AIAA JOURNAL
Vol. 41, No. 8, August 2003

Particle Swarm Optimization

Gerhard Venter*
Vanderplaats Research and Development, Inc., Colorado Springs, Colorado 80906
and
Jaroslaw Sobieszczanski-Sobieski†
NASA Langley Research Center, Hampton, Virginia 23681-2199

How the search algorithm known as particle swarm optimization performs is demonstrated. Particle swarm optimization is applied to structural design problems, but the method has a much wider range of possible applications. Improvements are contributed to the particle swarm optimization algorithm and conclusions and recommendations as to the utility of the algorithm are made. Results of numerical experiments for both continuous and discrete applications are presented. The results indicate that the particle swarm optimization algorithm does locate the constrained minimum design in both continuous and discrete applications and problems with multiple local minima, with very good precision. Additionally, particle swarm optimization has the potential of efficient computation with very large numbers of concurrently operating processors.

Introduction

MOST general-purpose optimization software used in industrial applications make use of gradient-based algorithms, mainly due to their computational efficiency. However, in recent years nongradient-based, probabilistic search algorithms have attracted much attention from the research community. These algorithms generally mimic some natural phenomena, for example, genetic algorithms and simulated annealing. Genetic algorithms model the evolution of a species, based on Darwin's principle of survival of the fittest (see Ref. 1), whereas simulated annealing is based on statistical mechanics and models the equilibrium of large numbers of atoms during an annealing process.[2]

Although these probabilistic search algorithms generally require many more function evaluations to find an optimum solution, as compared to gradient-based algorithms, they do provide several advantages. These algorithms are generally easy to program, can efficiently make use of large numbers of processors, do not require continuity in the problem definition, and are generally better suited for finding a global, or near global, solution. In particular, these algorithms are ideally suited for solving discrete and/or combinatorial-type optimization problems.

In this paper, a fairly recent type of probabilistic search algorithm, called particle swarm optimization (PSO), is investigated. The PSO algorithm is based on a simplified social model that is closely tied to swarming theory. The algorithm was first introduced by Kennedy and Eberhart[3] and Eberhart and Kennedy.[4] A physical analogy might be a swarm of bees searching for a food source. In this analogy, each bee (referred to as a particle here) makes use of its own memory as well as knowledge gained by the swarm as a whole to find the best available food source.

Since it was originally introduced, the PSO algorithm has been studied by a number of different authors.[5–8] These authors concentrated mostly on multimodal mathematical problems that are important in the initial research of any optimization algorithm but are of little practical interest. Few applications of the algorithm to structural and multidisciplinary optimization are known. Two examples are by Fourie and Groenwold, who considered an application to shape and size optimization,[9] and an application to topology optimization.[10]

The present paper focuses on enhancements to the basic PSO algorithm. These include the introduction of a convergence criterion and dealing with constrained and discrete problems. The basic algorithm is tested on a mathematical example problem with a large number of local minima, whereas the enhanced version is applied to the design of a 10 design variable cantilevered beam. Both continuous and integer/discrete versions of the cantilevered beam problem are studied.

Basic PSO Algorithm

PSO makes use of a velocity vector to update the current position of each particle in the swarm. The position of each particle is updated based on the social behavior that a population of individuals, the swarm in the case of PSO, adapts to its environment by returning to promising regions that were previously discovered.[5] The process is stochastic in nature and makes use of the memory of each particle, as well as the knowledge gained by the swarm as a whole. The outline of a basic PSO algorithm is as follows:

1) Start with an initial set of particles, typically randomly distributed throughout the design space.

2) Calculate a velocity vector for each particle in the swarm.

3) Update the position of each particle, using its previous position and the updated velocity vector.

4) Go to step 2 and repeat until convergence.

The scheme for updating the position of each particle is

$$\boldsymbol{x}^i_{k+1} = \boldsymbol{x}^i_k + \boldsymbol{v}^i_{k+1}\,\Delta t \tag{1}$$

where $\boldsymbol{x}^i_{k+1}$ is the position of particle i at iteration $k+1$ and $\boldsymbol{v}^i_{k+1}$ is the corresponding velocity vector. A unit time step Δt is used throughout the present work.

The scheme for updating the velocity vector of each particle depends on the particular PSO algorithm under consideration. A commonly used scheme, introduced by Shi and Eberhart,[6] is

$$\boldsymbol{v}^i_{k+1} = w\boldsymbol{v}^i_k + c_1 r_1 \frac{\left(\boldsymbol{p}^i - \boldsymbol{x}^i_k\right)}{\Delta t} + c_2 r_2 \frac{\left(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k\right)}{\Delta t} \tag{2}$$

where r_1 and r_2 are independent random numbers between 0 and 1, $\boldsymbol{p}^i$ is the best position found by particle i so far, and $\boldsymbol{p}^g_k$ is the best position in the swarm at time k. Again, a unit time step Δt is used throughout the present work. There are three problem-dependent

Received 28 August 2002; revision received 5 March 2003; accepted for publication 14 March 2003.

