Incremental Approximation of Nonlinear Constraint Functions for Evolutionary Constrained Optimization

Yaochu Jin, *Senior Member, IEEE*, Sanghoun Oh and Moongu Jeon

***Abstract*— This paper proposes an alternative approach to efficient solving of nonlinear constrained optimization problems using evolutionary algorithms. It is assumed that the separateness of the feasible regions, which imposes big difficulties for evolutionary search, is partially resulted from the complexity of the nonlinear constraint functions. Based on this hypothesis, an approximate model is built for each constraint function with an increasing accuracy, starting from a simple linear approximation. As a result, the feasible region based on the approximate constraint functions will be much simpler, and the isolated feasible regions will become more likely connected. As the evolutionary search goes on, the approximated feasible regions should gradually change back to the original one by increasing the accuracy of the approximate models to ensure that the optimum found by the evolutionary algorithm does not violate any of the original constraints. Empirical studies have been performed on 13 test problems and four engineering design optimization problems. Simulation results suggest that the proposed method is competitive compared to the state-of-the-art techniques for solving nonlinear constrained optimization problems.**

I. Introduction

Evolutionary algorithms (EAs) have been employed for solving many scientific and engineering optimization problems. In general, conventional EA-based optimization techniques are insufficient in searching highly constrained design spaces, particularly those with separated, small feasible regions. To handle constraints in evolutionary optimization, a variety of techniques have been developed [1], [2]. The most often used approaches in EAs apply a penalty function to infeasible solutions. Since equality constraints can usually be converted into inequality constraints, without the loss of generality, we consider the following minimization problem with nonlinear inequality constraints:

$$\min \quad f(\mathbf{x}), \quad \mathbf{x} = (x_1, ..., x_n), \tag{1}$$

$$\text{subject to} \quad g_j(\mathbf{x}) \leq 0, \quad j \in \{1, 2, ..., m\}, \tag{2}$$

where $g_j(\mathbf{x})$ are nonlinear functions. Penalty function approaches typically convert the constrained optimization problem in Eqns. (1)-(2) into an unconstrained one by adding a certain penalty value (always larger than zero for minimization problem) multiplied by a penalty factor $r_j > 0$ to the objective function as follows:

$$F(\mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^{m} r_j \, \phi(g_j(\mathbf{x})), \tag{3}$$

Yaochu Jin is with the Department of Computing, University of Surrey, Guildford, Surrey, GU2 7XH, UK. Email: yaochu.jin@surrey.ac.uk.

Sanghoun Oh and Moongu Jeon are with the Department of Information and Communications, Gwangju Institute of Science and Technology, Gwangju 500-712, Republic of Korea.

where $\phi(g_j(\mathbf{x}))$ is often defined as follows:

$$\phi(g_j(\mathbf{x})) = \max \{0, (g_j(\mathbf{x}))^{\alpha}\}, \tag{4}$$

where n is the dimension of the search space, m the number of constraints, and α is a constant often set to 1 or 2.

The main problem with the penalty function approaches is that there is no principled way to determine the penalty factor r_j. To address this problem, two classes of methods have been proposed for setting up the penalty factor, namely, dynamic penalty and adaptive penalty methods. The dynamic penalty methods monotonically increase the penalty factor over generations, e.g., in [3]. Dynamic penalty methods increase the penalty factor using a predefined rule regardless of the search dynamics of the population. A better alternative is to adjust the penalty factor based on the feedback of the search results. The basic idea is to increase the penalty factor if the best individual in the last generations is always feasible, and to decrease the penalty factor otherwise. For example, the adaptive segregational constraint handling evolutionary algorithm (ASCHEA) [4] allows for fine tunings of the penalty factor and maintains diversity using niching. Nevertheless, this method needs to include a few user-defined parameters and requires a high number of fitness evaluations. Another adaptive penalty method is termed as self-adaptive fitness formulation (SAFE) [5], where infeasible solutions that have a high fitness value are also favored in selection. SAFF does not require any extra user-defined parameters and its implementation is relatively easy, but it incurs a large number of fitness function evaluations.

An interesting method that is closely related to the penalty function approaches is the stochastic ranking evolution strategy (SRES) introduced in [6]. SR can be seen as a stochastic version of the bubble-sort algorithm. SRES is based on a (μ,λ) evolution strategy that employs a stochastic ranking (SR) selection scheme. The motivation of SR is to balance the influence of the objective function and the penalty function in selection by using the dominance comparison between the penalty and fitness. SRES has shown to be advantageous over the penalty function approaches on 13 test problems.

Although it attempts to achieve a good balance between the penalty and fitness in terms of dominance, SRES is still a single-objective evolutionary algorithm in principle. Recently, handling constrained optimization using the multi-objective approach has become increasingly popular [7]. In the multi-objective approaches, all or a few of the constrains are converted into objectives to be minimized [8], [9]. However, it has been found that treating the constraint

functions and the original objective equally in the multi-objective approaches is not a good idea. Instead, a bias towards the feasible solutions is more helpful [10], [11], [12]. Other approaches include repair algorithms [15], where the infeasible solutions are modified so that they become feasible, e.g., by replacing it with its closest feasible solution that can be found using a local search algorithm, and the use of problem-specific representations or genetic operators to avoid infeasible solutions [16], [14]. A comprehensive survey with detailed discussions of the existing techniques for handling constrained optimization can be found in [2]. A collection of selected research work on evolutionary optimization of constrained optimization is provided in [13].

