Varying Fitness Functions in Genetic Algorithm Constrained Optimization: The Cutting Stock and Unit Commitment Problems

Vassilios Petridis, *Member, IEEE*, Spyros Kazarlis, and Anastasios Bakirtzis, *Senior Member, IEEE*

***Abstract*— In this paper, we present a specific varying fitness function technique in genetic algorithm (GA) constrained optimization. This technique incorporates the problem's constraints into the fitness function in a dynamic way. It consists in forming a fitness function with varying penalty terms. The resulting varying fitness function facilitates the GA search. The performance of the technique is tested on two optimization problems: the cutting stock, and the unit commitment problems. Also, new domain-specific operators are introduced. Solutions obtained by means of the varying and the conventional (nonvarying) fitness function techniques are compared. The results show the superiority of the proposed technique.**

***Index Terms*—Constrained optimization, cutting stock, genetic algorithms, genetic operators, unit commitment.**

I. Introduction

GENETIC algorithms (GA's) turned out to be powerful tools in the field of global optimization [7], [10], [13]. They have been applied successfully to real-world problems and exhibited, in many cases, better search efficiency compared with traditional optimization algorithms. GA's are based on principles inspired from the genetic and evolution mechanisms observed in natural systems and populations of living beings [13]. Their basic principle is the maintenance of a population of encoded solutions to the problem (genotypes) that evolve in time. They are based on the triangle of genetic solution reproduction, solution evaluation and selection of the best genotypes. Genetic reproduction is performed by means of two basic genetic operators: Crossover [10], [13], [30] and Mutation [10]. Many other genetic operators are reported in the literature, including problem specific ones [10], [15], [20], [25]. Evaluation is performed by means of the Fitness Function which depends on the specific problem and is the optimization objective of the GA. Genotype selection is performed according to a selection scheme, that selects parent genotypes with probability proportional to their relative fitness [11]. In a GA application the formulation of the fitness function is of critical importance and determines the final shape of the hypersurface to be searched. In certain real-world problems, there is also a number of constraints to be satisfied. Such constraints can be incorporated into the fitness function by means of penalty terms which further complicate the search.

The authors of this paper were among the first that proposed the varying fitness function technique [9], [14], [25], [28], [29], [33]. The purpose of this paper is to present a specific varying fitness function technique. This technique incorporates the problem's constraints into the fitness function as penalty terms that vary with the generation index, resulting thus in a varying fitness function that facilitates the location of the general area of the global optimum.

This technique is applied to two small-scale versions of two hard real-world constrained optimization problems, that are used as benchmarks: the cutting stock and unit commitment problems. The cutting stock problem consists in cutting a number of predefined two-dimensional shapes out of a piece of stock material with minimum waste. It is a problem of geometrical nature with a continuous-variable encoding. The unit commitment problem consists in the determination of the optimum operating schedule of a number of electric power production units, in order to meet the forecasted demand over a short term period, with the minimum total operating cost. It is clearly a scheduling problem. It is not our intent to present complete solutions of the above problems but to demonstrate the effectiveness of the varying fitness function technique. We have chosen these particular problems because they are diverse in nature (each problem exhibits unique search space characteristics) and therefore they provide a rigorous test for the efficiency and robustness of our technique.

Section II discusses various methods that enable the application of GA's to constrained optimization problems. Section III contains a detailed analysis of the varying fitness function technique proposed in this paper. The application of this technique to the cutting stock and the unit commitment problems are presented in Sections IV and V, respectively. Finally conclusions are presented in Section VI.

II. General Considerations

In the sequel, without loss of generality, we assume that we deal with minimization problems. As it has been mentioned before, in many optimization problems there are a number of constraints to be satisfied. As far as we know, six basic methods have been reported in the literature that enable GA's to be applied to constrained optimization problems.

Manuscript received October 8, 1994; revised August 27, 1996 and August 5, 1997.

The authors are with the Department of Electrical and Computer Engineering, Faculty of Engineering, Aristotle University of Thessaloniki, Thessaloniki 54006, Greece.

Publisher Item Identifier S 1083-4419(98)07312-9.

1) The search space is restricted in such a way that it doesn't contain infeasible solutions. This is the simplest method for handling elementary constraints, and was used in the traditional GA implementations. This is a wise thing to do when it is possible (e.g., in the case of bounded problem variables), since it leads to a smaller search space. However, this method is of little use when dealing with the majority of real-world constrained problems (e.g., with coupling constraints involving a number of variables).
2) Infeasible solutions are discarded as soon as they are generated. This method doesn't utilize the information contained in infeasible solutions. Also in case the GA probability of producing a feasible solution is very small, a lot of CPU time is consumed in the effort of finding feasible solutions through the genetic operators [10], [18].
3) An invalid solution is approximated by its nearest valid one [22], or repaired to become a valid one [23]. Such approximation (or repair) algorithms, can be time consuming. Also the resulting valid solution may be substantially different from the originally produced solution. Moreover, in certain problems, finding a feasible approximation of an infeasible solution may be as difficult as the optimization problem (Constraint Satisfaction Problems [9]).
4) Penalty terms are added to the fitness function. In this way the invalid solutions are considered as valid but they are penalized according to the degree of violation of the constraints. This method is probably the most commonly used method for handling problem constraints and is implemented in many variations [9], [10], [14], [15], [25], [28], [29], [33]. However, it imposes the problem of building a suitable penalty function for the specific problem, based on the violation of the problem's constraints, that will help the GA to avoid infeasible solutions and converge to a feasible (and hopefully the optimal) one.
5) Special phenotype-to-genotype representation schemes (stated also as decoders) are used, that minimize or eliminate the possibility of producing infeasible solutions through the standard genetic operators, Crossover and Mutation [18].
6) Special problem-specific recombination and permutation operators are designed, which are similar to traditional crossover and mutation operators, and produce only feasible solutions [10]. Such operators, though, are sometimes difficult to construct and are usually strongly adapted to the problem they were originally designed for.

Recent work also reports the combined use of traditional calculus-based optimization methods together with GA's and a meta-level Simulated Annealing scheme for the solution of nonlinear optimization problems with linear and nonlinear constraints (Genocop II) [19].

In this paper we use method d), which adds penalty terms to the fitness function according to the constraint violation. As stated earlier, the problem of this method is the design of an appropriate penalty function that will enable the GA to converge to a feasible suboptimal or even optimal solution. Some guidelines for building appropriate penalty functions are given in [26], where it is proved that it is better that the penalty function be based on the distance-from-feasibility of the infeasible solution, than simply on the number of violated constraints. Other researchers [29], proposed an adaptive penalty function that depends on the number of violated constraints and the qualities of the best-so-far overall solution (feasible or infeasible) and the best-so-far feasible solution.

