An adaptive neural network strategy for improving the computational performance of evolutionary structural optimization

Nikolaos D. Lagaros, Dimos C. Charmpis, Manolis Papadrakakis *

Institute of Structural Analysis and Seismic Research, National Technical University of Athens, 9, Iroon Polytechniou Str., Zografou Campus, GR-15780 Athens, Greece

Received 20 January 2004; received in revised form 5 May 2004; accepted 6 December 2004

Abstract

This work is focused on improving the computational efficiency of evolutionary algorithms implemented in large-scale structural optimization problems. Locating optimal structural designs using evolutionary algorithms is a task associated with high computational cost, since a complete finite element (FE) analysis needs to be carried out for each parent and offspring design vector of the populations considered. Each of these FE solutions facilitates decision making regarding the feasibility or infeasibility of the corresponding structural design by evaluating the displacement and stress constraints specified for the structural problem at hand. This paper presents a neural network (NN) strategy to reliably predict, in the framework of an evolution strategies (ES) procedure for structural optimization, the feasibility or infeasibility of structural designs avoiding computationally expensive FE analyses. The proposed NN implementation is adaptive in the sense that the utilized NN configuration is appropriately updated as the ES process evolves by performing NN retrainings using information gradually accumulated during the ES execution. The prediction capabilities and the computational advantages offered by this adaptive NN scheme coupled with domain decomposition solution techniques are investigated in the context of design optimization of skeletal structures on both sequential and parallel computing environments.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Optimization; Evolution strategies; Neural networks; Parallel computing; Domain decomposition
1. Introduction

Evolutionary algorithms (EA), like genetic algorithms (GA) or evolution strategies (ES), are widely accepted today as a family of effective methods for handling large-scale structural optimization problems and have been successfully applied to a variety of applications in Computational Structural Mechanics [15,20,27,30,33]. EA are capable of locating (near-)optimum solutions within large and irregular search spaces by maintaining a population of potential solutions in the context of an evolution-based procedure. However, locating optimal structural designs using EA is a task associated with high computational cost, since a complete finite element (FE) analysis needs to be carried out for each parent and offspring design vector of the populations considered. The present work is motivated by the time-consuming repeated FE analyses performed in large-scale evolutionary structural optimization and is therefore focused on improving the computational performance of the optimization process.

Artificial intelligence or soft computing techniques have emerged over the last ten years as a valuable tool used to replace time consuming computational tasks in many scientific and engineering applications. The use of such techniques, like neural networks (NN), to predict FE analysis outputs has been previously studied in the context of optimal design of structural systems [1,2,19,22,32], as well as in some other areas of structural engineering applications, such as structural damage assessment, structural reliability analysis, FE mesh generation and fracture mechanics [25,34]. The main concern in substituting FE computations by NN-based schemes has been to determine properly trained NN configurations and mechanisms to detect and correct false NN predictions, in order to ensure the reliability and accuracy of NN results.

The design optimization of skeletal structures is studied in the present work by implementing an ES-based procedure. Each parent and offspring design of the populations involved in the ES process requires a constraints check, which facilitates decision making regarding the feasibility or infeasibility of the structural design considered. Thus, a FE solution is performed for each design, in order to evaluate the displacement and stress constraints specified for the structural problem at hand. This paper presents a NN strategy to reliably predict the feasibility or infeasibility of structural designs and therefore avoid computationally expensive FE analyses in the framework of the ES optimization procedure. The proposed NN implementation is adaptive in the sense that the utilized NN configuration is appropriately updated as the ES process evolves by performing NN retrainings using information gradually accumulated during ES optimization steps. Thus, this NN scheme is capable of adapting to a particular structural optimization run by gradually acquiring prediction capabilities for the regions of the overall design space that are actually visited by the ES procedure.

The prediction capabilities and the computational advantages offered by the proposed adaptive NN strategy in the context of ES-based structural design optimization are investigated on both sequential and parallel computing environments. Two algorithms are employed for the FE solutions performed: (a) a standard direct solver executed sequentially and (b) the FETI domain decomposition method implemented both sequentially and in parallel. The developed software incorporating the methodologies of the present work allows the optimum design of large-scale skeletal structures to be performed in affordable computing times, since it combines the effectiveness provided by soft computing with the speed of computations achieved through parallel processing.

The remainder of this paper is organized as follows. Section 2 briefly describes the implemented ES procedure for structural optimization. Section 3 gives an outline of the FE solution techniques employed and the parallelization concept followed in this work. In Section 4 previous attempts to exploit NN-based predictions in the framework of structural optimization are discussed and the proposed adaptive NN strategy is presented. Section 5 reports performance results of the methodologies described in this work. The paper concludes with some final remarks given in Section 6.
2. Evolution strategies for structural optimization problems

A discrete structural optimization problem can be formulated in the following form:

\[
\begin{align*}
\min \quad & F(s) \\
\text{subject to} \quad & g_j(s) \leq 0, \quad j = 1, \ldots, m \\
& s_i \in \mathbb{R}^d, \quad i = 1, \ldots, n,
\end{align*}
\]

where \( F(s) \) and \( g_j(s) \) denote the objective and \( m \) constraints functions respectively, \( \mathbb{R}^d \) is a given discrete set, while the design variables \( s_i (i = 1, \ldots, n) \) can take values only from this set. The ES algorithm for handling discrete optimization problems of the form (1) is an evolution-based procedure maintaining a population of potential solutions [3,20,26]. The ES process starts with a set of parent vectors; if any of these parent vectors corresponds to an infeasible design, then this parent vector is modified until it becomes feasible. Subsequently, appropriate recombination and mutation operators are utilized to form \( \lambda \) offsprings for each ES-generation, which are checked whether they are in the feasible region. The available selection schemes for the parents of each ES-generation correspond to \((\mu + \lambda)\)-ES and \((\mu, \lambda)\)-ES. Hence, the ES algorithm for structural optimization applications is an iterative process consisting of the following steps:

1. **Selection step:** Selection of parent vectors.
2. **Analysis step:** Solution of the FE problems corresponding to the parent vectors.
3. **Constraints check:** All parent vectors become feasible.
4. **Offspring generation:** Generation of the offspring vectors for the design variables using recombination and mutation operators.
5. **Analysis step:** Solution of the FE problems corresponding to the generated offspring vectors.
6. **Constraints check:** If satisfied continue, else go to step 4.
7. **Parents’ selection step:** Selection of the next generation parents according to \((\mu + \lambda)\) or \((\mu, \lambda)\) selection schemes.
8. **Convergence check:** If satisfied stop, else go to step 4.