This paper addresses nonlinear constrained optimization problems with separated feasible regions from a perspective that is fairly different from existing work. Our basic idea is to purposely simplify the nonlinear constraints in the early stage of the search so that the original separated feasible regions become connected to make it easier for evolutionary algorithms to find their way to the global optimum. To this end, we use a machine learning model, a neural network model in this work, to approximate each nonlinear constrain function. It is essential that the approximate model of the nonlinear constraints starts from a simple approximation (e.g., a linear model) in the early stage of evolutionary search. As the evolution proceeds, the accuracy of the approximate constraints should increase incrementally. At the late stage of the search, sufficiently accurate approximation of the constraints is desired.

The use of approximate models for nonlinear constraint functions is related to the research work on using surrogates in evolutionary optimization [17], [18], particularly the use of surrogates for constraints [19], [20]. However, the motivation for approximating the nonlinear constraint functions in this work is completely different from that in [19], [20], where the target for approximating the constraint functions was to reduce time-consuming evaluations, not to manipulate the complexity of the feasible regions. Consequently, the way of training and using the approximate constraints in this work is completely different from that in [19], [20]. Another related research topic is dynamic optimization [21], specifically when the constraint functions change over time [22]. But again, the motivation of changing the constraints in this work is different from that in [22], since in our work, we change the originally stationary constraint functions on purpose, whereas the constraint functions are themselves time-varying in [22].

The rest of this paper is organized as follows. In Section II-A, we discuss our hypothesis and the basic idea of the work, followed by a brief description of the SRES in Section II-B, and the details of the proposed method for synthesizing the constraints are presented in Section II-C. Empirical comparative studies the proposed algorithm and the SRES on 13 test problems and four engineering design problems are conducted and discussed in Section III. A summary and conclusion is provided in Section IV.

II. The Proposed Algorithm

A. Incremental Approximation of the Constraint Functions

A design space with separated feasible regions poses grand challenges to evolutionary search in that a path to the global optimum may be blocked by the infeasible regions, as illustrated in Fig. 1. To ease the evolutionary search, it is desired that the path to the global optimum can be made available temporarily, which can be realized by artificially expanding the feasible region by means of simplifying the nonlinear constraints.

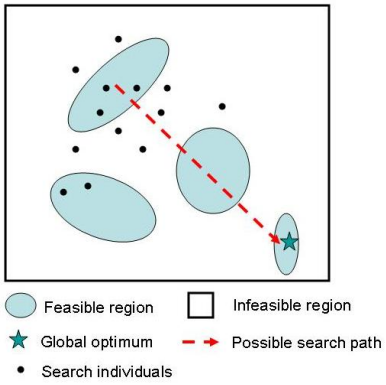
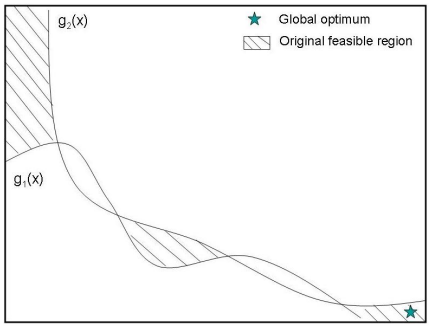


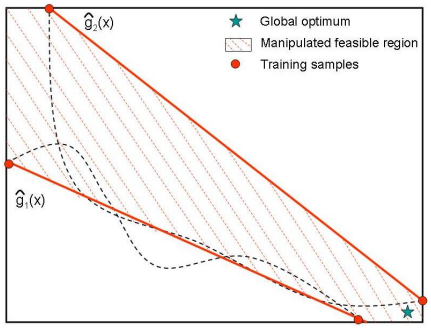
Fig. 1. Illustration of search space with separated feasible regions. The evolutionary path to the global optimum may be blocked by the infeasible regions.

The most important question to answer now is how to change the nonlinear constraints so that the topology of the resulting feasible regions will become simpler? Our idea in this work is to approximate the nonlinear constraint functions using a simpler model in the beginning, and then to increase the complexity of the model as the evolution proceeds. In this way, the topology of the feasible region can be simplified in the beginning and then it gradually goes back to its original complexity, as illustrated in Fig. 2 (a)-(c). According to this hypothesis, we can start from a linear approximation of the nonlinear constraints and then increase the complexity of the approximate constraints gradually by increasing the approximation accuracy. The next question is then, how to achieve an approximate model of the constraint functions with an increasing accuracy? Our idea here is to train a neural network model (or any other machine learning models) for each constraint function with an increasing number of training data. In the beginning, a very small number of training data are sampled from the constraint functions. Consequently, only a rough approximation (linear approximation) of the nonlinear constraint functions can be achieved. Attention should be paid to reduce overfitting of the learning models, which can be realized by using early stopping or regularization techniques [24]. Here, a slight underfitting will not be a big problem, since at the end of the evolutionary optimization, the algorithm will switch back to the original constraint functions to ensure that the obtained optimal solution is always feasible. As the evolution proceeds, additional data points will be sampled, resulting

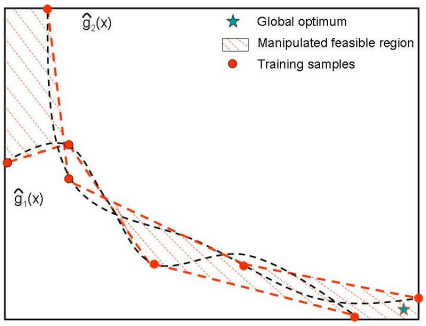
in an increasingly accurate approximation of the nonlinear constraints.



