probability values of the generator for the top 50 and 150 designs are shown in Table 1 and 2.

Experiment	1	2	3	4	5	6	7	8	9	10	Mean	Std. D
F(150)	0.46	0.56	0.50	0.62	0.58	0.57	0.58	0.53	0.43	0.58	0.54	0.06
F(50)	0.26	0.43	0.27	0.29	0.36	0.32	0.43	0.24	0.19	0.37	0.32	0.08

Table 1. The final cumulative probabilities of good designs for 10 independent runs where the target probability p_G of the good designs is 0.50 and the number of iterations is 250.

Experiment	1	2	3	4	5	6	7	8	9	10	Mean	Std. D
F(150)	0.98	0.97	0.96	0.96	0.98	0.97	0.98	0.95	0.96	0.95	0.96	0.011
F(50)	0.68	0.69	0.70	0.73	0.77	0.76	0.79	0.33	0.73	0.20	0.64	0.20

Table 2. The final cumulative probabilities of good designs for 10 independent runs where the target probability p_G of the good designs is 0.90 and the number of iterations is 600.

In this relatively simple problem, we observe that we do well for the solution generator with very little effort. Even though the evaluations of the cost values received for every selected design from the environment are in a huge error (a very rough model at best), it took a small number of iterations to increase the probability of the good designs considerably.

Figure 6 shows a path that the probabilities of the good designs follow through the iteration of the learning in a typical case.

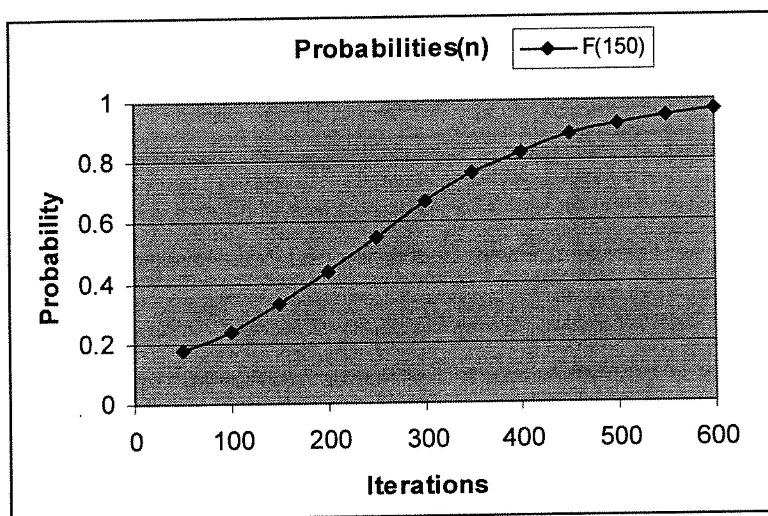


Figure 6. A typical path of the probabilities of good designs.

5.2 Exponential Cost with Proportional Errors

In this test problem, we have 1000 alternative designs with the cost functions defined as follows:

$$L(\theta_j) = 100 - 40.5\,e^{(-0.1j+1)},$$

$$L(\theta_j,\omega) = L(\theta_j) \pm 0.25r\,L(\theta_j), \text{ where } r \text{ is a random number and } j = 1,2,...1000$$

These functions are shown in Figure 7. We define the good designs as the any design that passes the threshold value T=70 in 50% of the time. They correspond to the first 13 designs with initial probability of P_0=0.013. Note that the first 9 designs pass the

threshold 100% of the time and the first 29 designs pass the threshold with some positive probability. The remaining 971 designs fail the threshold value all the time.