*VisualDOC Project Manager, 1767 South 8th Street, Suite 100; gventer@vrand.com. Member AIAA.

†Senior Research Scientist, Analytical and Computational Methods Branch, Mail Stop 240, Structures and Materials Competency; j.sobieski@larc.nasa.gov. Fellow AIAA.

parameters, the inertia of the particle w and two "trust" parameters c_1 and c_2. The inertia controls the exploration properties of the algorithm, with larger values facilitating a more global behavior and smaller values facilitating a more local behavior. The trust parameters indicate how much confidence the current particle has in itself, c_1, and how much confidence it has in the swarm, c_2.

Fourie and Groenwold[9] proposed a slight modification to Eq. (2) for their structural design applications. They proposed using the best position in the swarm to date, $\boldsymbol{p}^g$, instead of the best position in the swarm at iteration k, $\boldsymbol{p}^g_k$. Both approaches were investigated, and it was found that Eq. (2) works slightly better for the applications considered here. As a result, Eq. (2) is used throughout the present work.

Initial Swarm

The initial swarm is generally created such that the particles are randomly distributed throughout the design space, each with a random initial velocity vector. In the present work, the following equations are used to obtain the random initial position and velocity vectors:

$$\boldsymbol{x}^i_0 = \boldsymbol{x}_{\min} + r_3(\boldsymbol{x}_{\max} - \boldsymbol{x}_{\min}) \tag{3}$$

$$\boldsymbol{v}^i_0 = \frac{\boldsymbol{x}_{\min} + r_4(\boldsymbol{x}_{\max} - \boldsymbol{x}_{\min})}{\Delta t} \tag{4}$$

In Eqs. (3) and (4), $\boldsymbol{x}^i_0$ is to the initial position vector, $\boldsymbol{v}^i_0$ is the initial velocity vector of particle i, r_3 and r_4 are independent random numbers between 0 and 1, $\boldsymbol{x}_{\min}$ is the vector of lower bounds, and $\boldsymbol{x}_{\max}$ is the vector of upper bounds for the design variables.

The influence of the initial swarm distribution on the effectiveness of the PSO algorithm was studied by considering the initial particle distribution. Instead of using a random distribution, a space-filling design of experiments (DOE) was used to distribute the initial swarm in the design space. However, these numerical experiments indicated that the initial distribution of the particles (random as compared to using the space-filling DOE) is not important to the overall performance of the PSO algorithm. The reason is that the swarm changes dynamically throughout the optimization process until an optimum solution is reached. As a result, the precise distribution of the initial swarm is not important, as long as it is fairly well distributed throughout the design space. In the present work, all initial swarms were randomly distributed.

Problem Parameters

The basic PSO algorithm has three problem-dependent parameters, w, c_1, and c_2. The literature[3] proposes using $c_1 = c_2 = 2$, so that the mean of the stochastic multipliers of Eq. (2) is equal to 1. Additionally, Shi and Eberhart[7] suggest using $0.8 < w < 1.4$, starting with larger w values (a more global search behavior) that is dynamically reduced (a more local search behavior) during the optimization. Dynamically adjusting the w value has several advantages. First, it results in faster convergence to the optimum solution, and second, it makes the w parameter problem independent.

In the present work, the trust parameters were chosen so that each particle puts slightly more trust in the swarm (larger c_2 value) and slightly less trust in itself (smaller c_1 value). Based on numerical simulations using the example problems presented here, values of $c_1 = 1.5$ and $c_2 = 2.5$ were chosen for the two trust parameters.

The inertia weight parameter w was adjusted dynamically during the optimization, as suggested by Shi and Eberhart,[7] with an example implementation by Fourie and Groenwold.[9] Shi and Eberhart[7] proposed linearly decreasing w during the first part of the optimization, whereas Fourie and Groenwold[9] decreased the w value with a fraction if no improvement has been made for a predefined number of consecutive design iterations.

Here a different implementation is proposed, based on the coefficient of variation (COV) of the objective function values. The COV of the objective function values for a subset of best particles was monitored. If the COV fell below a specified threshold value, it was assumed that the algorithm was converging toward an optimum solution, and the w value was updated. A general equation to calculate the COV for a set of points is

$$\text{COV} = \frac{\text{StdDev}}{\text{mean}} \tag{5}$$

where StdDev is the standard deviation and mean is the mean value for the set of points. In the present work, a subset of the best 20% of particles from the swarm was monitored, and a COV threshold of 1.0 was used. A starting value of $w = 1.4$ was used to accommodate a more global search initially and was dynamically reduced to a minimum value of $w = 0.35$. The result is to terminate the PSO algorithm with a more local search. The w value was adjusted using

$$w_{\text{new}} = w_{\text{old}} f_w \tag{6}$$

where w_{new} is the newly adjusted w value, w_{old} is the previous w value, and f_w is a constant between 0 and 1. Smaller f_w values would result in a more dramatic reduction in w that would in turn result in a more local search. In the present work $f_w = 0.975$ was used throughout, resulting in a PSO algorithm with a fairly global search characteristic.