(a)



(b)



(c)

Fig. 2. Artificial change of the feasible regions by incremental approximation of the nonlinear constraint functions. (a) The design space has three separated feasible regions with the original nonlinear constraint functions. (b) With a linear approximation of the constrains, the approximated feasible region becomes connected. (c) The approximate constraints become more accurate and the approximated feasible region has two separated feasible regions.

B. Stochastic Ranking Evolution Strategy (SRES)

SRES is a variant of the canonical (μ,λ)-ES with individual stepsizes, i.e., each design variable has a separate stepsize. In the population, each individual is composed by real-valued vector $(\mathbf{x}, \boldsymbol{\sigma}) = \{(x_1, x_2, \cdots, x_n), (\sigma_1, \sigma_2, \cdots, \sigma_n)\}$, where n is the dimension of the search space of the given problem. During the initialization, $x_j, j = 1, 2, ..., n$, are generated according to a uniform distribution bounded by the interval $[\underline{x}_j, \overline{x}_j]$, where $\underline{x}_j$, and $\overline{x}_j$ are the lower and upper bounds of design variable x_j. The upper bound of the initial step size (σ_j) is defined as $(\overline{x}_j - \underline{x}_j)/\sqrt{n}$.

To generate λ offspring from μ parents ($\mu < \lambda$), global intermediate recombination and Gaussian mutations are applied to both the design variables and the strategy parameters. Two individuals are randomly chosen from the parent population and the arithmetic average is performed on both the design variables and strategy parameters:

$$(\hat{x}_{h,j}^{(g)}, \hat{\sigma}_{h,j}^{(g)}) = \left(\frac{(x_{i,j} + x_{k,j})}{2}, \frac{(\sigma_{i,j} + \sigma_{k,j})}{2} \right), \quad (5)$$

where $i = \{1, \cdots, \mu\}$, $h = \{1, \cdots, \lambda\}$, $j = \{1, \cdots, n\}$ and index k is randomly chosen from i. This process repeats until λ offspring are generated.

After recombination, the mutation operator is first applied to the step-size of each offspring individual:

$$\sigma_{h,j}^{(g+1)} = \hat{\sigma}_{h,j}^{(g)} \exp(\tau' N(0,1) + \tau N_j(0,1))), \quad (6)$$

where τ and τ' are two constants defined as $1/\sqrt{2\sqrt{n}}$ and $1/\sqrt{2n}$, respectively, and normal distribution is set to zero mean and one variance. Then, the design variables are mutated as follows:

$$x_{h,j}^{(g+1)} = x_{i,j}^{(g)} + \sigma_{h,j}^{(g+1)} N_j(0,1). \quad (7)$$

The main difference between the SRES and a canonical ES lies in the selection strategy. In the canonical (μ,λ)-ES the λ individuals are ranked deterministically according to their fitness value and the best μ individuals are selected as the parents of the next generation. In SRES, a stochastic bubble sorting strategy is adopted to balance between the objective and penalty. Note that in our work, we use the synthetic constrain functions for calculating the penalty:

$$\tilde{\phi}(\mathbf{x}) = \sum_{j=1}^{m} \max \{0, (\tilde{g_j}(\mathbf{x}))^{\alpha}\}, \quad (8)$$

where $\tilde{\phi}(\mathbf{x})$ is the penalty value of $\mathbf{x}$ calculated using the synthetic constrains:

$$\tilde{g_j}(\mathbf{x}) \in \{g_j(\mathbf{x}), \hat{g_j}(\mathbf{x})\}, \quad (9)$$

where $\hat{g_j}(\mathbf{x})$ is the approximate constrain function of $g_j(\mathbf{x})$. Refer to Section II-C for details.

Given the fitness and penalty pair $(f(\mathbf{x}_i), \tilde{\phi}(\mathbf{x}_i))$, where $\mathbf{x}_i$ denotes the solution of i-th offspring individual, $i = 1, 2, ..., \lambda$, they will be ranked according to the stochastic ranking algorithm as shown in Algorithm 1.

C. SRES with Synthesized Constraints

The basic idea for artificially expanding the feasible regions in the early stage of evolutionary search by means of incremental approximation of nonlinear constraint functions is implemented and incorporated within the the framework of SRES, which is termed as SRES-SC. The main components of the SRES-SC are illustrated in Fig. 3. Compared to the SRES, the major differences between SRES and SRES-SC are that a set of synthesized constraints will be created and used in stochastic ranking. The procedure for synthesizing

Algorithm 1 *The Stochastic ranking algorithm (adapted from [6]).*

for i=1 to λ, **do**
  for j=1 to $\lambda - 1$, **do**
    sample $u \in U(0,1)$, U is a uniform distribution
    if($\tilde{\phi}_j = \tilde{\phi}_{j+1} = 0$) **or** ($u < 0.45$), **then**
      if ($f_i > f_j$), **then**
        swap the order of individual i and j
      fi
    fi
  od
  if no swap done **break** **fi**
od

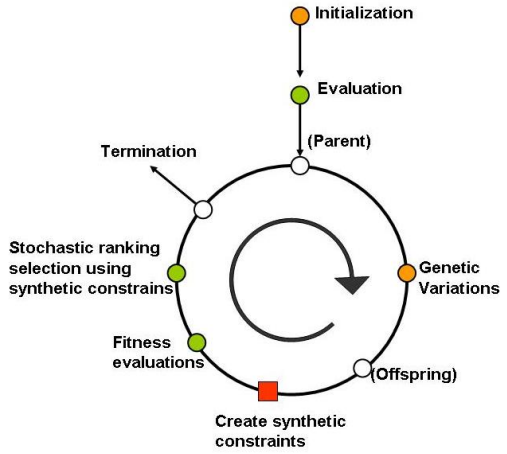


Fig. 3. Diagram of SRES-SC.

the constraints is provided in Fig. 4, which is composed of two main steps. First, an approximate model will be built for each constraint function. Second, the original constraints will compete with the approximate constrains in terms of the number of feasible solutions in the offspring population. In other words, for the j-th constraint, if the original constraint function produces more feasible solutions than the approximate constraint function, the original constraint function will be used for calculating the penalty in stochastic ranking. Otherwise, the approximate constraint function will be used.

