Steel and Composite Structures, *Vol. 18, No. 2 (2015) 289-303*
DOI: http://dx.doi.org/10.12989/scs.2015.18.2.289

Subspace search mechanism and cuckoo search algorithm for size optimization of space trusses

A. Kaveh * and T. Bakhshpoori

Centre of Excellence for Fundamental Studies in Structural Engineering, School of Civil Engineering, Iran University of Science and Technology, Narmak, Tehran-16, Iran

(*Received May 03, 2014, Revised June 09, 2014, Accepted June 14, 2014*)

Abstract. This study presents a strategy so-called Subspace Search Mechanism (SSM) for reducing the computational time for convergence of population based metaheusristic algorithms. The selected metaheuristic for this study is the Cuckoo Search algorithm (CS) dealing with size optimization of trusses. The complexity of structural optimization problems can be partially due to the presence of high-dimensional design variables. SSM approach aims to reduce dimension of the problem. Design variables are categorized to predefined groups (subspaces). SSM focuses on the multiple use of the metaheuristic at hand for each subspace. Optimizer updates the design variables for each subspace independently. Updating rules require candidate designs evaluation. Each candidate design is the assemblage of responsible set of design variables that define the subspace of interest. SSM is incorporated to the Cuckoo Search algorithm for size optimizing of three small, moderate and large space trusses. Optimization results indicate that SSM enables the CS to work with less number of population (42%), as a result reducing the time of convergence, in exchange for some accuracy (1.5%). It is shown that the loss of accuracy can be lessened with increasing the order of complexity. This suggests its applicability to other algorithms and other complex finite element-based engineering design problems.

Keywords: subspace search mechanism; structural optimization; population based meta-heuristic algorithms; cuckoo search; truss size optimization

1. Introduction

Since the 1960s, due to the significant development in numerical methods and computing, the finite element analysis has become a frequent tool to solve engineering problems (Reddy 1993). Parallel to the developments in FEM, optimization methods contributed to the discipline of structural optimization. The aim of structural optimization is to generate automated procedures for finding the best possible structure with respect to at least one criterion (the objective), satisfying a set of constraints (Davarynejad *et al*. 2012). Two major elements must be included in any structural optimization problem: one is the analysis or evaluation of the candidate design considered for structure, and the other is the optimization solver. For the second part, efficient and fast stochastic optimization algorithms (metaheuristic algorithms) are developed to overcome

*Corresponding author, Professor, E-mail: alikaveh@iust.ac.ir

http://www.techno-press.org/?journal=scs&subpage=8 ISSN: 1229-9367 (Print), 1598-6233 (Online)

disadvantages of traditional optimization solvers (Gradient-based optimization algorithms) and gained increasing popularity because of their ability to deliver satisfactory solutions in a reasonable time. Performance assessment of a meta-heuristic may be used by solution quality, computational effort, and robustness (Talbi 2009), directly affected by its two contradictory criteria: exploration of the search space (diversification) and exploitation of the best solutions found (intensification). To alleviate these two features over the last three decades various kinds of population based metaheuristic algorithms have been invented and modified, and applied successfully in structural optimization. Examples of these are: Genetic Algorithms (GAs) (Tang *et al*. 2005), Simulated Annealing (SA) (Lamberti 2008), Ant Colony Optimization (ACO) (Camp and Bichon 2004), Particle Swarm Optimization (PSO) (Talatahari *et al.* 2013), Harmony Search algorithm (HS) (Degertekin 2012), Big Bang-Big Crunch algorithm (BB-BC) (Camp 2007), Artificial Bee Colony algorithm (ABC) (Sonmez 2011), Charged System Search method (CSS) (Kaveh and Talatahari 2010a), Imperialist Competitive Algorithm (ICA) (Kaveh and Talatahari 2010b), Cuckoo Search Algorithm (CS) (Kaveh and Bakhshpoori 2013a), Teaching Learning Based Optimization algorithm (TLBO) (Degertekin 2013), and Colliding Bodies Optimization (CBO) (Kaveh and Mahdavi 2014) among many others (Hasançebi *et al*. 2009, Saka and Geem 2013). Further improvement of metaheuristics can be achieved by different strategies such as the following among many others: Hybridization approaches (Hybrid metaheuristics or combining with exact methods) (Adeli and Cheng 1994, Kaveh and Talatahari 2009), Parallel Design of metaheuristics (Sarma and Adeli 2001), approaches to modify the updating rules such as tuning the control parameters so as to maintain the balance between local search and global search (Liu *et al*. 2005), and strategies related to the search space such as search space reduction (Cheng and Yap 2008). The majority of the works are focused on improving the performance of the algorithms in terms of the fitness function considered, whereas the computational cost and time consumption of the algorithms have been considered as a secondary issue.

The contribution of this study is to present a strategy for reducing the time of convergence of population based metaheusristic algorithms in and inductive manner by Cuckoo Search algorithm (CS) for size optimization of trusses. CS as one of the population based algorithms, incorporates some cuckoo species lifestyle and Lévy flight behavior. This algorithm is developed by Yang and Deb (Yang 2008, Yang and Deb 2009), and has been used successfully in structural optimization problems (Kaveh *et al*. 2011, 2012, Saka and Dogan 2012, Shayanfar *et al.* 2013, Gandomi *et al*. 2013, Kaveh *et al*. 2014, 2015). Size optimization of trusses is often taken as benchmark in constrained global structural optimization because of the presence of many design variables (i.e., large search spaces) and nonlinear constraints.

