Evolutionary Algorithms for Constrained Engineering Problems

Zbigniew Michalewicz,* Dipankar Dasgupta,† Rodolphe G. Le Riche,‡ and Marc Schoenauer§

Abstract

Evolutionary computation techniques have been receiving increasing attention regarding their potential as optimization techniques for complex problems. Recently these techniques were applied in the area of industrial engineering; the most-known applications include scheduling and sequencing in manufacturing systems, computer-aided design, facility layout and location problems, distribution and transportation problems, and many others.

Industrial engineering problems usually are quite hard to solve due to a high complexity of the objective functions and a significant number of problem-specific constraints; often an algorithm to solve such problem requires incorporation of some heuristic methods.

In this paper we concentrate on constraint handling heuristics for evolutionary computation techniques. This general discussion is followed by three test case studies: truss structure optimization problem, design of a composite laminated plate, and the unit commitment problem. These are typical highly constrained engineering problems and the methods discussed here are directly transferrable to industrial engineering problems.

1 Introduction

Evolutionary computation techniques have drawn much attention as optimization methods in the last two decades [33, 39, 16]. Evolutionary computation algorithms are stochastic optimization methods; they are conveniently presented using the metaphor of natural evolution: a randomly initialized *population of individuals* (set of points of the search space at hand) evolves following a crude parody of the Darwinian principle of *the survival of the fittest*. New individuals are generated using simulated genetic operations such as mutation and crossover. The probability of survival of the newly generated solutions depends on their *fitness* (how well they perform with respect to the optimization problem at hand): the best are kept with a high probability, the worst are rapidly discarded. Three main

* Dept. of Computer Science, University of North Carolina, Charlotte, NC 28223, USA; e-mail: *zbyszek@uncc.edu* and Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland; e-mail: *zbyszek@ipipan.waw.pl.*

† Department of Mathematics and Computer Science, University of Missouri, St. Louis, MO 63121, USA; e-mail: *dasgupta@arch.umsl.edu.*

‡ Université de Technologie de Compiègne, LG2mS Laboratory, Division MNM, Compiègne 60200, France; e-mail: *leriche@univ-compiegne.fr.*

§ Ecole Polytechnique, CNRS–CMAP, Palaiseau 91128, France; e-mail: *marc.schoenauer@polytechnique.fr.*

algorithmic trends are based on such an evolutionary scheme: Genetic Algorithms (GA) [24, 18], Evolutionary Strategies (ES) [54, 1] and Evolutionary Programming (EP) [15, 16].

From the optimization point of view, one of the main advantages of evolutionary computation techniques is that they do not have much mathematical requirements about the optimization problem. They are 0–order methods (all they need is an evaluation of the objective function), they can handle nonlinear problems, defined on discrete, continuous or mixed search spaces, unconstrained or constrained. Moreover, the ergodicity of the evolution operators makes them global in scope (in probability).

Many industrial engineering activities involve unstructured, real-life problems that are hard to model, since they require inclusion of unusual factors (from accident risk factors to esthetics). Other industrial engineering problems are complex in nature: job shop scheduling, timetabling, traveling salesman or facility layout problems are examples of *NP-complete* problems. In both cases, evolutionary computation techniques represent a potential source of actual breakthroughs. Their ability to provide many near-optimal solutions at the end of an optimization run enables to choose the best solution afterwards, according to criteria that were either inarticulate from the expert, or badly modeled. Evolutionary algorithms can be made efficient because they are flexible, and relatively easy to hybridize with domain-dependent heuristics. Those features of evolutionary computation have already been acknowledged in the field of industrial engineering, and many applications have been reported (see, for example, [18, 47, 57]).

A vast majority of industrial engineering optimization problems are constrained problems. The presence of constraints significantly affects the performance of any optimization algorithm, including evolutionary search methods [34]. This paper focuses on the issue of constraints handling in evolutionary computation techniques. The general way of dealing with constraints — whatever the optimization method — is by penalizing infeasible points. However, there are no guidelines on designing penalty functions. Some suggestions for evolutionary algorithms are given in [50], but they do not generalize. Other techniques that can be used to handle constraints in evolutionary computation techniques are more or less problem dependent. For instance, the knowledge about linear constraints can be incorporated into specific operators [37], or a repair operator can be designed that projects infeasible points onto feasible ones [42]; Section 2 provides a general overview of constraints-handling methods for evolutionary computation techniques. Each of the next three sections presents a specific case study of both an engineering constrained optimization problem and a constraint handling algorithm for evolutionary algorithms. Section 3 introduces a truss structure optimization problem, for which results are obtained using a constraint handling method based on an evolutionary sampling of the feasible region; Section 4 presents an evolutionary method for the design of composite laminated plate, using the segregated genetic algorithm, where the difficult task of adjusting penalty parameters is avoided by using two sub-populations, evolving according to two slightly different fitness functions; Section 5 is devoted to the unit commitment problem, a highly-constrained nonlinear optimization problem. Section 6 concludes the paper.

2 Evolutionary Computation and Constrained Optimization

In this section we discuss several methods for handling feasible and infeasible solutions in a population; most of these methods emerged quite recently. Only a few years ago Richardson et al. [50] claimed: "Attempts to apply GA's with constrained optimization problems follow two different paradigms (1) modification of the genetic operators; and (2) penalizing strings which fail to satisfy all the constraints." This is no longer the case as a variety of heuristics have been proposed. Even the category of penalty functions consists of several methods which differ in many important details on how the penalty function is designed and applied to infeasible solutions. Other methods maintain the feasibility of the individuals in the population by means of specialized operators or decoders, impose a restriction that any feasible solution is 'better' than any infeasible solution, consider constraints one at the time in a particular linear order, repair infeasible solutions, use multiobjective optimization techniques, are based on cultural algorithms (i.e., algorithms with an additional layer of beliefs which undergoes evolution as well [48]), or rate solutions using a particular co-evolutionary model (i.e., model with more than one population, where the fitness of an individual in one population depends on the current state of evolution in the other population [46]).

2.1 Rejection of infeasible individuals

This "death penalty" heuristic is a popular option in many evolutionary techniques (e.g., evolution strategies). Note that rejection of infeasible individuals offers a few simplifications of the algorithm: for example, there is no need to evaluate infeasible solutions and to compare them with feasible ones.

The method of eliminating infeasible solutions from a population may work reasonably well when the feasible search space is convex and it constitutes a reasonable part of the whole search space (e.g., evolution strategies do not allow equality constraints since with such constraints the ratio between the sizes of feasible and infeasible search spaces is zero). Otherwise such an approach has serious limitations. For example, for many search problems where the initial population consists of infeasible individuals only, it might be essential to improve them (as opposed to rejecting them). Moreover, quite often the system can reach the optimum solution easier if it is possible to "cross" an infeasible region (especially in non-convex feasible search spaces).

2.2 Penalizing infeasible individuals

This is the most common approach in the genetic algorithms community. The domain of the objective function f is extended; the approach assumes that

$$eval(p) = f(p) \pm Q(p),$$

where $Q(p)$ represents either a penalty for infeasible individual p, or a cost for repairing such an individual. The major question is, how should such a penalty function $Q(p)$ be designed? The intuition is simple: the penalty should be kept as low as possible, just above the limit below which infeasible solutions are optimal (so-called *minimal penalty rule*) [29]. However, it is difficult to implement this rule effectively.

The relationship between infeasible individual 'p' and the feasible part of the search space plays a significant role in penalizing such individuals: an individual might be penalized just for being infeasible, the 'amount' of its infeasibility is measured to determine the penalty value, or the effort of 'repairing' the individual might be taken into account.

Several researchers studied heuristics on design of penalty functions. Some hypotheses were formulated [50]:

- "penalties which are functions of the distance from feasibility are better performers than those which are merely functions of the number of violated constraints,
- for a problem having few constraints, and few full solutions, penalties which are solely functions of the number of violated constraints are not likely to find solutions,
- good penalty functions can be constructed from two quantities, the *maximum completion cost* and the *expected completion cost*,
- penalties should be close to the *expected completion cost*, but should not frequently fall below it. The more accurate the penalty, the better will be the solutions found. When penalty often underestimates the completion cost, then the search may not find a solution."

and in [55]:

- "the genetic algorithm with a variable penalty coefficient outperforms the fixed penalty factor algorithm,"

where a variability of penalty coefficient was determined by a heuristic rule.

This last observation was further investigated by Smith and Tate [56]. In their work they experimented with dynamic penalties, where the penalty measure depends on the number of violated constraints, the best feasible objective function found, and the best objective function value found.

For numerical optimization problems,

$$\text{optimize } f(\overline{X}),\ \overline{X} = (x_1, \ldots, x_n) \in R^n,$$

where

$$g_j(\overline{X}) \leq 0, \text{ for } j = 1, \ldots, q, \text{ and } h_j(\overline{X}) = 0, \text{ for } j = q+1, \ldots, m,$$

penalties usually incorporate degrees of constraint violations. Most of these methods use constraint violation measures f_j (for the j-th constraint) for the construction of the *eval*; these functions are defined as

$$f_j(\overline{X}) = \begin{cases} \max\{0, g_j(\overline{X})\}, & if\ 1 \leq j \leq q \\ |h_j(\overline{X})|, & if\ q+1 \leq j \leq m \end{cases}$$

For example, Homaifar et al. [25] assume that for every constraint we establish a family of intervals that determines appropriate penalty values. A similar approach is presented in Section 5 of this paper (for the unit commitment problem).