Two questions remain to be answered. First, in which generations should the neural network models be updated? In this work, we specify that at generations $t = t + 10k^2$, where t is the generation number in which the neural network models are to be updated, $k = 0, 1, 2, ..., k_{\max}$. However, the condition $t \leq t_{\max}$ should be satisfied, where $t_{\max}$ is the allowed maximum number of generations, so that in the final generations, only the original constraint functions are used. The second question is how many samples should be used for training the neural networks? In this work, we again heuristically set $N = n_j k^2, k = 1, 2, ..., k_{\max}$, where N is the number of samples used for training the neural network

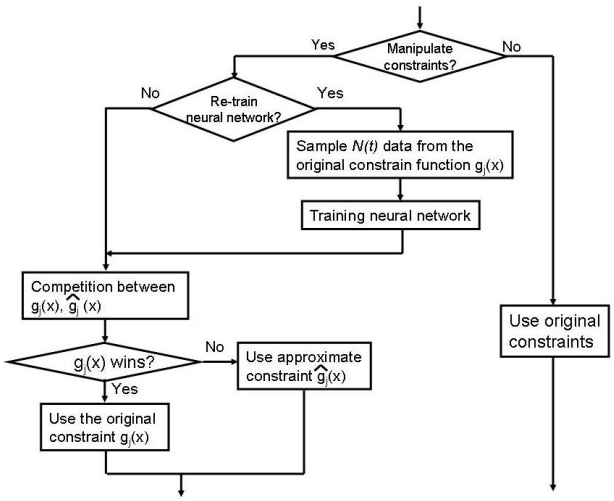


Fig. 4. Synthesizing the constraints via a competition between the original and approximate constraints.

in generation t, $n_j \leq n$ is the number of variables involved in $g_j(\mathbf{x})$. For instance, in the initial generation, if there are two variables in the constraint function, 2 pairs of training data are sampled. In generation 10, 8 samples are generated and so on. The location of the samples is determined by the Latin hypercube method.

In this work, we adopted a multi-layer perceptron (MLP) network with one hidden layer [24] (refer to Fig. 5) for approximating the nonlinear constraints. Both the hidden neurons and the output neurons use a tan-sigmoid transfer function. The number of input nodes equals the number of parameters in the constrain function plus one (a constant input as threshold), the number of hidden nodes is set to three times that of the input nodes, and the number of output node is one.

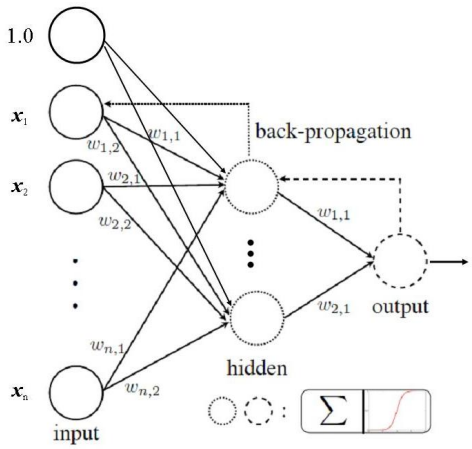


Fig. 5. Illustration of a multi-layer perceptron network.

TABLE I

PARAMETER SETTINGS USED BY THE COMPARED ALGORITHMS

	ATMES [12]	SRES [6]	SMES [25]	SRES-SC
Parent size	50	30	100	30
Offspring size	300	200	300	200
Generations	800	1750	800	1200

III. COMPARATIVE STUDIES

A. Results on Test Problems

In the first part of our empirical studies, we compare the performance of SRES-SC on 13 test problems [6] with the SRES [6], the adaptive tradeoff model evolutionary algorithm (ATMES) [12], and the simple multi-membered evolution strategy (SMES) [25]. The experimental setup of the four algorithms are listed in Table I. All equality constraints are transformed into inequality constraints using a tolerance value of 0.0001 for SRES and SRES-SC. The known optimal solution of the 13 test problems are (-15, -0.803619, -1, -30665.539, 5126.498, -6961.814, 24.306, -0.095825, 680.630, 7049.248, 0.75, -1, 0.053950).

The back-propagation learning algorithm [24] is used for training the MLP for 150 iterations every time when the MLP network models need to be updated, where the learning rate is set to 0.1. According to the rules set in Section II-C, the neural network based approximate constraint functions are updated in generation $t = \{0, 10, 50, 140, 300, 550, 910\}$ and from generations 911 to 1200, only the original constraint functions are used. Consequently, the number of training data to be sampled is set to $\{N_j, 4N_j, 9N_j, 16N_j, 25N_j, 36N_j, 49N_j\}$, where N_j is the number of variables involved in the j-th constraint function. Note, however, that if $N_j = 1$, the minimum number of samples should be 2.

