Clean Techn Environ Policy (2006) 8: 85–95
DOI 10.1007/s10098-006-0036-9

ORIGINAL PAPER

Elmer P. Dadios · Jamshaid Ashraf

Genetic algorithm with adaptive and dynamic penalty functions for the selection of cleaner production measures: a constrained optimization problem

Received: 2 July 2005 / Accepted: 5 December 2005 / Published online: 8 April 2006

Abstract This paper presents a new approach of genetic algorithm (GA) to solve the constrained optimization problem. In a constrained optimization problem, feasible and infeasible regions occupy the search space. The infeasible regions consist of the solutions that violate the constraint. Oftentimes classical genetic operators generate infeasible or invalid chromosomes. This situation takes a turn for the worse when infeasible chromosomes alone occupy the whole population. To address this problem, dynamic and adaptive penalty functions are proposed for the GA search process. This is a novel strategy because it will attempt to transform the constrained problem into an unconstrained problem by penalizing the GA fitness function dynamically and adaptively. New equations describing these functions are presented and tested. The effects of the proposed functions developed have been investigated and tested using different GA parameters such as mutation and crossover. Comparisons of the performance of the proposed adaptive and dynamic penalty functions with traditional static penalty functions are presented. The result from the experiments show that the proposed functions developed are more accurate, efficient, robust and easy to implement. The algorithms developed in this research can be applied to evaluate environmental impacts from process operations.

E. P. Dadios (✉)
Department of Manufacturing Engineering and Management,
De La Salle University, 2401 Taft Avenue,
Manila 1004, Philippines
E-mail: dadiose@dlsu.edu.ph

J. Ashraf
College of Computer Studies, De La Salle University,
2401 Taft Avenue, Manila 1004, Philippines

Introduction

Genetic Algorithms (GAs) and other evolutionary algorithms (EAs) are innovative and novel techniques that are based on the principle of nature (Goldberg 1989). For the past few years, GA has been successfully applied to a variety of complex real world problems like optimization, system identification, scheduling, data mining, nonlinear controls, robotics, and others (Ishibuchi and Murata 1996; Smith and Stonier 1996; Koza et al. 1996; Myung and Kim 1996; Dadios and Williams 1996; Barrido and Dadios 2002; Dozier et al. 1994).

John Holland originally developed GA in 1975 (Holland 1992). It is a generate-and-test method borrowed from the evolution theory of nature (Goldberg 1989). It evaluates the fitness of the candidate solution residing in the search space. The mission which GA carries in its operation is to find the best fit or approximately the fittest candidate solution(s) from the search space (Goldberg 1989; Man et al. 1999).

GA have been applied to various kinds of problems and is considered to be an ideal technique to solve optimization problems (Powell and Skolnick 1993; Riff Rojas 1996; Michalewicz and Attia 1994; Le 1996; Jiménez and Verdegay 1999; Deb and Agrawal 1999; Dodd et al. 2001). There are two types of optimization problems: (a) constrained and (b) unconstrained. Most of the hard problems and problems in operation research are constrained optimization problems. In this type of problem, a solution that must satisfy the constraints has to be found. Although the GA have been used to solve constrained optimization problems, there has been no standard procedure in handling the constraints (Holland 1992; Arabas et al. 1994; Michalewicz and Attia 1994). Most researchers used some ad hoc methods and heuristics to deal with constrained problems. The reason for this phenomenon is that there is experimental evidence that in EAs, incorporating the

problem specific knowledge can improve the performance of the system very significantly (Arabas et al. 1994; Michalewicz and Attia 1994; Michalewicz 1996). A comprehensive survey has been published pertaining to the constrained handling techniques and several approaches have been introduced in dealing with solutions that violate the constraints (Michalewicz and Attia 1994; Le 1996; Homaifar et al. 1994; Eiben et al. 1994; Michalewicz 1995a; b; Schoenauer and Xanthakis 1993; Hai-Lin and Yu-Ping 2003). The first approach is to eliminate the infeasible chromosome from the population by not considering its participation in reproduction process. This is known as reject strategy and classified as "death penalty" heuristic. The method of eliminating infeasible solutions from a population may work reasonably well when most of the search space is occupied by feasible solutions. On the other hand, it will introduce serious limitations when the initial population composed mostly of infeasible solutions.

The second approach in dealing with infeasible solution is to use special repair algorithms that will correct the generated infeasible solutions. This technique is known as repair strategy and it converts an infeasible offspring into a feasible one. The weakness of this method is the problem dependence that is computationally intensive and the resulting algorithm must be tailored to a particular application. For each particular problem, a specific repair algorithm should be designed especially when there is no standard heuristic available.

The third approach in dealing with infeasible solution is to use a special representation mapping which guarantees the generation of feasible chromosomes only. This is called decoder strategy. Decoder strategy has an advantage because it never generates infeasible solutions. However, the disadvantage of this approach is it considers no point outside the feasible region. It can be a serious limitation if the search space contains a large number of infeasible solutions. This mapping strategy is most of the time computationally expensive and needs to be tailored according to the problem in hand.

The fourth approach in dealing with infeasible solution is by using penalty strategy. In this technique, the potential solutions are generated without considering the constraints and then penalized by decreasing the "goodness" of the evaluation function. It means that the constrained problem is transformed into unconstrained problem by injecting penalty into all infeasible chromosomes. This penalty amount is computed through a penalty function. There are several ways to define penalty function but designing a very effective one is nontrivial (Myung and Kim 1996; Powell and Skolnick 1993; Michalewicz 1995a).