In the technique proposed in this paper the added penalty term is a function of the degree of violation of the constraints, so as to create a gradient toward valid solutions, which guides the search (especially in case hill-climbing techniques are used). The penalty term for any solution that violates the constraints can be formulated by using the following procedure: given an invalid solution S we must first represent quantitatively its degree of constraint violation. This is why, we introduce a quantity $d(S)$ which measures the degree of constraint violation of solution S. The next step is the formation of a penalty function, P, depending on $d(S)$. This can be any monotonically increasing function. We have chosen a linear function

$$P(d(S)) = A \cdot d(S) + B \tag{1}$$

where A is a "severity" factor that determines the slope of the penalty function and B is a penalty threshold factor. The penalty term is added to the objective function (to be optimized), $O(S)$, to form the final fitness function $Q(S)$

$$Q(S) = O(S) + P(d(S)) \cdot v \qquad \text{where } v = 0 \text{ for a valid solution and } v = 1 \text{ otherwise.} \tag{2}$$

The penalty factors A and B must be sufficiently large to guarantee the separation between feasible and infeasible solutions. On the other hand, the large penalty factors do not allow the invalid solutions to survive for long during the GA evolution, as they are most unlikely to be selected for reproduction due to their bad quality. Too-short lifetime of invalid solutions makes the GA unable to exploit the information contained in such solutions. If the penalty factors are low, then invalid solutions get a chance of surviving for some generations and their information can be exploited. In this case, however, their quality can be comparable to the quality of valid solutions and can thus represent possible suboptimal solutions to which the GA could converge. Therefore the choice of the penalty factors becomes a problem. In order to alleviate this problem we have used the varying fitness function technique.

III. The Varying Fitness Function Technique

The Varying Fitness Function technique alters the penalty function dynamically during the search procedure by dynamically changing the penalty factors A and B of (1). The technique of varying the penalty function dynamically during the run has been reported in the literature in a number of variations [9], [14], [15], [25], [28], [29], [33].

In [28] the authors use a fitness function of the form: $Q(S) = O(S) + \alpha \cdot P(S)$, where $O(S)$ is the objective function, $P(S)$ is a nonegative penalty function and α is a penalty coefficient that changes adaptively during the GA evolution. The value of α is selected at every generation based on statistical calculations for different values of α. The goal is to balance the penalty value with the objective function value and achieve a desired distribution of the population in the search space.

In [29] the penalty function is adaptively altered during the evolution of GA, depending on the number of violated constraints, n_i, the objective function value of the best-so-far feasible solution, O_{feas}, and the objective function value of the best-so-far overall solution O_{all}. It has the form $P(n_i) = (n_i/2)^k \cdot (O_{feas} - O_{all})$, where k is a severity parameter. According to the authors "this (method) allows effective penalty-guided search in cases where it is not known in advance how difficult it will be to find feasible solutions, or how much difference there is in objective function value between the best feasible solutions and the best solutions overall."

In [15] and [25] the authors propose a penalty function of the general form $P(S) = d(S) \cdot PF(g)$, where S is the genotype (solution) under evaluation, $d(S)$ is a measure of the constraint violation, and $PF(g)$ is a penalty factor increasing with the number of generations g. In [25] this factor is given by the linear formula $PF(g) = PF_{start} + g \cdot PF_{step}$, where PF_{start} is the factor's starting value (usually kept low) and PF_{step} is the factor's increment. In [15] the factor is determined as $PF(g) = PF_{\max} \cdot g/G$ where $PF_{\max}$ is a maximum value for the factor and G is the total number of generations.

In [14] the authors used a penalty function of the form: $P(S) = (C \cdot g)^{\alpha} \cdot d^{\beta}(S)$ where S is the solution under evaluation, C is a constant, g is the generation index, $d(S)$ is the distance-from-feasibility measurement, and α, β are penalty function coefficients. The authors tested this penalty function on four (4) problems with linear and nonlinear constraints and reported best results for $\alpha = 1$, $\beta = 1$.

In [33] the authors propose a penalty function for the infeasible individuals of the form: $P(S) = O(S) \cdot g^{\theta}$, where S is the solution under evaluation, $O(S)$ is the objective value of the solution, g is the generation index, and θ is a positive coefficient. This penalty function is not related to the degree of constraint violation. However, it ensures that all infeasible solutions are ranked worse than the feasible ones and the authors report better results using this penalty function than rejecting the infeasible solutions. Moreover, they suggest that θ should be in the range $0.5 \cdots 1$, for better performance.

In [9] the authors deal with a constraint satisfaction problem (CSP) and propose a method of penalizing the unsatisfied constraints with adjustable factors (weights) that evolve during the GA run, making thus the GA to learn the appropriate constraint weights. According to the authors, the method starts with standard initial constraint weights and every 100 generations raises the weights of the unsatisfied constraints.

In the technique presented in this paper, the penalty factors are kept low at the beginning of the GA evolution in order to simplify the search and give the GA the opportunity to explore the search space more efficiently. As the evolution proceeds the penalty factors increase linearly with the generation index, so that at the end of the run they reach appropriately large values, which result in the separation of valid solutions from invalid ones. This technique makes it easier for the GA to locate the general area where the global optimum lies, at the early stages of the search. As the search proceeds, the penalty factors increase and the GA adapts to the changing search hypersurface. Near the end of the run, when the penalty factors reach their appropriately large values, the final search hypersurface is formed, preventing thus the GA from converging to invalid solutions. A varying penalty term can be expressed as

$$P_{\mathrm{v}}(d(S), g) = a(g) \cdot d(S) + b(g) \tag{3}$$

where g is the generation index and $a(g)$, $b(g)$ are the penalty factors which are increasing functions of g.

A good choice is the linear functions

$$a(g) = \frac{g}{G} \cdot A \quad \text{and} \quad b(g) = \frac{g}{G} \cdot B \tag{4}$$

where G is the maximum value of the generation index and A, B are the maximum values of the penalty factors when $g = G$. So the penalty term becomes

$$P_{\mathrm{v}}(d(S), g) = \frac{g}{G} \cdot A \cdot d(S) + \frac{g}{G} \cdot B. \tag{5}$$

B is the penalty threshold factor and should be chosen in such a way that

$$O(S)_{\min} + B > O(S)_{\max} \quad \text{or} \quad B > O(S)_{\max} - O(S)_{\min} \tag{6}$$

so that no invalid solution is ranked better than the worst valid one. Inequalities (6) hold for minimization problems; they should be modified accordingly for maximization problems. The A parameter represents the slope of the penalty function. Some guidelines for the determination of this slope are given in [26]. It is stated that the penalty assigned to an infeasible solution should be close to the *expected completion cost*, which is an approximation of the additional objective cost needed for the repair (completion) of the infeasible solution. However, as the authors themselves admit, in real world problems it is very difficult to calculate this quantity, as it demands the existence of derivative information of the objective function. In this paper, A has been determined empirically in both the cutting stock and the unit commitment problems.

IV. The Cutting Stock Problem

The cutting stock problem [4], [8], [25], [31] belongs to a special category of problems named "Cutting and Packing

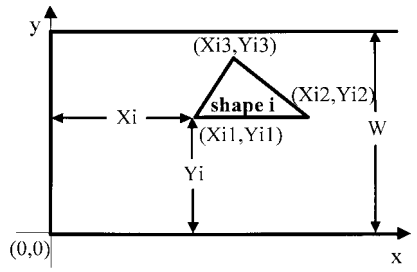

Fig. 1. The coordinate system of shapes within the stock material in the Cutting Stock problem.

problems" (C&P). The common objective of such problems is the determination of an optimal arrangement of a set of predefined objects (pieces) so that they fit within a container or stock material with minimum waste [8]. In this paper we consider a 2-D cutting stock problem. The specific problem that we consider consists in cutting a number of given two-dimensional shapes (to be referred to in the sequel as shapes) out of a large rectangular piece (the stock material), with standard width W and infinite length, so as to minimize the material waste. The constraint imposed on this problem is that no overlapping of shapes is allowed. For simplicity reasons, rotation of the shapes, in any angle, is not allowed, a restriction that is commonly applied in the industry (where the orientation of the shapes is specific, due to the nature of the material, e.g., decorations on textiles). As "material waste" we define the area of material not covered by the shapes, within the bounds of the smallest rectangle that contains the shapes (bounding rectangle). Before applying the GA, we must define a representation scheme to encode problem solutions into binary strings.