The procedure is terminated when the mean value of the objective values from all parent vectors in the last \( 2 \cdot n \cdot \mu/\lambda \) generations has been improved by less than 0.01%.

Although evolutionary-type algorithms were initially developed to solve unconstrained optimization problems [31], during the last decade several methods have been proposed for handling constrained optimization problems as well. The methods based on the use of penalty functions are employed in the majority of cases for treating constraint optimization problems. In a recent work by the authors [20] it has been observed that the death penalty method, according to which every infeasible design point is discarded, performs well in the framework of the ES procedure for the problems considered. A review of the constraint handling methods can be found in [7,20].

3. Solution of finite element equations and parallelization of the structural optimization process

The most time consuming process in structural FE analysis is usually the solution of the system of linear equilibrium equations:

\[
Ku = f,
\]

where \( K \) is the stiffness matrix of the structure, \( f \) is the vector of the external nodal loads and \( u \) is the vector of the unknown nodal displacements. The standard direct method based on Cholesky factorization remains
the most popular solution scheme for FE Eq. (2). According to this method, the stiffness matrix is factor-
ized usually in the form:

$$K = LDL^t,$$

(3)

where $D$ is a diagonal matrix and $L$ is a lower triangular matrix with unit elements on the leading diagonal. Eq. (2) is then solved for the displacements vector $u$ with a forward substitution using $L$, a vector operation involving $D$ and a backward substitution employing $L^t$. A widely known disadvantage of this solution approach is its poor performance when the problems handled contain large numbers of degrees of freedom and/or result in stiffness matrices with large bandwidths. Although this weakness can be alleviated with the use of more involved direct sparse solvers, the treatment of large-scale FE problems with a direct solution scheme constitutes a major computational task in the context of evolutionary structural optimization due to the successive FE solutions required.

This deficiency of the direct approach has led to the development of alternative solution procedures, which can handle large-scale FE problems encountered in evolutionary structural optimization in a computationally more efficient way. One category of methods suitable for this type of problems is based on the concept of domain decomposition (DD), which has served since the eighties as the basis for developing high performance solution schemes in Computational Mechanics. Such solvers, which are essentially iterative rather than direct, are collectively classified as domain decomposition methods (DDMs) (see e.g. [6,13] for DDM classifications).

3.1. The FETI method

The DDM employed as the DD solution kernel of the evolutionary structural optimization procedure is the dual substructuring method of finite element tearing and interconnecting (FETI) properly adjusted to the particular type of FE problems handled in the present work. According to the FETI method, a partitioning of the domain at hand into a set of $n_s$ totally disconnected subdomains is assumed. Thus, the global problem of Eq. (2) is replaced by subdomain equilibrium equations using Lagrange multipliers as interaction forces between the subdomains and enforcing the compatibility of displacements across the subdomain interfaces. FETI facilitates the adaptation of the optimization process to the DD concept and allows for computational gains compared to the direct solver on both sequential computers and parallel processing environments.

The DDM implemented in this paper is the so-called one-level FETI method. The particular FETI configuration utilized is basically described in [6], while further details on theoretical and practical aspects of this implementation can be found in [5,9,13,28]. The two-step mesh partitioning strategy presented in [10] is followed to generate subdomains with appropriate geometric characteristics for the FETI solver. The partitioning output is further processed to eliminate internal mechanisms in subdomains, which are often produced in decomposed skeletal structures due to ‘hanging’ elements arising at subdomain interfaces (i.e. elements with one end pinched to the main body of a subdomain and the other end free) [24].

3.2. Parallel evolutionary structural optimization using domain decomposition

The computing environment used in the present work as the experimental test bed is a dedicated cluster of homogeneous PCs networked via a 100 Mbits/s ethernet cable. The cluster consists of 12 PCs (each with a Pentium III 500 MHz processor and 256 MB RAM) running the Linux operating system and the message passing software parallel virtual machine (PVM) [14]. A single PC of this cluster is utilized for the sequential runs of the structural optimization procedure, while all 12 PCs are employed in parallel runs.
The FETI method is applied in general to partitions with $n_s > n_p$ ($n_p$ is the number of utilized processors), because this results in improved FETI computational performance both in terms of overall execution time and storage requirements [6,11]. Thus, an additional computational step termed as Subdomain Cluster GENeration (SCGEN) has to be performed just after mesh partitioning, in order to appropriately organize the $n_s$ subdomains into $n_p$ subdomain clusters and then assign each cluster to one of the $n_p$ available processors. The network-distributed FETI implementation employed in the present work [5,6] handles the SCGEN task with the graph partitioning optimization software METIS [18]. This FETI code is also used to carry out sequential runs by defining only one subdomain cluster, in order to assign all $n_s$ subdomains to a single processor.