The best, mean and worst solutions of the compared algorithms from 30 independent runs are listed in Table II, Table III, Table IV, respectively. From the three tables, we can see that the performance of proposed algorithm is better or comparable to SRES on all the 13 test problems with a much smaller number of fitness evaluations. Compared to SMES, our algorithm found a better best solution in four test functions ($g05, g07, g09, g10$). Compared to ATMES, our algorithm found the same best solution in 9 of the 13 test functions, and a better best solution in test function $g10$. Our algorithm also achieved a better mean and worst solution compared to ATMES in test function $g02$.

B. Results on the Engineering Optimization Problems

In this subsection, we investigate the performance of the proposed algorithm on four design optimization problems [33]. The first problem is to design a welded beam for minimum cost. The problem has four design variables and is subject to seven inequality constraints. The second problem is concerned with the optimization of a pressure vessel, which has four design variables and is subject to four inequality constraints. The third engineering problem optimizes the design of a speed reducer, where seven parameters are involved with six nonlinear inequality constraints. The last design optimization problem minimizes the weights of a tension spring subject to four nonlinear constraints.

TABLE II

THE BEST RESULTS ON 13 BENCHMARK FUNCTIONS OVER 30 INDEPENDENT RUNS

Fcn	ATMES [12]	SRES [6]	SMES [25]	SRES-SC
g01	**-15**	**-15**	**-15**	**-15**
g02	-0.803388	-0.803515	**-0.803601**	-0.8032295
g03	**-1**	**-1**	**-1**	**-1**
g04	**-30665.539**	**-30665.539**	**-30665.539**	**-30665.539**
g05	**5126.498**	**5126.498**	5126.599	5126.512
g06	**-6961.814**	**-6961.814**	**-6961.814**	-6957.633
g07	**24.306**	24.307	24.327	**24.306**
g08	**-0.095825**	**-0.095825**	**-0.095825**	**-0.095825**
g09	**680.630**	**680.630**	680.632	**680.630**
g10	7052.253	7054.316	7051.903	**7050.189**
g11	**0.75**	**0.75**	**0.75**	**0.75**
g12	**-1**	**-1**	**-1**	**-1**
g13	**0.053950**	0.053957	0.053986	0.053988

TABLE III

THE MEAN RESULTS ON 13 BENCHMARK FUNCTIONS OVER 30 INDEPENDENT RUNS

Fcn	ATMES [12]	SRES [6]	SMES [25]	SRES-SC
g01	**-15**	**-15**	**-15**	**-15**
g02	-0.790148	-0.781975	-0.785238	**-0.792114**
g03	**-1**	**-1**	**-1**	**-1**
g04	**-30665.539**	**-30665.539**	**-30665.539**	**-30665.539**
g05	**5127.648**	5128.881	5174.492	5129.823
g06	**-6961.814**	-6875.94	-6961.284	-6737.877
g07	**24.316**	24.374	24.475	24.323
g08	**-0.095825**	**-0.095825**	**-0.095825**	**-0.095825**
g09	**680.639**	680.665	680.643	680.646
g10	**7050.437**	7559.192	7253.047	7220.059
g11	**0.75**	**0.75**	**0.75**	**0.75**
g12	**-1**	**-1**	**-1**	**-1**
g13	**0.053950**	0.067543	0.166385	0.063862

We compared our algorithm with nine other algorithms that have been reported in the literature. The best, mean and worst solutions from 30 independent runs on the four design problems are provided in Table V, Table VI, Table VII and Table VIII, respectively. From these results, we can see that our algorithm obtained a better solution than the best optimal solution reported so far on three of the four design optimization problems. For the optimization of welded beam, our algorithm also achieved the second best solution and this solution has been found in all the 30 runs.

IV. CONCLUSIONS

This paper has proposed a new algorithm for solving nonlinear constrained problems from a perspective different from the existing approaches. Our basic idea is to manipulate the feasible regions by incrementally approximating the nonlinear constraints functions. In this way, the feasible region of the design space can be artificially enlarged in the beginning of the evolutionary search to facilitate the evolutionary algorithm to find its way to reach the feasible region where the global feasible optimum is located. Our algorithm has been compared with the state-of-the-art evolutionary methods

TABLE IV

THE WORST RESULTS ON 13 BENCHMARK FUNCTIONS OVER 30 INDEPENDENT RUNS

Fcn	ATMES [12]	SRES [6]	SMES [25]	SRES-SC
g01	**-15**	**-15**	**-15**	**-15**
g02	-0.756986	-0.726288	-0.751322	**-0.759766**
g03	**-1**	**-1**	**-1**	**-1**
g04	**-30665.539**	**-30665.539**	**-30665.539**	**-30665.539**
g05	**5135.256**	5142.472	5304.167	5149.931
g06	**-6961.814**	-6350.262	-6952.482	-6024.792
g07	**24.359**	24.642	24.483	24.395
g08	**-0.095825**	**-0.095825**	**-0.095825**	**-0.095825**
g09	**680.673**	680.763	680.719	680.725
g10	**7560.224**	8835.655	7638.366	7769.887
g11	**0.75**	**0.75**	**0.75**	**0.75**
g12	**-1**	**-1**	**-1**	**-1**
g13	**0.053999**	0.216915	0.468294	0.157467

TABLE V

THE SIMULATION RESULTS ON WELDED BEAM DESIGN OPTIMIZATION PROBLEM OVER 30 INDEPENDENT RUNS

Methods	Best	Mean	Worst
GA1 [23]	1.74831	1.77197	1.78584
GA2 [26]	1.72823	1.79265	1.99341
CAEP [27]	1.72485	1.97181	3.17971
Mezura [31]	1.72485	1.77760	NA
CPSO [28]	1.72802	1.74883	1.78214
HPSO [29]	1.72485	1.74904	1.81430
COPSO [32]	1.72485	**1.72485**	NA
SiC-PSO [33]	1.72485	2.05740	NA
NMPSO [30]	**1.72472**	1.72637	1.73339
SRES-SC	1.72485	**1.72485**	**1.72485**