A. Encoding

Consider a coordinate system with the origin at the left bottom corner of the stock material. Such a coordinate system is displayed in Fig. 1. Each of the N shapes is described by a set of vertices. The number of vertices of shape i is p_i. (X_{ij}, Y_{ij}), $j = 1 \cdots p_i$ are the coordinates of the vertices of shape i. The left bottom vertex is considered to correspond to $j = 1$; it is called the base vertex. The position of shape i is completely specified by the coordinates of the base vertex (X_{i1}, Y_{i1}), since rotation of the shapes is not allowed. The base coordinates are also denoted by (X_i, Y_i) and referred to as the shape's coordinates.

A solution is described by N pairs of shape coordinates $(X_i, Y_i)i = 1 \cdots N$ which represent a specific cutting pattern. A solution can be considered as a point in a $2N$-dimensional space. A 10-bit integer is used to encode a single coordinate, resulting in a genotype string of $2 \cdot 10 \cdot N$ bits in length. For example in a problem with 12 shapes the genotype string length is 240 bits long, producing a search space of 2^{240} or $1.767 \cdot 10^{72}$ different solutions.

B. Fitness Function

According to (2) and (3) the fitness function that incorporates the varying penalty term for the nonoverlapping constraint must be of the form

$$Q(S, g) = O(S) + P_{\mathrm{v}}(d(S), g) \cdot v \tag{7}$$

where $O(S)$ is the objective function, $P_{\mathrm{v}}(d(S), g)$ is the varying penalty term concerning the overlapping area, $v = 1$ if the solution is invalid and $v = 0$ if the solution is valid.

First we must define the objective function $O(S)$. The objective of the problem is to minimize the material waste, MW. For N nonoverlapping shapes the waste of a specific solution S can be calculated as the difference between the area of the bounding rectangle and the sum of the areas of the shapes

$$MW(S) = W \cdot \max_{i,j}(X_{ij}) - \sum_{i=1}^{N} E_i \tag{8}$$

where E_i, $i = 1 \cdots N$ is the area of shape i, and $\max(X_{ij})$ is the maximum X coordinate over all shapes' vertices (i.e., the farest point of the farest shape from the Y axis). Obviously, $W \cdot \max(X_{ij})$ is the area of the bounding rectangle. However if $MW(S)$ is taken as the objective function to be minimized and since no overlapping is allowed, the optimum solution becomes a "needle in a haystack" and the algorithm gets trapped at local minima very easily. To circumvent this problem, we have used another objective function

$$O(S) = \sum_{i=1}^{N} ((X_i \cdot E_i) + Y_i) \tag{9}$$

that creates a gradient toward positions with small X and Y values. Note that this function does not incorporate a "waste" factor directly, but drives the shapes toward small X's and Y's. More emphasis is given on the X coordinates of shapes, by multiplying them by the shape's area, a fact that also encourages the positioning of large shapes near the Y-axis. In this way, the bounding rectangle is kept small, resulting in a small total waste.

In order to define $P_{\mathrm{v}}(d(S), g)$, according to Chapter II we must first define $d(S)$ as a measure of the violation of the nonoverlapping constraint. A solution is to define $d(S)$ as the sum of the overlapping areas of the shapes within the material

$$d(S) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} OA_{ij} \tag{10}$$

where OA_{ij} is the overlapping area of shapes i and j. Then, the varying penalty term (5) is given by

$$P_{\mathrm{v}}(d(S), g) = \frac{g}{G} \cdot A \cdot \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} OA_{ij} + \frac{g}{G} \cdot B \tag{11}$$

where B should satisfy (6). The value of B has been kept as low as possible.

The corresponding nonvarying penalty term is of the form

$$P(d(S)) = A \cdot \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} OA_{ij} + B. \quad (12)$$

Finally, from (7), (9), and (11) the final fitness function is defined as

$$Q(S, g) = \sum_{i=1}^{N} ((X_i \cdot E_i) + Y_i) + \left(\frac{g}{G} \cdot A \cdot \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} OA_{ij} + \frac{g}{G} \cdot B \right) \cdot v. \quad (13)$$

Extensive GA runs have shown that the varying fitness function makes the GA population behave in a dynamic way, due to the change of the search hypersurface, making the GA able to avoid trapping at sub-optimal solutions. This behavior somehow resembles the Simulated Annealing optimization algorithm in that it tolerates bad solutions under certain circumstances.

C. The GA Implementation

In the GA used in this paper, the parent selection method is the **Roulette Wheel parent selection** mechanism [11]. The crossover scheme is a version of the multipoint crossover [30]. The mutation scheme used is **binary (bit) mutation** with uniform probability. The parent population is totally replaced by the offspring (generational replacement). Finally, some additional GA features have been used such as 1) **Elitism** [7]: as the preservation of the best solution is vital to the effectiveness of the GA [12]. 2) **Fitness scaling**: the genotype fitness is scaled by a nonlinear transformation in order to emphasize small differences between near optimal qualities in a converged population [10]. 3) **Adaptation of operator probabilities**: in GA's, it is well known that, by increasing the probability of the Crossover operator the convergence speed is increased. Convergence can be accelerated also by scaling the genotype fitnesses so as to apply more selection pressure on the high-quality genotypes. Conversely, increasing the probability of the Mutation operator enhances the population diversity. If pre-mature convergence or excessive diversity occurs the search becomes inefficient. So, during the whole GA run, the operator probabilities and the fitness scaling function are adjusted dynamically, to keep the population diversity at a reasonable state [6].

D. Additional Operators

In order to accelerate the search, we introduce a local gradient descend operator, which is called "**phenotype mutation**" [1], [25]. This operator is applied only to the best genotype of every generation. It is a form of local search by means of small changes along each coordinate of the real coordinate vector (phenotype) $[(X_1, Y_1), \cdots, (X_N, Y_N)]$. After each change, it evaluates the resulting solution and when a better solution is found it is kept. The operator is described in pseudocode in the following lines:

```
Operator Phenotype Mutation (S)
begin
  Q = evaluate(S)
  S_orig=S
  for shape i = 1 to N do
    for C in [(S → Xi), (S → Yi)] do
      for Step in [+BigStep,+SmallStep,
      -BigStep,-SmallStep] do
        C=C+Step
        Q_test=evaluate(S)
        if Q_test better than Q then Q=Q_test;
                                     goto EXIT
                                     else S=S_orig
      endfor
      label EXIT:
    endfor
  endfor
end.
```

As seen in the pseudocode, the operator begins with the X coordinate of the first shape (shape 1) in the phenotype vector and performs four (4) modifications to it by adding the values $+$BigStep, $+$SmallStep, $-$BigStep, $-$SmallStep to the original coordinate value, one by one. After each modification, the resulting solution is evaluated. If it is better than the original one it replaces it and the modifications of X stop, otherwise it is discarded and the modifications continue. The same procedure is repeated for the Y coordinate of the first shape and it continues for all other shapes' coordinates in the phenotype vector, performing thus a local search in the neighborhood of the solution produced by the genetic search. Although, for every generation, the operator adds $2 \cdot N \cdot 4 + 1$ fitness evaluations in the worst case, it accelerates dramatically the search speed toward the final optimum. The values BigStep and SmallStep represent a large and a small modification step for the shapes' coordinates. In our implementation, the values of 1 and 0.1 were chosen, respectively.