In each ES generation of either $(\mu + \lambda)$ or $(\mu, \lambda)$ selection schemes $\lambda$ offspring designs have to be produced, each of which requires a constraints check using the results of a complete FE analysis. In order to facilitate parallel computing on the $12$ PCs available, $\lambda = n_p = 12$ is adopted in this work. Thus, the FE analyses for the $\lambda = 12$ offsprings of the current ES generation are concurrently performed on the $n_p = 12$ PCs; each PC solves a complete FE problem with the FETI method executed sequentially. This process generally yields $\alpha$ infeasible and $\lambda - \alpha$ feasible designs ($\alpha$ is an integer with $0 \leq \alpha \leq \lambda$). Since only feasible offsprings are involved in the recombination and mutation operations of the ES procedure, the $\alpha$ infeasible designs are modified and therefore require new FE analyses on the $n_p$ available processors. Thus, generally, each of the $\alpha$ FE problems has to be solved by executing the FETI method in parallel on $n_p/\alpha$ PCs with a SCGEN partition containing $n_p/\alpha$ subdomain clusters. Two parallel FETI implementations are investigated to deal with FE reanalyses required during ES optimization steps:

- **Parallel implementation 1 (PI-1).** When $\alpha = 1, 2, 3, 4, 6$ or 12, the totally available $n_p = 12$ PCs can be split into $\alpha$ subgroups, each of which contains $n_p/\alpha = 12, 6, 4, 3, 2$ or $1$ PCs, respectively. Thus, by producing a SCGEN partition with $n_p/\alpha$ subdomain clusters for a particular $\alpha$-value, each of the $\alpha$ FE problems can be solved in parallel with the FETI method executed sequentially. This process generally yields $\alpha$ infeasible and $\lambda - \alpha$ feasible designs ($\alpha$ is an integer with $0 \leq \alpha \leq \lambda$). Since only feasible offsprings are involved in the recombination and mutation operations of the ES procedure, the $\alpha$ infeasible designs are modified and therefore require new FE analyses on the $n_p$ available processors. Thus, generally, each of the $\alpha$ FE problems has to be solved by executing the FETI method in parallel on $n_p/\alpha$ PCs with a SCGEN partition containing $n_p/\alpha$ subdomain clusters. Two parallel FETI implementations are investigated to deal with FE reanalyses required during ES optimization steps:

- **Parallel implementation 2 (PI-2).** When $\alpha = 1, 2, 3, 4, 6$ or 12, PI-2 follows the same parallelization concept applied in PI-1. However, when $\alpha = 5, 7, 8, 9, 10$ or 11, PI-2 manages to exploit the total number of available PCs by avoiding leaving idle PCs when $n_p$ cannot be exactly divided by $\alpha$ (i.e. when $\alpha = 5, 7, 8, 9, 10$ or 11 in our case), the available processing environment is not fully exploited, because a number of PCs is left idle during the parallel computations: when $\alpha = 5$, each of the 5 FE problems is solved in parallel with the FETI method executed on 2 PCs, i.e. $5 \times 2 = 10$ PCs are exploited and 2 PCs remain idle; when $\alpha = 7, 8, 9, 10$ or 11, each of the $\alpha$ FE problems is solved sequentially with the FETI method executed on 1 PC, i.e. $\alpha$ PCs are exploited and $12 - \alpha$ PCs remain idle. The parallelization of FETI solutions with the PI-1 concept in the framework of ES optimization has been first applied in [24]. This implementation becomes feasible due to the SCGEN partitioning scheme employed.

The process of modifying $\alpha$ designs and performing in parallel the corresponding FE reanalyses with properly clustered subdomains for the FETI method is recursively continued until all $\lambda = 12$ offsprings become feasible, i.e. until $\alpha = 0$ is obtained. Similar parallelization concepts are applied during ES initialization for the determination of $\mu$ feasible starting parent designs.
4. Structural optimization combining ES and NN

The use of powerful solution methods for FE equations and the exploitation of parallel processing presented in Section 3 constitute two effective means of improving the computational efficiency of large-scale evolutionary structural optimization. The present section is focused on further accelerating the optimization process with the use of NN-based schemes for the prediction of the constraints checks results.

A neural network attempts to create a desired relation for an input/output (I/O) set of \( n_{\text{IO}} \) learning patterns. This set, which is called training set, consists of a finite number of \( n_{\text{IO}} \) pairs \((\text{inp}, \text{tar}) \in \mathbb{R}^k \times \mathbb{R}^\ell \). The first coordinate is a position in \( k \)-dimensional space, corresponding to the input space, and the second coordinate is a position in \( \ell \)-dimensional space, corresponding to the desired or target space. The algorithm that is usually used in order to form the relation \( \mathbb{R}^k \rightarrow \mathbb{R}^\ell \) between these two spaces is the back propagation algorithm. This algorithm tries to determine a set of parameters called weights, in order to achieve the right response for each input vector applied to the network. If the training is successful, application of a set of inputs to the network produces the desired set of outputs. Based on previous experience [23,36], the Levenberg–Marquard method [17] is used in the present study to minimize the residual values between calculated and desired results in the NN training procedure. Moreover, NN generalization is improved by determining in an automated fashion regularization parameters with the Bayesian framework by MacKay [12,21].

The main objective of this paper is to justify the application of a properly configured metamodel in the framework of an evolutionary structural optimization process, in an effort to obtain inexpensive estimates regarding the feasibility or infeasibility of the candidate optimal designs considered. Thus, it should be mentioned that other metamodels with configuration capabilities and prediction behaviour similar to those of NN can be used in place of the NN-based schemes employed in the present study. Hence, Radial Basis Function models [4], Kriging-based approaches [16,29] or other learning systems following recent advances in statistical learning theory, like Support Vector Machines [35], are some other types of metamodels that could replace NN. Any of these models exhibits its own advantages and disadvantages making it more attractive than other alternatives in certain cases; a comparative investigation to assess the efficiency of each such approximation formula is however beyond the scope of the present work. The proposed prediction schemes and the resulting optimization methodologies work irrespective of the metamodel used and can be combined with any approximation formula similar to NN.