It is also possible (as suggested in [55] to adjust penalties in a dynamic way, taking into account the current state of the search or the generation number. For example, Joines and Houck [26] assumed dynamic penalties; individuals are evaluated (at the iteration t) by the following formula:

$$eval(\overline{X}) = f(\overline{X}) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(\overline{X}),$$

where C, α and β are constants.

Michalewicz and Attia [36] considered a method based on the idea of simulated annealing: the penalty coefficients are changed once in many generations (after the convergence of the algorithm to a local optima). At every iteration the algorithm considers active constraints only, the pressure on infeasible solutions is increased due to the decreasing values of the temperature of the system.

A method of adapting penalties was developed by Bean and Hadj-Alouane [4, 20]. As the previous method, it uses a penalty function, however, one component of the penalty function takes a feedback from the search process. Each individual is evaluated by the formula:

$$eval(\overline{X}) = f(\overline{X}) + \lambda(t) \sum_{j=1}^{m} f_j^2(\overline{X}),$$

where $\lambda(t)$ is updated every generation t in the following way:

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t), & if\ case\#1 \\ \beta_2 \cdot \lambda(t), & if\ case\#2 \\ \lambda(t), & otherwise, \end{cases}$$

where cases #1 and #2 denote situations where the best individual in the last k generation was always (case #1) or was never (case #2) feasible, $\beta_1, \beta_2 > 1$, and $\beta_1 \neq \beta_2$ (to avoid cycling).

Yet another approach was proposed recently by Le Riche et al. [29]. The authors designed a (segregated) genetic algorithm which uses two values of penalty parameters (for each constraint) instead of one; these two values aim at achieving a balance between heavy and moderate penalties by maintaining two subpopulations of individuals. The population is split into two cooperating groups, where individuals in each group are evaluated using either one of the two penalty parameters. This method is discussed in detail in Section 4 of this paper.

Some researchers [47, 40] reported good results of their evolutionary algorithms, which worked under the assumption that any feasible individual was better than any infeasible one. Powell and Skolnick [47] applied this heuristic rule for the numerical optimization problems: evaluations of feasible solutions were mapped into the interval $(-\infty, 1)$ and infeasible solutions—into the interval $(1, \infty)$ (for minimization problems). Michalewicz and Xiao [40] experimented with the path planning problem and used two separate evaluation functions for feasible and infeasible individuals. The values for infeasible solutions were increased (i.e., made less attractive) by adding such a constant, so that the best infeasible individual was worse that the worst feasible one.

It seems that the appropriate choice of the penalty method may depend on (1) the ratio between sizes of the feasible and the whole search space, (2) the topological properties of the feasible search space, (3) the type of the objective function, (4) the number of variables, (5) number of constraints, (6) types of constraints, and (7) number of active constraints at the optimum. Thus the use of penalty functions is not trivial and only some partial analysis of their properties is available. Also,

a promising direction for applying penalty functions is the use of adaptive penalties: penalty factors can be incorporated in the chromosome structures in a similar way as some control parameters are represented in the structures of evolution strategies and evolutionary programming.

2.3 Maintaining feasible population by special representations and genetic operators

One reasonable heuristic for dealing with the issue of feasibility is to use specialized representation and operators to maintain the feasibility of individuals in the population. During the last decade several specialized systems were developed for particular optimization problems; these systems use a unique chromosomal representations and specialized 'genetic' operators which alter their composition. Some of such systems were described in [12]; other examples include Genocop for optimizing numerical functions with linear constraints and Genetic-2N [33] for nonlinear transportation problem. For example, Genocop assumes linear constraints only and a feasible starting point (or feasible initial population). A closed set of operators maintains feasibility of solutions. For example, when a particular component x_i of a solution vector $\overline{X}$ is mutated, the system determines its current domain $dom(x_i)$ (which is a function of linear constraints and remaining values of the solution vector $\overline{X}$) and the new value of x_i is taken from this domain (either with flat probability distribution for uniform mutation, or other probability distributions for non-uniform and boundary mutations). In any case the offspring solution vector is always feasible. Similarly, arithmetic crossover of two feasible solution vectors $\overline{X}$ and $\overline{Y}$ yields always a feasible solution (for $0 \leq a \leq 1$) in convex search spaces (the system assumes linear constraints only which imply convexity of the feasible search space).

Often such systems are much more reliable than any other evolutionary techniques based on penalty approach. This is a quite popular trend: many practitioners use problem-specific representations and specialized operators in building very successful evolutionary algorithms in many areas; these include numerical optimization, machine learning, optimal control, cognitive modeling, classic operation research problems (traveling salesman problem, knapsack problems, transportation problems, assignment problems, bin packing, scheduling, partitioning, etc.), engineering design, system integration, iterated games, robotics, signal processing, and many others.

2.4 Repair of infeasible individuals

Repair algorithms enjoy a particular popularity in the evolutionary computation community: for many combinatorial optimization problems (e.g., traveling salesman problem, knapsack problem, set covering problem, etc.) it is relatively easy to 'repair' an infeasible individual. Such a repaired version can be used either for evaluation only, or it can also replace (with some probability) the original individual in the population.

The weakness of these methods is in their problem dependence. For each particular problem a specific repair algorithm should be designed. Moreover, there are no standard heuristics on design of such algorithms: usually it is possible to use a greedy repair, random repair, or any other heuristic which would guide the repair process. Also, for some problems the process of repairing infeasible individuals might be as complex as solving the original problem. This is the case for the nonlinear transportation problem [33], most scheduling and timetable problems, and many others.

On the other hand, the recently completed Genocop III system [38] for constrained numerical optimization (nonlinear constraints) is based on repair algorithms. Genocop III incorporates the original Genocop system [33] (which handles linear constraints only), but also extends it by maintaining two separate populations, where a development in one population influences evaluations of individuals in the other population. The first population P_s consists of so-called search points which satisfy linear constraints of the problem; the feasibility (in the sense of linear constraints) of these points is maintained by specialized operators (as in Genocop). The second population, P_r, consists of fully feasible reference points. These reference points, being feasible, are evaluated directly by the objective function, whereas search points are "repaired" for evaluation. The first results are very promising [38].

2.5 Replacement of individuals by their repaired versions

The question of replacing repaired individuals is related to so-called *Lamarckian evolution*, which assumes that an individual improves during its lifetime and that the resulting improvements are coded back into the chromosome.

Recently Orvosh and Davis [42] reported a so-called *5%-rule*: this heuristic rule states that in many combinatorial optimization problems, an evolutionary computation technique with a repair algorithm provides the best results when 5% of repaired individuals replace their infeasible originals. In continuous domains, a new replacement rule is emerging. As mentioned earlier, the Genocop III system for constrained numerical optimization problems with nonlinear constraints is based on repair approach. The first experiments (based on 10 test cases which have various numbers of variables, constraints, types of constraints, numbers of active constraints at the optimum, etc.) indicate that the 15% replacement rule is a clear winner: the results of the system are much better than with either lower or higher values of the replacement rate.

At present, it seems that the 'optimal' probability of replacement is problem-dependent and it may change over the evolution process as well. Further research is required for comparing different heuristics for setting this parameter, which is of great importance for all repair-based methods.

2.6 Use of decoders

Decoders offer an interesting option for all practitioners of evolutionary techniques. In these techniques a chromosome "gives instructions" on how to build a feasible solution. For example, a sequence of items for the knapsack problem can be interpreted as: "take an item if possible"—such interpretation would lead always to feasible solutions. However, it is important to point out that several factors should be taken into account while using decoders. Each decoder imposes a relationship T between a feasible solution and decoded solution.

It is important that several conditions are satisfied: (1) for each feasible solution s there is a decoded solution d, (2) each decoded solution d corresponds to a feasible solution s, and (3) all feasible solutions should be represented by the same number of decodings d. Additionally, it is reasonable to request that (4) the transformation T is computationally fast and (5) it has locality feature in the sense that small changes in the decoded solution result in small changes in the solution itself. An interesting study on coding trees in genetic algorithm was reported by Palmer and Kershenbaum [45], where the above conditions were formulated.

2.7 Separation of individuals and constraints

This is a general and interesting heuristic. The first possibility would include utilization of multi-objective optimization methods, where the objective function f and constraint violation measures f_j (for m constraints) constitute a $(m+1)$-dimensional vector $\vec{v}$:

$$\vec{v} = (f, f_1, \ldots, f_m).$$

Using some multi-objective optimization method, we can attempt to minimize its components: an ideal solution x would have $f_j(x) = 0$ for $1 \leq i \leq m$ and $f(x) \leq f(y)$ for all feasible y (minimization problems). A successful implementation of similar approach was presented recently in [57].

Another approach was recently reported by Paredis [46]. The method (described in the context of constraint satisfaction problems) is based on a co-evolutionary model, where a population of potential solutions co-evolves with a population of constraints: fitter solutions satisfy more constraints, whereas fitter constraints are violated by more solutions.

Yet another heuristic is based on the idea of handling constraints in a particular order; Schoenauer and Xanthakis [52] called this method a "behavioral memory" approach. Section 3 of this paper describes this approach in detail.

It is also possible to incorporate the knowledge of the constraints of the problem into the belief space of cultural algorithms [48, 49]. The general intuition behind belief spaces is to preserve those beliefs associated with "acceptable" behavior at the trait level (and, consequently, to prune away unacceptable beliefs). The acceptable beliefs serve as constraints that direct the population of traits. It seems that the cultural algorithms may serve as a very interesting tool for numerical optimization problems, where constraints influence the search in a direct way (consequently, the search in constrained spaces may be more efficient than in unconstrained ones!).