The complexity of structural optimization problems can be partially due to the presence of high-dimensional design variables (Davarynejad *et al*. 2012). The strategy that we introduce is a subspace search strategy (named as Subspace Search Mechanism (SSM)), This strategy reduces the dimension of the problem and lets the algorithm to work with a smaller number of population. Design variables are categorized to predefined groups (subspaces). SSM focuses on the multiple use of the metaheuristic at hand for each subspace. Optimization algorithm updates the design variables for each subspace independently. Updating rules require candidate designs evaluation. Each candidate design is the assemblage of responsible set of design variables that define the subspaces of interest. In order to demonstrate the efficiency and robustness of the proposed approach, SSM is applied in the context of the CS algorithm to three small, moderate and large weight minimization problems of truss structures with sizing variables (size optimization of trusses). Optimization results indicate that SSM enables the CS to work with a smaller number of

population (42%), as a result reduces the time of the convergence, in exchange for some accuracy (1.5%). It will be shown that this trade off will be lessen with increasing the order of complexity of the test problems.

The rest of this paper is organized as follows: The proposed SSM is introduced in Section 2 besides outlining the CS algorithm and statement of truss sizing optimization problem. Results of truss design problems using SSM are discussed in Section 3. Section 4 summarizes the main findings of this study along with some remarks for future research.

2. Subspace Search Mechanism (SSM) within the CS for truss sizing problem

In this section we describe the Subspace Search Mechanism. First the Cuckoo Search algorithm is outlined and in subsequent subsection the weight minimization problem of a truss structure is stated. At the end the Subspace Search Mechanism is described for the CS algorithm for size optimization of truss problems.

2.1 Cuckoo Search (CS) algorithm

Cuckoo Search is a meta-heuristic algorithm inspired by some species of a bird family called Cuckoo because of their special lifestyle and aggressive reproduction strategy (Yang 2008). These species lay their eggs in the nests of other host birds with amazing abilities like selecting the recently spawned nests and removing existing eggs that increase hatching probability of their eggs. The host takes care of the eggs presuming that the eggs are its own. However, some of host birds are able to combat with this parasites behavior of cuckoos, and throw out the discovered alien eggs or build their new nests in new locations. The cuckoo breeding analogy is used for developing new design optimization algorithm. A generation is represented by a set of host nests. Each nest carries an egg (solution). The quality of the solutions is improved by generating a new solution from an existing solution and modifying certain characteristics. The number of solutions remains fixed in each generation. In this study the later version of the CS algorithm is used for optimum design of frames (Yang and Deb 2009). The pseudo-code of the optimum design algorithm is as it follows (Kaveh and Bakhshpoori 2013b).

2.1.1 Initialize the cuckoo search algorithm parameters

The CS parameters are set in the first step. These parameters consist of the number of nests (n), the step size parameter (α), discovering probability (pa) and the maximum number of frame analyses as the stopping criterion.

2.1.2 Generate initial nests or eggs of the host birds

The initial locations of the nests are determined by the set of values randomly assigned to each decision variable as

$$nest_{i,j}^{(0)} = ROUND\left(x_{j,\min} + rand.(x_{j,\max} - x_{j,\min})\right) \tag{1}$$

where $nest_{i,j}^{(0)}$ determines the initial value of the jth variable for the ith nest; $x_{j,\min}$ and $x_{j,\max}$ are the minimum and the maximum allowable values for the jth variable; *rand* is a random number in the interval [0, 1]. The rounding function is introduced due to the discrete nature of the problem.

2.1.3 Generate new Cuckoos by Lévy flights

In this step, all the nests except for the best one are replaced based on their quality by new cuckoo eggs produced with Lévy flights from their positions as

$$nest_i^{(t+1)} = nest_i^{(t)} + \alpha.S.\left(nest_i^{(t)} - nest_{best}^{(t)}\right).r \tag{2}$$

where $nest_i^t$ is the ith nest current position, α is the step size parameter; r is a random number from a standard normal distribution and $nest_{best}$ is the position of the best nest so far; and S is a random walk based on the Lévy flights. The Lévy flight essentially provides a random walk while the random step length is drawn from a Lévy distribution. In fact, Lévy flights have been observed among foraging patterns of albatrosses, fruit flies and spider monkeys. One of the most efficient and yet straightforward ways of applying Lévy flights is to use the so-called Mantegna algorithm. In Mantegnas algorithm, the step length S can be calculated by

$$S = \frac{v}{|v|^{1/\beta}} \tag{3}$$

where β is a parameter between [1, 2] interval and here it is considered as 1.5; u and v are drawn from normal distribution as

$$u \sim N(0,\ \sigma_u^2), \quad v \sim N(0,\ \sigma_v^2), \tag{4}$$

$$\sigma_u = \left\{\frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\Gamma[(1+\beta)/2]\beta 2^{(\beta-1)/2}}\right\}^{1/\beta}, \quad \sigma_v = 1 \tag{5}$$

2.1.4 Alien eggs discovery

The alien eggs discovery is performed for each component of each solution in terms of the probability matrix such as

$$P_{ij} = \begin{cases} 1 & if \quad rand > pa \\ 0 & if \quad rand \geq pa \end{cases} \tag{6}$$

where *rand* is a random number in [0, 1] interval, and *pa* is the discovering probability. Existing eggs are replaced considering their quality by the newly generated ones from their current positions through random walks with step size such as

$$\begin{aligned} &S = rand.\left(nests(randperm1(n),:) - nests\left(randperm2(n),:\right)\right) \\ &nest^{t+1} = nest^t + S.*P \end{aligned} \tag{7}$$

where *randperm*1 and *randperm*2 are random permutation functions used for different rows permutation applied on nests matrix and P is the probability matrix.

2.1.5 Termination criterion

The generating new cuckoos and discovering alien eggs steps are alternatively performed until a termination criterion is satisfied. The maximum number of analyses is considered as termination

criterion of the algorithm.

2.2 Statement of the size optimization of truss problem

The weight minimization problem of a truss structure can be stated as follows

$$\begin{aligned} &\text{Find} && \{X\}=[x_1,x_2,...,x_{ng}], && x_i \in D \\ &\text{To minimize} && W(\{X\})=\sum_{i=1}^{ng} x_i \sum_{j=1}^{nm(i)} \rho_j . L_j && \\ &\text{Subject to:} && g_j(\{X\}) \le 0 && j=1,2,...,n \end{aligned} \tag{8}$$

where $\{X\}$ is the set of design variables; *ng* is the number of member groups (i.e., the number of optimization variables) defined according to structural symmetry; D represents the design space including the cross-sectional areas of truss elements that can take discrete or continuous values; $W(\{X\})$ is the weight of the structure; *nm(i)* is the number of members included in the *i*th group; ρ_j and L_j are respectively the material density and the length of the *j*th member included in the *i*th group; $g_j(\{X\})$ denote the *n* optimization constraints.

