Computer Assisted Mechanics and Engineering Sciences, 7: 735–742, 2000.

Parallel evolutionary optimization of structures

Georg Thierauf and Jianbo Cai
Department of Civil Engineering, University of Essen, 45117 Essen, Germany

(Received November 3, 1999)

By introducing a variable coding technique, a parallel optimization method based on a combination of GAs and ESs is presented. The advantages of both GAs and ESs, like coding of genetic information and adaptation of optimization parameters, are enhanced by this new method.

1. Introduction

Enhanced by the availability of high-speed parallel computing technique, zero-order and direct optimization methods gained renewed interest since the early eighties. Influenced by this development, stochastic search methods and Darwinian methods have been the subject of many publications during the past fifteen years. Among these, the genetic algorithms (GAs) and the evolution strategies (ESs) are two of the most discussed search strategies.

In the following, the basic concepts of GAs and ESs are first briefly described. Subsequently, by introducing a variable coding technique, a parallel optimization method based on a combination of GAs and ESs is presented.

2. The basic genetic algorithm

The basic genetic algorithm was proposed in 1975 by Holland [4] and Dejong [2]. A genetic algorithm can be considered as an iterative scheme, where each iteration cycle forms a generation of an evolutionary process.

Encoded representation of the design variables

As a main property of GAs the fact of using binary encoded individuals (genotype level) has to be noted. The components x_i of the vector

$$X = (x_1\,,\, x_2\,,\, \ldots\,,\, x_i\,,\, \ldots\,,\, x_n)^T \tag{1}$$

are considered as genes.

A component x_i can be represented by a substring with a length of m. The encoded components can be considered as chromosomes. A vector of design variables, usually called an individual, is characterized by a string of chromosomes:

$$\underbrace{0\ 1\ 1\ 0}_{x_1}\ \underbrace{1\ 0\ 1\ 1}_{x_2}\ \cdots\ \underbrace{1\ 0\ 1\ 0}_{x_n}.$$

For a discrete design variable, the length m of the substring depends on the number of the feasible discrete values of the variable. In case of $m = 4$, for example, 16 discrete values can be represented by the substrings (0 0 0 0) to (1 1 1 1). For a continuous design variable, the length m

of the substring depends on the required precision A_c, then the length m of the binary substring may be estimated from the following relationship [6]:

$$2^m \geq [(x_U - x_L)/A_c + 1]. \tag{2}$$

Here, x_L and x_U are the lower and upper bounds of a continuous variable x_i. It is obvious that a continuous variable is treated as a discrete one by dividing its feasible region $[x_L, x_U]$ into 2^m discrete values.

Mating, crossover and mutation

In the ν-th generation, a population of μ individuals, characterized by their genes (chromosomes) exists:

$$\begin{array}{lllllllllll} 1.: & 0 & 1 & 1 & 0 & \ldots & 1 & 0 & 1 & 1, \\ 2.: & 1 & 0 & 0 & 0 & \ldots & 1 & 0 & 0 & 1, \\ \vdots & & & & & & & & & & \\ \mu.: & 1 & 1 & 0 & 0 & \ldots & 1 & 0 & 1 & 0. \end{array}$$

The μ individuals are mated at random. As a result of mating, a crossover of chromosomes takes place. The crossover of two individuals A and B results in A^* and B^*. In a two-site crossover the chromosomes between two arbitrarily set borderlines are exchanged, e.g.