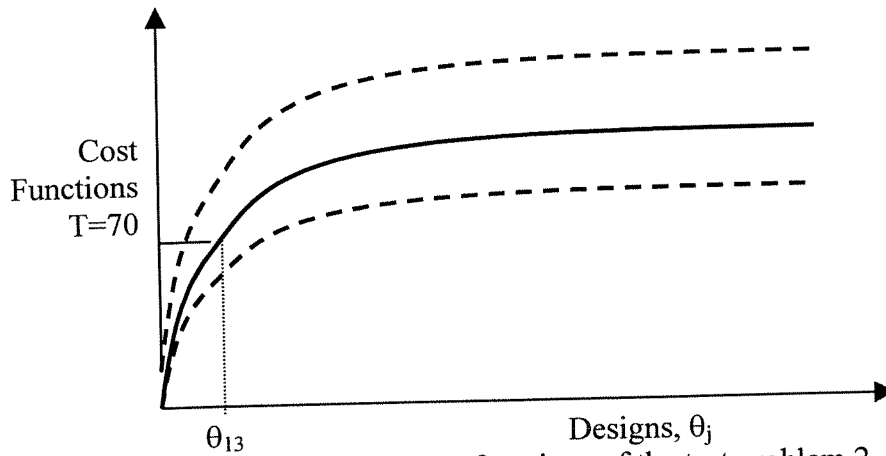


Figure 7. Exponential cost functions of the test problem 2.

We used the formulas of the Corollary to determine the number of iterations required to increase the probability of the first 13 designs to 0.15 in the generator. Using the average, minimum and maximum cost curves for the good and bad designs, we estimated the failure probabilities as c_G= 0.08 and c_B = 0.99. Using a learning coefficient value a =0.01, the number of iterations required turned out to be 300 and 475 for the target probabilities 0.15 and 0.5, respectively. In order to reduce the standard deviations of the probabilities, we then reduced the learning coefficient by a half (a=0.005). The number of iterations required has increased to 570 and 950 for the target probabilities 0.15 and 0.5, respectively. Table 3 summarizes the results of both cases.

Experiment	1	2	3	4	5	6	7	8	9	10	Mean	Std.Dv.
a=0.01,n=300	0.01	0.05	0.31	0.04	0.20	0.13	0.11	0.18	0.02	0.20	0.15	0.10
a=0.01,n=474	0.08	0.10	0.23	0.71	0.68	0.22	0.19	0.45	0.37	0.18	0.32	0.23
a=0.005,n=570	0.30	0.20	0.21	0.16	0.07	0.22	0.13	0.08	0.13	0.20	0.17	0.07
a=0.005,n=950	0.61	0.63	0.42	0.19	0.26	0.54	0.65	0.73	0.51	0.18	0.47	0.20

Table 3. The probabilities of the good designs after n iterations for Problem 2.

Due to the nature of the cost function and the threshold value selected in this problem, we had very small set of good designs (only 13 designs) and 4 of these good designs had a positive probability to fail the threshold due to an error in the estimation. On the other hand, relatively small percentage of the bad designs (15) had a chance to pass the threshold value. Because of this unusual nature of the problem, the target probability of our learning scheme has a large variation around the mean value identified by the Corollary, especially for the learning constant 0.01. We see that this variation can be reduced at the expense of a larger number of iterations by decreasing the learning constant's value.

5.3 Superimposed Exponential and Sin Cost Function with Proportional Errors

In this test problem, we have 1000 alternative designs whose expected and actual cost functions are defined as follows:

$$v = 100 - 40.5\, e^{(-0.1j + 1)}$$

$$L(\theta_j) = v[1 + 0.5\, Sin(v/5)]$$

$$L(\theta_j, \omega) = L(\theta_j) \pm 0.25 r\, L(\theta_j),\ \text{where r is a random number and } j = 1,2,...1000$$

These functions are shown in Figure 8. Again, we define the good designs as the any design that passes the threshold value T=70 in 50% of the time. This corresponds to two mutually exclusive sets: the first set being the first 11 designs with initial probability of 0.011 and second set being the 9 designs starting at the index number 17 with the initial probability 0.009.

In this problem, our main is to investigate how the local optima will affect the overall performance of the algorithm in terms of speed and accuracy. The proportion of the good to the bad designs is more or less the same as in the previous problem. In order to be able to compare the results of experiments to ones obtained form the previous test problem, we used exactly the same threshold value 70 and the same number of iterations with two different values of the learning constant. Table 4 displays the results of the experiment.