Additional Randomness

To avoid premature convergence of the algorithm, the literature mentions the possible use of a craziness operator[3] that adds randomness to the swarm. The craziness operator acts similarly to the mutation operator in genetic algorithms. However, there does not seem to be consensus in the literature whether the craziness operator should be applied. After introducing the craziness operator, Kennedy and Eberhart[3] concluded that this operator may not be necessary, whereas Fourie and Groenwold[9] reintroduced the craziness operator for their structural design problems. Thus, it was decided to implement a craziness operator here and to test its effectiveness for the structural design problem considered.

The originally proposed craziness operator identifies a small portion of randomly selected particles at each iteration for which the velocity vector is randomly changed. In the present work, the authors decided to introduce a new craziness operator. The proposed modification was an attempt to provide a craziness operator with better exploration of the design space. The craziness operator used here also identifies a small number of particles at each design iteration, but instead of changing the velocity vector, both the position and the velocity vector were changed. The position of the particles were changed randomly, whereas the velocity vector of each modified particle was reset to only the second component of Eq. (2) as

$$\boldsymbol{v}^i_{k+1} = c_1 r_1 \frac{\left(\boldsymbol{p}^i - \boldsymbol{x}^i_k\right)}{\Delta t} \tag{7}$$

In the present implementation, the particles to modify were identified using the COV of the objective function values of all particles, at the end of each design iteration. If the COV fell below a predefined threshold value, it was assumed that the swarm became too uniform. In this case, particles located far from the center of the swarm were identified, using the standard deviation of the position coordinates of the particles. Particles that were located more than two standard deviations from the center of the swarm were subjected to the craziness operator. In the present work, a COV threshold value of 0.1 was used.

Enhancements to the Basic Algorithm

The basic PSO algorithm summarized in Eqs. (1) and (2) were used in the literature to demonstrate the PSO algorithm on a number of test problems. However, the goal in the present work was to develop an implementation of the basic algorithm that would be more general in nature and applicable to a wide range of design problems. To achieve this goal, a number of enhancements to the basic algorithm were investigated. These enhancements are discussed in more detail in this section.

Convergence Criterion

A robust convergence criterion is important for any general-purpose optimizer to avoid additional function evaluations after an optimum solution is found. Ideally, the convergence criterion used should not have any problem-specific parameters. The convergence criterion considered here is very basic. The maximum change in the objective function was monitored for a specified number of consecutive design iterations. If the maximum change in the objective function was less than a predefined allowable change, convergence was assumed.

Constrained Optimization

The basic PSO algorithm is defined for unconstrained problems only. Because most engineering problems are constrained in one way or the other, it is important to add the capability of dealing with constrained optimization problems. It was decided to deal with constraints by making use of a quadratic exterior penalty function. This technique is often used to deal with constrained problems in genetic algorithms. In the current implementation, the objective function was penalized as follows when one or more of the constraints were violated:

$$\tilde{f}(\boldsymbol{x}) = f(\boldsymbol{x}) + \alpha + \beta \sum_{i=1}^{m} \max[0, g_i(\boldsymbol{x})]^2 \tag{8}$$

In Eq. (8), $f(\boldsymbol{x})$ is the original objective function, α and β are constant, positive penalty parameters that are applied to all infeasible design points, $g_i(\boldsymbol{x})$ is the set of all constraints (with violated constraints having values larger than zero), and $\tilde{f}(\boldsymbol{x})$ is the new, penalized, objective function. In the present work $\alpha = 1000$ and $\beta = 10^8$ were used as penalty parameters.

Particles with Violated Constraints

When dealing with constrained optimization problems, special attention needs to be paid to particles with violated constraints. This issue was not addressed in the literature, and thus a new enhancement to the basic PSO algorithm is proposed here.

It is preferable to restrict the velocity vector of a violated particle to a usable, feasible direction, for example, as by Vanderplaats,[11] a direction that would reduce the objective function while pointing back to the feasible region of the design space. Unfortunately, this would require gradient information for each violated particle. Currently, one of the attractive features of the PSO algorithm is that no gradient information is required, and thus, the concept of calculating a usable, feasible direction is not viable.

Instead, a simple modification to Eq. (2) for particles with one or more violated constraints is proposed. The modification can be explained by considering particle i, which is assumed to have one or more violated constraints at iteration k. When the velocity vector of particle i at iteration k is reset to zero, the velocity vector at iteration $k+1$ is obtained as

$$\boldsymbol{v}_{k+1}^{i} = c_1 r_1 \frac{\left(\boldsymbol{p}^{i} - \boldsymbol{x}_k^{i}\right)}{\Delta t} + c_2 r_2 \frac{\left(\boldsymbol{p}_k^{g} - \boldsymbol{x}_k^{i}\right)}{\Delta t} \tag{9}$$

The velocity vector of particle i at iteration $k+1$ is, thus, only influenced by the best point found so far for the particle itself and the current best point in the swarm. In most cases, this new velocity vector will point back to a feasible region of the design space. The result is moving the violated particle toward the feasible design space in the next design iteration.