4.1. Conventional NN training

ES and NN have been combined in previous publications by the authors [22,23], in an attempt to enhance the computational efficiency of the ES optimization procedure. The main objective in these implementations was to investigate the ability of the NN to perform the constraints check. The combined ES–NN optimization procedure is performed in two phases. The first phase includes the training set selection, the FE analyses required to obtain the necessary I/O data for NN training, and finally the selection, training and testing of a suitable NN configuration. The second phase is the ES optimization process, but instead of conducting conventional FE analyses the trained NN is used to predict the response of the structure in terms of objective and constraint functions values due to different sets of design variables.

The selection of appropriate I/O training data is one of the important factors in NN training. The number of training patterns used is not the only concern, since the distribution of samples may be of greater importance. Acceptable results can be expected by the trained NN scheme only if its training set includes data over the entire range of the output space. Thus, the selection of I/O training patterns is based on the requirement that the full range of possible results should be represented in the training procedure. In an effort to increase the robustness as well as the computational efficiency of the NN procedure, the training set can be chosen automatically based on a uniform distribution of the design variables in the design space.
When a NN scheme is utilized to predict results of ES constraints checks, the I/O data required for each design contributing to the NN training set consists of: (a) the values of the design variables for the particular design (input data) and (b) the outcome of the constraints check (determined through FE analysis) denoting feasibility or infeasibility of the design (output data).

The combined ES–NN(M) methodology, where M is the size of the NN training set, can be described with the following algorithm:

- **NN training phase:**
  2. *Constraints check:* Perform constraints check for each input pattern vector.
  3. *Training step:* Training of the NN.

- **ES–NN optimization phase:**
  5. *Parents initialization.*
  6. *NN constraints check:* All parent vectors become feasible.
  7. *Offspring generation.*
  8. *NN constraints check:* If satisfied continue, else go to step 7.
  10. *Convergence check:* If satisfied stop, else go to step 7.

In this study the number of patterns used for NN training in the framework of the ES–NN(M) algorithm is \( M = 200 \). This selection is based on a previous study by the authors [23].

4.2. Adaptive NN training

The training set for the conventional ES–NN scheme described in the previous subsection generally consists of I/O data corresponding to a limited number of structural designs, therefore the prediction capabilities offered cannot cover effectively the entire search space available. However, as the ES optimization process of a structural problem progresses, the designs produced after several generations become more and more different compared to the initial training data used for NN training at the beginning of the optimization procedure. Thus, a NN scheme trained according to an initially selected training set becomes gradually incapable of yielding reliable predictions for constraints check results, since the entire design space is poorly represented by the limited number of randomly generated training patterns. Eventually, the output of a poorly trained NN configuration may even resemble a random number generator producing 0 and 1 values (corresponding to feasible and infeasible designs, respectively), in which case NN predictions are completely unreliable.

Two undesirable cases may occur as a result of NN’s incapability to correctly predict the feasibility or infeasibility of offspring designs:

- **A feasible design may be wrongly predicted as infeasible.** This design will be immediately discarded based on the wrong NN prediction (without performing the corresponding FE analysis) and will actually not be involved in the optimization process. Although loosing such a design lies on the safe side, this may be a good-quality design (or even represent the global optimum), which could substantially benefit the ES procedure.

- **An infeasible design may be wrongly predicted as feasible.** This NN prediction claiming feasibility of the design will be checked with a standard FE analysis, which will reveal the wrong prediction made by the NN scheme. Thus, the design will finally not be involved in the optimization process, since it will be discarded based on computed (and not predicted) information obtained through FE analysis.
In the above cases it is assumed that predicting feasibility initiates a verification procedure to check the NN output through standard FE analysis (in order to prevent the ES process from considering infeasible designs), while a prediction nominating a design as infeasible is accepted without further checking (in order to avoid a time consuming FE analysis). Thus, the occurrence of the first case cannot be detected during the optimization process. On the other hand, detecting the occurrence of the second case actually signifies that the utilized NN configuration cannot act reliably upon the offspring vectors produced at the current stage of the optimization process.

To facilitate the discussion in the sequel of the present work, offspring structural designs are codified according to the NN predictions made for their feasibility or infeasibility. A NN prediction may be either Correct (C) or Wrong (W), while a NN configuration may predict either a Feasible (F) or an Infeasible (I) design. Hence, an offspring design can be characterized as either CF, CI, WF or WI, where the first letter of these abbreviations is associated with the correctness or not of the NN prediction (options: C or W), while the second letter expresses the NN output regarding feasibility or infeasibility of the design (options: F or I). For example, a WF-design is an offspring design, which has been Wrongly predicted by the NN scheme as Feasible. Following this codification, the undesirable situations caused by NN’s poor predictions correspond to WI and WF-designs.

In various methodologies combining ES and NN examined by the authors in the past [1,2,19,22,32], the NN training set was selected in advance, prior to initializing the optimization procedure, or during the optimization procedure by accumulating data up to a stationary point in order to collect enough patterns for NN training. In these attempts, the problem of controlling the inaccuracy in NN predictions emerged, which affected the optimization process and the optimum design finally achieved. These weak points of conventional NN-based ES procedures are overcome in the present work with an adaptive NN strategy, which reliably predicts the feasibility or infeasibility of structural designs by appropriately updating the utilized NN configuration as the ES process evolves. The proposed adaptive NN scheme performs NN retrainings using information gradually accumulated during ES execution and is therefore capable of adapting to a particular structural optimization run by gradually acquiring prediction capabilities for the regions of the overall design space that are actually visited by the ES procedure. Moreover, the convergence of the optimization process to an infeasible design due to possibly wrong NN predictions is avoided by performing conventional constraints checks through FE analysis every time a NN prediction nominates a design as feasible.