In order to handle optimization constraints, a penalty approach is utilized in this study by introducing the following pseudo-cost function

$$f_{cost}(\{X\})=(1+\varepsilon_1.\upsilon)^{\varepsilon_2} \times W(\{X\}), \quad \upsilon=\sum_{j=1}^{n} \max(0,\ g_j(\{X\})) \tag{9}$$

where υ is the total constraint violation. Constants ε_1 and ε_2 must be selected considering the exploration and the exploitation rate of the search space. In this study, ε_1 is set equal to one while ε_2 is selected so as to decrease the total penalty yet reducing cross-sectional areas. Thus, ε_2 increases from the value of 1.5 set in the first steps of the search process to the value of 3 set towards the end of the optimization process.

Stress limits on *nm* truss members are imposed as follows

$$g_j(\{X\})=\frac{|\sigma_j|}{|\sigma_j^u|}-1 \qquad j=1,2,...,nm \tag{10}$$

Optimization constraints on *nn* nodal displacements are set as follows

$$g_j(\{X\})=\frac{|\delta_j|}{\delta_j^u}-1 \qquad j=nm+1,nm+2,...,nm+nn=n \tag{11}$$

where: δ_i is the displacement of the *i*th node of the truss, δ_i^u is the corresponding allowable displacement, and *nn* is the number of nodes.

2.3 Subspace search mechanism (SSM)

Solving a *n*g-dimensional structural optimization problem using SSM involves three inter-related phases:

(1) *Modeling the optimization problem, and specifying the ns subspaces taken through the original ng-dimensional search space*:
A ng-dimensional search space the number of possible subspaces $s \subseteq \{1,...,ng\}$ is $\sum_{i=1}^{d}\binom{ng}{i} = 2^{ng} - 1$. The Specified set of $2^{ng} - 1$ subspaces cannot shares a proportion of dimensions and should cover the entire original search space. For example the possible sets of subspaces (S_i, i = 1,..., ns) for {1, 2, 3} as a 3-dimensional search space can be considered as: $S_1 = (s_1 = \{1\}, s_2 = \{2\}, s_3 = \{3\})$; $S_2 = (s_1 = \{1, 2\}, s_2 = \{3\})$; $S_3 = (s_1 = \{1, 3\}, s_2 = \{2\})$; $S_4 = (s_1 = \{1\}, s_2 = \{2, 3\})$; and $S_5 = (s_1 = \{1, 2, 3\})$. Subspaces can be specified by designer based on some criteria of interest. As an example, in truss sizing optimization problem type of members or the story levels can be used for sub-spacing.

(2) *Implementation of optimization algorithm for each subspace independently*:
An initial population (n individuals) is randomly generated (Eq. (1)) and progressively updated to search the optimum for each subspace independently, until the stopping criteria (maximum number of iterations) is satisfied. Updating rules (Eqs. (2) and (7)) are based on individual evaluations with some impression of randomness.

(3) *Individual evaluation*:
In each iteration of the algorithm, the ith individual (i = 1,…, n) of each subspaces are combined to form a candidate design. The candidate design evaluation (Eq. (9)) reveals the value of the corresponding objective function (usually, pseudo-cost function including penalty terms or fitness) for the ith individuals of each subspaces. In this way the number of optimization problem evaluations, in each iteration of optimization process, is equal to the number of population (n) considered for the optimizer. SSM can enable the optimization algorithm to work with less number of population, as a result reduces the time for convergence. It is known that the conventional structural analysis of candidate designs during the search usually consumes 85-95% of the total computing time of an optimization algorithm (Adeli and Cheng 1994). Extra computational cost for multiple use of optimization algorithm for subspaces can be negligible.

3. Computational tests and analysis

In this section to investigate the effectiveness of the proposed SSM, three space trusses consisting of a 72-bar, a 144-bar and a 216-bar, considered as the small, moderate and large test cases respectively, are optimized. The last two trusses are the scaled structures based on the first test problem. Three options are considered for the Cuckoo Search algorithm as the optimizer in this study: (i) standard CS (titled as SCS) with the following recommended parameter values for an efficient optimum design process of trusses (Kaveh and Bakhshpoori 2013a): n = 7, α = 0.1 and pa = 0.3; (ii) CS with the subspace search mechanism (titled as CS-SSM) in which CS can works with less number of population; (iii) standard CS with less number of population titled as reduced Cuckoo Search (RCS). Our simulation results show that SSM lets CS to reduce the number of population to 4 (42% saving in computational cost) which are considered in two last options. Two other parameters for two last options are considered the same as the first one. To investigate the effect of the initial solution on the final results, each example is solved independently 25 times

with random initial designs due to the stochastic nature of the algorithm. The optimizer and the proposed method are coded in MATLAB and structures are analyzed using the direct stiffness method.

3.1 The 72-bar space truss

Fig. 1 shows the schematic of the benchmark 72-bar space truss with its topology, geometry, element grouping, nodes and element numbering schemes. The material density is 0.1 lb/in^3 and the modulus of elasticity is 10^7 psi. The 72 structural members of this spatial truss are categorized into 16 groups using symmetry: (1) A_1–A_4; (2) A_5–A_{12}; (3) A_{13}–A_{16}; (4) A_{17}–A_{18}; (5) A_{19}–A_{22}; (6) A_{23}–A_{30}; (7) A_{31}–A_{34}; (8) A_{35}–A_{36}; (9) A_{37}–A_{40}; (10) A_{41}–A_{48}; (11) A_{49}–A_{52}; (12) A_{53}–A_{54}; (13) A_{55}–A_{58}; (14) A_{59}–A_{66}; (15) A_{67}– A_{70}; and (16) A_{71}–A_{72}. In this example, designs for a multiple load cases are performed. The values and directions of the two load cases applied to the 72-bar spatial truss are listed in Table 1. The members are subjected to the stress limits of ± 25 ksi. Maximum displacement limitations of ± 0.25 in are imposed on every node in every direction. The minimum value for the cross-sectional areas is 0.1 in^2 and the maximum value is limited to 4.0 in^2.