Discrete/Integer Design Variables

Unlike a genetic algorithm that is inherently a discrete algorithm, the PSO algorithm is inherently a continuous algorithm. The basic PSO algorithm is able to solve continuous design problems; however, the authors also wanted to explore the usefulness of the algorithm in discrete optimization applications. Two different modifications to the basic PSO algorithm that allow the solution of problems with discrete variables were considered. The first approach is straightforward. The position of each particle was modified to represent a discrete point, by rounding each position coordinate to its closest discrete value after applying Eq. (1).

The second approach is more elaborate. The position of each particle was modified to represent a discrete point, by considering a set of candidate discrete points about the new continuous point, obtained after applying Eq. (2). The candidate discrete points were obtained by rounding each continuous position coordinate to its closest upper and lower discrete values. For a problem with $nD\text{var}$ discrete design variables, this process would result in $2^{nD\text{var}}$ candidate discrete points. For example, a two-dimensional, discrete problem would produce four candidate discrete points, located at the vertices of a two-dimensional rectangle. The new discrete position for the particle is selected from the candidate set of discrete points as the point with the shortest perpendicular distance to the velocity vector. There are several enhancements to this scheme, based on the direction of the velocity vector, that could substantially reduce the number of candidate discrete points, but a more detailed discussion is beyond the scope of this paper. Reducing the number of candidate discrete points is especially important for problems with larger numbers of design variables.

The more elaborate approach was expected to result in a more efficient integer/discrete algorithm. In contrast to this expectation, numerical experiments indicated that there was no significant difference in the performance of the PSO algorithm when using the first as compared to the second approach. Because there is no advantage to use the more elaborate approach, this approach was discarded, and the simpler rounding approach was used for all integer/discrete problems considered here.

Mathematical Example Problem

As a first example problem, a two design variable, unconstrained mathematical function was considered. The objective function to be minimized is shown as follows with the two design variables were allowed to vary between −50 and 10:

$$F(x_1, x_2) = x_1^2 - 100\cos(x_1)^2 - 100\cos\left(x_1^2/30\right) + x_2^2 - 100\cos(x_2)^2 - 100\cos\left(x_2^2/30\right) + 1400 \tag{10}$$

This problem has many local minima, as shown in Fig. 1, which provides a contour plot of the objective function over the design space. The global optimum has a value of 1000 and is located at $x_1 = x_2 = 0$.

This example problem was included to verify the PSO implementation and to test the proposed scheme for dynamically updating the inertia weight considered as follows: 1) keeping w constant at a large value to provide a more global search, 2) keeping w constant at a small value to provide a more local search, and 3) using the proposed scheme to change the w value dynamically during the optimization. In all cases, a swarm size of 20 particles were considered

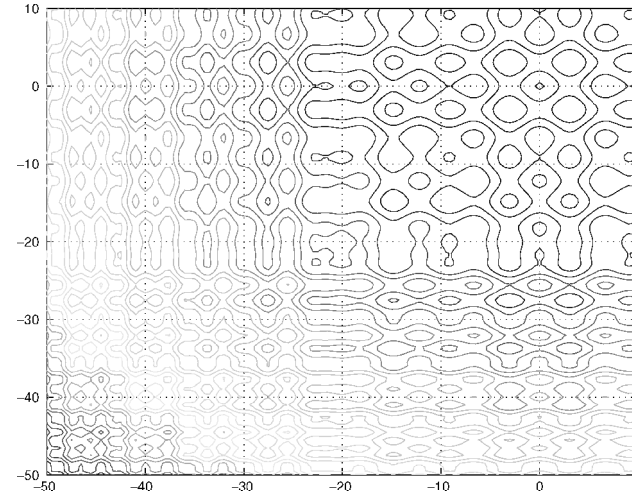

Fig. 1 Mathematical example problem.

Table 1 Mathematical example problem results

Option w	Cost				Objective				Success rate, %
	Mean	Standard deviation	Best	Worst	Mean	Standard deviation	Best	Worst	
0.5	345	48	240	480	1005.13	7.03	1000.00	1015.56	66
1.4	9940	374	7340	10000	1000.17	0.19	1000.00	1000.90	100
Dynamic	714	63	540	860	1000.09	0.10	1000.00	1000.45	100

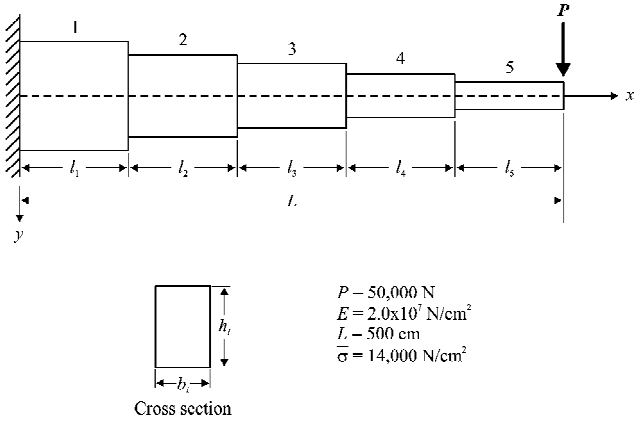

Fig. 2 Cantilevered beam example problem.

with $c_1 = 1.5$ and $c_2 = 2.5$, and the proposed craziness operator was applied. Each run was repeated 50 times and the best, worst, mean, and standard deviation of the best objective function and the number of function evaluations to convergence were recorded for each of the 50 repetitions. In addition, the reliability of the algorithm, measured in terms of the success rate by counting the number of optimization runs that found an objective function value within 1% of the best objective function value from all 50 runs, was also recorded. The results are presented in Table 1.