3 Truss Structure Optimization Problem

This section presents the problem of discrete optimization of truss structures using genetic algorithms. The original issues of this section are the constraint handling technique, based on the "behavioral memory" paradigm, and the experimental comparison of the performances of GAs on both the continuous and the discrete version of the same problem.

3.1 Problem description

3.1.1 Truss structure design

Truss structure optimization is a well-known problem of structural mechanics [22]. In its simplest form, the objective structure is made of *bars* linking fixed nodes, and the design variables are the areas of the sections of the bars (the mechanical behavior of bars only depends on their section areas).

The objective of the optimization is to find the structure of minimal weight meeting some given constraints on the maximal displacement or/and the maximal stress under prescribed loadings.

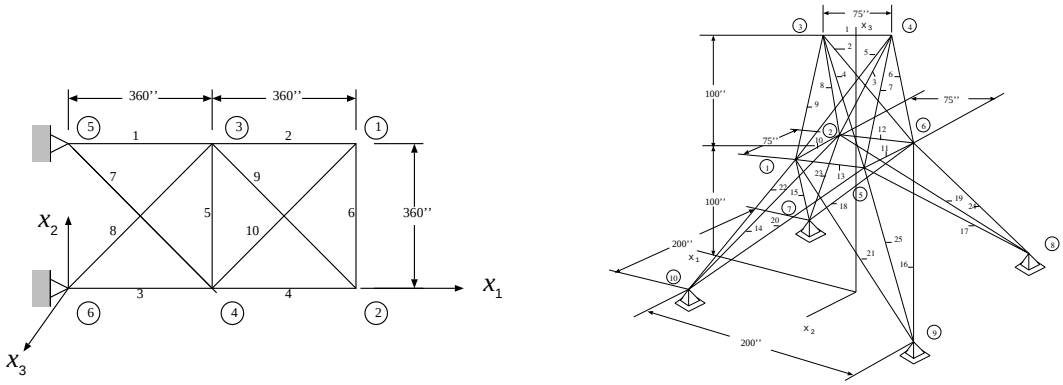

Figure 1: The 10-bars and 25-bars benchmark truss structures

Figure 1 shows the test cases considered in this section. Both are classical examples, used as a benchmarks for all truss structure optimization algorithms: the first problem is a bidimensional 10-bars truss, involving 10 design variables; in the second problem, a 3-dimensional truss is considered, made of 25 bars, but due to *a priori* symmetry conditions, only 7 independent variables are considered. Table 1 gives the mechanical parameters of the bars, together with the loadings considered during the optimization.

elasticity modulus $= 1.0 \times 10^4\ ksi$ density of material $= 0.10\ lb/in^3$ stress limits $= \pm 25.0\ ksi$ number of loadings $= 1$ (shown in *kips*)				
		direction of loading		
loading #	Node	x_1	x_2	x_3
1	2	0.0	-100.0	0.0
	4	0.0	-100.0	0.0

elasticity modulus $= 1.0 \times 10^4\ ksi$ density of material $= 0.10\ lb/in^3$ stress limits $= \pm 40\ ksi$ number of loadings $= 1$ (shown in *kips*)				
		direction of loading		
loading #	Node	x_1	x_2	x_3
1	3	0.0	20.0	-5.0
	4	0.0	-20.0	-5.0

Table 1: Mechanical parameters, constraints and loadings for both benchmark problems

3.1.2 The discrete problem

When the areas of the sections of the bars can take continuous values (in a prescribed interval), the problem of truss structure optimization, as stated in the preceding subsection, can be addressed by many deterministic optimization methods, based on gradient descent and projection operators for constraint handling (see [14] among others). But when a mechanical engineer faces such a problem, the actual solution must be technologically feasible: the bars have to be taken from suppliers stock, and the areas of the sections can only take a finite number of *discrete* values (the situation is even worse in the case of the beam model, as only a finite number of section shapes are available). A possible solution is to compute the solution of the continuous problem, and to take the closest possible values in the stock. But it is well-known that the discrete optimum might well be missed by such a simple strategy.

3.2 Constraints handling through behavioral memory

3.2.1 The behavioral memory

The "behavioral memory" paradigm, first introduced by De Garis [17], relies on the assumption that a population that has undergone artificial evolution contains more information than just the location of the point having the highest fitness: the localization of the whole population somehow witnesses the history of the population, how it did *behave* while evolving under the pressure due to the fitness function at hand, hence giving indirect insight about the fitness landscape. For instance, the mean distance between individuals can be an indication of the steepness of the fitness function around optimal value, giving hints on the stability of that solution. On the opposite, the existence of different clusters is a sign of the multi-modality of the fitness function.

The basic idea of behavioral memory-based evolution is to use a population resulting from a first fitness-driven evolution as the starting point for a second evolution, using another fitness function. This second evolution is biased by the information contained in its non-random initial population. Hopefully, a good choice of the first fitness function can help to find a good optimum to the second fitness function.

A common use of such iterated scheme amounts to gradually include more and more fitness cases in the computation of the fitness (e.g., more and more test points in regression problems). It has also been applied with completely different successive fitness functions [17, 53]. And it can be applied to handle constraints [52], as will be demonstrated in the following subsection.

One of the key issues for such an iterated scheme is the *genetic diversity*: if the population that has evolved in the context of the first fitness function has converged, the bias induced by using it as a starting point for the second evolution is too strong, and this second evolution reduces to a coupling between a local search (in the small region where the population is located) and hazardous random search: Only a lucky mutation can help escaping the (probably local) minimum. It is of utter importance to preserve the diversity in the population during the first evolution. Many schemes have been proposed to that aim, usually in the context of multi-modal function optimization. A good review of these schemes can be found in [31]. The sharing scheme (see [19] for details) has been used in this paper.

3.2.2 Iterated handling of constraints

The ideal way of handling constraints is to have the search space limited to the feasible space. In some cases, a suitable change of variables can transform the constraint problem at hand in such a desirable way [37]. But on the other hand, the main difficulty in many (deterministic) methods of constraint problem solving is to find even one feasible starting point.

The basic idea underlying the behavioral memory-based constraint handling algorithm (BMCHA) is to evolve the same population using different successive fitnesses to sample the feasible region, i.e., get a population of feasible points, regardless of the objective function at hand. It is then possible to use a standard GA *in the feasible region* to perform the optimization task on the objective function, throwing away the infeasible points by giving them zero fitness (death penalty; see section 2.1).

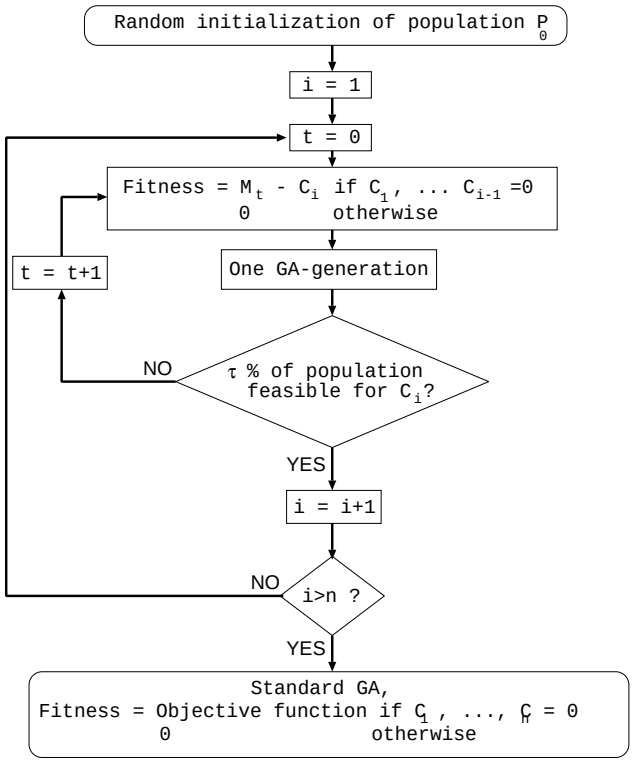

Figure 2: Flow-chart of behavioral memory-based constraint handling algorithm

Suppose the optimization problem is the following:

$$\begin{cases} \max\ f(x),\ x \in E \\ g_i(x) \leq 0 \quad for \quad i \in [0,n] \end{cases}$$

Define $C_i(x) = \max\{g_i(x), 0\},\ x \in E,\ i \in [0,n]$ and let M_i^t be the maximum value of C_i in the population at generation t of step i. Figure 2 shows the flow chart of the BMCHA, which will now be detailed.

Each one of the first steps of the algorithm is devoted to the satisfaction of a single constraint - while ensuring that the population remains feasible for the constraints used in the preceding steps. So the fitness for step i has to be a decreasing function of the violation C_i of the i^{th} constraint. Moreover, all feasible points must have the same fitness to prevent any convergence toward a particular feasible point. Finally, the maximum of these violations M_i^t over the current population (t is the generation counter) is computed, and the fitness of an individual *which is feasible for the constraints* $1, \ldots, i-1$ is set to $M_i^t - C_i$. Infeasible points for constraints $1, \ldots, i-1$ are assigned null fitness, and will be discarded by the following selection step. Step i ends when a large enough part of the population (ratio τ %, user-supplied parameter) is feasible for constraint i. As stated above, the sharing scheme is used throughout these first steps to keep the diversity of the population as large as possible.