Three more operators are used to assist the GA search, which are based on intelligent and problem oriented permutation mechanisms. The first, called the "**leap mutation**" operator, selects a shape at random and performs a leap, moving the shape in a random direction and distance, always within the bounds of the material. The second, called the "**shape swap**" operator, selects two shapes at random and swaps their positions. These operators are applied with a probability of 0.1 each.

The third and most significant operator, called the "**gap filling**" operator [25], scans the area of the stock material covered by the shapes trying to locate gaps (bounded areas not occupied by a shape) with an area greater than a predefined threshold (to avoid very small gaps). If an appropriate gap is located, the operator tries to fill the gap by repositioning a suitable shape with an X coordinate greater than that of the gap. The shape that achieves the smallest overlapping area wins and finally fills the gap. Since this operator is time consuming, it is applied only to the best genotype of every

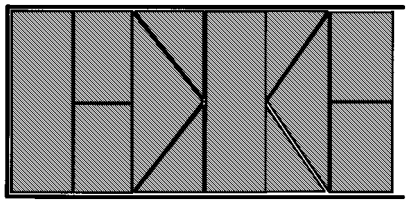

Fig. 2. Optimum arrangement of 12 shapes (Example 1).

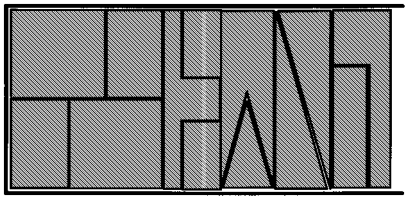

Fig. 3. Optimum arrangement of 13 shapes (Example 2).

TABLE I
COMPARISON OF RESULTS REGARDING THE CUTTING STOCK PROBLEM

Results over 20 runs	Example 1 (12 shapes)	
	Nonvarying Fitness Function GA	Varying Fitness Function GA
Success	60 %	90 %
Average time	335.2 sec.	335.3 sec.
Results over 20 runs	Example 2 (13 shapes)	
	Nonvarying Fitness Function GA	Varying Fitness Function GA
Success	0 %	30 %
Average time	401.14 sec.	402.7 sec.

generation. These additional operators were incorporated into the GA in both cases of the varying and the conventional (nonvarying) fitness function implementations, as they resulted in better GA performance compared with that of a simple GA implementation.

E. Simulation Results

The effectiveness of the varying fitness function has been compared with that of the nonvarying fitness function using two examples. In Example 1, a set of 12 convex shapes, and in Example 2 a set of 13 shapes, some of which are not convex, have been used. An optimal cutting pattern of the shapes of Example 1 is shown in Fig. 2. Fig. 3 displays an optimal cutting pattern of Example 2. The shapes in both examples have been selected so that they fit completely (leaving no material waste) in many combinations. Therefore, there are more than one globally optimal solutions for both sets of shapes. Also, there is a large number of local minima in both problems. Problem 2 is clearly more difficult to solve than problem 1, having more local minima, a fact that is justified by the simulation results.

The nonvarying fitness function GA and the varying fitness function GA have used the same operators and techniques described earlier, and have run for 300 generations with a population of 100 genotypes. Twenty runs have been performed for each technique. A run is considered successful if it has converged to an optimal solution (i.e., with zero material waste). The success percentage results are shown in Table I. The varying fitness function GA outperforms its nonvarying counterpart, with respect to the success percentage, while requiring the same CPU time. Slight differences in the average time figures between the two algorithms are mostly due to the stochastic nature of GA's (i.e., the number of Crossovers, Mutations, and other operators performed during each run is not constant but varies in a probabilistic manner). The simulation examples have run on a HP Apollo 720 workstation.

V. THE UNIT COMMITMENT PROBLEM

The second problem selected to demonstrate the efficiency of the varying fitness function technique comes from power systems engineering. GA's have been recently used to solve a variety of power system engineering problems such as distribution network planning [21], reactive power planning [16] and economic dispatch [1]. Here GA's are used for the solution of the well known unit commitment problem [2], [3], [5], [15], [17], [24], [27], [32], [34], [35]. The unit commitment (UC) problem in a power system is the determination of the start-up and shut down schedules of thermal units, to meet forecasted demand over a future short term (24–168 h) period. The objective is to minimize total production cost of the operating units while satisfying a large set of operating constraints. The UC problem is a complex mathematical optimization problem with both integer and continuous variables. The exact solution to the problem can be obtained by complete enumeration, which cannot be applied to realistic power systems due to combinatorial explosion [34].

In order to reduce the storage and computation time requirements of the unit commitment of realistic power systems, a number of suboptimal solution methods have been proposed. The basic UC methods reported in the literature can be classified into five categories: **Priority List** [2], **Dynamic Programming** [24], **Lagrangian Relaxation** [17], **Branch-and-Bound** [5], and **Benders Decomposition** [3]. Recent efforts include application of simulated annealing [35], expert systems [32] and Hopfield neural networks [27] for the solution of the UC problem. In this paper we present a small-scale version of the UC problem in order to test the effectiveness of the varying fitness function technique. A full-scale version has been solved using GA's in [15].

As mentioned above, the objective of the UC problem is the minimization of the total production cost over the scheduling horizon. The total cost consists of fuel costs, start-up costs and shut-down costs. Fuel costs are calculated using unit heat rate and fuel price information. For simplicity reasons, start-up costs are expressed as a fixed dollar amount for each unit per start-up. Shut-down costs are also defined as a fixed dollar amount for each unit per shut-down. The constraints which must be satisfied during the optimization process are: 1) system power balance (demand plus losses plus exports), 2) system reserve requirements, 3) unit initial conditions, 4) unit high and low MegaWatt (MW) limits (economic/operating), 5) unit minimum up-time, 6) unit minimum down-time, 7) unit status

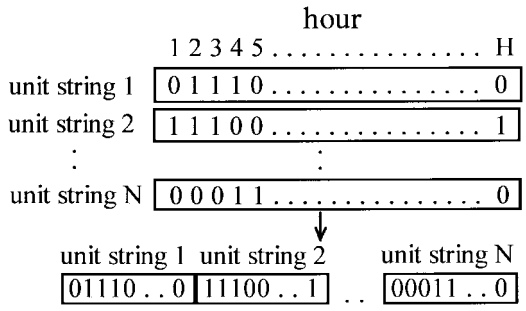

Fig. 4. The binary representation of a unit commitment problem solution.

restrictions (must-run, fixed-MW, unavailable, available), 8) unit or Plant fuel availability, and 9) plant crew constraints.

A. UC Encoding

We have applied GA's to the unit commitment problem using a simple binary alphabet to encode a solution. If N represents the number of units and H the number of hours of the scheduling period, an H-bit string (which is called the unit string in the sequel) is required to describe the operation schedule of a single unit, since at every hour a unit can be either on or off. In a unit string, a "1" at a certain location indicates that the unit is operating at this particular hour while a "0" indicates that the unit is down. By concatenating the N unit strings, a $N \cdot H$ bit genotype string is formed. This encoding is displayed in Fig. 4. As seen in this figure, there are N bit strings of H-bit length each that represent the schedule of the N units over an H-hour period and these unit strings are concatenated to form the genotype.