Michalewicz applied GA for 0/1 knapsack problem and uses static penalty function (SPF) to penalize infeasible chromosomes. In his work, the penalty function depends on the degree of violation. The bigger the violation the greater the penalty imposed. However, the growth of the function is uncontrollable with respect to the size of the violation (can be logarithmic, linear, quadratic, exponential, etc.). According to Michalewicz, a penalty function Pen(x) can effectively guide (drive) the search towards a promising area of the solution space. However, there is no general guideline or standard instruction on the design of the effective function because of its problem dependence.

The SPF does not change its amount according to the on-going changes in the course of genetic search. Hence, it is considered to be a blind function. If the process just concentrates on applying constant amount of penalty then it cannot explore the other regions of the search space. This will only direct the search process to concentrate on a small portion of the space that will eventually result in premature convergence. The weakness of static function can be resolved by incorporating a criterion of the process that will help the genetic search explore most of the regions at the start and then later converge to a promising area. This concept can be achieved using two strategies: (1) by designing a dynamic penalty function (DPF) that changes its penalty amount according to the state of the search, (2) by making the penalty function adaptive to the search and adjust the penalty amount according to the state of the feasibility in the search. This adaptive penalty function (APF) changes its value adaptively by exploiting the available current information about the feasibility and infeasibility in the population pool.

In this research, two functions, namely an APF and a DPF have been designed and applied to solve the 0/1 knapsack problem. The problem belongs to a family of NP-complete problems and is considered to be a benchmarking problem because it can be viewed as a generalization of various industrial problems such as cargo loading (Michalewicz 1995b, 1996). Hence, it is very significant to test these penalty functions on 0/1 knapsack problem because the conclusions and results can further be applied to many constrained optimization problems like the product life cycle (LC) assessment dynamic modeling; for example, in areas of costs, sensitivity analysis and optimization of LC process parameters to economically reduce the total environment load (Kantardgi et al. 2005).

The 0/1 knapsack problem: a model for selecting cleaner production measures

The 0/1 knapsack problem is a classical problem in the field of Operations Research and can be viewed as a general form of many practical and industrial application problems. This is considered as one of the nonlinear programming problems (NLP). The goal here is to find the greatest value (profit) for a given subset of objects that will fit inside the bag. Mathematically 0/1 knapsack is defined as follows (Michalewicz 1996):

For a given set of weights $w[i]$, profit $p[i]$, and the capacity C of the knapsack, find the binary vector x

$$x = \langle x[1], \ldots, x[n] \rangle \tag{1}$$

such that

$$\sum_{i=1}^{n} x[i] \cdot w[i] \leq C \tag{2}$$

and for which

$$P(x) = \sum_{i=1}^{n} x[i] \cdot p[i] \tag{3}$$

is maximum, where $x[i] \in \{0,1\}$; 0 for not selected and 1 for selected item.

As has been shown, this is an optimization problem which aims to find the best result under given circumstances (Myung and Kim 1996). In operations research, the ultimate goal is to minimize the effort required and to maximize the desired benefit. The effort required or the benefit desired in any practical situation can be expressed as a function of certain decision variables. Hence, optimization can be defined as the process of finding the values (conditions) that give the maximum or minimum value of a function. Mathematically, this can be stated as follows.

$$\text{Find } X = \begin{Bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{Bmatrix} \text{ which minimizes} f(X)$$

Subject to the constraints

$$g_j(X) = 0,\ j = 1, 2, \ldots, m$$
$$l_j(X) = 0,\ j = 1, 2, \ldots, p$$

where X is an n-dimensional vector called the design vector, $f(X)$ is termed the objective function, and $g_j(X)$ and $l_j(X)$ are known as inequality and equality constraints, respectively. The number of variable n and the number of constraint m and p need not be related in any way. The problem stated above is called a constrained optimization problem.

In the application of selecting cleaner production measures, the 0/1 Knapsack parameters are equivalent to (Kantardgi et al. 2005):

n-dimensional vector X	= the *Technologies applied*, for example, modifying the time temperature profile, increasing the turbulence in the pre-heat zone, increasing the interaction between the product and the flue gas, etc.
The *weights* $w[i]$	= *Technology indicators* like efficiency, etc.
The *profit* $p[i]$	= *Actual cost* of the technology applied, and
Capacity C	= *Level of waste/emission*.

For more details of these parameters, refer to the work of Kantardgi et al. (2005).

Constraints

In many practical problems, the decision variables cannot be chosen arbitrarily, rather they have to satisfy certain specified functional and other requirements. The restrictions that must be satisfied to produce an acceptable solution are collectively called decision constraints. Many practical problems in operations research are optimization problems and they are characterized by mixed continuous–discrete variables, and discontinuous and non-convex search spaces. If standard nonlinear programming techniques are used for this type of problem, they will be inefficient, computationally expensive, and in most cases, find a relative optimum that is closest to the starting point. GA promised a better solution to this problem because it can find global optimum solution with high probability (Goldberg 1989).

In order to solve a constrained optimization problem, there is a need to incorporate a constrained handling technique. This is the central issue which should be dealt with when applying EAs to these problems. The taxonomy of constraints can be considered as (Eiben et al. 1994):

a) Number: number of the constraints associated with problem.
b) Metric: either the constraint is continuous or discrete so that a violation of the constraint can be assessed in distance from satisfaction using that metric.
c) Criticality: how critical constraint is to the problem (soft, hard).
d) Difficulty: difficulty in satisfying the constraint. This difficulty can be characterized by the size of the feasible region (F) compared to the size of the sample space (S).

According to Michalewicz (1995b), there is no known method of determining the global maximum (or minimum) to the general nonlinear optimization problem. If the objective function f and the constraint c satisfy certain properties, the global optimum can sometimes be found. Several algorithms were developed for unconstrained problems such as direct search method and gradient method. In constrained problems, an indirect method extracts one or more linear problems from the original before a direct search is applied. Despite the active search and progress in the global optimization in recent years, there is still no efficient solution presented which is applicable to this type of problem.