TABLE VI

THE SIMULATION RESULTS ON PRESSURE VESSEL DESIGN OPTIMIZATION PROBLEM OVER 30 INDEPENDENT RUNS

Methods	Best	Mean	Worst
GA1 [23]	6288.7445	6293.8432	6308.1497
GA2 [26]	6059.9463	6177.2533	6469.3220
CAEP [27]	NA	NA	NA
Mezura [31]	6059.7143	6379.9380	NA
CPSO [28]	6061.0777	6147.1332	6368.8041
HPSO [29]	6059.7143	6099.9323	6288.6770
COPSO [32]	6059.1743	6071.0133	NA
SiC-PSO [33]	6059.7143	6092.0498	NA
NMPSO [30]	5930.3137	5946.7901	**5960.0557**
SRES-SC	**5885.3330**	**5923.5820**	6255.2580

TABLE VII

THE SIMULATION RESULTS ON SPREED REDUCER DESIGN OPTIMIZATION PROBLEM OVER 30 INDEPENDENT RUNS

Methods	Best	Mean	Worst
GA1 [23]	NA	NA	NA
GA2 [26]	NA	NA	NA
CAEP [27]	NA	NA	NA
Mezura [31]	2996.3481	2996.3480	NA
CPSO [28]	NA	NA	NA
HPSO [29]	NA	NA	NA
COPSO [32]	2996.3724	2996.4085	NN
SiC-PSO [33]	2996.3482	2996.3482	NA
NMPSO [30]	NA	NA	NA
SRES-SC	**2996.2314**	**2996.2314**	**2996.2314**

TABLE VIII

THE SIMULATION RESULTS ON TENSION/COMPRESSION SPRING DESIGN OPTIMIZATION PROBLEM OVER 30 INDEPENDENT RUNS

Methods	Best	Mean	Worst
GA1 [23]	0.012705	0.012769	0.012822
GA2 [26]	0.012681	0.012742	1.012973
CAEP [27]	0.012721	0.013568	0.015116
Mezura [31]	0.012689	0.013100	NA
CPSO [28]	0.012675	0.012730	0.012924
HPSO [29]	0.012665	0.012707	0.012719
COPSO [32]	0.012665	0.012600	NA
SiC-PSO [33]	0.012665	0.013100	NA
NMPSO [30]	0.012630	0.012631	0.012633
SRES-SC	**0.009872**	**0.009876**	**0.009909**

for solving constrained optimization problems on 13 test problems and four engineering design optimization problems. The proposed algorithm has shown comparable performance (but not as good as that of the ATMES) on the 13 test problems and achieved better results on three of the four engineering design problems, particularly on the pressure vessel and spring design optimization problem.

Several issues remain open regarding the applicability of the proposed method. First, the assumption that incremental approximation of the complex constraint functions will change the feasible regions in such a way that the feasible region will be enlarged in the beginning of the search and gradually converge to the original ones is to be verified on more test problems. This may depend on both the nature of the problems and the way how data for training the approximate models are sampled. Obviously, more sophisticated sampling methods than the Latin hypercube method can be explored. Second, many real-world optimization problems do not have explicit constraint functions, or the evaluation of the constraint functions is very time-consuming. In both cases, the number of samples for training the meta-models may be very limited, and therefore the strategy for sampling becomes more important.

ACKNOWLEDGMENT

YJ would like to thank Bernhard Sendhoff and Andreas Richter for their support. The work was done when the YJ was with the Honda Research Institute Europe.

REFERENCES

[1] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems,",*Evolutionary Computation*, vol. 4, pp. 1–32, Mar. 1996.

[2] C.A.C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, vol. 191, pp. 1245–1287, 2002.

[3] J.A. Joines and C.R. Houck, "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems,"*Proc. IEEE Conference on Evolutionary Computation*. pp.579-584, 1994.

[4] S.B. Hamida and M. Schoenauer, "An adaptive algorithm for constrained optimization problems," *Proc. Parallel Problem Solving form Nature*, Paris, Sept. 2000, pp. 529-538.

[5] M. Farmabi and J.A. Wright, "Self-adaptive fitness formulation for constrained optimization," *IEEE Trans. Evolutionary Computation*, vol. 7, pp. 445-455, Oct. 2003.