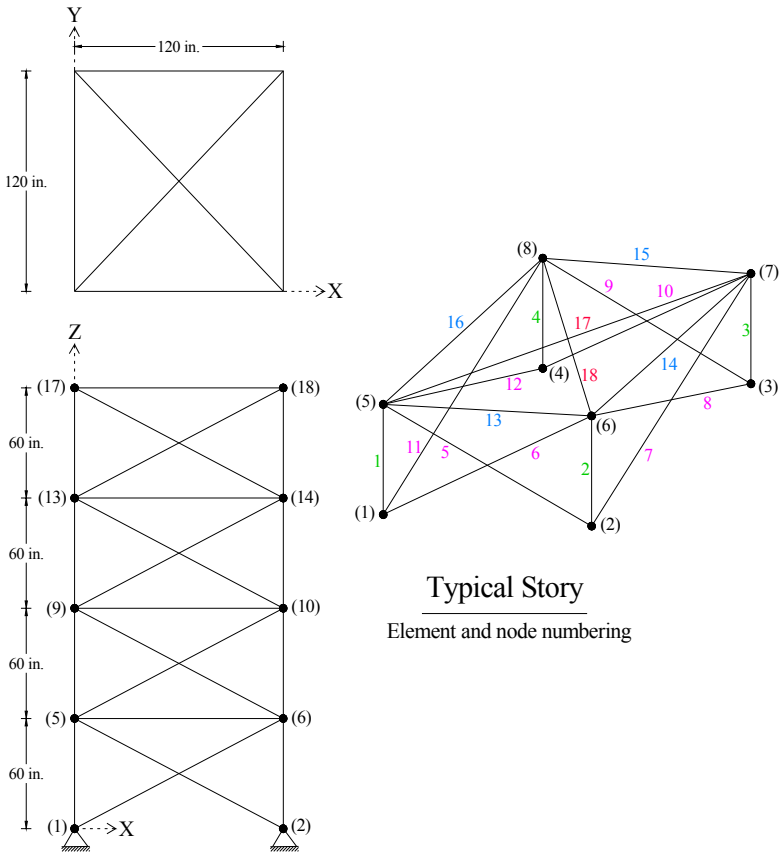

Fig. 1 Schematic of the spatial 72-bar truss

Table 1 Multiple loading conditions (kips) for the 72-bar truss

Case	Node	F_x	F_y	F_z
1	17	0.0	0.0	−5.0
	18	0.0	0.0	−5.0
	19	0.0	0.0	−5.0
	20	0.0	0.0	−5.0
2	17	5.0	5.0	−5.0

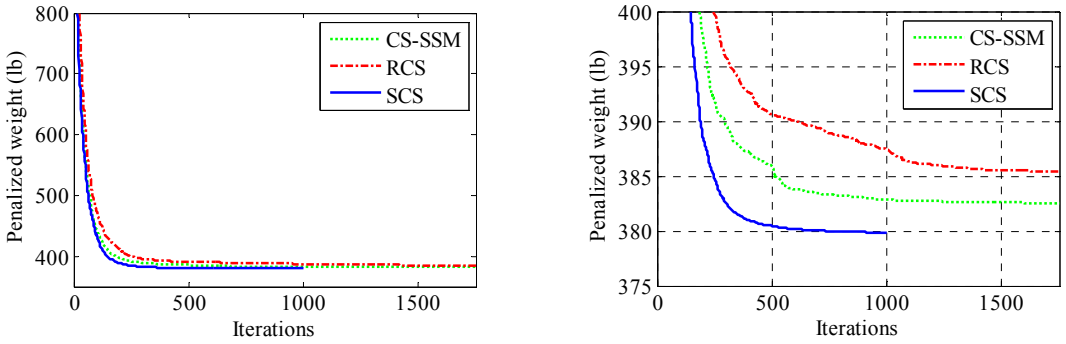

Fig. 2 Comparison of convergence curves obtained for the 72-bar truss relative to the average optimization of 25 independent runs

Table 2 Performance comparison of SCS, RCS and CS-SSM for the 72-bar truss in 25 runs

Option	Weight (lb)			
	Best	Average	Worst	SD
SCS	379.69	379.85	380.37	0.14
RCS	380.06 (0.10%)	387.42 (1.99%)	450.74 (18.50%)	17.77
CS-SSM	379.85 (0.04%)	382.84 (0.79%)	412.48 (8.44%)	7.79

Sub-spacing is performed for two adjacent stories and each type of members (columns, vertical braces, beams and horizontal braces). All 16 design variables are sub-spaced as: S = {[1 5] [9 13] [2 6] [10 14] [3 7] [11 15] [4 8] [12 16]}. The maximum number of truss analyses equal to 14000 is considered as the stopping criteria which results in 1000 and 1750 number of algorithm iterations for the first option and two for the last options, respectively. The convergence history of average optimization results achieved for 25 independent runs based on the standard Cuckoo Search (SCS), Cuckoo Search with Subspace Search Mechanisim (CS-SSM) and Cuckoo Search with less number of analyses (RCS) are compared in Fig. 2. As it is clear, the CS-SSM diagram is consistent with the SCS in the global search phase and traces the SCS in the local search phase with minor loss of accuracy. Table 2 represents the statistical results of 25 independent runs for three options with 1000 as the number of iterations. It can be seen that the SSM lets CS to work with less number of individuals. SSM enables the CS for 42% saving in computational cost to reach to the designs slightly heavier (average 0.79%) than ones obtained by the standard CS (CS-SSM works with 4 populations instead 7, saving 6 truss analyses for each iteration). Although

CS with 4 individuals results in practically the same design as those of SCS and CS-SSM. However, the average performance of the algorithm is not acceptable. Also from Fig. 2 it is clear that the RCS cannot trace the SCS converging behavior. Table 3 lists the best obtained designs by three options considering 1000 number of iterations as the stopping criteria. Optimum reported designs for this test case using other algorithms available at the optimization literature (Kaveh and Bakhshpoori 2013a), are also included in this table to show the reliability of the coded CS algorithm.

