Structure and Infrastructure Engineering, Vol. 3, No. 1, March 2007, 53–60

Taylor & Francis
Taylor & Francis Group

A modified shuffled frog-leaping optimization algorithm: applications to project management

EMAD ELBELTAGI†*, TAREK HEGAZY‡ and DONALD GRIERSON‡

†Assistant Professor of Construction Management, Structural Engineering Department, Faculty of Engineering, Mansoura University, Mansoura 35516, Egypt
‡Department of Civil Engineering, University of Waterloo, Waterloo, Ontario, N2L 3G1 Canada

(*Received 28 January 2005; accepted in final form 1 May 2005*)

Evolutionary algorithms, such as shuffled frog-leaping, are stochastic search methods that mimic natural biological evolution and/or the social behavior of species. Such algorithms have been developed to arrive at near-optimum solutions to complex and large-scale optimization problems which cannot be solved by gradient-based mathematical programming techniques. The shuffled frog-leaping algorithm draws its formulation from two other search techniques: the local search of the 'particle swarm optimization' technique; and the competitiveness mixing of information of the 'shuffled complex evolution' technique. A brief description of the original algorithm is presented along with a pseudocode and flowchart to facilitate its implementation. This paper then introduces a new search-acceleration parameter into the formulation of the original shuffled frog-leaping algorithm to create a modified form of the algorithm. A number of simulations are carried out for two continuous optimization problems (known as benchmark test problems) and two discrete optimization problems (large scale problems in the project management domain) to illustrate the positive impact of this new parameter on the performance of the shuffled frog-leaping algorithm. A range of 'best' usable values for the search-acceleration parameter is identified and discussed.

Keywords: Evolutionary algorithms; Shuffled frog leaping; Shuffled complex evolution; Particle swarm; Optimization

1. Introduction

The optimization of systems and processes is very important to the efficiency and economics of many science and engineering domains. Optimization problems are solved by using rigorous or approximate mathematical search techniques. Rigorous approaches have employed linear programming, integer programming, dynamic programming or branch-and-bound techniques to arrive at the optimum solution for moderate-size problems. However, optimizing real-life problems of the scale often encountered in engineering practice is much more challenging because of the huge and complex solution space. Finding exact solutions to these problems turns out to be *NP*-hard. This kind of complex problem requires an exponential amount of computing power and time, as the number of decision variables increases (Lovbjerg 2002). To overcome these problems, researchers have proposed approximate evolutionary-based algorithms as a means to search for near-optimum solutions.

Evolutionary algorithms (EAs) are stochastic search methods that mimic the metaphor of natural biological evolution and/or the social behavior of species. The optimized behavior of such species is guided by learning, adaptation and evolution (Lovbjerg 2002). Researchers have developed computational systems that mimic the

*Corresponding author. Email: eelbelta@mans.edu.eg

Structure and Infrastructure Engineering
ISSN 1573-2479 print/ISSN 1744-8980 online © 2007 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/15732470500254535

efficient behavior of species such as ants, bees, birds and frogs, as a means to seek faster and more robust solutions to complex optimization problems. The first evolutionary-based technique introduced in the literature was the genetic algorithm (Holland 1975). In an attempt to reduce processing time and improve the quality of solutions, particularly to avoid local optima, various genetic algorithm improvements and other EAs have been proposed during the past 10 years, with the latest and perhaps most promising technique being the shuffled frog-leaping (SFL) algorithm (Eusuff and Lansey 2003, Elbeltagi *et al*. 2005).

This paper first presents the SFL algorithm. Then, a new search–acceleration parameter is added to the original formulation to create a modified shuffled frog-leaping (MSFL) algorithm. A parametric study is conducted to determine the best range of values for this new parameter. To examine the efficacy of the acceleration parameter on solution quality and convergence speed of the MSFL algorithm, various example problems are considered. The examples include two well-known benchmark test problems of continuous optimization, in addition to two discrete optimization problems (one for optimizing time–cost trade-off decisions for construction projects, and the other for optimizing repair decisions of large bridge–deck infrastructure networks).