In order to realize the idea of updating the utilized NN scheme with retrainings performed during the ES process, two issues have to be dealt with:

1. *When should the NN scheme be retrained?* This issue is concerned with defining a criterion to decide whether a particular NN configuration yields acceptably reliable predictions or not. Using this criterion we can decide at any instance of the ES process whether the current NN configuration should be retrained or not.

2. *How should the NN scheme be retrained?* This issue refers to the composition of the NN retraining set. A number of offspring designs have to be carefully selected, in order to contribute I/O data to the NN retraining set.

These two issues are discussed in the following sub-subsections.

### 4.2.1. Definition of criterion for deciding NN retraining

An obvious response to the first of the issues above is to request a NN retraining every time a wrong NN prediction is detected. However, although NN updates are not as time consuming as the FE analyses conducted, each NN retraining performed during the ES process is associated with some computational cost. Therefore, the criterion, according to which the resorting to NN retraining is decided, must be allowed to
show some tolerance with wrong NN predictions, in order to avoid wasting computing time due to unnec-
essarily large numbers of NN retrainings. Thus, the detection of a single or a few wrong NN predictions
should not immediately invoke the NN retraining procedure, but rather act as an alarm sign that the cur-
rent NN configuration may be loosing its prediction reliability. After all, a single or a few mistakes does not
mean that NN prediction capabilities are completely lost at once.

In the present work NN retraining is requested only after detecting 10 wrong NN predictions corre-
sponding to WF-designs. This relaxed criterion shows some tolerance in the mistakes made by the current
NN configuration on the anticipation that the correct NN predictions in forthcoming constraints checks
will be substantially more than the wrong ones. This tolerance is lifted once the critical number of 10 wrong
NN predictions is reached, which signifies that the current NN configuration needs to be updated. Fur-
thermore, in order to escape from a particularly poor NN configuration, the NN retraining criterion is en-
hanced by appending an additional requirement discouraging consecutively wrong predictions: NN
retraining is requested after detecting 10 wrong NN predictions overall or 5 in a row, whichever comes first.
Although this appended requirement regarding consecutively wrong predictions has never been fulfilled in
any test run performed in this work (the detection of 10 non-consecutively wrong predictions has been more
critical in all cases considered), it may prove useful in other applications.

4.2.2. Composition of NN retraining set

The straightforward solution to the second issue posed regarding the composition of the NN retraining
set is to add to the previous training set the I/O data of the 10 WF-designs, which have contributed to the
fulfilment of the criterion for NN retraining. This retraining approach is based on the rationale that the
current NN configuration should learn from its mistakes, in order to avoid repeating these again. However,
a retraining oriented towards healing specific NN prediction weaknesses may confuse the utilized NN
scheme, since NN prediction reliability may be enhanced in some regions of the design space and disturbed
elsewhere. Therefore, it may be more effective to enrich the NN retraining set with I/O data belonging not
only to WF-designs.

Seven NN retraining schemes are examined in this study. They are summarized in Table 1 and explained
in the sequel of this sub-subsection. The idea employed by these schemes is to enrich the NN training set
with I/O data corresponding to a relatively small number of wrong or nearly wrong NN predictions and
retrain the NN by initiating the training with the weights coefficients of the previous NN configuration.
To explain the meaning of nearly wrong NN predictions, it should be mentioned first that the NN imple-
mentation used in the present work employs a sigmoid transfer function, whose output values are real num-
bers in the interval [0,1]. A NN output in the range [0,0.5] nominates a design as feasible, while a NN
output within (0.5,1] nominates a design as infeasible, i.e. the border value of 0.5 distinguishes predicted
feasibility and infeasibility. A nearly wrong NN prediction about a CF-design is a prediction, which nom-
inates the design as feasible (i.e. the prediction is based on a NN output in the interval [0,0.5]) and is con-

<table>
<thead>
<tr>
<th>Adaptive NN training scheme</th>
<th>Number of designs contributing I/O data to the NN retraining set</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10 WF-designs - CF-designs - WI-designs 10</td>
</tr>
<tr>
<td>B</td>
<td>10 WF-designs 2 CF-designs - WI-designs 12</td>
</tr>
<tr>
<td>C</td>
<td>10 WF-designs 4 CF-designs - WI-designs 14</td>
</tr>
<tr>
<td>D</td>
<td>10 WF-designs 2 CF-designs 0-2 WI-designs 12-14</td>
</tr>
<tr>
<td>E</td>
<td>10 WF-designs 2 CF-designs 0-4 WI-designs 12-16</td>
</tr>
<tr>
<td>F</td>
<td>10 WF-designs 4 CF-designs 0-2 WI-designs 14-16</td>
</tr>
<tr>
<td>G</td>
<td>10 WF-designs 4 CF-designs 0-4 WI-designs 14-18</td>
</tr>
</tbody>
</table>
firmed regarding its correctness through FE analysis results, but the associated NN output is close to the border value of 0.5 (i.e. it nearly lies within (0.5,1], which would yield a wrong NN prediction). Hence, the first three NN retraining schemes examined are: (i) the A-scheme, in which NN retraining is performed by adding to the training set the 10 extra training patterns of detected WF-designs; (ii) the B-scheme, which adds to the training set I/O data of the 10 WF-designs plus 2 CF-designs with nearly wrong NN predictions; (iii) the C-scheme, which exploits additional training patterns of 10 WF-designs plus 4 CF-designs with nearly wrong NN predictions.

If the initial NN training set is small (e.g. consists of only 6 patterns, as in certain runs of the present work), then the retraining schemes A, B and C, which take into account only WF-designs and maybe some CF-designs with nearly wrong NN predictions, will force the retrained NN configuration to nominate almost every design considered as an infeasible one. This observation has led to the investigation of four additional retraining schemes making use of I/O data belonging also to WI-designs. As a general rule, however, all designs contributing I/O data to the NN training set should have their associated NN predictions regarding feasibility or infeasibility checked through FE analysis. As already stated, WF and CF-designs are detected using FE results, but WI-designs are generally not detectable during the ES process (NN outputs predicting infeasibility are accepted without verification through FE analysis). Thus, in order to detect WI-designs for the four additional retraining schemes, a number of NN predictions nominating designs as infeasible have to be checked through FE analysis. Such checks yield the desired WI-designs, but reveal also CI-designs, which are ignored and not included in the retraining set. It is pointed out that the criterion for retraining refers always to WF-designs only, i.e. detected WI-designs do not count among the 10 wrongly predicted designs inducing NN retraining.