3.2 The 144-bar space truss

This truss is considered as a moderate test case which is the developed form of the first test problem with 8 stories. All 144 members are categorized like the first example which results in 32 member grouping. Maximum displacement limitations of ± 0.48 in are imposed on every node in every direction. The minimum value for the cross-sectional areas is 0.1 in^2 (0.6452 cm^2) and the maximum value is limited to 10.0 in^2. The same load conditions are applied to the same nodes of as those of the first problem in the last story. Remaining details are identical to the first example.

Sub-spacing for 32 design variables are considered the same as that of the 72-bar truss (for each type of members in two adjacent stories). This leads to a set of subspaces for use of SSM as: S = {[1 5] [9 13] [17 21] [25 29] [2 6] [10 14] [18 22] [26 30] [3 7] [11 15] [19 23] [27 31] [4 8] [12

Table 3 Optimum design by different methods for the 72-bar spatial truss

Element group		Optimal cross-sectional areas (in^2)							
							Present work		
		GA	ACO	PSO	BB–BC	HBB–BC	SCS	RCS	CS-SSM
1	A_1–A_4	1.755	1.948	1.7427	1.8577	1.9042	1.8687	1.8642	1.9049
2	A_5–A_{12}	0.505	0.508	0.5185	0.5059	0.5162	0.5219	0.5429	0.5201
3	A_{13}–A_{16}	0.105	0.101	0.1000	0.1000	0.1000	0.1002	0.1000	0.1000
4	A_{17}–A_{18}	0.155	0.102	0.1000	0.1000	0.1000	0.1001	0.1000	0.1000
5	A_{19}–A_{22}	1.155	1.303	1.3079	1.2476	1.2582	1.2815	1.2806	1.2384
6	A_{23}–A_{30}	0.585	0.511	0.5193	0.5269	0.5035	0.5158	0.5011	0.5145
7	A_{31}–A_{34}	0.100	0.101	0.1000	0.1000	0.1000	0.1000	0.1000	0.1009
8	A_{35}–A_{36}	0.100	0.100	0.1000	0.1012	0.1000	0.1000	0.1000	0.1005
9	A_{37}–A_{40}	0.460	0.561	0.5142	0.5209	0.5178	0.5230	0.5055	0.5417
10	A_{41}–A_{48}	0.530	0.492	0.5464	0.5172	0.5214	0.5113	0.5092	0.5089
11	A_{49}–A_{52}	0.120	0.100	0.1000	0.1004	0.1000	0.1001	0.1000	0.1011
12	A_{53}–A_{54}	0.165	0.107	0.1095	0.1005	0.1007	0.1001	0.1000	0.1000
13	A_{55}–A_{58}	0.155	0.156	0.1615	0.1565	0.1566	0.1569	0.1610	0.1566
14	A_{59}–A_{66}	0.535	0.550	0.5092	0.5507	0.5421	0.5371	0.5689	0.5493
15	A_{67}–A_{70}	0.480	0.390	0.4967	0.3922	0.4132	0.4150	0.3816	0.4006
16	A_{71}–A_{72}	0.520	0.592	0.5619	0.5922	0.5756	0.5698	0.5444	0.5632
Weight (lb)		385.76	380.24	381.91	379.85	379.66	379.69	380.06	379.85

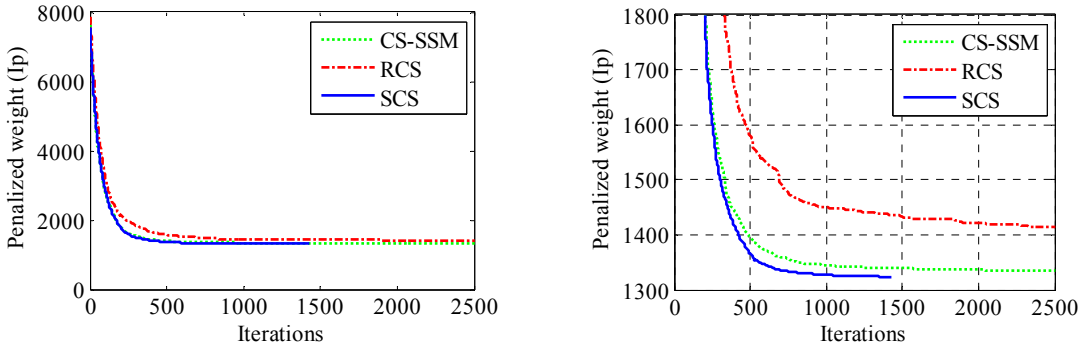

Fig. 3 Comparison of the convergence curves obtained for the 144-bar truss relative to the average optimization of 25 independent runs

Table 4 Performance comparison of the SCS, RCS and CS-SSM for the 144-bar truss in 25 runs

Option	Weight (lb)			
	Best	Average	Worst	SD
SCS	1320.43	1323.68	1370.79	9.89
RCS	1322.76 (0.18%)	1435.60 (8.46%)	1899.43 (38.57%)	152.88
CS-SSM	1321.65 (0.09%)	1339.28 (1.18%)	1446.27 (5.51%)	29.16

16] [20 24] [28 32]}. The maximum number of truss analyses equal to 20000 is considered as the stopping criteria that results in 1429 and 2500 number of iterations for the first option and for the two last options, respectively. The convergence history of average optimization results achieved for 25 independent runs based on the SCS, CS-SSM and RCS are compared in Fig. 3. Again the CS-SSM diagram is consistent with that of the SCS in the global search phase and traces the SCS in the local search phase with minor loss of accuracy. Table 4 represents the statistical results of 25 independent runs for three options with 1429 number of iterations. As it is clear the SSM lets CS to work with less number of individuals. SSM enables CS to reach the designs averagely of 1.18% heavier than ones obtained via the standard CS with 42% saving in computational cost (CS-SSM works with 4 number of population instead 7, which means the saving of 6 truss analyses for each iteration). Although CS with 4 individuals yields practically the same design to SCS and CS-SSM but the average performance of the algorithm is not acceptable (is heavier more than 8%). Comparing to first test case as the small one it is clear that SSM effectively refines the performance of the RCS with increasing the order of the complexity of problem. Also it is clear from Fig. 2 that the RCS cannot trace the SCS convergence. Table 5 lists the best obtained designs by three options considering 1429 number of iterations as the stopping criteria.