The results of Table 1 clearly illustrate that the implemented algorithm is capable of solving this unconstrained, continuous example problem very accurately. The algorithm is able to deal with the many local minima present in the example problem and finds the global optimum reliably. The influence of the w value is also clearly illustrated, with $w = 0.5$ resulting in a more efficient (in terms of function evaluations) but local search that sometimes converge on a local optimum. In contrast, $w = 1.4$ provides a more global search that finds the global optimum reliably, but at much higher computation cost. The proposed scheme to change the w value dynamically from an initially more global search to a more local search at the end of the optimization seems to work very well. It increases the robustness of the algorithm (compared to the $w = 0.5$ case), while reducing the computational cost (compared to the $w = 1.4$ case).

Cantilevered Beam Example Problem

To study the influence of the proposed enhancements of the PSO algorithm, a cantilevered beam example, simular to that considered by Vanderplaats[11] was chosen. A schematic representation of the example problem, including material properties, is shown in Fig. 2.

The beam was modeled using five segments of equal length, and the design problem was defined as minimizing the material volume of the beam, subject to maximum bending stress constraints for each segment. The design variables were the height h and width b of each segment, resulting in 10 independent design variables. Two cases were considered. The first was a continuous design problem where the height of each segment was allowed to vary between 10 and 200 cm, while the width was allowed to vary between 0.5 and 50 cm. The second was an integer/discrete case where all 10 design variables were restricted to integer values only. For the second case, the height of each segment was allowed to vary between 10 and 200 cm, while the width was allowed to vary between 1 and 50 cm.

Table 2 Comparison between continuous GENESIS and theoretical results

Parameter	Baseline	GENESIS	Theory
Volume, cm^3	36,683	27,498	27,438
b_1, cm	0.5	0.5	0.5
b_2, cm	0.5	0.5	0.5
b_3, cm	0.5	0.5	0.5
b_4, cm	0.5	0.5	0.5
b_5, cm	0.5	0.5	0.5
h_1, cm	146.73	146.73	146.39
h_2, cm	146.73	131.16	130.93
h_3, cm	146.73	113.16	113.39
h_4, cm	146.73	92.78	92.58
h_5, cm	146.73	65.61	65.47

The bending stress was obtained from the bending stress equation

$$\sigma = My/I \tag{11}$$

where σ is the bending stress, M the applied bending moment, I the moment of inertia, and y is the vertical distance, measured from the neutral axis, where the stress is calculated. For this problem, $M = P(L - x)$ and $I = \frac{1}{12}bh^3$, whereas the maximum stress value occurs at $y = h/2$. The height has a much larger influence on the bending stress, as compared to the width of the beam. It is reasonable to assume that the optimizer would minimize the weight by keeping the width constant and equal to its lower bound, while changing the height of the beam. In this case, the theoretical solution for the height is

$$h = \sqrt{\frac{6P(L - x)}{\bar{\sigma} b}} \tag{12}$$

where P is the applied tip load, x is the horizontal distance measured from the root of the beam, and $\bar{\sigma}$ is the allowable stress limit.

To verify the assumptions and to study how well a gradient-based optimizer would solve this problem, it was decided to model and solve the problem using the GENESIS[12] structural analysis and optimization code. The theoretical optimum results for a beam with uniform height and width across the span, as well as the GENESIS and theoretical optimum results for a beam with five segments, are summarized in Table 2. The case with uniform height and width was obtained by setting the width equal to its lower bound and calculating the height from Eq. (12) to have the maximum stress at the root equal to the allowable stress. The case with constant height and width may be considered as a baseline from which the optimizer makes an improvement.

Table 2 verifies both the assumptions that lead to Eq. (12) and the GENESIS results. GENESIS solved the problem using a total of 12 finite element analyses. Note that GENESIS makes use of advanced approximation techniques to reduce the required function evaluations for solving structural optimization problems and does not need to perform finite difference gradient calculations. In practice, most general purpose gradient-based optimizers obtain gradient information using finite difference gradient calculations. As a result a typical general-purpose gradient-based optimizer would most probably require between 65 and 130 function evaluations to solve this problem. The number of function evaluations are estimated based on 5–10 design iterations for convergence, with each design iteration requiring 3 one-dimensional search calculations and 10 finite difference gradient calculations.