The population after step n (the number of constraints) is then made of τ % of points which are

feasible for the problem at hand (i.e. satisfy all constraints). The last step addresses the optimization of the objective function f in a standard way, except that the infeasible points are discarded (see section 2.1) by being assigned null fitness. Note the use of the sharing scheme is not necessary in this last step.

3.3 Results

This subsection presents results obtained using the BMCHA on the truss structure optimization problem of section 3.1. More details can be found in [51].

3.3.1 Experimental settings

The GA used for these experiments is a home made package based on standards: stochastic remainder selection, linear fitness scaling with selective pressure of 2 [18], real encoding of the design variables, with linear cross-over [33] and ES-like Gaussian mutation [54]. In the discrete case, all floating point numbers are rounded to the nearest authorized value after application of the operators. One single step is used to handle the constraints (only constraints on the stress are considered), and the switch parameter τ is set to 80%.

3.3.2 Comparative results

Tables 2 and 3 present the results obtained for both problems (the 10 bars and 25 bars structures), for both the benchmark continuous problem and a discretized version, where only 36 values were allowed for the areas of the bars. These 36 values included the optimal values found by the gradient-like method in [22].

		ΔV	Section areas per bar									
Pb	Gen.	(%)	1	2	3	4	5	6	7	8	9	10
Disc.	487	0	5.948	.1	10.052	3.948	.1	2.052	8.559	2.754	5.583	.1
Cont.	4000	9.7	6.465	.279	9.531	3.837	.827	2.179	7.823	4.192	5.428	.409

Table 2: Results for the 10-bars problem. Population size = 100

		ΔV	Section areas per bar group						
Pb	Gen.	(%)	1	2 to 5	6 to 9	10 to 13	14 to 17	18 to 21	22 to 25
Disc.	176	0	.1	.3446	.4968	.1	.1	.308	.1
Cont.	2000	27	.361	.487	.438	.151	.1	.274	.505

Table 3: Results for the 25-bars problem. Population size = 70

The results are averages over 10 independent runs: in all runs, the discrete solution was found exactly. The average number of generations needed to reach the optimum is indicated in the tables. For the continuous case, good solutions were found, but a maximum number of generations was prescribed, as the exact values will not be reached exactly by that type of GA.

3.3.3 Discussion

The first important conclusion from these experiments is the ability for evolutionary algorithms to handle both discrete and continuous versions of the same problem with only minor changes in the cross-over and mutation operators. From the point of view of deterministic methods, the two problems look completely different, and the same kind of work (tailor a method to a problem) has to be done twice.

Of course, the results for the continuous problem are not very accurate. But a simple hill-climbing method should be used at the end of any evolutionary method, to locate the nearest local optimum, which hopefully is global. Furthermore, the evolutionary method requires a computational effort about 100 times larger than the gradient-based method: it is clear on that case that the latter method should be used.

Regarding now the constraint handling method, it is fair to say that we did succeed also using a simple penalty method, after a little tuning of the penalty parameter. But fair comparisons between the two approaches are very difficult to perform, as the effort can only be measured in terms of engineer time. In all runs of the discrete problem, the population reached the feasibility threshold in less than 50 generations.

It is clear than the BMCHA is not a universal method for constraint handling. If the feasible region is made of discrete points, for instance, it will not give any result. And the unavoidable genetic drift will certainly degrade its performances in problems with a large number of constraints. Moreover, the results seem to depend on the order in which the constraints are presented to the algorithm. Nevertheless, we believe that it can be useful in cases where the feasible region is "large but sparse" in the search space, being fairly easy to implement and tune.

4 Design of a Composite Laminated Plate

In this section, the problem of minimizing the weight of a composite laminated plate subjected to various failure constraints is considered. The optimization is carried out using an evolutionary algorithm where constraints are accounted for through a double series of penalty functions, the segregated genetic algorithm or SGGA [29].

4.1 Problem description

4.1.1 Stacking sequence design

Composite materials typically consist of fibers made of stiff materials, such as graphite, embedded in a soft matrix, such as epoxy resin. Their high strength-to-weight and stiffness-to-weight ratios make them attractive for aerospace applications. The simply-supported laminated plate shown in Figure 3 is loaded in the x and y directions by X_i and Y_i, respectively. The laminate is composed of N plies, each of thickness t. Because of manufacturing considerations,

- the orientation of the fibers in the plies are restricted to the discrete set (0^o, $+45^o$, -45^o, and 90^o),

- the laminate must be symmetric about its mid-plane,

- the laminate must be balanced (same number of $+45^o$ and -45^o layers).

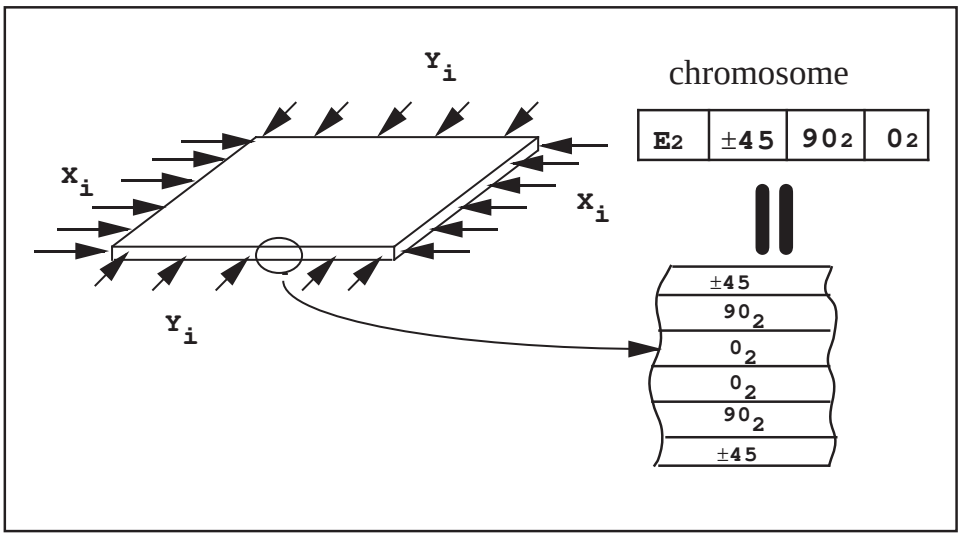

Figure 3: Simply supported laminated plate subjected to normal in-plane loads and coding

The objective of the optimization is to minimize the number of plies in the laminate, N, such that the manufacturing constraints are satisfied, and such that the plate does not fail, neither from buckling nor from insufficient strength. Analytical expressions are available [28] to predict failure. Failure modes are accounted for by the constraint $\lambda_{cr} \geq 1$, where λ_{cr} is the critical load factor (or safety load factor). The optimization is performed through the choice of the total number of plies and the fiber angles for each ply, i.e., we optimize the "stacking sequence" of the laminate.

To reduce the size of the optimization problem and to automatically satisfy the balance condition we consider only laminates made up of 2-ply stacks 0_2 (two 0^o plies), 90_2 (two 90^o plies), or ± 45 (a pair of $+45^o$ and -45^o plies). The constraint on symmetry is readily implemented by considering only one half of the laminate (the other half being deduced by symmetry). As a result a laminate with under N plies can be represented by a chromosome of length $N/4$. To accommodate variable thickness in a fixed string length we add an empty stack and denote it as E_2. Figure 3 shows an example of a laminate cross-section, and the associated coded chromosome assuming that the maximum thickness needed is $N = 16$ plies.

We minimize the objective function f which is defined as,

$$\begin{aligned} &\text{if } \lambda_{cr} \geq 1, \quad f = N + \varepsilon[1 - \lambda_{cr}], \\ &\text{if } \lambda_{cr} < 1, \quad f = \tfrac{N}{\lambda_{cr}^p}. \end{aligned} \tag{1}$$

The constraint on failure of the laminate is enforced by the penalty functions $(1/\lambda_{cr}^p)$. p is a penalty parameters for failure. Because of the discrete nature of the problem, there may be several feasible designs of the same minimum thickness. Of these designs, we define the optimum to be the design with the largest failure load λ_{cr}. Therefore, the objective function is linearly reduced in proportion to

the failure margin for designs that satisfy the failure constraint (term $\varepsilon[1-\lambda_{cr}]$). A value of ε ($\varepsilon = 6$) was derived in [28] along with a more extensive analysis.

The setting of p is the subject of the present study.

4.2 The segregated genetic algorithm

4.2.1 Motivations

As it was seen in Section 2, a general approach to handling constraints in evolutionary computation techniques is the use of penalty functions. The amount of penalty for each constraint violation is typically controlled by a penalty parameter that has a crucial influence on the performance of the genetic algorithm. If the amount of penalty for being infeasible is too small, the search will yield infeasible solutions. Reciprocally, the flaw of heavily penalizing infeasible designs is that it limits the exploration of the design space to feasible regions, precluding short cuts through the infeasible domain. The general procedure is to tune the penalty functions on each problem case to assure convergence in the feasible domain while permitting an efficient search. Apart from tuning, there is no general solution to the problem of optimally adjusting penalties, neither in classical numerical methods, nor in evolutionary calculation. Instead of fine tuning penalties, the segregated genetic algorithm is an attempt at desensitizing the method to the choice of penalty parameters.