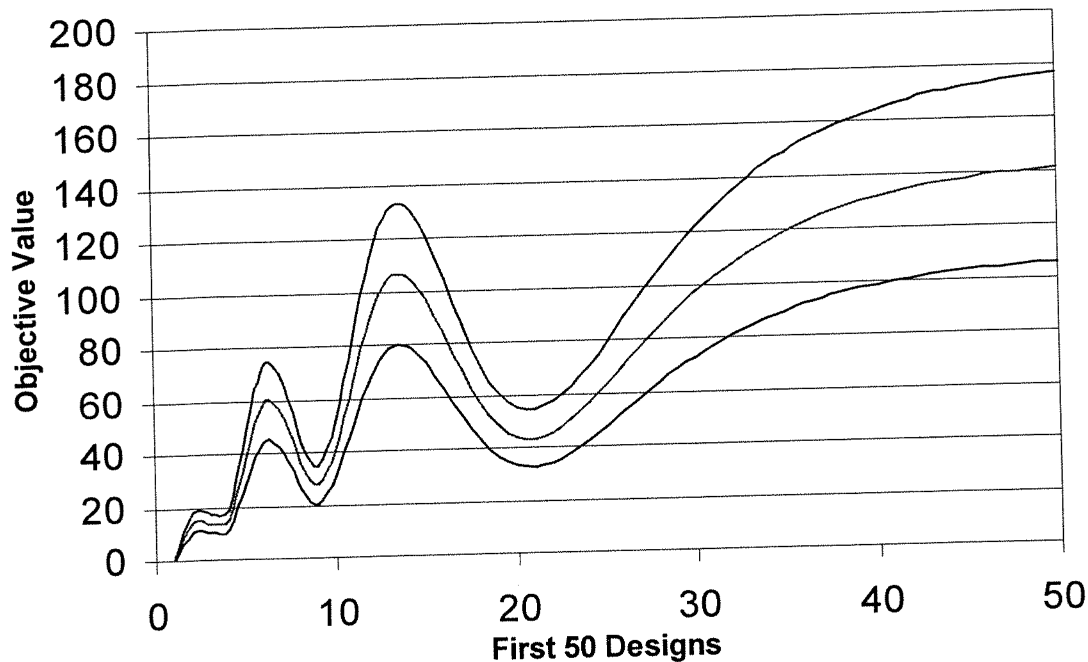


Figure 8. Expected cost function and error bands for the first 50 designs of the problem 3.

Experiment	1	2	3	4	5	6	7	8	9	10	Mean	Std.Dv.
a=0.01,n=300	0.13	0.30	0.05	0.10	0.30	0.30	0.47	0.21	0.21	0.17	0.22	0.13
a=0.01,n=474	0.62	0.81	0.54	0.17	0.62	0.53	0.27	0.61	0.18	0.47	0.48	0.21
a=0.005,n=570	0.23	0.32	0.18	0.21	0.22	0.16	0.10	0.21	0.24	0.30	0.22	0.06
a=0.005,n=950	0.48	0.75	0.64	0.59	0.58	0.70	0.46	0.57	0.44	0.43	0.56	0.11

Table 4. Probabilities of the good designs after n iterations for Problem 3.

By comparing the summary statistics in the last two columns of Table 3 and 4, we find that the average performance has suffered somewhat due to the undulations embedded in the cost function of the problem 3. But the target probabilities for the good designs are still within the one standard deviation from the average values. Again, we observe that a reduced learning coefficient results in less variation in the good probabilities at the expense of increased number of iterations. The good news is that the local optima do not cause any serious degradation in the performance of the algorithm. We will investigate this point further in the next problem.

5.4 Nonlinear Cost Function of Multiple Decisions with Proportional Errors

Here, we used a modified version of a test problem, which was also employed in the genetic algorithms, Michalewicz, 1994. The expected cost function is defined over designs θ, each of which is a feasible combination of five decision variables. Hence, there are 4,084,101 alternative designs and maximum F(X) value is 2,316,082 and the optimal solution F(X)=0 is reached when all variables are assigned the value 1. The cost functions are as follows:

$$F(X)= 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1) + 30(x_1 - x_5)^2$$

$$\text{st}: \quad -10 \le x_i \le 10 , \ \text{for } i = 1,2,\ldots,5$$

And

$$J(\theta,\omega) = F(X) (1 \pm 0.75 r), \quad \text{where r is a random number.}$$

The statistics about the expected cost function for some good designs are displayed in Table 4.

F(X) < Y	1,000	2,000	5,000	10,000
Design number	6,056	21,587	91,889	223,150
Percentage	0.0015	0.0053	0.0225	0.0546

Table 4. Distributions of some good designs with respect to values of the expected cost function.