Table 3 GENESIS discrete and upper and lower bound solutions for the integer/discrete case

Parameter	Lower bound	GENESIS discrete	Upper bound
Volume, cm^3	38,803	47,000	39,100
b_1, cm	1.0	1.0	1.0
b_2, cm	1.0	2.0	1.0
b_3, cm	1.0	1.0	1.0
b_4, cm	1.0	1.0	1.0
b_5, cm	1.0	1.0	1.0
h_1, cm	103.51	104	104
h_2, cm	92.58	93	93
h_3, cm	80.18	80	81
h_4, cm	65.47	66	66
h_5, cm	46.29	47	47

Finding the theoretical optimum for the integer/discrete case is a more daunting task. Instead, we will determine tight upper and lower bounds for the objective function value, by considering a partially discrete solution, rounding and the discrete optimizer provided in GENESIS. A lower bound for the discrete problem may be obtained from Eq. (12), using the lower bound values of 1.0 for all b_i, thus producing an optimum answer with half the variables being integer/discrete and the other half being continuous. An upper bound may be obtained by considering three discrete points, obtained by rounding the continuous h_i values obtained from Eq. (12). The three points are obtained by rounding all h_i values up, rounding all h_i values down, and rounding all h_i values to their closest integer value. It turns out that only the case where all h_i values are rounded up produce a feasible design. Alternatively, an upper bound may be obtained by using the discrete optimizer provided in GENESIS. The discrete optimizer in GENESIS makes use of a sequential unconstrained minimization technique to find good feasible discrete solutions, without providing a guarantee of finding the best discrete solution. The algorithm is particularly powerful when solving discrete problems with large numbers (thousands) of discrete variables.

The results obtained from the partial discrete solution, the rounding technique, and the GENESIS discrete optimizer are shown in Table 3. Although the rounding technique and the GENESIS discrete optimizer provided very simular results, the tightest discrete/integer upper bound is provided by rounding rather than by the GENESIS discrete optimizer. The difference in the objective function values of the calculated upper and lower bounds is less than 1%, and Table 3, thus, provides tight upper and lower bounds for the integer/discrete solution.

Cantilevered Beam Results

The PSO algorithm was used to analyze the beam example problem, employing the elementary strength of materials approach described in the preceding section. As discussed, two design problems were considered; the first is a continuous problem and the second an integer/discrete problem. For each design problem, the influence of two enhancements to the basic algorithm was considered. The first is the proposed craziness operator. As already mentioned, there seem to be disagreement in the literature as to the usefulness of the craziness operator. The second is resetting the velocity vectors of violated design points. Resetting the velocity vectors is a new feature that has not been studied previously. To investigate fully the influence of these two enhancements, all possible combinations of using and not using the enhancements were considered, resulting in four possible combinations. The PSO algorithm was first run for a fixed number of function evaluations and then using the proposed convergence criterion. In all cases, a swarm size of 300 particles was used.

Each run was repeated 50 times and the best, worst, mean, and standard deviation of the best objective function from each of the 50 repetitions were recorded. For the runs where the convergence criterion was used, the same statistics were also recorded for the number of function evaluations to convergence for each of the 50 repetitions. In addition, the reliability of the algorithm in terms of the number of optimization runs that found an optimum solution within 1% of the best optimum found from all 50 runs was also recorded.

Table 4 PSO parameters used in example problems

Parameter	Value
Number of particles	300
Initial inertia weight w	1.4
Trust parameter 1 c_1	1.50
Trust parameter 2 c_2	2.50

Table 5 PSO enhancement summary

Option	Definition
C	Apply craziness operator
R	Reset velocities of violated particles

Table 6 Objective function (material volume) statistics for the continuous design problem

Option	Mean	Standard deviation	Best	Worst	Success rate, %
——	39,273	15,395	28,142	119,501	2
R	29,789	6,419	27,438	54,283	76
C	39,199	16,378	27,572	118,182	4
CR	31,220	13,115	27,438	93,454	82

Table 7 Objective function (material volume) statistics for the integer design problem

Option	Mean	Standard deviation	Best	Worst	Success rate, %
——	61,339	15,411	39,600	102,384	4
R	42,525	10,018	39,100	85,085	78
C	65,079	23,501	39,300	165,624	10
CR	40,790	6,753	39,100	73,108	82

For all runs, the same PSO parameters were used, as summarized in Table 4. The w, c_1, and c_2 values were determined as discussed in earlier sections of this paper.

The number of particles (swarm size) was selected as a tradeoff between cost and reliability. Smaller swarm sizes required less function evaluations for convergence but decreased the reliability of the algorithm. Larger swarm sizes required more function evaluations for convergence, but increased the reliability of the algorithm. Additionally, a larger swarm size has the advantage of exploiting more processors in a massively parallel processing environment. The authors did not perform exhaustive testing to find the best swarm size. Instead, some preliminary testing was performed, and a larger swarm size was selected, keeping in mind the added advantage in a parallel computing environment. Future work should include not only a study to find the best swarm size, but also an investigation into performing multiple runs, each with less function evaluations. For example, Le Riche and Haftka[13] showed that it might be more advantageous to perform two genetic searches with, for example, 50 iterations each, as compared to performing a single genetic search with 100 iterations. Performing multiple searches at the same time can help reduce the overall cost and has the added advantage of exploiting even more processors in a massively parallel environment.