3.3 The 216-bar space truss

The 216-bar truss as the last test case is the scaled structure of the first example with 12 stories considered as the large test problem. Members are categorized like the first example resulting in 48 design variables. Maximum displacement limitations of ± 0.72 in are imposed on every node in every direction. The minimum value for the cross-sectional areas is 0.1 in^2 and the maximum

Table 5 Optimum designs for two last test cases using three options considered for CS

Element group		Optimal cross-sectional areas (in^2)					
		144-bar truss			216-bar truss		
		SCS	RCS	CSS-SSM	SCS	RCS	CSS-SSM
1	A_1–A_4	6.3375	6.5747	6.6133	17.4604	17.6797	17.3243
2	A_5–A_{12}	0.7122	0.7006	0.6898	1.5577	1.7164	1.6410
3	A_{13}–A_{16}	0.1000	0.1121	0.1025	0.1003	0.1000	0.1000
4	A_{17}–A_{18}	0.1005	0.1002	0.1000	0.1016	0.1001	0.1002
5	A_{19}–A_{22}	5.2825	5.3041	5.5013	16.1305	16.1796	15.9162
6	A_{23}–A_{30}	0.6927	0.6997	0.6861	1.6031	1.7381	1.6942
7	A_{31}–A_{34}	0.1000	0.1000	0.1008	0.1037	0.1004	0.1022
8	A_{35}–A_{36}	0.1008	0.1015	0.1000	0.1011	0.1063	0.1042
9	A_{37}–A_{40}	4.1698	4.2555	4.3684	14.7491	14.8640	14.4843
10	A_{41}–A_{48}	0.7452	0.7034	0.7144	1.6122	1.5923	1.6549
11	A_{49}–A_{52}	0.1004	0.1000	0.1000	0.1006	0.1000	0.1007
12	A_{53}–A_{54}	0.1004	0.1212	0.1003	0.1006	0.1336	0.1067
13	A_{55}–A_{58}	3.5476	3.7397	3.7117	12.6186	13.2090	12.5319
14	A_{59}–A_{66}	0.7118	0.6678	0.7070	1.6686	1.7446	1.5406
15	A_{67}–A_{70}	0.1011	0.1002	0.1000	0.1006	0.1000	0.1000
16	A_{71}–A_{72}	0.1010	0.1000	0.1086	0.1077	0.1635	0.1000
17	A_{73}–A_{76}	2.5643	2.5070	2.4391	11.5102	11.0379	11.3060
18	A_{77}–A_{84}	0.6874	0.7657	0.7066	1.6569	1.6586	1.6602
19	A_{85}–A_{88}	0.1018	0.1008	0.1003	0.1035	0.1000	0.1044
20	A_{89}–A_{90}	0.1004	0.1003	0.1000	0.1029	0.1001	0.1000
21	A_{91}–A_{94}	1.8255	1.6544	1.7368	10.4172	10.1130	10.2644
22	A_{95}–A_{102}	0.7081	0.6830	0.6891	1.6617	1.6460	1.7035
23	A_{103}–A_{106}	0.1020	0.1000	0.1000	0.1020	0.1005	0.1000
24	A_{107}–A_{108}	0.1006	0.1000	0.1000	0.1000	0.1000	0.1000
25	A_{109}–A_{112}	0.7408	0.8135	0.7709	8.7940	8.1296	9.0349
26	A_{113}–A_{120}	0.6806	0.7345	0.6846	1.6559	1.4398	1.6536
27	A_{121}–A_{124}	0.1010	0.1007	0.1000	0.1009	0.1000	0.1022
28	A_{125}–A_{126}	0.2348	0.1000	0.1257	0.1000	0.1005	0.1084
29	A_{127}–A_{130}	0.1346	0.1431	0.1427	7.4271	7.3187	7.2980
30	A_{131}–A_{138}	0.7894	0.7409	0.7635	1.6321	1.6768	1.6309
31	A_{139}–A_{142}	0.5660	0.5539	0.5311	0.1007	0.1000	0.1040
32	A_{143}–A_{144}	0.7570	0.7732	0.7417	0.1000	0.1008	0.1069
33	A_{145}–A_{148}	-	-	-	5.7125	5.8723	5.8620
34	A_{149}–A_{156}	-	-	-	1.6405	1.6077	1.6473
35	A_{157}–A_{160}	-	-	-	0.1000	0.1471	0.1004
36	A_{161}–A_{162}	-	-	-	0.1005	0.1002	0.1000
37	A_{166}–A_{166}	-	-	-	4.3148	3.8847	4.2788