Ga system for 0/1 knapsack problem

The process involved in formulating the GA system for 0/1 knapsack problem is shown in Fig. 1. This architecture shows the involved modules and flow of the process. The initialization module generates the data set and stores it in a database. The computation module

performs the necessary computation before passing the data to GA-cycle module. Computation module involves computation of ratio term and capacity C of the knapsack. GA-cycle module involves the search algorithm for 0/1 knapsack problem.

The required data for GA of 0/1 knapsack problem is generated randomly. Here, each item $i = 1,..., n$ has weight $w[i]$ and profit $p[i]$. Weight and profit are also generated randomly and stored in the database. Table 1 shows a sample data for the GA operation. The weight and profit of $N = 8$ items are displayed.

In this work, the chromosome is represented with a binary string of known length n. The value of the bit position indicates the presence of that particular item. For example, if the value of the ith bit position is 0 then this means that i item is not in the chromosome. Individual solution is represented as vector $x = (x_1, x_2,..., x_n)$,

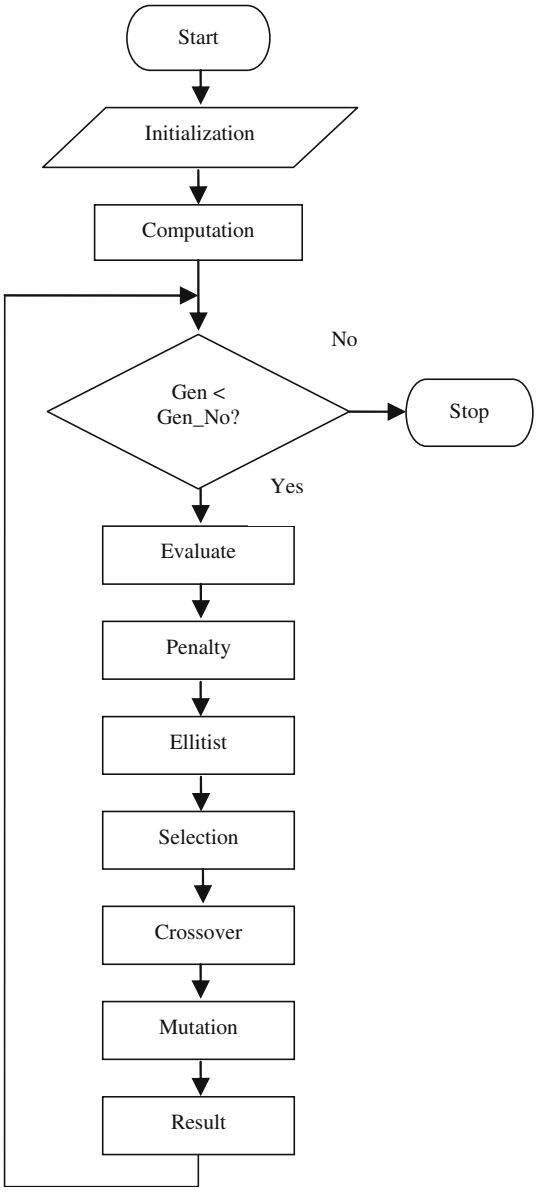

Fig. 1 Program flowchart

where n is the number of the items and $x_i = \{0, 1\}$. For example, $x = (1, 0, 0, 1, 0, 1, 1, 0)$, where $n = 8$ objects and the first, fourth, sixth and seventh items are selected and considered to be inside the knapsack of a limited weight capacity.

Since this work is for research purposes, a random initial population is considered. The ideal initial population is the one that covers approximately a good portion of the solution space. The more it is scattered the better performance it gets because it covers a wide area of the search space. There are several ways of choosing the size of the population. Note that once the population size is too large, the system may spend that much time to converge. Also, once the population is too small, this may not find the optimum solution and might get stuck in the local maxima. Nature reveals that larger populations are more stable and will have a better effect on evolution than a small population. Alander in his work "Optimal Population Size of Genetic Algorithm" suggested that the optimal population size for GA problems of varying complexity.

$$S_{\text{opt}}(n) \text{ is } : n \leq \boldsymbol{S}_{\text{opt}}(\boldsymbol{n}) \leq \boldsymbol{2n} \tag{4}$$

where:

n is the number of variables or length of the chromosome.

During evaluation, the fitness value of all individuals in the current population is determined by applying the fitness function $f(x)$. In most optimization problems, the fitness function $f(x)$ is quite straightforward (measurement of some distance). In the case of 0/1 knapsack problem, the evaluation function is

$$\text{Eval}(x) = \sum_{i=1}^{n} x[i] \cdot p[i] \tag{5}$$

Here, the above data are followed for weight and profit. For example, let $x = (1, 0, 0, 1, 0, 1, 1, 0)$ then from Table 1: eval(x) = summation of the profit of the first, fourth, sixth and seventh items $= 5 + 6 + 4 + 7 = 22$.