The resulting search space is vast. For example, for a 20-unit system and 24-h scheduling period the genotype strings are $20 \cdot 24 = 480$ bits long resulting in a search space of $2^{480} = 3.12 \cdot 10^{144}$ different solutions.

B. Fitness Function

Here again, the fitness function that incorporates the varying penalty term for the constraints must be of the form

$$Q(S, g) = O(S) + P_{\mathrm{v}}(d(S), g) \cdot v \tag{14}$$

where $O(S)$ is the objective function, $P_{\mathrm{v}}(d(S), g)$ is the varying penalty term, $v = 1$ if the solution is invalid and $v = 0$ if the solution is valid. First we must define the objective function $O(S)$. The objective of the problem is to minimize the total power production cost, over the scheduling horizon, which consists of the total fuel cost $fc(S)$ the sum of the units' startup costs $su(S)$ and the sum of the units' shutdown costs $sd(S)$. With a given operating schedule, for every hour t, a dispatch algorithm [1] calculates the optimum power output values, PO_i^t, for every operating unit i in order to meet the given forecasted demand D^t at that hour. The operating units are the ones that have, their genotype bit corresponding to hour t equal to "1" (see Fig. 4). The high and low limits of the units are also taken into account.

The total fuel cost $fc(S)$ of the operating units is calculated by

$$fc(S) = \sum_{i=1}^{N} \sum_{t=1}^{H} u_i^t \cdot F_i(PO_i^t) \tag{15}$$

where u_i^t is the state (0, 1) of unit i at time t and $F_i(\cdot)$ is the fuel cost function of unit i.

The units' start-up cost $su(S)$ and shut-down cost $sd(S)$ are calculated by

$$su(S) = \sum_{i=1}^{N} \sum_{t=1}^{H} u_i^t \cdot (1 - u_i^{t-1}) \cdot SU_i \tag{16}$$

$$sd(S) = \sum_{i=1}^{N} \sum_{t=1}^{H} u_i^{t-1} \cdot (1 - u_i^t) \cdot SD_i \tag{17}$$

where SU_i is the start-up cost of unit i and SD_i is the shut-down cost of unit i. More details on the calculation of F_i, SU_i, and SD_i will be given in the Section V-E.

In order, again, to form the penalty function for the violation of the constraints, we must first define a measure $d(S)$ of the degree of constraint violation. Five constraints are considered in this paper: the power balance constraint, the reserve requirement constraint, the unit high and low MW limits, the minimum up time constraint and the minimum down time constraint. Additional constraints can be taken into account easily by adding an appropriate term that represents a measure of the degree of violation of the specific constraint.

The system power balance equation should be satisfied for every hour t

$$\sum_{i=1}^{N} (u_i^t \cdot PO_i^t) = D^t. \tag{18}$$

Also the system reserve requirements must be met

$$\sum_{i=1}^{N} (u_i^t \cdot PO_i \max) \geq D^t + R^t \tag{19}$$

where $PO_i \max$ is the maximum power output of unit i and R^t is the reserve requirement at hour t. In case, on the basis of a schedule, (18) and (19) are not satisfied at hour t, due to the limits of the operating units, i.e.,

$$\sum_{i=1}^{N} u_i^t \cdot PO_i \max < D^t + R^t \quad \text{or}$$
$$\sum_{i=1}^{N} u_i^t \cdot PO_i \min > D^t \tag{20}$$

where $PO_i \min$ is the minimum power output of unit i, then we have to calculate a measure, d_{pb}^t, of the degree of violation of the power balance and the reserve requirement constraints at hour t, to form the corresponding penalty term. d_{pb}^t is

calculated by

$$d_{pb}{}^t = D^t + R^t - \sum_{i=1}^{N} u_i^t \cdot PO_{i\max},$$

$$\text{if} \quad \sum_{i=1}^{N} u_i^t \cdot PO_i \max < D^t + R^t \quad \text{or,}$$

$$d_{pb}^t = \sum_{i=1}^{N} u_i^t \cdot PO_i \min - D^t \quad \text{if} \quad \sum_{i=1}^{N} u_i^t \cdot PO_i \min > D^t. \tag{21}$$

It should be noted that invalid solutions are not discarded but are left to compete according to their quality.

Finally a measure of the degree of violation of the "minimum up-time" constraint, d_{mu}, and the "minimum down-time" constraint, d_{md}, are calculated. According to a specific solution, for every unit string i, a number K_i^{up} of time-zones of continuous operation are scheduled, interrupted by a number K_i^{dn} of time zones, during which the unit i is continuously down. These zones are displayed in Fig. 5. Every time zone $z_{ij}^{up}, j = 1 \cdots K_i^{up}$ consists of a number of hours $h(z_{ij}^{up})$ where the unit i is continuously up. Also every time zone $z_{ij}^{dn}, j = 1 \cdots K_i^{dn}$ consists of a number of hours $h(z_{ij}^{dn})$ where the unit i is continuously down. For a solution to be valid, $h(z_{ij}^{up}) \geq M_i^{up}$, for $i = 1 \cdots N$ and $j = 1 \cdots K_i^{up}$, where M_i^{up} is the "minimum up-time" for unit i. d_{mu} is then given by

$$d_{mu} = \sum_{i=1}^{N} \sum_{j=1}^{K_i^{up}} ((M_i^{up} - h(z_{ij}^{up})) \cdot r_{ij}) \tag{22}$$

with

$$r_{ij} = \begin{cases} 1, & \text{if } h(z_{ij}^{up}) < M_i^{up} \\ 0, & \text{otherwise.} \end{cases}$$

The degree of violation of the "minimum down-time" constraint, d_{md}, is calculated through (22) by simply substituting the symbol "dn" for the symbol "up". However, for clarity reasons, the corresponding equation is given below. For a solution to be valid $h(z_{ij}^{dn}) \geq M_i^{dn}$, for $i = 1 \cdots N$ and $j = 1 \cdots K_i^{dn}$, where M_i^{dn} is the "minimum down-time" for unit i. d_{md} is then given by

$$d_{md} = \sum_{i=1}^{N} \sum_{j=1}^{K_i^{dn}} ((M_i^{dn} - h(z_{ij}^{dn})) \cdot r_{ij}) \tag{23}$$

with

$$r_{ij} = \begin{cases} 1, & \text{if } h(z_{ij}^{dn}) < M_i^{dn} \\ 0, & \text{otherwise.} \end{cases}$$

So the overall measure, $d(S)$, of the degree of the constraint violation is given by

$$d(S) = \sum_{t=1}^{H} (d_{pb}^t) + d_{mu} + d_{md}. \tag{24}$$

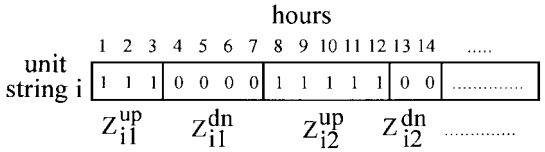

Fig. 5. The zones of operation and nonoperation in the scheduling of unit i.