Table 5 Continued

Element group		Optimal cross-sectional areas (in^2)					
		144-bar truss			216-bar truss		
		SCS	RCS	CSS-SSM	SCS	RCS	CSS-SSM
38	A_{167}–A_{174}	-	-	-	1.6313	1.6473	1.6021
39	A_{175}–A_{178}	-	-	-	0.1013	0.1000	0.1002
40	A_{179}–A_{180}	-	-	-	0.1000	0.1003	0.1000
41	A_{181}–A_{184}	-	-	-	2.9194	2.8857	2.9805
42	A_{185}–A_{192}	-	-	-	1.6131	1.6369	1.6198
43	A_{193}–A_{196}	-	-	-	0.1000	0.1000	0.1000
44	A_{197}–A_{198}	-	-	-	0.1004	0.1000	0.1000
45	A_{199}–A_{202}	-	-	-	1.3295	1.3971	1.3919
46	A_{203}–A_{210}	-	-	-	1.6328	1.6605	1.6465
47	A_{211}–A_{214}	-	-	-	1.3035	1.4017	1.3918
48	A_{215}–A_{216}	-	-	-	0.4610	0.4682	0.4739
Weight (lb)		1320.43	1322.76	1321.65	4990.73	5002.31	4992.61

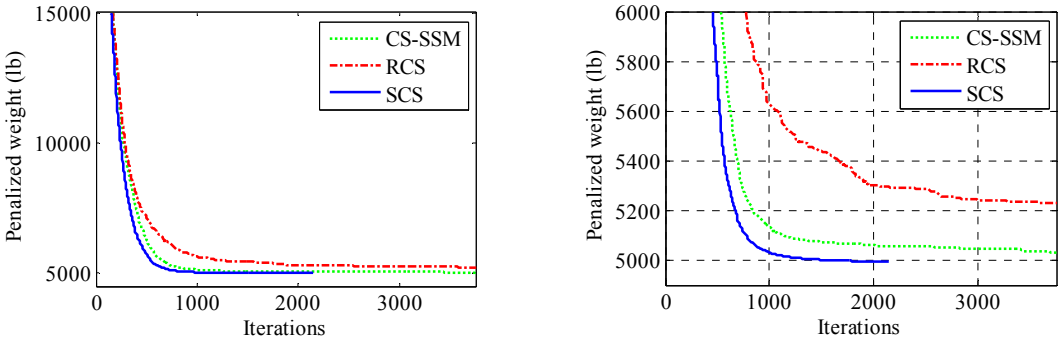

Fig. 4 Comparison of the convergence curves obtained for the 216-bar truss relative to the average optimization of 25 independent runs

value is limited to 30.0 in^2. The same load conditions are applied to the same nodes as the first problem in the last story. Remaining details are same as the first example.

Sub-spacing for 48 design variables are considered for each type of members in four adjacent stories leading to a set of subspaces using by SSM as: S = {[1 5 9 13] [17 21 25 29] [33 37 41 45] [2 6 10 14] [18 22 26 30] [34 38 42 46] [3 7 11 15] [19 23 27 31] [35 39 43 47] [4 8 12 16] [20 24 28 32] [36 40 44 48]}. The maximum number of truss analyses equal to 30000 is considered as the stopping criteria which results in 2143 and 3750 number of algorithm iterations for the first option and the last two options, respectively. The convergence history of the average optimization results achieved for 25 independent runs based on the SCS, CS-SSM and RCS are compared in Fig. 4. As it is clear the CS-SSM diagram again is consistent with the SCS in the global search phase and traces the SCS in the local search phase with minor loss of accuracy. Table 6 represents the statistical results of 25 independent runs for three options with 2143 iterations. SSM enables CS to

Table 6 Performance comparison of the SCS, RCS and CS-SSM for the 216-bar truss in 25 runs

Option	Weight (lb)			
	Best	Average	Worst	SD
SCS	4990.73	4994.73	5002.94	2.86
RCS	5002.31 (0.23%)	5491.43 (9.94%)	6737.88 (34.68%)	560.47
CS-SSM	4992.61 (0.04%)	5057.19 (1.25%)	5417.47 (8.29%)	127.65

reach the designs averagely 1.25% heavier than ones obtained via the standard CS with 42% saving in computational cost. Comparing to the previous test cases, it is clear that the SSM effectively refines the performance of the RCS with increasing order of the complexity of problem. Also it is clear from Fig. 2 that the RCS cannot trace the converging of SCS. Table 5 lists the best obtained designs by three options considering 2143 number of iterations as the stopping criteria.

4. Conclusions

This paper proposes a strategy named Subspace Search Mechanism (SSM) for population based metaheuristic algorithms dealing with structural optimization problems, which enables optimizers to work with less number of population in exchange for some loss of accuracy. SSM is inspired by this idea that the complexity of structural optimization problems can be partially due to the presence of high-dimensional design variables. SSM approach aims to dimensional reduction of the problem. Design variables of a *ng*-dimentional optimization problem are categorized to predefined groups (subspaces). SSM focuses on the multiple employment of the metaheuristic at hand to each subspace. Optimization algorithm updates the design variables for each subspace independently. Updating rules require the candidate designs evaluation. Each candidate design is the assemblage of responsible set of design variables for subspaces specified based on some criteria of interest.

SSM is utilized in conjunction with the Cuckoo Search algorithm for truss sizing optimization problem (three small, moderate and large space trusses are considered) as a common global constrained structural optimization problem. CS is successfully applied to structural optimization problems of the literature. Optimization results indicate that SSM enables the CS to work with less number of population (42%), as a result reducing the time of convergence, in exchange to some reduction of accuracy (1.5%). It is shown that the loss of accuracy reduces with the increase of the order of complexity. This suggests its applicability to other algorithms and other complex finite element-based engineering design problems. SSM is easy to implement since it needs considering only a sub-spacing. In solving continuous sizing optimization of trusses, our proposed mechanism is an effective tool.

Acknowledgments

The first author is grateful to Iran National Science Foundation for the support.

References

Adeli, H. and Cheng, N. (1994), "Augmented Lagrangian genetic algorithm for structural optimization", *J. Aerosp. Eng.*, **7**(1), 104-118.

Camp, C.V. (2007), "Design of space trusses using big bang-big crunch optimization", *J. Struct. Eng.*, **133**(7), 999-1008.

Camp, C.V. and Bichon, B.J. (2004), "Design of space trusses using ant colony optimization", *J. Struct. Eng.*, **130**(5), 741-751.