(before crossover)

$$\begin{array}{lllllllllllll} A & = & (\mathbf{0} & \mathbf{1} & \mathbf{1} & \mid & \mathbf{0} & \ldots & \mathbf{1} & \mathbf{0} & \mid & \mathbf{1} & \mathbf{1}) \\ B & = & (1 & 0 & 0 & \mid & 1 & \ldots & 1 & 1 & \mid & 0 & 1) \end{array}$$

(after a two-site crossover)

$$\begin{array}{lllllllllllll} A^* & = & (\mathbf{0} & \mathbf{1} & \mathbf{1} & \mid & 1 & \ldots & 1 & 1 & \mid & \mathbf{1} & \mathbf{1}) \\ B^* & = & (1 & 0 & 0 & \mid & \mathbf{0} & \ldots & \mathbf{1} & \mathbf{0} & \mid & 0 & 1) \end{array}$$

As rare and secondary events, mutations occur: With a low probability, certain chromosomes in the genetic code of individuals, selected at random, are exchanged from 1 to 0 or inversely, e.g.:

$$\begin{array}{lllllllllll} \text{from} & A' & = & (0 & 1 & 1 & \mathbf{1} & \ldots & 1 & 1 & 1 & 1) \\ \text{to} & A'' & = & (0 & 1 & 1 & \mathbf{0} & \ldots & 1 & 1 & 1 & 1) \end{array}$$

After crossover and mutation μ new individuals are generated. The μ old individuals die out.

Reproduction

For a specific optimization problem of the form

$$\begin{array}{ll} \text{minimize} & f(X) \\ \text{subject to} & g_i(X) \leq 0 \quad i = 1, 2, \ldots, m \end{array} \tag{3}$$

the fitness of each individual must be defined. For an admissible point X the value of the objective function can be used for the definition of its fitness. In general it is difficult to generate admissible points in a random process. For this reason, a penalty transformation

$$\text{minimize} \quad f(X) + \rho \sum_{i=1}^{m} \Phi[g_i(X)] \tag{4}$$

can be used, where Φ is an appropriate penalty function [5] and ρ is a penalty coefficient.

With considering Eq. (3), the fitness of an individual X_j can be defined as follows [8]:

$$F_j = (f_{\max} + f_{\min}) - f_j \,, \tag{5}$$

where f_{max} is the maximal value of the objective function between the μ individuals and f_{min} is the minimal one. According to their fitness values, each of the μ new individuals gets α copies into the mating pool of the next generation, where α is calculated as follows, with rounding off:

$$\alpha_j = \frac{F_j}{\sum_{j=1}^{\mu} F_j/\mu} \,. \tag{6}$$

A good individual X_j (with $\alpha_j > 1$) will get more than one copy in the next generation and a bad one gets no copy. This process is called reproduction following the Darwinian principle "survival of the fittest". After reproduction a mating pool of the $(\nu+1)$-th generation with μ individuals is formed, which has a higher average fitness value than that of the ν-th generation.

3. The basic multi-membered evolution strategy

The evolution strategies were first proposed by Rechenberg [9] in 1964. Applications to optimization of technical systems were proposed by Rechenberg in 1973 [10]. A comprehensive description is given by Schwefel [11].

In the following we first restrict ourselves to a basic form of the multi-membered evolution.

Recombination and mutation

A main property of ESs that differs from GAs is that ESs work with real values of the variables (phenotype) instead of encoded binary strings.

In the ν-th generation a population of μ design points, called μ parent vectors, are given:

$$P^{\nu} = (X_1^P,\ X_2^P,\ \cdots\ X_{\mu}^P) \tag{7}$$

with

$$\begin{aligned} X_1^P &= [x_{1,1},\ x_{1,2},\ \cdots\ x_{1,n}] \\ X_2^P &= [x_{2,1},\ x_{2,2},\ \cdots\ x_{2,n}] \\ &\vdots \\ X_{\mu}^P &= [x_{\mu,1},\ x_{\mu,2},\ \cdots\ x_{\mu,n}] \end{aligned} \tag{8}$$