Then a varying penalty term similar to that of (5) is formed

$$P_{\text{v}}(d(S), g) = \frac{g}{G} \cdot A \cdot \left(\sum_{t=1}^{H} (d_{pb}^t) + d_{mu} + d_{md} \right) + \frac{g}{G} \cdot B. \tag{25}$$

B should satisfy (6).

The corresponding nonvarying penalty term is of the form

$$P(d(S)) = A \cdot \left(\sum_{t=1}^{H} (d_{pb}^t) + d_{mu} + d_{md} \right) + B. \tag{26}$$

In case that a number of constraints contribute to the penalty function [as in (25) and (26)] there is usually the problem of constraint normalization or weighting, as the constraints may be violated by quantities that have large scaling differences. In such cases, the GA gets "more interested" in satisfying the constraints with high penalty values, and consequently, the rest of the constraints will be satisfied with low priority, or not satisfied at all. In general, the constraint penalties should reflect how important or rather how difficult a specific constraint is. The importance of individual constraints may even be determined adaptively [9]. In the penalty functions of (25) and (26) the sum of the various degree-of-constraint-violation quantities is not weighted, since their values were of the same scale (d_{pb} is in MW, d_{mu} and d_{md} are in hours) and the constraints were considered of equal importance or difficulty. Finally, the fitness function is given by summing all the cost and penalty terms

$$Q(S, g) = fc(S) + su(S) + sd(S) + P_{\text{v}}(d(S), g) \cdot v \tag{27}$$

where $v = 1$ if the solution is invalid and $v = 0$ if the solution is valid. The GA has to find the optimum unit commitment schedule that minimizes the fitness function $Q(S, g)$ (27).

C. The GA Implementation

The GA implemented for the UC problem, used the same techniques as the ones used for the Cutting Stock problem, which are described in Subsection IV-C. The techniques are again, the Roulette Wheel parent selection mechanism, multipoint crossover, binary mutation, generational replacement, elitism, fitness scaling and adaptation of operator probabilities.

D. Additional Operators

The algorithm's search efficiency is strengthened by using additional operators described below.

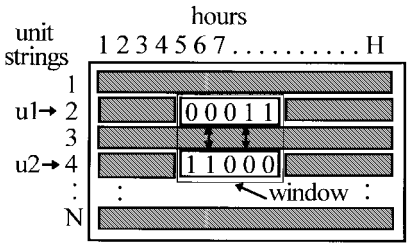

Fig. 6. The swap-window operator.

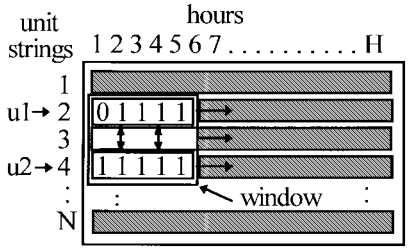

Fig. 7. The swap-window hill-climbing operator.

1) Swap-Window Operator: This operator is applied to all the population genotypes with a probability of 0.2. It selects two arbitrary unit strings u1, u2, a "time window" of random width w $(0 < w \leq H)$ and a random window position taking values in the range $1 \cdots (H - w)$. Then the bits of the two unit strings u1, u2 within the window are interchanged. The function of this operator is displayed in Fig. 6.

2) Window-Mutation Operator: This operator is also applied to all the population genotypes with a probability of 0.2. It selects a random unit string, a "time window" of random width w $(0 < w \leq H)$ and a random window position in the range $1 \cdots (H - w)$. Then it mutates all the bits within the window turning all of them to 1 or all of them to 0 with equal probability.

3) Swap-Mutation Operator: This operator is applied only to the best genotype of every generation. For a given hour t, it performs with equal probability one of the following two operations: 1) it selects two arbitrary unit strings and swaps their bits corresponding to hour t, or 2) it selects a single arbitrary unit string and flips its bit corresponding to hour t, from "0" to "1" or vice versa. If a better solution is produced, it replaces the old one otherwise it is restored to its initial state. This procedure is repeated for every value of t in the range $1 \cdots H$.

4) Swap-Window Hill—Climbing Operator: This operator is also applied only to the best genotype of every generation with a probability of 0.3. It selects two arbitrary unit strings u1, u2 and a "time window" of length w $(0 < w \leq H)$. The time window is first positioned at the first hour of the scheduling period. Then the bits of the two unit strings u1, u2 within the window are exchanged. If a better solution is produced it replaces the old one otherwise it is restored to its initial state. Then the window is shifted one hour up and the above procedure is repeated until the window reaches the last hour of the scheduling period. The function of this operator as explained above is displayed in Fig. 7.

In this problem too, these additional operators were incorporated into the GA in both cases: the varying and the conventional (nonvarying) fitness function implementations.

TABLE II
UNIT PARAMETERS OF THE 10-UNIT COMMITMENT PROBLEM

Unit	Pmax (MW)	Pmin (MW)	c_1 ($)	c_2 ($/MWh)	c_3 ($/MW^2-h)
1	455	150	1000	16.19	0.00048
2	455	150	970	17.26	0.00031
3	130	20	700	16.60	0.00200
4	130	20	680	16.50	0.00211
5	162	25	450	19.70	0.00398
6	80	20	370	22.26	0.00712
7	85	25	480	27.74	0.00079
8	55	55	660	25.92	0.00413
9	55	55	665	27.27	0.00222
10	55	55	670	27.79	0.00173

Unit	min up time (h)	min down time (h)	start -up cost ($)	initial status (h)
1	8	8	9000	8
2	8	8	10000	8
3	5	5	1100	-5
4	5	5	1120	-5
5	6	6	1800	-6
6	3	3	340	-3
7	3	3	520	-3
8	1	1	60	-1
9	1	1	60	-1
10	1	1	60	-1

E. Simulation Results

The varying fitness function technique (27) that uses the varying penalty term of (25) has been tested on two UC problems: a 10-unit problem and a 20-unit problem. In both cases the results obtained through the varying fitness function are compared with those obtained through the nonvarying fitness function [with the penalty term of (26)]. Each of the 10 units of the first problem has a fuel cost function of the form $F_i(PO) = c_1 + c_2 \cdot PO + c_3 \cdot PO^2$, where PO is the power output and c_1, c_2, c_3 are fuel cost function coefficients. The resulting fuel cost values as well as the start-up costs, are in U.S. dollars. The scheduling period is 24 h. The parameters for each unit are shown in Table II and the demand data in MW are shown in Table III.

The units' start up cost SU is defined as a fixed amount per start-up. The shut-down cost has been taken equal to 0 for every unit. The "initial status" figure, if it is positive, indicates the number of hours the unit is already up, and if it is negative, indicates the number of hours the unit has been already down.

The 20-unit problem has been generated from the 10-unit problem by simply duplicating the generating units and doubling the demand values. The implementations for both techniques (the nonvarying fitness function GA and the varying fitness function GA), used the same operators described above

TABLE III
DEMAND DATA OF THE 10-UNIT COMMITMENT PROBLEM

Hour	demand (MW)	Hour	demand (MW)
1	800	13	1500
2	850	14	1400
3	950	15	1300
4	1050	16	1150
5	1100	17	1100
6	1200	18	1200
7	1250	19	1300
8	1300	20	1500
9	1400	21	1400
10	1500	22	1200
11	1550	23	1000
12	1600	24	900

and the same number of generations. For the 10-unit problem the GA has run for 500 generations and for the 20-unit problem for 750 generations. In all cases the population has been 50 genotypes. Twenty runs have been performed for each technique and problem.