Davarynejad, M., Vrancken, J., van-den-Berg, J. and Coello-Coello, C.A. (2012), "A fitness granulation approach for large-scale structural design optimization", In: *Variants of Evolutionary Algorithms for Real-World Applications*, (Raymond Chiong, Thomas Weise, and Zbigniew Michalewicz Eds.), Springer-Verlag, pp. 245-280.

Degertekin, S.O. (2012), "Improved harmony search algorithms for sizing optimization of truss structures", *Comput. Struct.*, **92-93**, 229-241.

Degertekin, S.O. (2013), "Sizing truss structures using teaching-learning-based optimization", *Comput. Struct.*, **119**, 177-188.

Gandomi, A.H., Yang, X.S. and Alavi, A.H. (2013), "Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems", *Eng. Comput.*, **29**(1), 17-35.

Hasançebi, O., Çarbaş, S., Doğan, E., Erdal, F. and Saka, M.P. (2009), "Performance evaluation of metaheuristic search techniques in the optimum design of real size pin jointed structures", *Comput. Struct.*, **87**(5-6), 284-302.

Kaveh, A. and Bakhshpoori, T. (2013a), "Optimum design of space trusses using cuckoo search algorithm with Lévy flights", *IJST, Trans. Civil. Eng.*, **37**(C1), 1-15.

Kaveh, A. and Bakhshpoori, T. (2013b), "Optimum design of steel frames using cuckoo search algorithm with Lévy flights", *Struct. Design. Tall. Spec. Build.*, **22**(13), 1023-1036.

Kaveh, A. and Mahdavi, V.R. (2014), "Colliding Bodies Optimization method for optimum design of truss structures with continuous variables", *Adv. Eng. Softw.*, **70**, 1-12.

Kaveh, A. and Talatahari, S. (2009) "Particle swarm optimizer, ant colony strategy and harmony search scheme hybridized for optimization of truss structures", *Comput. Struct.*, **87**(5-6), 267-283.

Kaveh, A. and Talatahari, S. (2010a), "Optimal design of skeletal structures via the charged system search algorithm", *Struct. Multidiscip. Optim.*, **41**(6), 893-911.

Kaveh, A. and Talatahari, S. (2010b), "Optimum design of skeletal structures using imperialist competitive algorithm", *Comput. Struct.*, **88**(21-22), 1220-1229.

Kaveh, A., Bakhshpoori, T. and Afshary, E. (2011), "An optimization-based comparative study of double layer grids with two different configurations using cuckoo search algorithm", *Int. J. Optim. Civil. Eng.*, **1**, 507-520.

Kaveh, A., Bakhshpoori, T. and Ashoory, M. (2012), "An efficient optimization procedure based on cuckoo search algorithm for practical design of steel structures", *Int. J. Optim. Civil. Eng.*, **2**(1), 1-14.

Kaveh, A., Bakhshpoori, T. and Barkhori, M., (2014), "Optimum design of multi-span composite box girder bridges using cuckoo Search algorithm", *Steel. Compos. Struct., Int. J.*, **17**(5), 705-719.

Kaveh, A., Bakhshpoori, T. and Azimi, M. (2015), "Seismic optimal design of 3D steel frames using cuckoo search algorithm", *Struct. Design. Tall. Spec. Build.*, **24**(3), 210-227. DOI: 10.1002/tal.1162

Cheng, K.C.K. and Yap, R.H.C. (2008), "Search space reduction for constraint optimization problems", Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, 5202:635-639, Springer-Verlag.

Lamberti, L. (2008), "An efficient simulated annealing algorithm for design optimization of truss structures", *Comput. Struct.*, **86**(19-20), 1936-1953.

Liu, B., Wang, L., Jin, Y.H., Tang, F. and Huang, D.X. (2005), "Improved particle swarm optimization combined with chaos", *Chaos, Solitons & Fractals.*, **25**(5), 1261-1271.

Reddy, J.N. (1993), *Introduction to the Finite Element Method*, McGraw-Hill, New York, USA.

Saka, M.P. and Dogan, E. (2012), "Design optimization of moment resisting steel frames using a cuckoo salgorithm", *Proceedings of the Eleventh International Conference on Computational Structures Technology*, (B.H.V. Topping, Ed.), Civil-Comp Press, Stirlingshire, UK, Paper 71. DOI: 10.4203/ccp.99.71

Saka, M.P. and Geem, Z.W. (2013), "Mathematical and metaheuristic applications in design optimization of steel frame structures: an extensive review", *Math. Probl. Eng.*, DOI: 10.1155/2013/271031

Sarma, K.C. and Adeli, H. (2001), "Bilevel parallel genetic algorithms for optimization of large steel structures", *Comput. Aided. Civ. Infrastruct. Eng.*, **16**(5), 295-304.

Shayanfar, M.A., Ashoory, M., Bakhshpoori, T. and Farhadi, B. (2013), "Optimization of modal load pattern for pushover analysis of building structures", *Struct. Eng. Mech., Int. J.*, **47**(1), 119-129.

Sonmez, M. (2011), "Artificial bee colony algorithm for optimization of truss optimization", *Appl. Soft. Comput.*, **11**(2), 2406-2418.

Talatahari, S., Kheirollahi, M., Farahmandpour, C. and Gandomi, A.H. (2013), "A multi-stage particle swarm for optimum design of truss structures", *Neural. Comput. Applic.*, **23**(5), 1297-1309.

Talbi, E.G. (2009), *Metaheuristics: From Design to Implementation*, John Wiley & Sons, Hoboken, NJ, USA.

Tang, W., Tong, L. and Gu, Y. (2005), "Improved genetic algorithm for design optimization of truss structures with sizing, shape and topology variables", *Int. J. Numer. Meth. Eng*., **62**(13), 1737-1762.

Yang, X.S. (2008), *Nature-Inspired Metaheuristic Algorithms*, Luniver Press.

Yang, X.S. and Deb, S. (2009), "Engineering optimization by cuckoo search", *Int. J. Math. Model. Num. Optim.*, **1**, 330-343.

BU