[6] T.P. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization", *IEEE Trans. Evolutionary Computation*, vol. 4, pp. 284-294, Sep. 2000.
[7] E. Mezura-Montes and C.A.C. Coello, "Constrained optimization via multi-objective evolutionary algorithms," In: [13], pp.1-24, 2009.
[8] P.D. Surrey, N.J. Radcliffe and I.D. Boyd, "A multi-objective approach to constrained optimization of gas supply networks,"*Proc. AISB Workshop on Evolutionary Computation*, pp.166-180, 1995.
[9] Z. Cai and Y. Wang, "A multi-objective optimization-based evolutionary algorithms for constrained optimization," *IEEE Transactions on Evolutionary Computation*, vol. 10, pp. 658–675, 2006.
[10] T.P. Runarsson and X. Yao, "Search biases in constrained evolutionary optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 35, 233–243, 2005.
[11] S. Venkatraman and G.G. Yen, "A generic framework for constrained optimization using genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol.9, pp.424-435, 2005.
[12] Y. Wang, Z. Cai, Y. Zhou and W. Zeng, "An adaptive trade-off model for constrained evolutionary optimization," *IEEE Transactions on Evolutionary Computation*, vol. 12, pp. 80-92, 2008.
[13] E. Mezura-Montes (Ed.), Constraint-Handling in Evolutionary Optimization. Springer, 2009.
[14] I.G. Tsoulos, "Solving constrained optimization problems using a novel genetic algorithms," *Applied Mathematica and Computation*, vol.208, pp.273-283, 2009.
[15] G.E. Liepins, M.D. and Vose, "Representational issues in genetic optimization," *Journal of Experimental Theoretical Artificial Intelligence*, vol. 2, pp.101–115, 1990.
[16] S. Bagchi, S. Uchun, Y. Miyabe and K. Kawamura, "Exploring problem-specific recombination operators for job shop scheduling," *Proc. 4th. Int. Conf. on Genetic Algorithms*, pp.10-17, 1991.
[17] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol.9, pp. 3–12, 2005.
[18] Y. Jin and B. Sendhoff. "A systems approach to evolutionary multiobjective structural optimization and beyond," *IEEE Computational Intelligence Magazine*, vol.4, pp.62–76, 2009.
[19] T.P. Runarsson, "Constrained evolutionary optimization by approximate ranking and surrogate models,", *Proc. Parallel Problem Solving from Nature VII*, LNCS 3242, pp.401–410, 2004.
[20] E. Mezura-Montes and C.A.C. Coello, "Saving evaluations in differential evolution for constrained optimization," *Proc. Sixth Mexican Conference on Computer Science*, pp.274–281, 2005.
[21] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments - A survey," *IEEE Transactions on Evolutionary Computation*, vol.9, pp.303–317, 2005.
[22] T.T. Nguyen and X. Yao, "Benchmarking and solving dynamic constrained problems," *Proc. IEEE Congress on Evolutionary Computation*, pp.690–697, 2009.
[23] Carlos A. Coello Coello1, "Use of a self-adaptive penalty approach for engineering optimization problems," *Computers in Industry*, vol. 41, pp. 113–127, Mar. 2000.
[24] R.D. Reed and R.J. Marks II, "Neural Smitning - Supervised Learning in Feedforward Artificial Neural Networks," The MIT Press, 1999.
[25] Efren Mezura-Montes and Carlos A. Coello Coello, "A simple multimembered evolution strategy to solve constrained optimization problems," *IEEE Trans. Evolutionary Computation*, vol. 9, pp. 1–17, Feb. 2005.
[26] Carlos A. Coello Coello1 and Efrn Mezura Montes, "Constraint-handling in genetic algorithms through the use of dominance-based tournament selection," *Advanced Engineering Informatics*, vol. 16, pp. 193–203, Jul. 2002.
[27] Carlos A. Coello Coello1 and Ricardo Landa Becerra, "Efficient evolutionary optimization through the use of a cultural algorithm," *Engineering Optimization*, vol. 36, pp. 219–236, Apr. 2004.
[28] Qie He and Ling Wang, "An effective co-evolutionary particle swarm optimization for constrained engineering design problems," *Engineering Applications of Artificial Intelligence*, vol. 20, pp. 89–99, Feb. 2006.
[29] Qie He and Ling Wang, "A hybrid particle swarm optimization with a feasibility based rule for constrained optimization," *Applied Mathematics and Computation*, vol. 186, pp. 1407–1422, Mar. 2007.
[30] Erwie Zahara and Yi-Tung Kao, "Hybrid Nelder-Mead simplex search and particle swarm optimization for constrained engineering design problems," *Expert Systems with Applications*, vol. 36, pp. 3880-3886, Mar. 2009.
[31] Efrn Mezura-Montes and Carlos A. Coello Coello, "Useful Infeasible Solutions in Engineering Optimization with Evolutionary Algorithms," *Proc. 4th Mexican International Conference on Artificial Intelligence*, Mexico, Nov. 2005, pp. 652–662.
[32] Arturo Hernandez Aguirre, Angel E. Munoz Zavala, E. Villa Diharce and S. Botello Rionda, "Constrained optimization with an improved particle swarm optimization algorithm," *International Journal of Intelligent Computing and Cybernetics*, vol. 1, pp. 425–453, Mar. 2008.
[33] Leticia C. Cagnina, Susana C. Esquivel and Carlos A. Coello Coello, "Solving Engineering Optimization Problems with the Simple Constrained Particle Swarm Optimizer," *Informatica*, vol. 32, pp. 319–326, Oct. 2008.

APPENDIX: THE CONSTRAINED DESIGN OPTIMIZATION PROBLEMS

Details of the four engineering design optimization problems are listed below, which are taken from [7] and [33].

- **Design Problem 1**: Welded beam design optimization

Minimize:

$$f = \quad 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2);$$

Subject to:

$$\begin{aligned}
g_1 &= \quad \tau - 13600 \leq 0;\\
g_2 &= \quad \sigma - 30000 \leq 0;\\
g_3 &= \quad x_1 - x_4 \leq 0;\\
g_4 &= \quad 0.10471x_1^2 + 0.04811x_3x_4(14 + x_2) - 5.0 \leq 0;\\
g_5 &= \quad 0.125 - x_1 \leq 0;\\
g_6 &= \quad \delta - 0.25 \leq 0;\\
g_7 &= \quad 6000 - pc \leq 0;
\end{aligned}$$