In order to judge the success of the varying fitness and nonvarying fitness function techniques we have compared the results with what we call the estimated optimum. The estimated optimum in the case of the 10-unit problem coincides with the actual optimum which has been calculated using a Dynamic Programming algorithm (DP). The problem, however, of the DP methods is the exponential growth of the search space as a function of the input dimensions. Therefore, in the case of the 20-unit problem, we could not use the DP algorithm for the calculation of the actual optimum, due to the excessive computation time and storage required. So we had to rely on the estimated optimum which in this case has been calculated by doubling the actual optimum of the 10-unit problem, which is slightly higher than the actual 20-unit optimum.

A comparison of the progress of the two techniques regarding the 10-unit problem and the 20-unit problem are shown in Figs. 8 and 9, respectively. Fig. 8 displays the progress of the GA with the varying fitness function and the GA with the nonvarying fitness function on the 10-unit problem. The two trajectories are computed by averaging the corresponding progress trajectories over all twenty runs. It is clear that the varying fitness function GA finds better solutions more quickly than its nonvarying counterpart and manages to reach the optimum within the limit of 500 generations. The nonvarying fitness function GA doesn't reach the optimum in 500 generations and may need twice as much to finally reach it. Fig. 9 displays the progress of the varying and the nonvarying fitness function GA's on the 20-unit problem. Again here, the two trajectories are computed by averaging the corresponding progress trajectories over all twenty runs. Although the two trajectories are almost identical in the beginning, the varying fitness function trajectory quickly falls below the one of its nonvarying counterpart. The difference here between the trajectories seems smaller than that of the 10-unit case but this is due to the larger operating cost scale. As seen in the

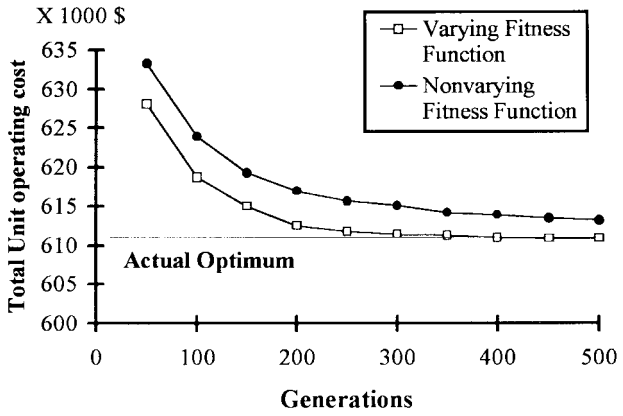

Fig. 8. Average progress of the two techniques on the 10-unit problem.

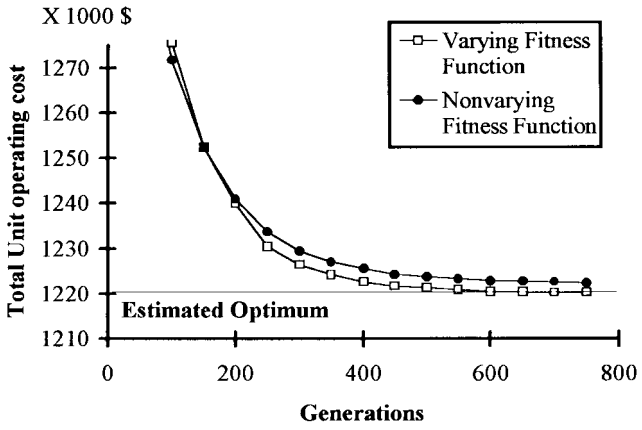

Fig. 9. Average progress of the two techniques on the 20-unit problem.

figure the varying fitness function GA manages to reach the estimated optimum within the limit of 750 generations while the nonvarying counterpart does not.

Details of the simulation results are shown in Table IV. A run is considered successful if it obtains a solution equal to or better than the estimated optimal solution. The dispersion figure in Table IV expresses the difference between the best and the worst solution obtained among the 20 runs as a percentage of the best solution. The varying fitness function GA outperforms again its nonvarying counterpart, while requiring the same computational time. Again here, slight differences in the average time figures between the two algorithms are due to the stochastic nature of the GA execution. The simulation examples have run on a HP Apollo 720 workstation.

VI. CONCLUSIONS

The choice of appropriate penalty terms for constrained optimization is a serious problem. Large constraint penalties separate the invalid solutions from the valid ones but lead to a more complicated hypersurface to be searched, whereas small penalties result in a smoother hypersurface but increase the possibility of misleading the GA toward invalid solutions. An answer to this problem can be the use of varying penalty terms, less stringent at the beginning

TABLE IV
SIMULATION RESULTS OF THE UNIT COMMITMENT PROBLEM

Results over 20 runs	10-unit problem	
	Nonvarying Fitness Function GA	Varying Fitness Function GA
Est.optimum ($)	610646.5	
Success	0 %	65 %
Best Quality ($)	611758.1	610646.5
Worst Qual. ($)	615199.9	612326.6
Dispersion	0.56 %	0.27 %
Average Time	262 sec.	258 sec.
Results over 20 runs	20-unit problem	
	Nonvarying Fitness Function GA	Varying Fitness Function GA
Est.optimum ($)	1221293	
Success	40 %	80 %
Best Quality ($)	1219314	1218922
Worst Qual. ($)	1226840	1223426
Dispersion	0.62 %	0.37 %
Average Time	633 sec.	626 sec.

and rising gradually to appropriately large values at later stages. The penalty terms used are linearly proportional to the generation index. The most effective penalty function form (e.g., a quadratic function might be better than a linear one) is an open question and further research is required in that direction. The presented technique gives the GA a significantly better chance of locating the global optimum especially in case of problems with many constraints that result in a complicated search hypersurface. The results show that the varying fitness function outperforms the traditional nonvarying fitness function technique.

REFERENCES

[1] A. Bakirtzis, V. Petridis, and S. Kazarlis, "A genetic algorithm solution to the economic dispatch problem," *Proc. Inst. Elect. Eng.*, vol. 141, pp. 377–382, July 1994.

[2] C. J. Baldwin, K. M. Dale, and R. F. Dittrich, "A study of economic shutdown of generating units in daily dispatch," *AIEE Trans. Power Apparat. Syst.*, vol. 78, pp. 1272–1284, 1960.

[3] L. F. B. Baptistella and J. C. Geromel, "A decomposition approach to problem of unit commitment schedule for hydrothermal systems," *Proc. Inst. Elect. Eng.*, vol. 127, pt. D, pp. 250–258, Nov. 1980.

[4] A. R. Brown, *Optimum Packing and Depletion.* New York: American Elsevier, 1971.

[5] A. I. Cohen and M. Yoshimura, "A branch-and-bound algorithm for unit commitment," *IEEE Trans. Power Apparat. Syst.*, vol. PAS-102, pp. 444–451, Feb. 1983.

[6] L. Davis, "Adapting operator probabilities in genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms Applications*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, June 1989, pp. 61–69.

[7] L. Davis, Ed., *Handbook of Genetic Algorithms.* New York: Van Nostrand, 1991.

[8] H. Dyckhoff, "A typology of cutting and packing problems," *Eur. J. Oper. Res.*, vol. 44, pp. 145–159, 1990.