This problem is quite different from the previous two test problems in that each design has five components as the decision variables resulting in a huge feasible design space. Here, we use a separate automaton for each component of the design. Each automaton starts with a uniform distribution defined over 21 different values of each decision variable. When a feedback value is received from the environment, every automaton

updates its own distribution using the same feedback value in the learning algorithm. Our goal is to obtain the probability 0.50 for the good designs identified by the threshold value of 5000 in the generator. Using similar parameters ($a=0.01$, $c_B= 1$, and $c_G=0.8$) as in the previous problem, we estimated the iteration number as 1500. After completing the experiments, we calculated the cumulative probabilities for several top designs values by exhaustively enumerating all alternatives for the entire design population with the final distributions of the automata. We have performed five independent runs. Table 5 summarizes the results for these runs.

	P(Cost≤Y)	1,000	2,000	5,000	10,000	0(optimal)
EXPER. n	1	0.102	0.229	0.508	0.710	0.000033
	2	0.09	0.235	0.567	0.778	0.000008
	3	0.09	0.243	0.579	0.789	0.00002
	4	0.08	0.220	0.570	0.771	0.000006
	5	0.127	0.306	0.713	0.901	0.000036

Table 5. The final cumulative probabilities of designs whose costs are below some Y values for five independent experiments.

The results of this last problem have surprised us pleasantly because this is a problem with the largest design population, and a nonlinear noisy cost function with decisions in a high interaction. But still we were able to increase the probability of good designs whose average cost values are less than 5000 about 25 folds in short 1500 iterations. We believe that the reason behind this performance is that the five different automata were able to capture the structure of the problem better than it would have been possible with a single automaton built for the entire design space. In this way, any combination of decision values that have worked well together in a solution and therefore got awarded, stood a good chance of being selected together again and to work well or even better in some other combination of the decisions. In other words, these smaller combinations of decisions form the building blocks for the larger, better solutions. And the multiple automata used in this problem were able to capture the inner workings of the decisions in the objective function well.

6. Conclusion

In this paper, we developed a probabilistic generator, which is capable of producing the good enough solutions with a desired probability in an uncertain environment. The good solutions are defined as the subset of solutions whose average objective values are better than a predefined threshold value. We built our method on the basis of a theory that guarantees that the expected probabilities of the good enough solutions will be improved strictly even with a very noisy estimation of the objective function values in every iteration. Using this theory, we determine how many iterations it will take to increase the probabilities of the good designs to a desired level on the average.

We applied our approach to four challenging test problems. The results have conformed perfectly to the theory and performed beyond our expectations. All calculations have been executed with an insignificant computation time on a laptop computer.

Our approach confirms the insights of many researchers who believe in probabilistic learning, goal softening, and ordinal evaluation of the alternative solutions in lieu of searching for a good (not necessarily optimal) solution in a noisy environment. In conclusion, our approach holds a great promise as an integrated stochastic "optimizer" in simulation models. In the future, we will investigate how the short simulation runs can serve as a rough-cut performance evaluator for building the solution generator.

References

Andradottir, S., 1996. A global search method for discrete stochastic optimization, *SIAM Journal on Optimization* 6.

Dai, L. Y., 1996. Convergence properties of ordinal comparison in the simulation of discrete event dynamic systems, *Journal of Optimization Theory and Applications* 91.

Glover, F., Kelly, J. P., Laguna, M. 1996. New advances and applications of combining simulation and optimization, Proceedings of Winter Simulation Conference.

Gutjahr, W. J., Pflug, G. Ch. 1996. Simulated annealing for noisy cost functions. *Journal of Global Optimization* 8.

Haddock, J., Mittenthal, J. 1992. Simulation optimization using simulated annealing, Computers and Industrial Engineering 22.

Ho, Y. C., Sreenivas, R., Vakili, P. 1992. Ordinal optimization of discrete event dynamic systems. *Journal of Discrete Event Dynamic Systems* 2.

Ho, Y. C. 1999. An explanation of ordinal optimization: soft computing for hard problems. *Int. Journal of Information Sciences* 113.

Lacksonen, T., Anussornnitisarn, P. 1995. Empirical comparison of discrete event simulation optimization techniques, Proceedings of the SCSC95, Ottowa, Canada.