Following the above discussion, the four additional NN adaptive training schemes are defined as follows: (iv) the D-scheme adds to the training set extra training patterns of 10 WF-designs plus 2 CF-designs with nearly wrong NN predictions plus a number of WI-designs, which are detected among 2 randomly chosen designs predicted as infeasible (i.e. the WI-designs detected among the 2 randomly chosen designs predicted as infeasible are added to the training set, while the remaining CI-designs are ignored); (v) the E-scheme takes into account I/O data of 10 WF-designs plus 2 CF-designs with nearly wrong NN predictions plus a number of WI-designs, which are detected among 4 randomly chosen designs predicted as infeasible; (vi) the F-scheme enriches the retraining set with I/O data of 10 WF-designs plus 4 CF-designs with nearly wrong NN predictions plus a number of WI-designs, which are detected among 2 randomly chosen designs predicted as infeasible; (vii) the G-scheme adds training patterns due to 10 WF-designs plus 4 CF-designs with nearly wrong NN predictions plus a number of WI-designs, which are detected among 4 randomly chosen designs predicted as infeasible.

4.2.3. ES–NNX algorithms

Two algorithms are proposed for implementing the seven adaptive NN training schemes A–G in the framework of ES optimization. The first algorithm denoted as ES–NNX(M), where X is the adaptive training scheme adopted and M is the size of the starting NN training set, consists of the following steps:

- **First ES optimization phase:**
  1. Parents initialization.
  3. Constraints check: All parent vectors become feasible.
  4. Offspring generation.
  5. Analysis step.
  6. Constraints check.
  7. Selection step.
8. **First ES phase stop check**: The first phase ES procedure stops when a data base of $M$ training patterns have been accumulated, else go to step 4.

- **NN training phase**:
  9. **Training set accumulation step**: Depending on the adopted training scheme $X$, add $N$ training patterns to the existing training set; in the first training: $N = M$, in all subsequent trainings: $N = 10–18$ (see Table 1).
  10. **Training step**: Training of the NN.

- **ES–NNX($M$) optimization phase (second ES optimization phase)**:
  11. Offspring generation.
  12. NN constraints check.
  13. Retraining check: If the criterion for retraining is satisfied then go to step 9, else continue.
  15. **Second ES phase convergence check**.

In this study the number of patterns used for initial NN training in the framework of the ES–NNX($M$) algorithm is $M = 50$ or $M = 100$. These selections correspond respectively to 25% or 50% of the training set size used in the conventional ES–NN($M$) methodology (see Section 4.1).

The second algorithm denoted as ES–NNX($\mu$), where $X$ is the adaptive training scheme adopted and $\mu$ is the size of the starting NN training set coinciding with the number of ES parent designs, consists of the following steps:

- **First ES optimization phase**:
  1. Parents initialization.
  3. Constraints check: All parent vectors become feasible.

- **NN training phase**:
  4. **Training set accumulation step**: Depending on the adopted training scheme $X$, add $N$ training patterns to the existing training set; in the first training: $N = \mu$, in all subsequent trainings: $N = 10–18$ (see Table 1).
  5. **Training step**: Training of the NN.

- **ES–NNX($\mu$) optimization phase (second ES optimization phase)**:
  6. Offspring generation.
  7. NN constraints check.
  8. Retraining check: If the criterion for retraining is satisfied then go to step 4, else continue.
  10. **Second ES phase convergence check**.

5. **Numerical results**

The performance of the optimization methodologies discussed in this work is investigated with two sizing optimization test problems. A 3D bus frame and an aircraft hangar space truss have been considered. The following abbreviations are used in this section: ES refers to the standard implementation of the ES optimization procedure, in which constraints checks are performed using conventional FE analyses without resorting to NN predictions; ES–NN($M$) refers to the conventional ES combined with the NN methodology described in Section 4.1; ES–NNX($M$) and ES–NNX($\mu$) refer to the algorithms of Section 4.2.3 combining ES with an adaptively trained NN (the adaptive NN training scheme adopted is designated by $NNX$).
5.1. 3D bus frame

The first test example considered is the 3D bus frame shown in Fig. 1, which consists of 1269 elements, 753 nodes and 4489 degrees of freedom. The cross-sections of all frame members are assumed to be tubular. The members of the 3D bus involved in the optimization process are divided into 15 independent groups, which are illustrated in Fig. 1 with different colors. Each of these groups corresponds to a design variable taking values from a database containing members with various cross-sectional geometric properties. The objective function minimized in this sizing optimization problem is the weight of the structure, while only stress constraints are imposed in the bus frame's elements. The bus structure is designed taking into account a combination of loading conditions including dead loads, breaking (mass × negative acceleration loading at the y-direction) and asymmetrical vertical loading (mass × acceleration loading at the z-direction).

Table 2 investigates the performance of the \((\mu + \lambda)\)-ES selection scheme for various numbers of parents and offsprings. It can be seen that the three schemes examined perform barely the same in terms of the optimum design achieved. However, the \((6 + 12)\)-ES scheme outperforms the other two in terms of convergence speed, since the \((6 + 6)\)-ES scheme requires almost double number and the \((12 + 12)\)-ES scheme almost triple number of FE analyses compared to \((6 + 12)\)-ES for achieving roughly the same output quality. Based on these results the \((6 + 12)\)-ES scheme is selected for the remaining test runs of this test example. This selection is enforced by the performance statistics of the three \((\mu + \lambda)\)-ES selection schemes over 100 runs given in Table 3.