Evaluation function with penalty term

In general, there are two ways to construct evaluation function with penalty term. One is to multiply the penalty function with the objective function and the other is to add or subtract the penalty term from the evaluation function for minimization and maximization problems, respectively. This research uses the second approach. In

Table 1 Sample of randomly generated data: weight and profit of each item

$\boldsymbol{x[I]}$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
$w[I]$	2	5	3	7	4	5	2	6
$p[i]$	5	6	3	6	1	4	7	2

the case of 0/1 knapsack problem, the objective is to maximize profit, therefore, it penalizes the infeasible solution(s) by reducing the "goodness" of the infeasible individual. Equation 6 shows the evaluation or fitness function.

$$\text{Eval}(x) = \sum_{i=1}^{n} x[i] \cdot p[i] - \text{Pen}(x) \tag{6}$$

where:
Pen(x) is the penalty function.
Pen(x) = 0; if x is feasible
Pen(x) > 0; otherwise

Here, the solution x is considered feasible when

$$\sum_{i=1}^{n} x[i] \cdot w[i] \leq C; \text{Otherwise, it is infeasible.}$$

Static penalty function

The SPF was designed and applied by Michalewicz for 0/1 knapsack problem (Michalewicz 1995b). This is the simplest way to penalize infeasible individuals by giving constant amount of penalty. Equation 7 shows the SPF and uses distance metric unit of infeasible individual to measure the distance from the feasible region.

$$\text{PenSPF}(x) = \log 2\left(1 + \text{ratio}\left(\sum_{i=1}^{n} x[i] \cdot w[i] - C\right)\right) \tag{7}$$

where:
$\sum_{i=1}^{n} x[i] \cdot w[i] - C$ = distance from feasible region
ratio = max $i = 1 \ldots n$ $p[i]/w[i]$);
$p[i]$ = the profit of ith item
C = capacity of the knapsack

The SPF applies static amount of penalty only and it can be considered as weak solution because it is not capable of changing its value as the search process go over generations. Therefore, dynamic and APFs are proposed in this work to overcome the limitations of the SPF.

Dynamic penalty function

In SPF, the imposed penalty amount to infeasible solution is static throughout the process of GA and is considered blind during the progress of search. It seems worthwhile to change the penalty amount dynamically as the search progresses. The main idea here is to allow light penalty in the beginning and to increase the severity of penalty at the later part. The purpose of increasing the severity penalty amount at the later part is to help the system converge the search to near optimal or sub-optimal region. Two parameters may be considered to incorporate the dynamic aspects:

1. The number of the feasible/infeasible solution found in the population pool.
2. The number of iteration or generation number.

Let gen represent the number of generation; then the proposed dynamic penalty function (PenDPF) is as follows:

$$\text{PenDPF}(x, \text{gen}) = \text{dist} + \log 2(\text{gen} + \text{ratio}) \tag{8}$$

where:
dist = distance from the feasible region
gen = generation number.
ratio = max $i = 1 \ldots n$ $\{p[i]/w[i]\}$; used as a heuristic to influence the search.
C = capacity of the knapsack

Adaptive penalty function

Incorporating the length of the search into the penalty function has been generally effective but on the other side these penalties ignore any other aspect of the search. It means that they are not adaptive to the ongoing success of the search. This gives rise to the need to make the system adaptive in such a way that the search will explore the region that seems attractive and stay away from the region that is unattractive to the search.

The main objective behind the design of the APF is to come up with a function that changes the penalty according to the state of the infeasibility in the current population. Equation 9 gives the proposed APF. This function exploits the on-going information to change its penalty amount adaptively. It incorporates problem specific, population specific and chromosome specific information to make it completely adaptive. This proposed penalty function can alter the magnitude of the penalty dynamically by counting the number of the infeasible solutions found in the current population pool. If considerable feasible solutions are present in the current population, then, regions not that far from the boundary are considered in the search. For example, there are eight infeasible chromosomes in a population of size 10. If only feasible solutions are used for reproduction then the search will explore less search space. Therefore, the APF adaptively adjusts the penalty amount in such a way that it encourages the infeasible chromosome to participate in reproduction and share more information. In this case, the adaptive term helps decide 'how far' an infeasible candidate should consider participating in the genetic search.

$$\text{Pen}_{\text{APF}}(x, t) = \text{dist} \times \frac{\text{POP\_SIZE}}{\text{infeas}} + \log_{10}(\text{ratio}) \tag{9}$$

where:

$\sum_{i=1}^{n} x[i] \cdot w[i] - C$ = distance from the feasible region.

POP_SIZE = population size (number of solution candidates in one generation pool).

infeas = number of infeasible chromosomes in current population of *i*th generation.

ratio = max $i = 1..., n$ $p[i]/w[i]$

C = capacity of knapsack

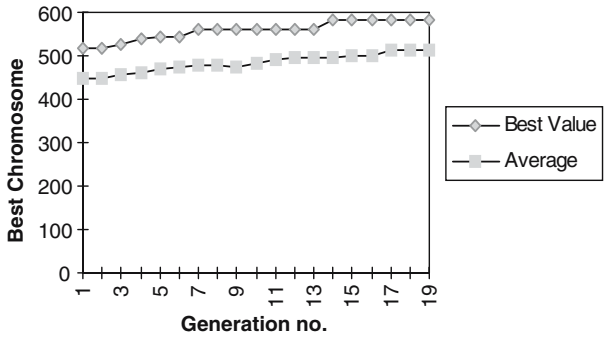

Fig. 2 Best chromosomes fitness value and average value during genetic algorithm search process

Experiment results

The performance of the proposed penalty functions developed was tested several times with different parameters and data set. The robustness of the proposed system was tested based from randomly generated data sets. The following are to be observed in the experiments:

1. Performance improvement along with the generation number.
2. Performance of the proposed DPF against the SPF and APF against the SPF.
3. Performance of the system by using different kind of randomly generated data set.
4. Effect of the relationship between weight and profit of the item over the performance of the system.
5. GA parameters suited for 0/1 knapsack problem.

The experiment was conducted in two phases. The first phase observed the performance of the proposed adaptive and DPFs. The second phase made extensive observation of the system to decide the environment and parameters required to achieve better results for 0/1 knapsack problem.

The analysis of GA convergence into promising area of the search space is presented in Fig. 2. The fitness value of the best chromosome found during each GA-cycle is plotted against the generation number. The figure shows improvements in the search along generation number. The continuous rise in the graph (best, average) proves the GA's capability of searching the best solution through evolution. It is important to take note that once no significant improvement is attained, the penalty functions and mutation operator comes in to perturb the system structure with small probability. This helps the GA search in finding solutions that may not be covered by classical GA operators.