From these μ parent vectors, λ new design points, called offspring vectors, will be generated. For every offspring vector a temporary parent vector $\tilde{X}^P = [\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n]^T$ should be first built by means of recombination. For a continuous problem five recombination cases which can be used selectively are given by Hoffmeister and Baeck [3]:

$$\tilde{x}_i = \begin{cases} x_{a,i} \text{ or } x_{b,i} \text{ randomly} & \text{(A)} \\ \frac{1}{2}(x_{a,i} + x_{b,i}) & \text{(B)} \\ x_{bj,i} & \text{(C)} \\ x_{a,i} \text{ or } x_{bj,i} \text{ randomly} & \text{(D)} \\ \frac{1}{2}(x_{a,i} + x_{bj,i}) & \text{(E)} \end{cases} \tag{9}$$

where $\tilde{x}_i$ is the i-th component of the temporary parent vector $\tilde{X}^P$, $x_{a,i}$ and $x_{b,i}$ are the i-th components of the vectors X_a^P and X_b^P which are two parent vectors randomly chosen from the

population. In case (C) of Eq. (9), $\tilde{x}_i = x_{bj,i}$ means that the i-th component of $\tilde{X}^P$ is chosen randomly from the i-th components of all μ parent vectors.

For discrete optimization problems the following recombinations are employed [1]:

$$\tilde{x}_i = \begin{cases} x_{a,i} \text{ or } x_{b,i} \text{ randomly} & \text{(A)} \\ x_{bj,i} & \text{(B)} \\ x_{a,i} \text{ or } x_{bj,i} \text{ randomly} & \text{(C)} \\ x_{m,i} \text{ or } x_{b,i} \text{ randomly} & \text{(D)} \\ x_{m,i} \text{ or } x_{bj,i} \text{ randomly} & \text{(E)} \end{cases} \tag{10}$$

where the vector X_m^P is not chosen at random but as the best of the μ parent vectors in the ν-th generation. In cases (D) and (E) of Eq. (10) the information from the best parent can be used which results in a better convergence for many problems.

From the temporary parent $\tilde{X}^P$ an offspring vector X_j^O can be created by means of mutation as follows:

$$X_j^O = \tilde{X}_j^P + Z_j\,, \tag{11}$$

where $Z_j = [z_{j,1}\,, z_{j,2}\,, \ldots\,, z_{j,n}]^T$ is a vector of random change. For continuous problems the components $z_{j,i}$ are random numbers from a normal distribution [11]:

$$p(z_{j,i}) = \frac{1}{\sqrt{(2\pi)}\,\sigma_i} \exp\left(-\frac{(z_{j,i} - \xi_i)^2}{2\sigma_i^2}\right), \tag{12}$$

where ξ_i is the expectation, which should have the value zero, and σ_i^2 is the variance, which should be small.

For a discrete problem, the components of the random change vector Z_j have the form [1]

$$z_{j,i} = \begin{cases} (\kappa + 1)\delta x_i & l\,(l < n) & \text{randomly chosen components,} \\ 0 & n - l & \text{other components,} \end{cases} \tag{13}$$

where δx_i is the current difference between two adjacent values in the discrete set and κ is a Poisson-distributed integer random number with the distribution

$$p(\kappa) = \frac{(\gamma)^\kappa}{\kappa!}\, e^{-\gamma}\,, \tag{14}$$

where γ is the deviation of the random number κ and should range from 0.001 to 0.1. A uniformly distributed random choice decides which l components should be changed for a mutation according to Eq. (13). For structural optimization problems, according to our research, a suitable l value ranges from 8 to 12 [1].

Selection

Now we have a population of $(\mu + \lambda)$ individuals. Following the Darwinian principle "survival of the fittest" μ best individuals will be selected according to their fitness for surviving to the next generation. The fitness of an individual is defined as its value of the objective function.

There are two variants of the multi-membered evolution strategy: $(\mu + \lambda)$-ES and (μ, λ)-ES. In the case of $(\mu + \lambda)$-ES, all the $\mu + \lambda$ individuals are ordered according to their fitness values, and in the case of (μ, λ)-ES $(\lambda > \mu)$, only the λ offspring vectors of the ν-generation are ordered according to their fitness values. The first set of the μ elements are chosen as the parent vectors of the next generation.