A performance comparison of various adaptive retraining schemes versus the conventional non-adaptive one is shown in Table 4. According to these results, the non-adaptive scheme requires the largest number of training patterns \((M = 200)\). The adaptive NN configurations require substantially smaller numbers of training patterns in total, while they perform a small number of adaptive training steps. In particular, the A-scheme using \(\mu = 6\) patterns as an initial training set results in 16 accumulated training patterns only and just 2 adaptive steps. However, in order to draw a firm conclusion on the robustness and advantages of
each optimization methodology, it remains to be seen how the performance of the conventional ES approach compares against ES–NN and ES–NNX in terms of optimum design achieved and computing time consumed.
The performance of various ES methodologies in terms of both computing time required and optimum design achieved is presented in Table 5, where the computations are performed in a sequential computing environment. A first remark on the results shown is that the overall computing time consumed by the combined methodologies ES–NNX is about 60% to 80% less than the time required by standard ES. From Table 5 it can be seen that despite the use of a large number of patterns for NN training by the non-adaptive ES–NN(200) methodology the optimization algorithm has not managed to converge to a feasible design. Among all ES methodologies examined, ES–NNA(6) performs best in terms of computing time demands, but the small number of 16 training patterns used overall seems to be inadequate to lead the ES process to the optimum design achieved by other ES methodologies. Therefore, ES–NNE(6), ES–NNF(6) and ES–NNG(6) can be considered as the overall best performing methodologies for this test example, since they achieve the best quality design (15% less weight) at a 30% more computing time compared to the entire optimization process. Finally, Fig. 2 depicts the convergence history of some of the NN-based ES methodologies in comparison with the standard ES. As a general remark, it can be said that the best adaptive scheme corresponds to a compromise between the total training patterns used, the retraining steps required and the optimum design achieved.

5.2. Aircraft hangar

The preliminary design of a space truss roof for an aircraft hangar to service 747–400 aircrafts, which is a typical double-layered space truss structure, is considered in this second test example. The hangar is intended to service three planes at a time requiring a total clear width of 194 m and a clear depth of 89 m, therefore an area of 100 m x 200 m is designed to be covered. The space truss is shown in Fig. 3 and consists of 51448 elements, 13031 nodes and 38961 degrees of freedom. The total number of design variables is eight, since the hangar’s elements follow the pattern of the member groups schematically depicted in the coarse truss of Fig. 4 (elements located in a range of 5 m from each supported side are assigned to different groups compared to the elements of the remaining structure). Each truss member is assumed to have a thin-walled tubular cross-section, while group elements are selected from a database containing members with
various cross-sectional geometric properties. The objective function to be minimized is the weight of the structure, while both stress and displacement constraints are imposed to this sizing optimization problem according to Eurocode 3 [8]. Loading conditions are grouped in two categories: (i) Dead loads ($G$) and (ii) Live loads ($Q$), thus each truss member is checked for actions that correspond to the load combination $1.35G + 1.50Q$.

Table 6 examines the performance of the $(\mu + \lambda)$-ES selection scheme for various numbers of parents and offsprings. The trend observed is similar to the previous test example, i.e. all three configurations considered perform similarly in terms of the optimum design achieved, but the $(6 + 12)$-ES scheme outperforms the other two in terms of the number of FE analyses required. Thus, based on these results the $(6 + 12)$-ES scheme is selected for the remaining test runs of this test example. Performance statistics of the selected $(6 + 12)$-ES selection scheme over 100 runs are shown in Table 7.
Table 8 illustrates a performance comparison of various adaptive retraining schemes versus the conventional non-adaptive one. As in the bus frame example, the non-adaptive scheme requires the largest number of training patterns to yield acceptably accurate NN predictions, while the adaptive NN configurations require a few training patterns and retraining steps. In particular, the A-scheme using only \( \mu = 6 \) patterns as an initial training set results in 26 training patterns in total and 3 adaptive steps. These results verify the advantages offered by the proposed adaptive strategy, which collects the fewest data possible for NN training, without setting limits to the NN capability in predicting the outcomes of constraints checks.

Both sequential and parallel test runs are carried out for this example. The FETI domain decomposition method is applied to a mesh partition containing \( n_s = 48 \) subdomains with optimal aspect ratios (Fig. 5). The graph representation of this single mesh partition is used as input to the METIS software, in order to produce optimal SCGEN solutions for all required \( x \)-values (see Section 3.2) before invoking the ES algorithm and be able to allocate \( n_p/x \) PCs for each associated parallel run of the FETI method. Note that the production of the several SCGEN partitions required is a very fast computational task due to the small size of the processed graphs [6]. The SCGEN results obtained are then distributed across the 12 networked PCs and the corresponding message passing information is stored. Thus, each PC possesses the required data for parallel FETI execution, which is exploited according to the needs encountered as the parallel optimization process evolves.