GA and penalty function

For each penalty functions developed, five experiments were conducted with different GA parameters (number of items, population size and number of generations). Each experiment is composed of 50 runs. For each run, data set (weight and profit of *N* items) and initial population is generated randomly. The idea behind this is to observe the system thoroughly by providing different and diverse inputs and find out the behavior of the system.

Adaptive versus SPF

Table 2 shows the parameters used during experiments and results comparing the fitness performance of using APF and SPF. The performance of penalty function is measured by the maximum fitness value of the best chromosome found. Therefore, both penalty functions are tested with same data set (input) for each experiment to see which one helped the system got better fitness value for the best chromosome. Note that each experiment constitutes 50 runs and comparison of fitness values for penalty functions are taken for every run. Each run uses different randomly generated weight and profit of *N* items, and initial population.

From Table 2, experiment 1 has 34 runs having best fitness for APF compared to 16 runs for SPF. The overall comparison showed that the APF performed 68 best fitness against 32% of the SPF for a total of 50 runs.

Experiment 2 changes the number of item (*N*) for knapsack and population size of the initial population. Out of 50 runs, SPF got 12 with highest fitness value against 38 of APF. It is shown from here that the APF remains dominant over SPF with 76 against 24% of SPF.

Table 2 Comparison of the results of APF versus SPF

Experiment	GA parameters			Result[a]	
	No. of items (*N*)	POP_SIZE	GEN	APF	SPF
1	50	50	100	34	16
2	75	75	100	38	12
3	75	125	100	39	11
4	100	100	300	41	9
5	75	150	400	36	14

[a]Number of runs with the highest fitness based on 50 runs

Experiment 3 was conducted to test the effect of increasing the population size from 75 to 125. Out of 50 runs of the search, the APF has 39 better fit against 11 of SPF. This result shows that even with increased population size, the APF still performed better with 78% of the total 50 runs as compared to 22% of the SPF.

In experiment 4, all the genetic parameters were changed. In this experiment, the effect of penalty functions is observed based on large generation number. The reason for this is to see the effectiveness of penalty function when there is less diversification in the population pool. Here, the APF has 41 runs with highest fitness compared to 9 of SPF. This result shows that increasing the generation number increases the effectiveness of APF also.

Experiment 5 doubles the population size. The result of this experiment shows that the APF got 36 best fitness runs compared to 14 of SPF. Therefore, based on the overall results in Table 2, it was found that the APF was able to achieve superior behavior compared to SPF.

Dynamic versus SPF

In this experiment DPF is tested against SPF. Five experiments were conducted to evaluate the performance of penalty functions as shown in Table 3. Different combinations of genetic parameter (N, POP_SIZE, GEN_NO) values are used for each experiment. However, crossover and mutation rate were fixed probabilities and were set to 0.65 and 0.03, respectively, to insure uniformity.

From Table 3, experiment 1 sets the GA parameters to small values. The item in the knapsack is 50, the initial population is 50, and the generation number is 100. The DPF in most of the run directed the search to the promising area. Here, out of 50 runs, 43 gives better result for DPF against 7 for SPF. It is shown during this experiment that DPF achieved very high results of 86 against 14% of SPF. For experiments 2 and 3, the population size has been increased keeping the other parameters similar to the previous experiment. It is clear from the result that DPF exhibits superior behavior compared to SPF. For experiments 4 and 5, both the population size and GA parameters are changed. Again, the DPF performs better results compared to SPF. It is, therefore, observed that the DPF consistency helped the system direct the search to the promising area.

Table 3 Comparison of the results of DPF versus SPF

Experiment	GA parameters			Result[a]	
	N	POP_SIZE	GEN	DPF	SPF
1	50	50	100	43	7
2	50	75	100	43	7
3	50	100	100	35	15
4	50	75	100	35	15
5	75	150	400	43	7

[a]Number of runs with the highest fitness based on 50 runs

GA crossover parameter

As has been mentioned in (Dadios and Williams 1996), crossover is one of the important processes for GA. Traditional GA uses only 1-point crossover (Goldberg 1989). The extension of 1-point crossover can be useful depending on the structure of the problem to solve. In this study, 1-point and 2-point crossovers were investigated and applied with different penalty functions to analyze their effects over the convergence of genetic search. Genetic parameters (items, pop_size, gen) were kept the same in each experiment. Both crossover operators are applied on the same data set with constant mutation rate. The experiment is composed of 50 program runs. Both crossover operators are applied with APF, DPF and SPF separately. This is conducted in order to study the application of crossover operators considering the effect of penalty functions.

Table 4 shows the result of the experiment conducted. Both 1-point and 2-point crossover operators were applied with APF, DPF, and SPF and test runs were taken under each penalty function. For example, in the case of APF, out of 50 runs 27 times 1-point crossover generates highest fitness value compared to 23 times for the 2-point crossover. Same figures are achieved in the case of DPF. In the case of SPF, for a total of 50 runs, 1-point crossover performed 32 times highest fitness value compared to 18 times for the 2-point crossover. Hence, from these results the 1-point crossover technique was chosen for the 0/1-knapsack problem.

GA mutation parameter

Mutation is a GA operator whose main purpose is to reintroduce lost alleles into the population (Dadios and Williams 1996). Mutation is applied with declared mutation rate. One of the goals of this operator is to introduce a perturbation in the search so that the system will not be stuck on local maxima. If the mutation rate is very high, then it might fail to achieve its goal because it might drive away the search from the feasible region. Therefore, a moderate and well-studied mutation rate should be used in order to achieve better performance of

Table 4 Crossover result from 50 runs of the program with different penalty functions

Crossover	APF[a]	DPF[a]	SPF[a]
1-point	27	27	32
2-point	23	23	18

[a]Number of runs having highest fitness

the system. In this research, four mutation rates (0.01, 0.03, 0.05, 0.08) were examined.