4. GAs and ESs, a comparison

With respect to the major working scheme both algorithms, evolution strategies and genetic algorithms, are identical, but the details of their implementations establish some significant differences, especially the parameter representations and the selection schemes. In their research, Hoffmeister and Baeck [3], have made a detailed comparison between GAs and ESs. In the following, the results in [3] will be briefly described. Subsequently, with respect to the applications of both algorithms in engineering optimization, some aspects will be discussed.

Many differences between ESs and GAs directly or indirectly stem from a substantial difference in the underlying "genetic" representation used by the algorithms. While in general GAs operate on fixed-sized bit strings which are subsequently mapped to the values of the object variables, ESs work on real-valued vectors. This must not be confused with real-valued "strings" which in a second stage are mapped to the object variables in question. Instead, ESs are operating completely on a phenotypic level, hence they can utilize much more knowledge about the application domain.

When considering the working schemes of GAs and ESs it is apparent that both algorithms have different handling of reproduction rates. With proportional selection in GAs there is a dynamic assignment of reproduction rates to the individuals with respect to their relative fitness, even the worst individual has a minor chance to reproduce. The reproduction rates may differ significantly allowing a super-individual to dominate the next generations quickly, thus leading to early convergence. In ESs there is a static assignment of reproduction rates the μ best individuals within a population with no respect to their relative fitness, i.e. every selected individual reproduces with a rate of $1/\mu$.

Although ESs and GAs use mutation and recombination (crossover), the role of these genetic operators is different. While in GAs mutation only serves to recover lost alleles, in ESs mutation implements some kind of hill-climbing search procedure with self-adapting step sizes σ (or γ). When an ES is trapped on a local optimum the step sizes are reduced to get closer to the optimum. Due to normally (or Poisson-) distributed mutations, occasionally large changes are realized which may give the chance to escape from the basin of attraction of a local optimum. In both algorithms recombination serves to virtually enlarge the population, and thus the covered search space. In ESs it is also an effective means to lessen the tendency of ESs to reduce the search space in order to achieve a higher rate of convergence.

In GAs the effect of a single bit (small) mutation on the genotype level is not easily predictable on the phenotypic level and depends considerably on the used coding function. For a binary coding function, a single mutation causes a position-dependent change $\Delta x \in \{2^1\epsilon, \ldots, 2^i\epsilon, \ldots 2^m\epsilon\}$ of an object variable where i refers to the mutated position and $\epsilon \in R$ denotes the resolution of the coding scheme of the object variables.

In basic GAs recombination is realized by one- or two-site crossover which provides only a limited mixing of information (building blocks) and which does not take into account the internal substring boundaries imposed by the coding function. Recombination as realized in ESs (see Eqs. (9) and (10)) achieves a much better mixing of the genetic information which implicitly obeys the boundaries between object variables since it operates on the real-valued vectors.

The main differences between ESs and GAs are listed by Hoffmeister and Baeck in Table 1 [3].

For engineering optimization problems, according to our experience [1], the following aspects should be considered.

Due to the discrete nature of the binary representation schemes, GAs are suitable for solving discrete optimization problems. A continuous problem can only be translated in to a discrete problem according to the required solution precision A_c (see Eq. (2)). The smaller the value A_c (namely the higher the accuracy), the longer is the length of the binary string, and then, the slower is the convergence. ESs have different mutation schemes for continuous and discrete variables (see Eqs. (12), (13) and (14)), and no limitation for both continuous and discrete problems.