Each run is identified by the combination of enhancements used during that run, as summarized in Table 5. From Table 5, *R* would represent resetting the velocities of violated particles only, whereas *CR* would represent using the craziness operator and resetting the velocities of violated particles.

Fixed Number of Function Evaluations

First, each of the four combinations was evaluated using a fixed number of design iterations equal to 50, resulting in a total number of function evaluations equal to 15,000. The statistical results obtained from 50 repetitions for each combination are summarized in Table 6 for the continuous design problem and in Table 7 for the integer/discrete design problem.

When the statistical data, especially the mean and standard deviation values and the success rate are compared, from Tables 6 and 7

it is clear that resetting the velocity vectors of the violated design points had a significant and positive influence on the performance of the PSO algorithm. In contrast, the craziness operator did not appear to have a big influence. It is not clear whether combining the craziness operator and resetting the velocity vectors of the violated design points resulted in any additional improvement over just resetting the velocity vectors without the craziness operator. Finally, the standard deviation clearly shows that the algorithm is more successful in solving the discrete problem (Table 7) than the continuous problem (Table 6). This was expected because the discrete problem has a smaller design space as compared to the continuous problem.

For the case with a fixed number of function evaluations, it is possible to compare the optimum results obtained by the PSO algorithm against other systematic sampling techniques. An exhaustive comparison is beyond the scope of the current paper. However, it was decided to compare the performance of the PSO algorithm against a simple random search, using the same number of function evaluations. A random search with 15,000 analyses was performed, repeating the process 50 times and recording the statistics for the best objective function from each repetition. The results are summarized in Table 8.

Table 8 clearly indicates that the random search performed significantly worse as compared to the PSO algorithm (Table 6). In fact, the random search did not find a single feasible optimum solution.

Convergence Criterion

Next the runs of Tables 6 and 7 were repeated, using the proposed convergence criterion. For convergence, the objective function was required not to change more than 0.1% in 10 consecutive design iterations. A maximum of 500 design iterations, equivalent to 150,000 analyses, was allowed for cases where the algorithm did not converge.

The statistical results for both the cost (number of function evaluations) and the objective function values obtained from 50 repetitions are summarized in Table 9 for the continuous design problem. Table 10 contains the corresponding results for the integer/discrete design problem.

The results are simular to that of Tables 6 and 7. Resetting the velocity vectors of violated particles had a big influence on the performance of the PSO algorithm, whereas applying the craziness operator had a much smaller influence. It would appear that combining the two enhancements would have a positive influence by reducing the number of function evaluations, while providing more consistent results (smaller mean and standard deviation for the objective function values). Again, the discrete problem was solved more efficiently than the continuous problem.

The best continuous design point found by the PSO had an objective function of 27,440 cm^3, whereas the best discrete solution had an objective function value of 39,100 cm^3. The best results found by the PSO algorithm are compared to the theoretical answer for the continuous problem in Table 11.

Table 11 shows that the PSO algorithm found a very accurate optimum solution for the continuous design problem as compared to the theoretical answer. The best integer/discrete solution found is within the calculated upper and lower bounds summarized from Table 3. In fact, the best integer/discrete solution found is equal to the upper bound shown in Table 3.

Massively Parallel Computing

Although the authors did not apply the algorithm in a parallel computing environment, the algorithm is ideally suited for application with large numbers of processors. Within each design iteration, the analyses are independent of each other and can easily be performed on multiple processors with little communication between the processors. It is, thus, reasonable to expect near perfect speedup within a design iteration, when adding more processors.

The present work showed that the algorithm worked very well with a larger swarm size, which allows for using more processors in a massively parallel environment. If one considers the cases where both enhancements to the basic algorithm are considered, an average of 49.24 design iterations is required to solve the continuous problem and 32.62 to solve the integer/discrete problem. When using 300 processors, a parallel implementation should allow the designer to solve the 10 design variable continuous problem in the equivalent time of 49 analyses, whereas the integer/discrete case can be solved in the equivalent time of only 32 analyses. If more processors are available, the number of particles considered can be increased to the number of available processors.

Even larger numbers of processors could potentially be utilized by extending the work of Le Riche and Haftka[13] that showed that

Table 8 Objective function (material volume) statistics for the continuous problem using a random search

Statistic	Value
Mean	176,956
Standard deviation	28,349
Best	117,115
Worst	261,619

Table 11 Comparison between PSO and theoretical results

Parameter	Theoretical continuous	PSO	
		Continuous	Discrete
Volume	27,429	27,438	39,100
b_1	0.5	0.5	1.0
b_2	0.5	0.5	1.0
b_3	0.5	0.5	1.0
b_4	0.5	0.5	1.0
b_5	0.5	0.5	1.0
h_1	146.27	146.39	104.0
h_2	130.85	130.93	93.0
h_3	113.31	113.39	81.0
h_4	92.54	92.58	66.0
h_5	65.44	65.47	47.0

Table 9 Cost and objective function (material volume) statistics for the continuous design problem