Figure 3 shows the result of this experiment in radar-type graph. Here, the number of axes present the number of runs of the system. Hence, the number of the axis is from 1 to 50. Each label represents the mutation rate and their mapping is shown in the graph legend. The circular rings in the graph show the range of the fitness values of the best chromosome. If one particular label has more range than the other label, then it means that the label representing the mutation rate has generated highest fitness value of the best chromosome. Hence, it is observed that mutation rate with smaller value shows better results compared to higher value for this constrained optimization problem.

Discussion and analysis of results

It has been discussed in section 3 that the SPF is bound to access limited information about the system's on-going situation, hence, it not utilizing any other aspect except the distance from the feasible region. Though this distance is the metric unit that helps adjust the penalty for a particular solution this is still naive in accommodating any other useful information. In order to come up with more effective function, two techniques have been designed in this work: (1) make the function adaptive and (2) consider the dynamic behavior of the search.

Adaptive penalty function

The main objective of APF is to change the amount of penalty according to the state of infeasibility in the current population. State of infeasibility is measured by analyzing the population pool. If there are very few feasible chromosomes in population pool, then it means that the state of feasibility is very low. In such a case, infeasible chromosomes are necessary to participate in the reproduction so that the system can diversify. In order to adjust the penalty amount according to the state of feasibility or infeasibility, feedback from the search regarding the on-going change is required. There are three domains that can be exploited to provide specific information (feedback) to the search:

1. Problem specific information (i.e., heuristic data set)
2. Population specific information (ratio between feasible and infeasible regions)
3. Chromosome specific information (distance of the infeasible chromosome from the feasible region)

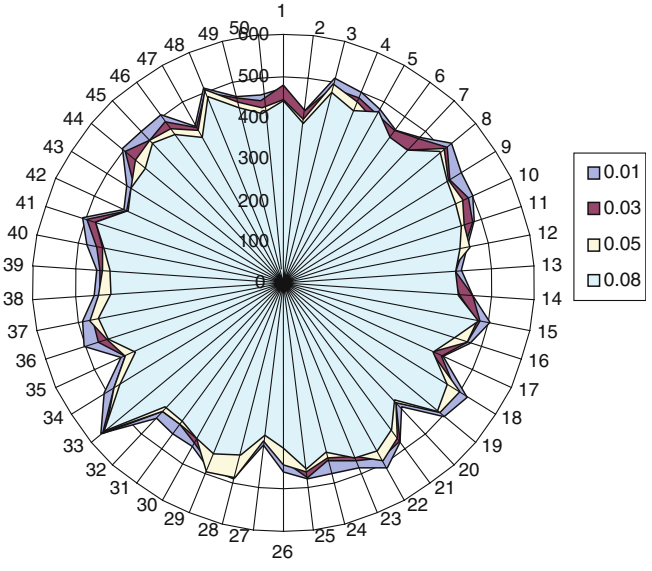

Fig. 3 Fitness values of best chromosomes achieved in 50 runs by applying different mutation rates using DPF and 1-point crossover

The adaptive function exploits the above information to compute the amount of penalty. The problem specific information is the maximum ratio of profit and weight of the items that are generated randomly. This ratio value has two roles in this research. One is to provide problem specific information to compute the penalty amount and the second is for heuristic application. The ratio term gives the profit-weight ratio that is more profitable and consumed less space.

Equation 9 of section 3 shows the APF. The adaptive term exploits the population specific information as well as the chromosome specific information. After evaluating the population pool, the system analyzes the state of feasibility in the current population. If there are enough feasible chromosomes present in the population, then infeasible chromosomes are discouraged to participate in the reproduction. This is achieved adaptively by giving severe penalty to infeasible chromosomes. If the population contains very few feasible chromosomes, during reproduction these chromosomes become dominant, hence the new population pool developed will constitute chromosomes with less diversification. This can further introduce two limitations:

1. Population might contain identical copies of chromosomes.
2. Due to less diversification, dominant structure (schemata) forces the search to concentrate on a small portion of the search space. This eventually leads the search towards premature convergence.

To avoid this premature convergence and to diversify further, infeasible chromosomes are encouraged to participate in reproduction by giving them a less penalty. This is achieved through infeas variable in Eq. 9. The variable infeas updates its value adaptively in every new population to provide feedback to the search regarding the state of the feasibility. This adaptive term (POP_SIZE/infeas) is then applied to the system with

the metric unit dist. The variable dist provides the chromosomes specific information and further tunes the penalty term making it more effective. This function penalizes the infeasible chromosomes that cannot be achieved through static penalties operation.

During experiments, it was observed that chromosomes that reside near the boundary get dominant over the population and restrict the search. If it happens in the early stage of the search, then it can be stuck in the local maxima. To avert this situation, there is a need to introduce a perturbation in the chromosomal structure. This can be achieved by adjusting the mutation rate accordingly. This helps the search explore possibly the whole search space and find promising solutions. Here, it was observed that the APF along with adaptive mutation rate makes the system perform better compared to traditional GA parameters.

Figure 4 shows that the fitness values of the chromosomes using APF have better results compared to SPF. The graph shows 50 runs arranged in circular formation with the corresponding fitness values. The outermost part of the circle gives the highest value. Here, the result of APF is plotted first followed by the result of SPF. The parameters used are $N = 100$, POP_SIZE = 100, GEN = 300 and with different data sets.