In GAs mutation plays a secondary role in the optimization process. The improvements of the population are mainly gained by the operator crossover. In point of view of linear algebra, a crossover

Table 1. Differences of GAs and ESs

Genetic algorithms	Evolution strategies
Genotype level of individuals (binary coding)	Phenotype level of individuals (real-value representation)
No knowledge about the objective function's properties	Knowledge of the dimension of the objective function (i.e. number of variables)
Parameter space restrictions for coding purpose	No parameter restricts apart from machine-dependencies
Dynamic preservative or static preservative selection	Static, extinctive selection (equal probabilities); more or less selective
Recombination serves as the main search operator	Mutation serves as the main search operator
Secondary role of mutation	Different recombination schemes
No collective self-learning of parameter settings	Collective self-learning of strategy parameters

is nothing else than a linear combination of given vectors. To avoid an improvement in a subspace, the minimal number of given vectors (namely the base for the vector space of dimension n) should at least equal to the number of the design variables n. This forms the lower limit of the population size for GAs. In ESs, mutation works as a main operator, and there is no lower limit to the population size. Hence, by using a (1+1)-ES (two-membered evolution strategy), good results can be obtained for many problems.

5. A COMBINED ALGORITHM

The basic idea to build a combined algorithm is to introduce a new coding technique into GAs. In the basic GAs the base of the binary coding is 2. Each position of a binary code can only take the values 0 or 1. As an improvement Lin *et al.* [7] introduce a decimal coding technique into GAs. Thus, the length of a substring can be sufficiently reduced. In each portion of a decimal code 10 values from 0 to 9 can be taken. Based on this idea, we extend the decimal coding into a "variable coding". For a discrete problem, we take n_d, the number of the feasible discrete values of a variable, as the base of the coding. Then, the length of a substring is always 1, and for each position n_d values can be taken. It is obvious, this is the way of ESs for representation of discrete variables. The same way can be used for the continuous variables. A significant benefit is that the mutation can be used as a main search operator. Thus, in the combined algorithm, we use the variable representation and the operator mutation from ESs and the operators crossover and reproduction from GAs.

The combined algorithm can be formulated as follows.

In the ν-th generation a population of μ design points, called μ individuals, are given:

$$P^{\nu} = (X_1\,,\, X_2\,,\, \ldots\,,\, X_\mu) \tag{15}$$

with

$$\begin{aligned} X_1 &= [x_{1,1}\,,\, x_{1,2}\,,\, \ldots\,,\, x_{1,n}]\,, \\ X_2 &= [x_{2,1}\,,\, x_{2,2}\,,\, \ldots\,,\, x_{2,n}]\,, \\ &\vdots \\ X_\mu &= [x_{\mu,1}\,,\, x_{\mu,2}\,,\, \ldots\,,\, x_{\mu,n}]\,. \end{aligned} \tag{16}$$

The μ individuals are mated at random. As a result of mating, a crossover of chromosomes (one- or two-site) takes place. After crossover μ new individuals are obtained:

$$\begin{aligned} \bar{X}_1 &= [\bar{x}_{1,1}, \bar{x}_{1,2}, \ldots, \bar{x}_{1,n}], \\ \bar{X}_2 &= [\bar{x}_{2,1}, \bar{x}_{2,2}, \ldots, \bar{x}_{2,n}], \\ &\vdots \\ \bar{X}_\mu &= [\bar{x}_{\mu,1}, \bar{x}_{\mu,2}, \ldots, \bar{x}_{\mu,n}]. \end{aligned} \tag{17}$$

For each individual $\bar{X}_i$, a mutation according to (12) or (13) can be taken and the following individuals are generated:

$$\begin{aligned} \hat{X}_1 &= [\hat{x}_{1,1}, \hat{x}_{1,2}, \ldots, \hat{x}_{1,n}], \\ \hat{X}_2 &= [\hat{x}_{2,1}, \hat{x}_{2,2}, \ldots, \hat{x}_{2,n}], \\ &\vdots \\ \hat{X}_\mu &= [\hat{x}_{\mu,1}, \hat{x}_{\mu,2}, \ldots, \hat{x}_{\mu,n}]. \end{aligned} \tag{18}$$

According to their fitness values (see Eq. (5)) each of the μ individuals $(\hat{X}_1, \ldots, \hat{X}_\mu)$ gets α (see Eq. (6)) copies into the mating pool of the next generation which form the μ initial individuals $P^{\nu+1}$ of the $(\nu+1)$-th generation.