Option	Cost				Objective				Success rate, %
	Mean	Standard deviation	Best	Worst	Mean	Standard deviation	Best	Worst	
——	33,228	36,963	15,000	150,000	39,296	16,840	27,440	85,974	6
R	14,370	3,386	8,700	24,000	28,997	5,383	27,442	61,007	80
C	51,660	52,845	17,400	150,000	37,015	14,718	27,440	123,451	10
CR	15,012	3,895	8,700	26,100	29,372	4,951	27,441	49,699	72

Table 10 Cost and objective function (material volume) statistics for the integer design problem

Option	Cost				Objective				Success rate, %
	Mean	Standard deviation	Best	Worst	Mean	Standard deviation	Best	Worst	
——	73,938	62,708	17,100	150,000	57,592	16,394	39,100	108,282	24
R	10,134	1,807	7,800	16,800	41,233	7,711	39,100	76,588	88
C	77,532	63,703	16,500	150,000	56,709	20,687	39,100	124,901	36
CR	10,026	1,815	6,900	17,100	40,628	5,820	39,100	73,108	78

it might be more advantageous to run multiple optimizations, each with less design iterations as compared to a single optimization with more design iterations. The use of the PSO algorithm in a massively parallel environment is a topic for future research, especially when considering problems with many design variables.

Conclusions

The PSO algorithm was applied to an unconstrained, continuous mathematical example problem with many local minima as well as both a continuous and an integer/discrete structural design problem with constraints.

The results indicate that the PSO algorithm is able to deal with both unconstrained and constrained as well as the continuous and integer/discrete design problems considered here. The algorithm was also successful at solving a design problem with many local minima. Although the PSO algorithm was able to solve the continuous design problems very accurately, the computational cost was higher than that of traditional gradient-based optimizers. For the integer/discrete problem considered, where use of a gradient-based optimizer may not be appropriate, the PSO algorithm performed even better. The integer/discrete design problem was solved more accurately and using less function evaluations as compared to the corresponding continuous design problem.

Although only two variants of the same design problem were studied, the results show that the newly introduced idea of resetting the velocity vectors of violated design points has a significant positive influence on the performance of the algorithm. The craziness operator does not have a big influence. However, it seems that there might be a small advantage to combining the two enhancements as compared to just using the idea of resetting the velocity vectors. Note that a fairly large swarm size was considered in the present work, which might explain why the craziness operator was not particularly effective in the current example problems.

References

[1]Michalewicz, Z., and Dasgupta, D. (eds.), *Evolutionary Algorithms in Engineering Applications*, Springer-Verlag, Berlin, 1997.

[2]Nemhauser, G. L., and Wolsey, L. A., *Integer and Combinatorial Optimization*, Wiley, 1988, New York, Chap. 3.

[3]Kennedy, J., and Eberhart, R. C., "Particle Swarm Optimization," *Proceedings of the 1995 IEEE International Conference on Neural Networks*, Vol. 4, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, 1995, pp. 1942–1948.

[4]Eberhart, R. C., and Kennedy, J., "A New Optimizer Using Particle Swarm Theory," *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, 1995, pp. 39–43.

[5]Kennedy, J., and Spears, W. M., "Matching Algorithms to Problems: An Experimental Test of the Particle Swarm and Some Genetic Algorithms on the Multimodal Problem Generator," *Proceedings of the 1998 International Conference on Evolutionary Computation*, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, 1998, pp. 78–83.

[6]Shi, Y., and Eberhart, R. C., "A Modified Particle Swarm Optimizer," *Proceedings of the International Conference on Evolutionary Computation*, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, 1998, pp. 69–73.

[7]Shi, Y. H., and Eberhart, R. C., "Parameter Selection in Particle Swarm Optimization," *Evolutionary Programming VII: Proc. EP98*, Lecture Notes in Computer Science, Springer-Verlag, New York, 1998, pp. 591–600.

[8]Clerc, M., "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization," *Proceedings of the Congress of Evolutionary Computation*, Vol. 3, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, 1999, pp. 1951–1957.

[9]Fourie, P. C., and Groenwold, A. A., "The Particle Swarm Optimization Algorithm in Size and Shape Optimization," *Structural and Multidisciplinary Optimization*, Vol. 23, No. 4, 2002, pp. 259–267.

[10]Fourie, P. C., and Groenwold, A. A., "Particle Swarms in Topology Optimization," *Proceedings of the Fourth World Congress of Structural and Multidisciplinary Optimization*, Liaoning Electronic Press, 2001, pp. 177.1–177.6

[11]Vanderplaats, G. N., *Numerical Optimization Techniques for Engineering Design*, 3rd ed., Vanderplaats Research and Development, Inc., Colorado Springs, CO, 1999, pp. 185–189, 237–250.

[12]"GENESIS Version 7.0 Users Manual," Vanderplaats Research and Development, Inc., Colorado Springs, CO, 2001.

[13]Le Riche, R., and Haftka, R., "Improved Genetic Algorithm for Minimum Thickness Composite Laminate Design," *Composite Engineering*, Vol. 5, No. 2, 1995, pp. 143–161.

A. Chattopadhyay
Associate Editor