4.2.2 Description

In the segregated genetic algorithm (SGGA), the population is split into two co-existing and co-operating groups that differ in the way the fitnesses of their members are calculated. Each group uses a different value of the penalty parameter p. Each of the groups corresponds to the best performing individuals with respect to one penalty parameter. The two groups interbreed, but they are segregated in terms of rank. Two advantages are expected. First, because the penalty parameters are different, the two groups will have distinct trajectories in the design space. Because the two groups interbreed, they can help each other out of local optima. The SGGA is thus expected to be more robust than the GA. Second, in constrained optimization problems, the optimum is typically located at the boundary between feasible and infeasible domains. If one selects one of the penalty parameters large (say p_1) and the other small (p_2), one can achieve simultaneous convergence from both the feasible and the infeasible side. The global optimum will then be rapidly encircled by the two groups of solutions, and since the two groups of designs interbreed, the global optimum should be located faster.

For example, in structural optimization, one usually seeks to minimize the weight of a structure. The "p_1 group" contains heavy designs that do not fail, while the "p_2 group" contains light designs that fail. The optimum design, which is a compromise between weight and safety, is located somewhere between the p_1 and p_2 groups

Figure 4 gives the flow chart of the SGGA. Note that when the individuals are ranked, duplicates are pushed to the bottom (low rank) of the lists. This is a protection against premature uniformization of the population. Then, from the two lists of $2m$ ranked individuals, one single population of m individuals is built which mixes the relative influences of the two lists (this step is referred to as "merge the two lists ..." in Figure 4). One starts by selecting the best individual of the list established

using the highest penalty parameter (p_1). Then, one chooses the best individual of the other list that has not yet been selected. The process is repeated alternatively on each list until m individuals have been selected. Then, reproduction occurs as usual by application of linear ranking selection, crossover, mutation, and stack swap to the combined list, creating m offspring. They are added to the m parents, and the entire process is repeated. Note that the crossover applied here is a one-point thick crossover (probability = 1) which ressembles one-point crossover, the mutation can independently change the thickness of the laminate or change the orientation of stacks (probabilities = 0.05 and 0.01, respectively), and the stack swap operator (probability = 1) shuffles a little the ordering of the plies in the laminate. The population has 8 individuals, 4 in each group. The interested reader should refer to [28] for more details.

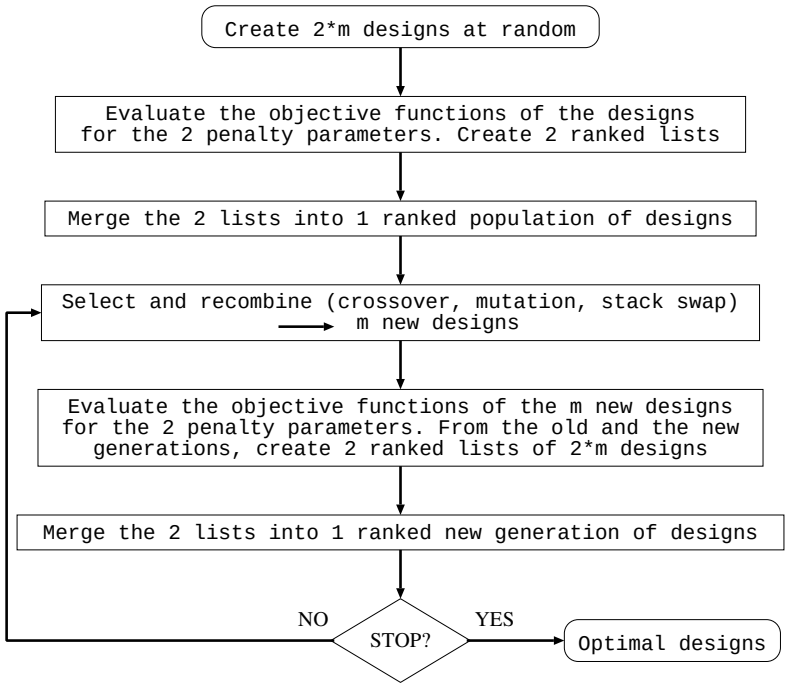

Figure 4: Flow-chart of a segregated genetic algorithm

4.3 Optimization of composite laminates by SGGA

We now turn to the performance of the segregation strategy for handling constraints. We first define a performance criterion and then compare the SGGA with an equivalent algorithm that does not rely on segregation to handle constraints.

4.3.1 Performance criterion

It is common in stacking sequence problems to have many near optimal designs. For this reason, we define *practical optima* as feasible optimal weight designs for which λ_{cr} is within a 10th of a percent

of λ_{cr} of the global optimum. The measure of the performance of the algorithm is the *reliability* of the algorithm. It is the probability the algorithm has of finding a practical optimum after a given number of analyses (evaluations of the objective function). In this study, the reliability is calculated by averaging the results of 3000 independent searches for each implementation that is being tested. Each search is 6000 analyses long. The *price of the search* is the number of analyses necessary to reach a reliability of 80%.

4.3.2 Results

GA type	Price of search
EGA p=0.4	$r = 0.15^*$
EGA p=0.5	1380
EGA p=5.0	3980
SGGA p_1=0.5 p_2=0.4	1270
SGGA p_1=5.0 p_2=0.4	3350
SGGA p_1=0.5 p_2=0.5	990

Table 4: Price of the search of EGA and SGGA. * : reliability at 6000 analyses. The price is not defined because 80% reliability was not achieved within 6000 analyses

We consider a graphite-epoxy plate with longitudinal and lateral dimensions $a = 20$ in. and $b = 5$ in., respectively. The material properties are: $E_1 = 18.50 \times 10^6$ psi; $E_2 = 1.89 \times 10^6$ psi; $G_{12} = 0.93 \times 10^6$ psi; $\nu_{12} = 0.3$; $t = 0.005$ in. (t is the basic ply thickness). The maximum thickness for a laminate is assumed to be 64 plies, i.e., the string length is $64/4 = 16$. Three loadings are considered simultaneously: [X_1 = 12,000. lb/in, Y_1 = 1,500. lb/in], [X_2 = 10,800. lb/in, Y_2 = 2,700. lb/in], [X_3 = 9,000. lb/in, Y_3 = 4,500. lb/in], and the ultimate allowable strains are $\epsilon_1^{ua} = 0.008$, $\epsilon_2^{ua} = 0.029$, $\gamma_{12}^{ua} = 0.015$. This optimization problem has been treated over 100000 times, so the best-known design is taken as the optimum.

The performance of the SGGA ($p_1 \neq p_2$) is compared with the performance of the same algorithm when $p_1 = p_2$. When $p_1 = p_2$, the co-evolutionary aspect of the SGGA is lost to yield an algorithm that we call superElitist Genetic Algorithm (EGA). Table 4 gives the prices of the search for EGA and SGGA. Three values of the penalty parameter p, 0.4, 0.5 and 5.0. These values were found experimentally. $p = 0.5$ is the optimal setting of the penalty parameter, as it can be seen in Table 4. $p = 0.4$ is a small value of the penalty parameter, and the search often gets trapped in the infeasible region of the design space (although the global optimum is still feasible for this value of p). $p = 5.0$ is a large value of the penalty parameter. All three combinations of p_1 and p_2, $p_1 \neq p_2$, are tested with the SGGA.

Table 4 shows that EGA endures a dramatic decrease in performance if p is poorly chosen. For $p = 0.4$, the reliability achieved by EGA is only 15% after 6000 analyses. This is because EGA gets trapped at a local infeasible optimum ($N = 44$ plies, $\lambda_{cr} \approx 0.8$ when the global optimum has $N = 48$ plies, $\lambda_{cr} \approx 1.01$). SGGA on the contrary maintains a decent level of performance for the worst possible choice of penalty parameters. For p_1 too large (=5.0) and p_2 too small (=0.4), the price of

SGGA is 3350 analyses. Providing that p_1 is taken small and p_2 large, SGGA is less sensitive to the tuning of p than EGA is.

Furthermore, segregation speeds up the search since the best price with SGGA is 990 analyses against 1380 analyses for EGA. Those figures, compared to the size of the design space ($4^{16} \approx 4.\ 10^9$), show how efficient specialized evolutionary algorithms can be on this type of problem.

5 Unit Commitment Problem

The unit commitment in a power system is a complex optimization problem because of multiple nonlinear constraints which can not be violated while finding the optimal schedule. An electric power network usually operates under continuous variation of consumer load demand. This demand for electricity exhibits such large variations between weekdays and weekends, and between peak and off-peak hours that it is not economical to keep all the generating units continuously on-line; since fuel expenses constitute a significant part of the overall generation costs. In a system, there exist various types of generating units that are categorized on the basis of fuel used (e.g., coal, natural gas, oil), production costs, generating capacities and operating characteristics. Thus determining which units should be kept on-line and which ones should not, constitutes a difficult decision-making task for the operators seeking to minimize the system operational cost.

Several mathematical programming techniques have been reported in the literature [6, 32, 44] to solve the unit commitment problem. They primarily include priority list and heuristic methods, dynamic programming, method of local variations, mixed integer programming, lagrangian relaxation, branch and bound, bender decomposition, etc. Among these classical techniques, dynamic programming methods based on the priority list have been used most extensively throughout the power industry. However, different strategies (for selecting a set of units from priority list) have been adopted with dynamic programming to limit the search space and execution time.

In recent work, some researchers have suggested artificial intelligence based techniques to supplement the limitation of mathematical programming methods. These are simulated annealing [59], expert systems [41], heuristic rule-based systems [58] and neural networks [43]; these hybrid approaches have demonstrated some improvement in solving unit commitment problems. However, heuristic and expert system based mathematical approaches require a lot of operator interaction which is troublesome and time-consuming [21].