where

$$\begin{aligned}
pc &= \frac{4.013(30\times10^6)\sqrt{\frac{x_3^2x_4^6}{36}}}{196}\left(1-\frac{x_3\sqrt{\frac{30\times10^6}{4(12\times10^6)}}}{28}\right);\\
\delta &= \frac{65856000}{(30\times10^6)x_3^3x_4};\\
\sigma &= \frac{504000}{x_3^2x_4};\\
J &= 2\left\{x_1x_2\sqrt{2}\left[\frac{x_2^2}{12}+\left(\frac{x_1+x_3}{2}\right)^2\right]\right\};\\
R &= \sqrt{\frac{x_2^2}{4}+\left(\frac{x_1+x_3}{2}\right)^2};\\
M &= 6000\left(14+\frac{x_2}{2}\right);\\
\tau &= \sqrt{t_1^2+(2t_1t_2)\frac{x_2}{2R}+t_2^2};\\
t_1 &= \frac{6000}{\sqrt{2}x_1x_2};\\
t_2 &= \frac{MR}{J};
\end{aligned}$$

with $0.1 \leq x_1, x_4 \leq 2.0$, and $0.1 \leq x_2, x_3 \leq 10$.

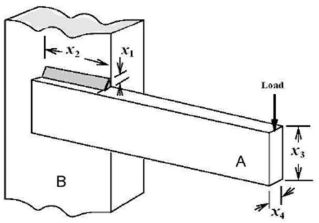


Fig. 6. Welded beam.

- **Design Problem 2**: Pressure vessel design optimization

Minimize:

$$f = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3;$$

Subject to:

$$g_1 = -x_1 + 0.0193x_3 \leq 0;$$
$$g_2 = -x_2 + 0.00954x_3 \leq 0;$$
$$g_3 = -\pi x_3^2x_4 - \tfrac{4}{3}\pi x_3^3 + 1296000 \leq 0;$$
$$g_4 = x_4 - 240 \leq 0;$$

with $1 \leq x_1, x_2 \leq 99$ and $10 \leq x_3, x_4 \leq 200$.

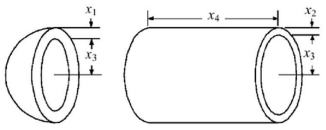


Fig. 7. Pressure vessel.

- **Design Problem 3**: Speed reducer design optimization

Minimize:

$$f = 0.7854x_1x_2^2(3.333x_3^2 + 14.9334x_3 - 43.0934) - 1.508x_1(x_6^2 + x_7^2) + 7.4777(x_6^3 + x_7^3) + 0.7854(x_4x_6^2 + x_5x_7^2);$$

Subject to:

$$g_1 = \frac{27}{x_1x_2^2x_3} - 1 \leq 0;$$
$$g_2 = \frac{397.5}{x_1x_2^2x_3^2} - 1 \leq 0;$$
$$g_3 = \frac{1.93x_4^3}{x_2x_3x_6^4} - 1 \leq 0;$$
$$g_4 = \frac{1.93x_5^3}{x_2x_3x_7^4} - 1 \leq 0;$$
$$g_5 = \frac{1.0}{110x_6^3}\sqrt{\left(\frac{745.0x_4}{x_2x_3}\right)^2 + 16.9 \times 10^6} - 1 \leq 0;$$
$$g_6 = \frac{1.0}{85x_7^3}\sqrt{\left(\frac{745.0x_5}{x_2x_3}\right)^2 + 157.5 \times 10^6} - 1 \leq 0;$$
$$g_7 = \frac{x_2x_3}{40} - 1 \leq 0;$$
$$g_8 = \frac{5x_2}{x_1} - 1 \leq 0;$$
$$g_9 = \frac{x_1}{12x_2} - 1 \leq 0;$$
$$g_{10} = \frac{1.5x_6+1.9}{x_4} - 1 \leq 0;$$
$$g_{11} = \frac{1.1x_7+1.9}{x_5} - 1 \leq 0;$$

with $2.6 \leq x_1 \leq 3.6$, $0.7 \leq x_2 \leq 0.8$, $17 \leq x_3 \leq 28$, $7.3 \leq x_4 \leq 8.3$, $7.8 \leq x_5 \leq 8.3$, $2.9 \leq x_6 \leq 3.9$, and $5.0 \leq x_7 \leq 5.5$.

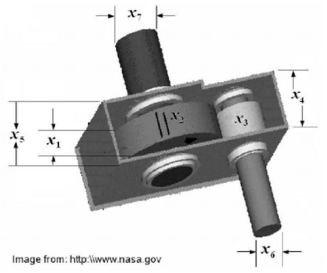


Fig. 8. Speed reducer.

- **Design Problem 4**: Spring design optimization

Minimize:

$$f = (x_3 + 2)x_1^2x_2; \tag{10}$$

Subject to:

$$g_1 = 1 - \frac{x_2^3x_3}{71785x_1^4} \leq 0;$$
$$g_2 = \frac{4x_2^2 - x_1x_2}{12566(x_1^3x_2 - x_1^4)} + \frac{1}{5108x_1^2} - 1 \leq 0;$$
$$g_3 = 1 - \frac{140.45x_1}{x_2^2x_3} \leq 0;$$
$$g_4 = \frac{x_1+x_1}{1.5} - 1 \leq 0;$$

with $0.05 \leq x_1 \leq 2.0$, $0.25 \leq x_2 \leq 1.3$, and $2.0 \leq x_3 \leq 15.0$.

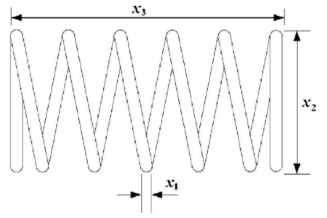


Fig. 9. Tension-compression spring.