6. Parallel sub-evolution-strategy

Similar to ESs and GAs, the combined algorithm works simultaneously with a population of design points in the space of variables. This inherent parallelism allows for an implementation in a parallel computing environment.

With an increasing size of the population, the probability to obtain a global optimum increases almost proportionally. However, large scale problems with an increasing size of population also require much computing time. Following an idea of natural evolution, a parallel sub-evolution-strategy (PSES) was suggested by the authors [1]. The idea of PSES or of the "islands model" can be used for the parallelization of the combined algorithm. The population can be divided into several smaller subpopulations which can undergo their evolution separately and in parallel. In order to prevent the development of evolutionary niches, migration between the subpopulations must be allowed.

In the parallel implementation on a n-processor computer, the whole population is divided into n subpopulations, each processor runs the combined algorithm on its own subpopulation. Periodically some good individuals will be selected and copies of them will be sent to one of its neighbors (migration). Every subpopulation also receives copies from its neighbors, which replace its own "bad" individuals.

The information exchange will be carried out in a determined sequence, For example, from processor $i-1$ to i and from i to $i+1$ [1]. During an optimization process, except for the information exchange, the job on every processor runs independently and a local search can be stopped according to its own termination criterion. If a job on a processor terminated normally, it sends a signal to its neighbours. To keep the cyclic exchange working, exchange between the processor and its neighbours is carried out until all computations are terminated.

7. Concluding remarks

Evolution strategies and genetic algorithms work with the same searching scheme, but the details of their implementations establish some significant differences, especially the parameter representations and the selection schemes. With respect to applications of both algorithms in engineering

optimization, because of the real value representation of variables, evolution strategies seem to have a better flexibility.

In the new algorithm presented above, the variable representation and the operator mutation from evolution strategies are combined with the operators crossover and reproduction from genetic algorithms. In this way, the advantages of both evolution strategies and genetic algorithms, like coding of genetic information, unlimited size of population and adaptation of optimization parameters, are included in the new algorithm.

REFERENCES

[1] J. Cai. *Discrete Optimization of Structures under Dynamic Loading by Using Sequential and Parallel Evolution Strategies* (in German). Doctoral Dissertation. Department of Civil Engineering, University of Essen, Germany, 1995.

[2] K.A. Dejong. *Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* Ph. D. Thesis, University of Michigan, Ann Arbor, MI. 1975

[3] F. Hoffmeister, T. Baeck. *Genetic Algorithms and Evolution Strategies: Similarities and Differences.* Technical Report No. SYS-1/92, University of Dortmund, 1992.

[4] J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Mich., 1975.

[5] U. Kirsch. *Structural Optimization – Fundamentals and Applications.* Springer-Verlag, Berlin, 1993.

[6] C.-Y. Lin, P. Hajela. Genetic algorithms in optimization problems with discrete and integer design variables. *Eng. Opt.*, **19**: 309–327, 1992.

[7] S.-S. Lin, C. Zhang, H.-P. Wang. On mixed-discrete nonlinear optimization problems: A comparative study. *Eng. Opt.*, **23**: 287–300, 1995.

[8] S. Rajeev, C.S. Krishnamoorthy. Discrete optimization of structures using genetic algorithms. *J. Struct. Engrg., ASCE*, **118**(5): 1233–1250, 1992.

[9] I. Rechenberg. *Cybernetic Solution Path of an Experimental Problem.* Royal Aircraft Establishment, Library Translation 1122, Farnborough, England, 1965 (in German: 1964).

[10] I. Rechenberg. *Evolution Strategy: Optimization of Technical Systems According to the Principles of Biological Evolution* (in German). Frommann–Holzboog, Stuttgart, 1973.

[11] H.-P. Schwefel. *Numerical Optimization of Computer Models.* Wiley & Sons, Chichester, 1981 (translated from German, 1977).