This section presents an evolutionary algorithm to solve the unit commitment problem. The main purpose of using the evolutionary approach is to replace classical solution methods with a population-based global search procedure which has some distinct advantages.

5.1 Problem Description

The unit commitment problem involves in determining which of the generating units are to be committed (on or off) in every time interval during the scheduling time horizon. This decision must take into account load forecast information and the economic implications of the startup or shutdown of various units. The transition between their commitment states must satisfy the operating (minimum up and

down time) constraints. So the demand and the reserve requirement impose global constraints in coupling all active generating units, while the different operating characteristics of each unit constitute local constraints. Also some stand-by capacity (called spinning reserve) is required to be maintained at all time in addition to the forecasted load in order to meet unexpected load demand and sudden unit failures.

5.2 Objective function

The objective of the unit commitment problem is to determine the state of each unit u_i^t (0 or 1) at each time period t, where unit number $i = 1 \ldots U_{max}$, and time periods $t = 1 \ldots T_{max}$, so that the overall operation cost is a minimum within the scheduling time horizon.

$$\min \ \sum_{t=1}^{T_{max}} \sum_{i=1}^{U_{max}} [u_i^t \ AFLC_i + u_i^t(1 - u_i^{t-1}) \ S_i(x_i^t) + u_i^{t-1}(1 - u_i^t) \ D_i] \qquad (1)$$

For each committed unit, the cost involved is the *start-up* cost (S_i) and the *Average Full Load Cost* ($AFLC_i$) per MWh, according to the unit's maximum capacity such that

$$\sum_{i=1}^{U_{max}} {P_i}^{max} \geq R^t + L^t, \qquad (2)$$

where ${P_i}^{max}$ is the maximum output capacity of unit i, L^t is the demand and R^t is the spinning reserve in time period t. The above objective function should satisfy minimum *up-time* and *down-time* constraints of generating units.

The start-up cost is expressed as a function of the number of hours (x_i^t) the unit has been down and the shut-down cost is considered as a fixed amount (D_i) for each unit per shut-down, and these state transition costs are applied in the period when the unit is committed or taken off-line respectively [23].

However, unit commitment decisions based solely on the unit-$AFLC$ usually do not provide sufficient information about the impact of system load conditions on how efficiently (e.g., fully) the committed units being utilized while determining a near-optimal commitment [27]. To compensate the deficiency associated with committed decisions based on the classical $AFLC$ of units, an index is used to measure the utility of each commitment decision, while satisfying the global constraint (equ. (2)). This is called an utility factor and can be calculated as

$Utility \ Factor = \frac{Load - Reserve \ Requirements}{Total \ committed \ output}$. So during the performance evaluation, commitment decisions having a low utility factor are penalized accordingly.

5.3 GA-based Unit Commitment

The task of the GA-based commitment scheduler is to ensure the adequate power supply over the entire scheduling period, in a most cost-effective manner while satisfying the system operational constraints. For developing a GA-based scheduler, we first map the problem space into the framework of genetic algorithm. Accordingly, each chromosome is encoded in binary form (bit string) to represent the state of commitment variables for the units that are available in the system. So the allele value at loci give the state (on/off) of the units as a commitment decision at each time period. In this study, a short-term commitment is considered with a 24-hour time horizon. Since the system load

varies substantially over a 24-hour period and the cost of operation over this time span depends on the timing and frequency of unit's start-ups and shut-downs, this commitment problem is generally viewed as a multi-period problem where the commitment horizon is divided into a number of periods of shorter length (usually a one-hour commitment interval).

During the genetic optimization process, the evolved commitment decisions satisfying both generation constraints and unit constraints are regarded as feasible solutions. Any violation of these constraints is penalized through a penalty function. Accordingly, the raw fitness function is formulated using a weighted sum of the objective function and values of the penalty function based on the number of constraints violated and the extent of these violations. The fitness is then scaled to get a non-negative figure of merit for each commitment decision. The scaled fitness is subsequently used to determine the probability of selecting the members in the population for breeding.

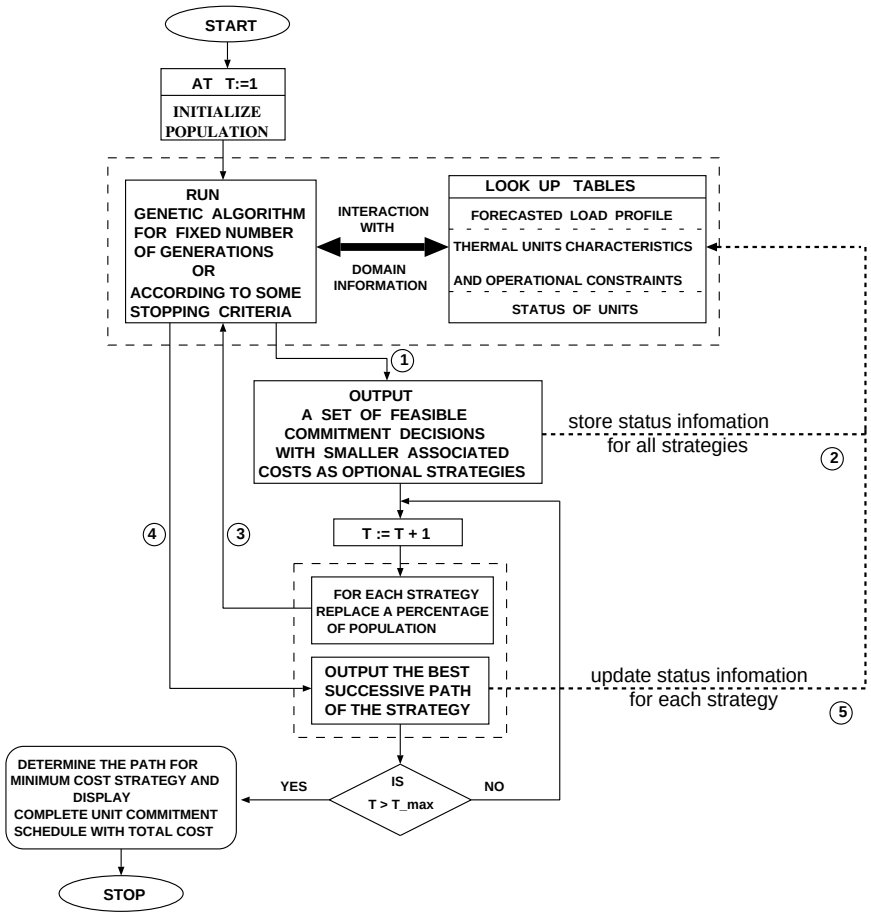

Figure 5: A flow diagram for unit commitment using genetic algorithms

The flow chart of implementing the GA-based unit commitment is shown in figure 5. For better understanding of the diagram, some of the flow lines connected to the repeatedly used program module (GA routines) and domain information (look-up tables) are numbered in accordance with the sequence

of execution. The GA-based unit commitment program starts with a random initial population (at T=1) and computes the fitness of each individual (commitment decision) using the forecasted load demand at each period, and the operating constraints of the units (using lookup tables). Each time the genetic optimizer is called, it runs for a fixed number of generations or until the best individual remains unchanged for a long time (here 100 successive generations).

Since the unit commitment problem is time-dependent, these piecewise approaches (moving window) of working forward in time and retaining the best decision, can not be guaranteed to find the optimal commitment schedule. The reason for this is that a decision with significantly higher costs during the early hours of scheduling could lead to significant savings later and may produce a lower overall cost commitment schedule. For finding near-optimal solutions, a number of feasible commitment decisions (less than or equal to a predefined value S) with smaller associated costs are saved at each time period. These strategies[1] determine how many possible alternative paths are available at each period for finding the overall operation cost. The selection of S is effective in economical scheduling (in finding an optimal solution), memory requirement and computation time. In order to save computation time the same strategies are carried forward to the next period if the load remains unaltered or varies slightly in the current period such that load-reserve requirements are satisfied by all strategies. If a strategy cannot meet the demand of present period, the genetic optimization process is performed for the period to find a feasible successor paths with smaller cost. This approach increases the likelihood of finding the path of minimum cumulative cost.

These temporary commitment strategies are used to update the status information of the units (up-time/down-time counter) to keep track of the units in service or shutdown for a number of successive hours. In the next time period, half of the population is replaced by randomly generated individuals to introduce diversity in the population so that the search for new commitment strategy can proceed according to the load demand. The purpose of keeping half of the previous population is that in most situations the load varies slightly in some successive time intervals and the previous better individuals (commitment strategies) are likely to perform well in the current period. However, if there is a drastic change in load demand, newly generated individuals can explore the commitment space for finding the best solution. The iterative process continues for each period in the scheduling horizon, and the accumulated cost associated with each commitment strategy gives the overall cost for the commitment path.

During these multi-period optimization processes, if in a particular period no feasible solution (strategy) is found, the process is repeated so that at least one feasible solution is found before shifting to the next period. However, in our test example such repetitions are required only in a few occasions in later periods.

5.4 Experimental results

The GA-based unit commitment scheduler has been tested on a number of example cases and some were reported in [9, 7]. Due to space restrictions we report results of one example problem which consists of 10 thermal units (please refer to [9] for details). It is to be noted that the capacities, costs and operating constraints varied greatly among the various generating units in this example system.

[1] A strategy is a sequence of commitment decisions from the starting period to the current period with its accumulated cost.