Lau, T.W.E., Ho, Y.C. 1997. Universal alignment probabilities and subset selection for ordinal optimization, JOTA 39.

Lee, L. H., Lau, T.W.E., Ho, Y.C. 1999. Explanation of Goal Softening in Ordinal Optimization, IEEE Trans. On Automatic Control 44.

Michalewicz, Z. 1994. Genetic Algorithms + Data Structures = Evolution Programs, 2nd Ed, Springer-Verlag.

Narendra, K., Thathachar, M. A. L. 1989. Learning automata an Introduction, Prentice Hall.

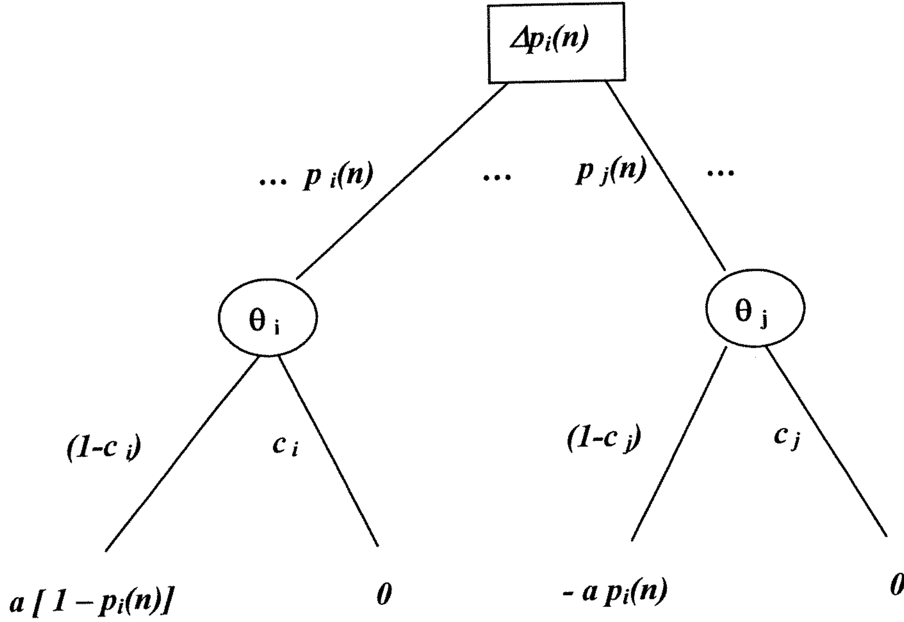
Ozden, M. 1994. Intelligent objects in simulation, *ORSA Journal on Computing* 5.

Tsetlin, M. L. 1962. On the behavior of finite automata in random media. *Automation and Remote Control* 22.

Yan, D., Mukai, H. 1992. Stochastic discrete optimization. *SIAM Journal on Control and Optimization* 30.

APPENDIX A

The change in the probability of a good design θ_i in iteration n can be visualized as in the following figure:



By definition, the change in the probability of the design θ_i in iteration n is

$$\Delta p_i(n) = E[p_i(n+1) \mid P(n)] - p_i(n)$$

$$\Delta p_i(n) = p_i(n)(1-c_i)a[1-p_i(n)] + \sum_{j \neq i} p_j(n)(1-c_j)[-ap_i(n)]$$

$$\Delta p_i(n) = a\, p_i(n)\{(1-c_i)[1-p_i(n)] - \sum_{j \neq i} p_j(n)(1-c_j)\}$$

$$\Delta p_i(n) = a\, p_i(n)[(1-c_i)\sum_{j \neq i} p_j(n) - \sum_{j \neq i} p_j(n)(1-c_j)]$$

$$\Delta p_i(n) = a\, p_i(n)[\sum_{j \neq i} p_j(n) - \sum_{j \neq i} p_j(n)c_i - \sum_{j \neq i} p_j(n) + \sum_{j \neq i} p_j(n)c_j]$$

$$\Delta p_i(n) = a\, p_i(n)[-\sum_{j \neq i} p_j(n)c_i + \sum_{j \neq i} p_j(n)c_j]$$

$$\Delta p_i(n) = a\, p_i(n)\sum_{j \neq i} p_j(n)(c_j - c_i) \qquad \otimes$$