Table 9 summarizes the performance of various ES methodologies in terms of both elapsed time required and optimum design achieved. The parallel test runs reported are performed with the two FETI implementations PI-1 and PI-2 described in Section 3.2. The conclusions drawn from this table regarding sequential runs are similar to those of the first test example, since the overall time required by the combined methodologies ES–NN and ES–NNX is about 60–70\% less than that of the standard ES procedure with either Direct or FETI solvers. The corresponding computational gains achieved through parallel processing with ES–NN and ES–NNX compared to conventional ES are about 15–35\% in the case of FETI implementation PI-1 and 40–50\% in the case of PI-2. Clearly, due to the capability of PI-2 to fully exploit the total number

<table>
<thead>
<tr>
<th>Starting training set</th>
<th>Adaptive NN training scheme</th>
<th>Final training set</th>
<th>Retraining steps</th>
<th>FE analyses</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>–</td>
<td>200</td>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>100</td>
<td>A</td>
<td>120</td>
<td>3</td>
<td>243</td>
</tr>
<tr>
<td>50</td>
<td>A</td>
<td>80</td>
<td>4</td>
<td>242</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>26</td>
<td>3</td>
<td>232</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>54</td>
<td>5</td>
<td>246</td>
</tr>
<tr>
<td>6</td>
<td>C</td>
<td>62</td>
<td>5</td>
<td>244</td>
</tr>
<tr>
<td>6</td>
<td>D</td>
<td>60</td>
<td>5</td>
<td>244</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>47</td>
<td>4</td>
<td>230</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>49</td>
<td>4</td>
<td>254</td>
</tr>
<tr>
<td>6</td>
<td>G</td>
<td>51</td>
<td>4</td>
<td>250</td>
</tr>
</tbody>
</table>
of available PCs when \( n_p \) cannot be exactly divided by \( z \), PI-2 yields substantially reduced processing times compared to PI-1 in the case of ES–NN and ES–NN.X methodologies. The overall computational gains are smaller in parallel compared to sequential runs when ES–NN or ES–NN.X are employed instead of ES, because NN predictions succeed in substituting a significant number of FE analyses and therefore in reducing the computationally intensive operations performed during the optimization procedure, leaving this way fewer computations to be performed in parallel as the ES process evolves.

Fig. 5. Mesh partition of the curved space truss with 48 subdomains. Interface elements are split among neighboring subdomains to avoid mechanisms caused by ‘hanging’ truss bars.

<table>
<thead>
<tr>
<th>Optimization methodology</th>
<th>Time (s)</th>
<th>Weight (kN) of optimum design achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequential execution (direct solver)</td>
<td>Sequential execution (FETI solver)</td>
</tr>
<tr>
<td>ES</td>
<td>14529</td>
<td>7977</td>
</tr>
<tr>
<td>ES–NN(200)</td>
<td>4492</td>
<td>2476</td>
</tr>
<tr>
<td>ES–NNA(100)</td>
<td>5453</td>
<td>3001</td>
</tr>
<tr>
<td>ES–NNA(50)</td>
<td>5443</td>
<td>2996</td>
</tr>
<tr>
<td>ES–NNA(6)</td>
<td>5206</td>
<td>2865</td>
</tr>
<tr>
<td>ES–NNB(6)</td>
<td>5517</td>
<td>3038</td>
</tr>
<tr>
<td>ES–NNC(6)</td>
<td>5473</td>
<td>3013</td>
</tr>
<tr>
<td>ES–NND(6)</td>
<td>5488</td>
<td>3019</td>
</tr>
<tr>
<td>ES–NNE(6)</td>
<td>5170</td>
<td>2846</td>
</tr>
<tr>
<td>ES–NNF(6)</td>
<td>5701</td>
<td>3138</td>
</tr>
<tr>
<td>ES–NNG(6)</td>
<td>5626</td>
<td>3097</td>
</tr>
</tbody>
</table>
According to Table 9, ES–NN(200) requires less time than any adaptive ES–NNX methodology both in sequential and parallel computing environments, but converges to the worst-quality design. This is due to the incapability of the conventionally trained NN to provide acceptable prediction results. In this test example all the adaptive schemes perform equally well in terms of optimum design achieved apart from the ES–NNA(6) and ES–NNB(6) methodologies. In terms of time required, ES–NNE(6) is the fastest in both sequential and parallel computing environments. Compared to the standard ES, ES–NNE(6) demands about 65% less computing time on a single processor and 45% less in the parallel processing environment using the FETI implementation PI-2. Finally, Fig. 6 depicts the convergence history of some of the NN-based ES methodologies in comparison with the conventional ES.

6. Conclusions

The adaptive strategy for NN training proposed in this work substantially improves the prediction reliability of a NN configuration in the context of the ES optimization procedure. This combination of ES and NN is motivated by the time consuming repeated FE analyses encountered in evolutionary structural optimization problems, since the computational effort required by the optimization procedure becomes excessive in large-scale problems. The use of NN-based schemes to predict the outcome of constraints checks during the ES procedure can drastically accelerate the overall optimization process, since NN-predicted structural response was found to fall within acceptable accuracy tolerances.

A number of adaptive NN training schemes has been examined in this study. These schemes are based on two concepts using either a small or a large starting training set. The test examples examined in this study lead to the conclusion that a small initial training set performs better. This is explained by the fact that the proposed strategy of adaptively creating the NN training set is based on the idea of gradually providing a NN configuration with prediction capabilities for the regions of the overall design space that are actually
visited by the ES procedure. This concept is best realized using a small initial training set, which is adaptively enriched with few but really effective NN training patterns. It has been observed that, when a small initial training set is used, it is necessary to gradually enhance the training set with additional training patterns corresponding to both feasible and infeasible designs, as implemented in adaptive training schemes D–G.

The proposed optimization methodology, which combines the ES algorithm with an adaptively trained NN predictor for constraints checks, is an efficient, robust and generally applicable procedure capable of finding the global optimum design in complicated structural optimization problems. This combined methodology can effectively handle computationally intensive optimization problems by implementing efficient domain decomposition solvers in both sequential and parallel computing environments. The numerical results reported in this study reveal the computational advantages offered by the proposed methodology over standard ES. More specifically, if we consider the conventional ES procedure equipped with a sequential direct solver as the reference optimization methodology, then for the test examples examined in this study the proposed algorithm combining ES and NN achieves the same optimum weight obtained by standard ES in about 4 to 5 times less computing time on a single computer and in 30 times less computing time using a parallel processing environment of 12 processors. To conclude, the ES algorithm combined with adaptive NN was found to provide high-quality results in affordable computing time, making this way the structural optimization process more tractable in engineering practice.

References