Different type of load profiles are tested which represent typical operating circumstances in the studied power system. We have considered a short term scheduling where the time horizon is 24-hour and scheduling for an entire day is done in advance which may be repeated using load profile of each day for a long-term scheduling.

In these experiments, the spinning reserve requirement is assumed to be 10% of the expected hourly peak load. A program implementing the algorithm has been run on a SUN (sparc 2) workstation under UNIX 4.1.1 operating system. The experiment is conducted with a population size of 250 using different crossover and mutation rates. For the result reported here (shown in a tabular form), a crossover probability of 78% and mutation rate of 15% were used along with a stochastic remainder selection scheme [5] for reproduction. We also used an elitist scheme which passes the best individual unaltered to the succeeding generation. Each run was allowed to continue up to 500 generations and the strategy path with minimum cumulative cost gives a near-optimal commitment for the whole scheduling period.

For this example, Table 5 gives the characteristics and the initial states of the generating units. Table 6 gives commitment schedule for two cases, which were run independently. In first case, one best solution is saved at each time period and in second case, multiple least cost strategies are saved for determining the minimum cost path. A comparison shows that substantial reduction in overall cost can be achieved when the best commitment schedule is determined from multiple least cost strategies.

In this table, the second column gives the hourly load demand, the third column shows total requirement after adding spinning reserve, the rest columns give the total output capacity (in MW) of the committed units and the state of units in each case, where '1' is used to indicate a unit is committed, '0' to indicate that a unit is decommitted. The GA-based unit-commitment system have been tested under different operating conditions in order to evaluate the algorithm's performance.

It is observed that the scheduling which produces optimal power output does not always give the overall minimum cost scheduling, and also the minimum cost scheduling is very sensitive to the system parameters and the operating constraints of the generating units.

5.5 Discussion

The unit commitment is a highly-constrained decision-making problem, and traditional methods make several assumptions to solve the problem. Most of these traditional methods require well-defined performance indices and explicitly use unit selection list or priority order list for determining the commitment decisions. The major advantages of using GAs are that they can eliminate some limitations of mathematical programming methods. Particularly, the GA-based unit commitment scheduler evaluates the priority of the units dynamically considering the system parameters, operating constraints and the load profile at each time period while evolving near-optimal schedules. Though global optimality is desirable, but in most practical purposes near-optimal (or good feasible) solutions are generally sufficient. This evolutionary approach attempts to find the best schedule from a set of good feasible commitment decisions. Also the method presented in this section can include more of the constraints that are encountered in real-world applications of this type.

This study suggests that the GA-based method for short-term unit commitment is a feasible alternative approach and is easy to implement. One disadvantage of this approach is the computational

Unit No.	Maximum Capacity (MW)	Min. Up Time (hr)	Min. Down Time (hr)	Initial Status (hr)	St_Up cost			Sh_Down Cost	$AFLC$
					b_1	b_2	b_3		
1	60	3	1	-1	85	20.588	0.2	15	15.3
2	80	3	1	-1	101	20.594	0.2	25	16
3	100	4	2	1	114	22.57	0.2	40	20.2
4	120	4	2	5	94	10.65	0.18	32	20.2
5	150	5	3	-7	113	18.639	0.18	29	25.6
6	280	5	2	3	176	27.568	0.15	42	30.5
7	520	8	4	-5	267	34.749	0.09	75	32.5
8	150	4	2	3	282	45.749	0.09	49	26.0
9	320	5	2	-6	187	38.617	0.130	70	25.8
10	200	5	2	-3	227	26.641	0.11	62	27.0

(-) indicates unit is down for hours and positive otherwise.

*We used start-up cost $= b_{1,i}(1 - e^{-b_{3,i}.(x_i^t)}) + b_{2,i}$.

Table 5: Characteristics and initial state of the thermal units

			CASE - 1		CASE - 2	
			When only best strategy is saved at each hour		Best of five least cost strategies saved	
Time in (hr)	Load Demand (MW)	Load + Reserve (MW)	Committed Output (MW)	State of units	Committed Output (MW)	State of Units
1	1459.00	1677.85	1700.00	1011111110	1710.00	1110011111
2	1372.00	1577.80	1710.00	1110111011	1860.00	1110111111
3	1299.00	1493.85	1710.00	1110111011	1550.00	1111101011
4	1280.00	1472.00	1490.00	0111101011	1490.00	0111101011
5	1271.00	1461.65	1470.00	1011101011	1470.00	1011101011
6	1314.00	1511.10	1550.00	1111101011	1550.00	1111101011
7	1372.00	1577.80	1600.00	1101101111	1580.00	1110101111
8	1314.00	1511.10	1600.00	1101101111	1520.00	0110101111
9	1271.00	1461.65	1500.00	1111101110	1500.00	1111101110
10	1242.00	1428.30	1630.00	1111011110	1500.00	1111101110
11	1197.00	1376.55	1420.00	0111011010	1380.00	1011110111
12	1182.00	1359.30	1360.00	1111011001	1360.00	1101110111
13	1154.00	1327.10	1330.00	1001111001	1360.00	1101110111
14	1138.00	1308.70	1410.00	1101111001	1310.00	1111110011
15	1124.00	1292.60	1310.00	1111110011	1310.00	1111110011
16	1095.00	1259.25	1280.00	0110110111	1260.00	1111110110
17	1066.00	1225.90	1260.00	1010110111	1260.00	1111110110
18	1037.00	1192.55	1260.00	1010110111	1200.00	0111110110
19	993.00	1141.95	1180.00	1111100111	1180.00	1111100111
20	978.00	1124.70	1200.00	1111001010	1180.00	1111100111
21	963.00	1107.45	1320.00	0101011010	1180.00	1111100111
22	1022.00	1175.30	1210.00	1101011100	1180.00	1111100111
23	1081.00	1243.15	1340.00	1110111100	1250.00	0111110011
24	1459.00	1677.85	1900.00	1011111111	1680.00	1111011011
			Cumulative scheduling cost = 940101.65		Cumulative scheduling cost = 877854.32	

The difference in cost is approximately 7% in these two cases.

Table 6: Unit commitment schedules determined by the genetic algorithm

time needed to evaluate the population in each generation, but since genetic algorithms can efficiently be implemented in a highly parallel fashion, this drawback becomes less significant with its implementation in a parallel machine environment. Further research should address a number of issues to solve the commitment problem: Experiments should be carried out with large power systems having hundreds of units in multiple areas. One possible approach may be to use indirect encoding or use of some grammar rule (as used in other GA applications) for representing a cluster of units in a chromosome. Also in a large power plant, the unit-commitment task may be formulated as a multi-objective constrined optimization problem; where it is necessary to take into account not only the operational cost but also the emission of pollutant and other environmental factors as mutually conflicting objectives. Finally, it may be also possible to integrate both the unit commitment and the economic dispatch (i.e. the optimal allocation of the load among the committed units) in a single evolutionary optimization framework using our recently-developed structured genetic model [8].

6 Conclusions

The paper surveys many heuristics which support the most important step of any evolutionary technique: evaluation of the population. It is clear that further studies in this area are necessary: different problems require different "treatment". It is also possible to mix different strategies described in this paper. The authors are not aware of any results which provide heuristics on relationships between categories of optimization problems and evaluation techniques in the presence of infeasible individuals; this is an important area of future research.

References

[1] Bäck, T., *Evolutionary Algorithms in Theory and Practice*, New-York, Oxford University Press, 1995.

[2] Baptistella, L.F.B. and Geromel, J.C., *A Decomposition Approach to Problem of Unit Commitment Schedule for Hydrothermal Systems*, IEE Proceedings – Part C, Vol.127, Np.6, pp.250–258, November 1980.

[3] Bard, J.F., *Short-Term Scheduling of Thermal-Electric Generators using Lagrangian Relaxation*, Operations Research, Vol.36, No.5, pp.756–766, Sept/Oct. 1988.

[4] Bean, J.C. and Hadj-Alouane, A.B., *A Dual Genetic Algorithm for Bounded Integer Programs*, Department of Industrial and Operations Engineering, The University of Michigan, TR 92-53, 1992.

[5] Booker, L.B., *Intelligent Behavior as an Adaptation to the Task Environment*, PhD thesis, Computer Science, University of Michigan, Ann Arbor, 1982.

[6] Cohen, A.I. and Yoshimura, M., *A Branch-and-Bound Algorithm for Unit Commitment*, IEEE Transactions on Power Apparatus and Systems, Vol.102, No.2, pp.444–449, February 1983.

[7] Dasgupta, D., *Unit Commitment in Thermal Power Generation using Genetic Algorithms*, Sixth International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE), 1993, pp.374–383.

[8] Dasgupta, D., *Structured Genetic Algorithms in Search and Optimization*, Ph.D. Thesis, Dept. of Computer Science, University of Strathclyde, Glasgow, UK, 1993.

[9] Dasgupta, D., and McGregor, D.R., *Thermal Unit Commitment using Genetic Algorithms*, IEE Proceedings – Part C: Generation, Transmission and Distribution, Vol.141, No.5, September 1994, pp.459–465.

[10] Dasgupta, D., and McGregor, D.R., *Short-Term Unit Commitment using Genetic Algorithms*, Proceedings of 5th IEEE International Conference on Tools with Artificial Intelligence (TAI'93), November 8–11, 1993, Boston, USA, pp.240–247.

[11] Davidor, Y., *Epistasis Variance: Suitability of a Representation to Genetic Algorithms*, Complex Systems, Vol.4, pp.369–383, 1990.