In this research, the mutation rate was adjusted adaptively. During experiments, it was observed that adaptive mutation helps the system find the best chromosome rapidly because of perturbation scale adjustment. If the system search is not exploring other regions and moving toward only one direction then a high rate perturbation is required so that other regions will be included in the search. Here, the mutation rate was adjusted to 0.3 for the minimum need and 0.08 for the maximum need of perturbations.

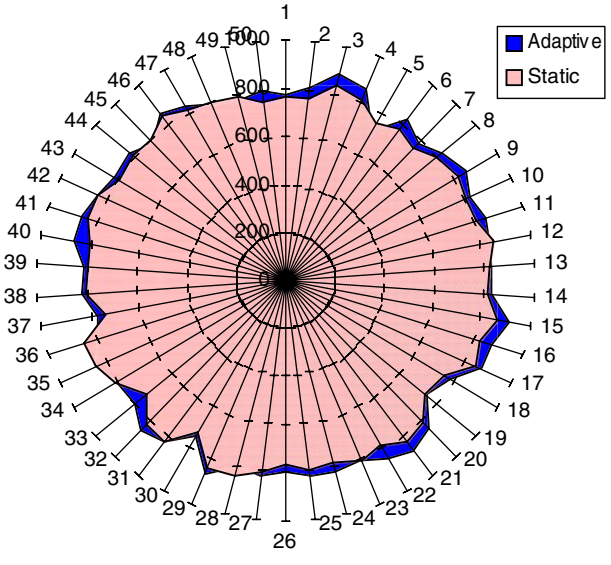

Fig. 4 Radar-type graph showing the fitness values of static and adaptive penalty functions

Dynamic penalty function

Dynamic approach, constantly injects increasing penalty amount to the infeasible chromosomes as GA-cycle increases. The main idea behind the dynamic approach is to allow the infeasible individual at the beginning of the search to participate in the reproduction process and at the later part increase the penalty so that infeasible individuals hardly qualify for reproduction. With this approach, the system barely allows infeasible chromosomes in the later part of the search. This diversifies the search in all possible directions at early stages, exploring the whole search space and converging it at the end by giving severe penalty. This idea may work well if the optimal solution lies in between the feasible region. However, if the optimal solution resides outside the feasible region and the system failed to locate it in the early stages then it will be hard for the system to find it during the latter part of the search. In order to make a correct choice of penalty function, it is better to study the landscape of the problem very well. Once the structure and behavior of the problem is known, then the choice for the penalty function becomes easier.

In this research, Eq. 8 gives the DPF designed to apply the dynamic behavior of the search. Here, variable dist measures the distance of the infeasible chromosome from the feasible region. This variable also gives the measure of the overflow and drags the infeasible individual inside the feasible region. In the early stage of the search, when the variable gen is very small and the logarithmic part of the equation is less significant, the dist value is injected into the infeasible chromosome that attempts to bring it back near the boundary of the feasible region. This term is problem dependent and plays a critical role in finding the optimal or sub-optimal point. Problems with multi-constraint and other combinatorial problems usually contain optimal points on the fringe of boundary. In such case, it would be better to place the infeasible individual outside but near the

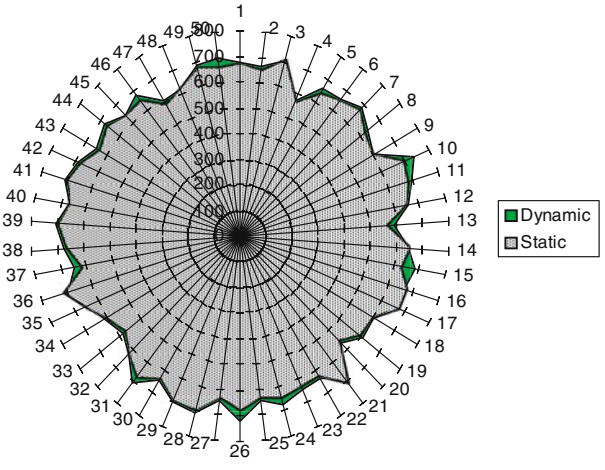

Fig. 5 Graph showing the dominance of the dynamic penalty function over static penalty function

boundary. This could be achieved by dividing the dist term with some constant value. As shown in eq. 8, the dynamic aspect of the function is the generation number gen. This allows highly infeasible solutions to stay in the search in the early stage and ignore them in the later part of the search.

Figure 5 shows the effectiveness of DPF in finding the maximum fitness value of the best chromosome. It can be seen from this graph that the DPF has a better result compared to the SPF. The result of dynamic function is superior and its application justifies the above discussion.

Conclusion

In this research, DPF is designed to incorporate dynamic behavior of the search. The main objective of this function is to punish the system as the search proceeds to infeasible regions. In the early stage of the process, this will allow highly infeasible chromosomes to participate in the search by giving less penalty and discourage them in the later part of the search by intensifying the amount of penalty. It was observed that this function works very well because of the dynamism of the search. In most optimization problems, the optimal point usually resides near the boundary of the search space and the dynamic function converges it to a promising area by limiting the diversification.

Besides DPF, in this research, an APF was designed to exploit the on-going information about the search. The applied adaptive function tailored the penalty amount according to the current situation of the feasibility in the population. In here, feasibility is the region of the search space occupied by feasible points. The ratio of feasible and infeasible region decides the severity of the penalty. If a certain population contains enough feasible chromosomes, then the infeasible chromosome are discouraged to participate in the reproduction of the next generation. Penalty function counts the number of infeasible chromosomes in a population and adjusts the penalty amount depending on the state of feasibility.

It was concluded in this research that huge improvement of the GA performance was obtained by experimenting other promising techniques and concepts like adaptive and dynamic learning behavior of the system. For successful implementation, a careful study of the problem and its search space is required. Before applying GA, one should analyze empirically the relation between the performance of the GA and the fitness landscape it has to optimize. From this experiment, it is shown that GA with variable penalty function outperform the genetic search with fixed penalty function.