[9] A. E. Eiben and Z. Ruttkay, "Self-adaptivity for constraint satisfaction: Learning penalty functions," in *Proc. 3rd IEEE Conf. Evolutionary Computation*, IEEE Service Center, 1996, pp. 258–261.

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

[11] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms.* San Mateo CA: Morgan Kaufmann, 1991, pp. 69–93.

[12] R. Gunter, "Convergence analysis of canonical genetic algorithms," *IEEE Trans. Neural Networks*, vol. 5, pp. 96–101, Jan. 1994.

[13] J. H. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: Univ. Michigan Press, 1975.

[14] J. A. Joines and C. R. Houck, "On the use of nonstationary penalty functions to solve nonlinear constrained optimization problems with GA's," in *Proc. 1st IEEE Conf. Evolutionary Computation*, IEEE Service Center, 1994, pp. 579–584.

[15] S. A. Kazarlis, A. G. Bakirtzis, and V. Petridis, "A genetic algorithm solution to the unit commitment problem," *IEEE Trans. Power Syst.*, vol. 11, pp. 83–92, Feb. 1996.

[16] K. Y. Lee, X. Bai, and Y. Park, "Optimization method for reactive power planning by using a modified genetic algorithm," *IEEE Trans. Power Syst.*, vol. 10, pp. 1843–1850, Nov. 1995.

[17] A. Merlin and P. Sandrin, "A new method for unit commitment at electricite de France," *IEEE Trans. Power Apparat. Syst.*, vol. PAS-102, pp. 1218–1225, May 1983.

[18] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd ed. Berlin, Germany: Springer-Verlag, 1992.

[19] Z. Michalewicz and N. Attia, "Evolutionary optimization of constrained problems," in *Proc. 3rd Annu. Conf. Evolutionary Programming*, A. V. Sebald and L. J. Fogel, Eds. River Edge, NJ: World Scientific, 1994, pp. 98–108.

[20] J. A. Miller, W. D. Potter, R. V. Gandham, and C. N. Lapena, "An evaluation of local improvement operators for genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 1340–1351, Sept./Oct. 1993.

[21] V. Miranda, J. V. Rantino, and L. M. Proenca, "Genetic algorithms in optimal multistage distribution network planning," *IEEE Trans. Power Syst.*, vol. 9, pp. 1927–1934, Nov. 1994.

[22] R. Nakano and T. Yamada, "Conventional genetic algorithm for job shop problems," in *Proc. 4th Int. Conf. Genetic Algorithms Applications.* San Mateo, CA: Morgan Kaufmann, 1991, pp. 474–479.

[23] D. Orvosh and L. Davis, "Using a genetic algorithm to optimize problems with feasibility constraints," in *Proc. 1st IEEE Conf. Evolutionary Computation*, IEEE Service Center, 1994, pp. 548–553.

[24] C. K. Pang and H. C. Chen, "Optimal short-term thermal unit commitment," *IEEE Trans. Power Apparat. Syst.*, vol. PAS-95, pp. 1336–1346, July–Aug. 1976.

[25] V. Petridis and S. Kazarlis, "Varying quality function in genetic algorithms and the cutting problem," in *Proc. 1st IEEE Conf. Evolutionary Computation*, IEEE Service Center, 1994, vol. 1, pp. 166–169.

[26] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard, "Some guidelines for genetic algorithms with penalty functions," in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, June 1989, pp. 191–197.

[27] H. Sasaki, M. Watanabe, and R. Yokoyama, "A solution method of unit commitment by artificial neural networks," *IEEE Trans. Power Syst.*, vol. 7, pp. 974–981, Aug. 1992.

[28] W. Siedlecki and J. Sklanski, "Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition," in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, June 1989, pp. 141–150.

[29] A. E. Smith and D. M. Tate, "Genetic optimization using a penalty function," in *Proc. 5th Int. Conf. Genetic Algorithms*, S. Forrest, Ed. Los Altos, CA: Morgan Kaufmann, 1993, pp. 499–505.

[30] W. M. Spears and K. A. De Jong, "An analysis of multipoint crossover," in *Foundations of Genetic Algorithms*, G. Rawlins, Ed. San Mateo, CA: Morgan Kaufman, 1991, pp. 301–315.

[31] P. E. Sweeney and E. Ridenour Paternoster, "Cutting and packing problems: A categorized, application—Oriented research bibliography," *J. Oper. Res. Soc.*, vol. 43, no. 7, pp. 691–706, 1992.

[32] C. Wang and S. M. Shahidehpour, "A decomposition approach to nonlinear multiarea generation scheduling with tie-line constraints using expert systems," *IEEE Trans. Power Syst.*, vol. 7, pp. 1409–1418, Nov. 1992.

[33] H. Wang, Z. Ma, and K. Nakayama, "Effectiveness of penalty function in solving the subset sum problem," in *Proc. 3rd IEEE Conf. Evolutionary Computation*, IEEE Service Center, 1996, pp. 422–425.

[34] A. J. Wood and B. F. Wollenberg, *Power Generation Operation and Control.* New York: Wiley, 1984.

[35] F. Zhuang and F. D. Galiana, "Unit commitment by simulated annealing," *IEEE Trans. Power Syst.*, vol. 5, pp. 311–318, Feb. 1990.

Vassilios Petridis (M'77) received the diploma in electrical engineering from the National Technical University, Athens, Greece, in 1969, and the M.Sc. and Ph.D. degrees in electronics and systems from King's College, University of London, London, U.K., in 1970 and 1974, respectively.

He has been Consultant of the Naval Research Centre in Greece, Director of the Department of Electronics and Computer Engineering, and Vice-Chairman of the Faculty of Electrical and Computer Engineering with Aristotle University, Thessaloniki, Greece. He is currently Professor in the Department of Electronics and Computer Engineering, Aristotle University. He is the author of three books on control and measurement systems and more than 85 research papers. His research interests include control systems, intelligent and autonomous systems, artificial neural networks, evolutionary algorithms, modeling and identification, robotics, and industrial automation.

Spyros Kazarlis was born in Thessaloniki, Greece, in June 1966. He received the Dipl. Eng. degree from the Department of Electrical Engineering, Aristotle University, Thessaloniki, in 1990 and the Ph.D. degree from the same university in 1998.

Since 1986, he has been working as a Computer Analyst, Programmer, and Lecturer for public and private companies. Since 1990, he has also been working as a Researcher at Aristotle University. His research interests are in evolutionary computation (genetic algorithms, evolutionary programming, etc.), artificial neural networks, software engineering and computer technology.

Dr. Kazarlis is a Member of the Society of Professional Engineers of Greece and the Research Committee of EVONET.

Anastasios Bakirtzis (S'77–M'79–SM'95) was born in Serres, Greece, in February 1956. He received the Dipl. Eng. degree from the Department of Electrical Engineering, National Technical University, Athens, Greece, in 1979 and the M.S.E.E. and Ph.D. degrees from Georgia Institute of Technology, Atlanta, in 1981 and 1984, respectively.

In 1984, was a Consultant to Southern Company. Since 1986, he has been with the Electrical Engineering Department, Aristotle University of Thessaloniki, Greece, where he is an Associate Professor. His research interests are in power system operation and control, reliability analysis and in alternative energy sources.

Dr. Bakirtzis is a member of the Society of Professional Engineers of Greece.