[12] Davis, L., *Handbook of Genetic Algorithms*, New York, Van Nostrand Reinhold, 1991.

[13] Eshelman, L.J. and Schaffer, J.D, *Real-Coded Genetic Algorithms and Interval Schemata*, Foundations of Genetic Algorithms – 2, Morgan Kaufmann, Los Altos, CA, 1993, pp.187–202.

[14] Fletcher, R., *Practical Methods of Optimization*, second edition, John Wiley & Sons, 1987.

[15] Fogel, L. J., Owens, A. J. and Walsh M. J., *Artificial Intelligence through Simulated Evolution*, New York, John Wiley, 1966.

[16] Fogel, D. B., *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*, IEEE Press, 1995.

[17] de Garis, H., *Genetic Programming: Building Artificial Nervous Systems using Genetically Programmed Neural Networks Modules*, Proceedings of the 7th International Conference on Machine Learning, R. Porter and B. Mooney (Eds), Morgan Kaufmann, pp.132–139, 1990.

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

[19] Goldberg D. E. and Richardson, J, *Genetic Algorithms with Sharing for Multi-Modal Function Optimization*, Proceedings of the Second ICGA, Lawrence Erlbaum Associates, pp.41–49, 1987.

[20] Hadj-Alouane, A.B. and Bean, J.C., *A Genetic Algorithm for the Multiple-Choice Integer Program*, Department of Industrial and Operations Engineering, The University of Michigan, TR 92-50, 1992.

[21] Hamdam, A.R. and Mohamed-Nor, K., *Integrating an Expert System into a Thermal Unit-Commitment Algorithm*, IEE Proceedings – Part C, Vol.138, No.6, pp.553–559, November 1991.

[22] Haug, E.J. and Arrora, J.S., *Applied Optimal Design,* John Wiley and Sons, Inc., New York, 1979.

[23] Hobbs, W.J., Hermon, G., Warner, S., and Sheble, G.B., *An Advanced Dynamic Programming Approach for Unit Commitment*, IEEE Transactions on Power Systems, Vol.3, No.3, pp.1201–1205, August 1988.

[24] Holland, J., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Harbor, 1975.

[25] Homaifar, A., Lai, S. H.-Y., Qi, X., *Constrained Optimization via Genetic Algorithms*, Simulation, Vol.62, No.4, 1994, pp.242–254.

[26] Joines, J.A. and Houck, C.R., *On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems With GAs*, Proceedings of the First IEEE ICEC 1994, pp.579–584.

[27] Lee, F.N., *The Application of Commitment Utilization Factor (CUF) to Thermal Unit Commitment*, IEEE Transactions on Power Systems, Vol.6, No.2, pp.691–698, May 1991.

[28] Le Riche, R., and Haftka, R.T., *Improved Genetic Algorithm for Minimum Thickness Composite Laminate Design*, Composites Engineering, Vol.3, No.1, pp. 121-139, 1995.

[29] Le Riche, R., Knopf-Lenoir, C., and Haftka, R.T., *A Segregated Genetic Algorithm for Constrained Structural Optimization*, Proceedings of the Sixth ICGA, Morgan Kaufmann, pp.558–565, 1995.

[30] Lowery, P.G., *Generating Unit Commitment by Dynamic Programming*, IEEE Transactions on Power Apparatus and Systems, Vol.85, No.5, pp.422–426, May 1966.

[31] Mahfoud, S.W., *A Comparison of Parallel and Sequential Niching Methods*, Proceedings of the Sixth ICGA, Morgan Kaufmann, pp.136–143, 1995.

[32] Merlin, A. and Sandrin, P., *A New Method for Unit Commitment at Electricite De France*, IEEE Transactions on Power Apparatus and Systems, Vol.102, No.5, pp.1218–1225, May 1983.

[33] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, New York, 2nd edition, 1994.

[34] Michalewicz, Z., *A Survey of Constraint Handling Techniques in Evolutionary Computation Methods*, Proceedings of the 4th Annual Conference on EP, MIT Press, Cambridge, MA, pp.135–155, 1995.

[35] Michalewicz, Z., *Genetic Algorithms, Numerical Optimization, and Constraints*, Proceedings of the Sixth ICGA, Morgan Kaufmann, pp.151–158, 1995.

[36] Michalewicz, Z., and Attia, N., *Evolutionary Optimization of Constrained Problems*, Proceedings of the 3rd Annual Conference on EP, World Scientific, 1994, pp.98–108.

[37] Michalewicz, Z. and Janikow, C., *Handling Constraints in Genetic Algorithms*, Proceedings of the Fourth ICGA, Morgan Kaufmann, 1991, pp.151–157.

[38] Michalewicz, Z. and Nazhiyath, G., *Genocop III: A Co-evolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints*, Proceedings of the Second IEEE ICEC, Perth, Australia, 29 November – 1 December 1995.

[39] Michalewicz, Z., Schaffer, J. D., Schwefel, H.-P., Fogel, D. B. and Kitano, H., Eds, *Proceedings of the First IEEE International Conference on Evolutionary Computation*, Orlando, June 1994, IEEE Press.

[40] Michalewicz, Z. and Xiao, J., *Evaluation of Paths in Evolutionary Planner/Navigator*, Proceedings of the 1995 International Workshop on Biologically Inspired Evolutionary Systems, Tokyo, Japan, May 30–31, 1995, pp.45–52.

[41] Mukhtari, S., Singh, J., and Wollenberg, B., *A Unit Commitment Expert System*, IEEE Transactions on Power Systems, Vol.3, No.1, pp.272–277, February 1988.

[42] Orvosh, D. and Davis, L., *Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints*, Proceedings of the Fifth ICGA, Morgan Kaufmann, p.650, 1993.

[43] Ouyang, Z. and Shahidehpour, S.M., *A Hybrid Artificial Neural Network-Dynamic Programming approach to Unit Commitment*, IEEE Transactions on Power Systems, Vol.7, No.1, pp.236–242, February 1992.

[44] Ouyang, Z. and Shahidehpour, S.M., *An Intelligent Dynamic Programming for Unit Commitment Application*, IEEE Transactions on Power Systems, Vol.6, No.3, pp.1203–1209, August 1991.

[45] Palmer, C.C. and Kershenbaum, A., *Representing Trees in Genetic Algorithms*, Proceedings of the IEEE International Conference on Evolutionary Computation, 27–29 June 1994, pp.379–384, 1994.

[46] Paredis, J., *Co-evolutionary Constraint Satisfaction*, Proceedings of the 3rd PPSN Conference, Springer-Verlag, pp.46–55, 1994.

[47] Powell, D. and Skolnick, M.M., *Using Genetic Algorithms in Engineering Design Optimization with Non-linear Constraints*, Proceedings of the Fifth ICGA, Morgan Kaufmann, pp.424–430, 1993.

[48] Reynolds, R.G., *An Introduction to Cultural Algorithms*, Proceedings of the Third Annual Conference on Evolutionary Programming, River Edge, NJ, World Scientific, pp.131–139, 1994.

[49] Reynolds, R.G., Michalewicz, Z., and Cavaretta, M., *Using Cultural Algorithms for Constraint Handling in Genocop*, Proceedings of the 4th Annual Conference on Evolutionary Programming, San Diego, CA, pp.289–305, March 1–3, 1995.

[50] Richardson, J.T., Palmer, M.R., Liepins, G., and Hilliard, M., *Some Guidelines for Genetic Algorithms with Penalty Functions*, in Proceedings of the Third ICGA, Morgan Kaufmann, pp.191–197, 1989.

[51] Schoenauer, M., and Wu, Z., *Discrete Optimal Design of Structures by Genetic Algorithms*, Proceedings of the Conférence Nationale sur le Calcul de Structures, Hermes, 1993 (in French).

[52] Schoenauer, M., and Xanthakis, S., *Constrained GA Optimization*, Proceedings of the Fifth ICGA, Morgan Kaufmann, pp.573–580, 1993.

[53] Schoenauer, M., *Iterated Genetic Algorithms*, Technical report # 304, CMAP, Ecole Polytechnique, Oct. 1994.

[54] Schwefel, H.-P., *Evolution and Optimum Seeking*, John Wiley & Sons, Chichester, UK, 1995.

[55] Siedlecki, W. and Sklanski, J., *Constrained Genetic Optimization via Dynamic Reward–Penalty Balancing and Its Use in Pattern Recognition*, Proceedings of the Third International Conference on Genetic Algorithms, Los Altos, CA, Morgan Kaufmann Publishers, pp.141–150, 1989.

[56] Smith, A. and Tate, D., *Genetic Optimization Using A Penalty Function*, Proceedings of the Fifth ICGA, Morgan Kaufmann, pp.499–503, 1993.

[57] Surry, P.D., N.J. Radcliffe, and I.D. Boyd, *A Multi-objective Approach to Constrained Optimization of Gas Supply Networks*. Presented at the AISB-95 Workshop on Evolutionary Computing, Sheffield, UK, April 3–4, 1995.

[58] Tong, S.K., Shahidehpour, S.M., and Ouyang, Z., *A Heuristic Short-term Unit Commitment*, IEEE Transactions on Power Systems, Vol.6, No.3, pp.1210–1216, August 1991.

[59] Zhuang, F. and Galiana, F.D., *Unit Commitment by Simulated Annealing*, IEEE Transactions on Power Systems, Vol.5, No.1, pp.311–317, February 1990.