In future works, the authors would like to implement the algorithms developed to model energy utilization in product life cycles. Also, the authors recommend study of the effects of integrating both the dynamic and APF into a single GA search process for constrained optimization problem.

Acknowledgments The authors would like to acknowledge De La Salle University -Manila URCO/College of Engineering Research Fund and the DLSU-Science Foundation for funding this project.

References

Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading

Ishibuchi H, Murata T (1996) Multi-objective genetic local search algorithm. In: Proceedings of IEEE international conference on evolutionary computation, Nagoya, Japan, May 20–22, 1996. JOB SHOP SCHEDULING, pp 119–124

Smith S, Stonier R (1996) Applying evolution program techniques to constrained continuous optimal control problems. In: Proceedings of IEEE international conference on evolutionary computation, Nagoya, Japan, May 20–22, 1996, pp 285–290

Koza JR, Bennett FH III, Andre D, Keane M (1996) Four Problems for which a computer program evolved by genetic programming is competitive with human performance. In: Proceedings of IEEE international conference on evolutionary computation, Nagoya, Japan, May 20–22, 1996, pp 1–10

Myung H, Kim JH (1996) Constrained optimization using two-phase evolutionary programming. Proceedings of IEEE international conference on evolutionary computation, Nagoya, Japan, May 20–22, 1996, pp 262–266

Dadios EP, Williams DJ (1996) A fuzzy-genetic controller for the flexible pole-cart balancing problem. In: Proceedings of IEEE international conference on evolutionary computation, Nagoya, Japan, May 20–22, 1996, pp 223–229

Barrido S, Dadios EP (2002) Online robot tracking using genetic algorithms. In: Proceedings of IEEE international symposium on intelligent control (ISIC) 2002, The Coast Plaza Hotel, Vancouver, British Columbia, Canada, 27–30 October 2002

Holland JH (1992) Adaptation in natural and artificial systems. University of Michigan Press, Michigan

Man KF, Tang KS, Kwong S (1999) Genetic algorithms concepts and design. Springer, London

Arabas J, Michalewicz Z, Mulawka J (1994) Genetic algorithms with varying population size. Proceedings of the 1st IEEE world conference on evolutionary computation, pp 306–311

Powell D, Skolnick MM (1993) Using genetic algorithms in engineering design optimization with non-linear constraint. Proceedings of the 5th ICGA, Morgan Kaufmann, San Mateo, pp 424–430

Riff Rojas MC (1996) Using the knowledge of the constraints network to design an evolutionary algorithms that solves CSP. In: Proceedings of IEEE international conference on evolutionary computation, Nagoya, Japan, May 20–22, 1996, pp 279–284

Michalewicz Z, Attia N (1994) Evolutionary optimization of constrained problems. In: Proceedings of the 3rd annual conference on EP, World Scientific, pp 98–108

Le TV (1996) A fuzzy evolutionary approach to constrained continuous optimal control problems. In: Proceedings of IEEE international conference on evolutionary computation, Nagoya, Japan, May 20–22, 1996, pp 274–278

Homaifar A, Lai SH, Qi X (1994) Constrained optimization via genetic algorithms. Simulation 62(4):242–254

Eiben AE, Raue PE, Ruttkay Z (1994) Solving constraint satisfaction problems using genetic algorithms. In: Proceedings of IEEE international conference on evolutionary computation, Orlando 1994, pp 542–547

Michalewicz Z (1995) A survey on constraint handling techniques in evolutionary computation methods. In: Mcdonnel JR, Reynolds RG, Fogel DB (eds) Proceedings of the 4th annual conference on evolutionary programming. MIT Press, Cambridge, pp 135–155

Michalewicz Z (1995) Genetic algorithms, numerical optimization, and constraints. Proceedings of the 6th ICGA, Morgan Kaufmann, San Mateo, pp 151–158

Schoenauer M, Xanthakis S (1993) Constrained GA optimization. Proceedings of the Fifth ICGA, Morgan Kaufmann, San Mateo, pp 573–580

Dozier G, Bowen J, Bahler D (1994) Solving small and large scale constraint satisfaction problems using a heuristic-based microgenetic algorithm. Proceedings of the 1st IEEE conference on evolutionary computation, Orlando, pp 306–311

Michalewicz Z (1996) Genetic algorithms + data structure = evolutionary programs, 3rd, revised and extended edition, Springer, Berlin Heidelberg New York

Jiménez F, Verdegay J (1999) Evolutionary techniques for constrained optimization problems. In: 7th European Congress on intelligent techniques and soft computing (EUFIT'99), Aachen, Germany. Springer, Berlin Heidelberg New York

Deb K, Agrawal S (1999) A niched-penalty approach for constraint handling in genetic algorithms. In: Proceedings of the ICANNGA, Portoroz, Slovenia

Dodd TJ, Tutty OR, Rogers E (2001) Genetic algorithm based constrained optimization for laminar flow control. In: Proceedings of the European Control Conference ECC'01, pp 2970–2974

Hai-Lin Liu, Yu-Ping Wang (2003) Solving constrained optimization problem by a specific-design multiobjective genetic algorithm. In: 5th international conference on computational intelligence and multimedia applications (ICCIMA'03), Xi'an, China, September 27–30, 2003, p 200

Kantardgi I, Purvis MRI, Cherviakov L, Khudoshina M (2005) Approaches to the modelling of energy utilization in product life cycles. Second Humanoid, Nanotechnology, Information Technology, Communication and Control Environment and Management (HNICEM) International Conference of the Institute of Electrical and Electronics Engineers Inc. (IEEE)–Philippine Section, HYATT Regency Hotel, Manila, Philippines, Plenary paper, March 17–20, 2005