2. Shuffled frog-leaping algorithm

The shuffled frog-leaping algorithm is a memetic meta-heuristic that is designed to seek a global optimal solution by performing a heuristic search. It is based on the evolution of memes carried by individuals and a global exchange of information among the population (Eusuff and Lansey 2003). In essence, it combines the benefits of the local search tool of the particle swarm optimization (Kennedy and Eberhart 1995), and the idea of mixing information from parallel local searches to move toward a global solution (Duan *et al*. 1993). The SFL algorithm has been tested on several combinatorial problems and found to be efficient in finding global solutions (Eusuff and Lansey 2003).

The SFL algorithm involves a population of possible solutions defined by a set of frogs (i.e. solutions) that is partitioned into subsets referred to as memeplexes. The different memeplexes are considered as different cultures of frogs, each performing a local search. Within each memeplex, the individual frogs hold ideas, that can be influenced by the ideas of other frogs, and evolve through a process of memetic evolution. After a number of memetic evolution steps, ideas are passed among memeplexes in a shuffling process (Liong and Atiquzzaman 2004). The local search and the shuffling processes continue until convergence criteria are satisfied (Eusuff and Lansey 2003).

The SFL algorithm is described by the pseudocode in Appendix 1 and the corresponding flowchart in figure 1. First, an initial population of 'P' frogs is created randomly. For S-dimensional problems, each frog i is represented by S variables as $X_i = (x_{i1}, x_{i2}, \ldots\ldots, x_{iS})$. The frogs are sorted in a descending order according to their fitness. Then, the entire population is divided into m memeplexes, each containing n frogs (i.e. $P = m \times n$). In this process, the first frog goes to the first memeplex, the second frog goes to the second memeplex, frog m goes to the mth memeplex, and frog $m+1$ goes to the first memeplex, and so on.

Within each memeplex (figure 1b), the frogs with the best and the worst fitness are identified as X_b and X_w, respectively. Also, the frog with the global best fitness is identified as X_g. Then, an evolution process is applied to improve only the frog with the worst fitness (i.e. not all frogs) in each cycle. Accordingly, the position of the frog with the worst fitness is adjusted as follows:

$$\text{Change in frog position } (D_i) = rand() \cdot (X_b - X_w) \quad (1)$$

$$\text{New position } X_w = \text{current position } X_w + D_i; \quad (D_{max} \geq D_i \geq -D_{max}) \quad (2)$$

where $rand()$ is a random number between 0 and 1; and D_{max} is the maximum allowed change in a frog's position. If this process produces a better frog (solution), it replaces the worst frog. Otherwise, the calculations in equations (1) and (2) are repeated with respect to the global best frog (i.e. X_g replaces X_b). If no improvement becomes possible in this latter case, then a new solution is randomly generated to replace the worst frog with another frog having any arbitrary fitness (as shown in figure 1b). The calculations then continue for a specific number of evolutionary iterations within each memeplex (Eusuff and Lansey 2003). The main parameters of the SFL algorithm are: number of frogs P, number of memeplexes, and number of evolutionary iterations for each memeplex before shuffling.

3. A modified shuffled frog-leaping algorithm

In the SFL algorithm, each memeplex is allowed to evolve independently to locally search at different regions of the solution space. In addition, shuffling all the memeplexes and re-dividing them again into a new set of memeplexes results in a global search through changing the information between memeplexes. As such, the SFL algorithm attempts to balance between a wide search of the solution space and a deep search of promising locations that are close to a local optimum.

As expressed by equation (1), each individual frog (solution) in a memeplex is trying to change its position towards the best frog within the memeplex or the overall best frog. As shown in this equation, when the difference in position between the worst frog X_w (i.e. the frog under evolution) and the best frogs (X_b or X_g) becomes small, the

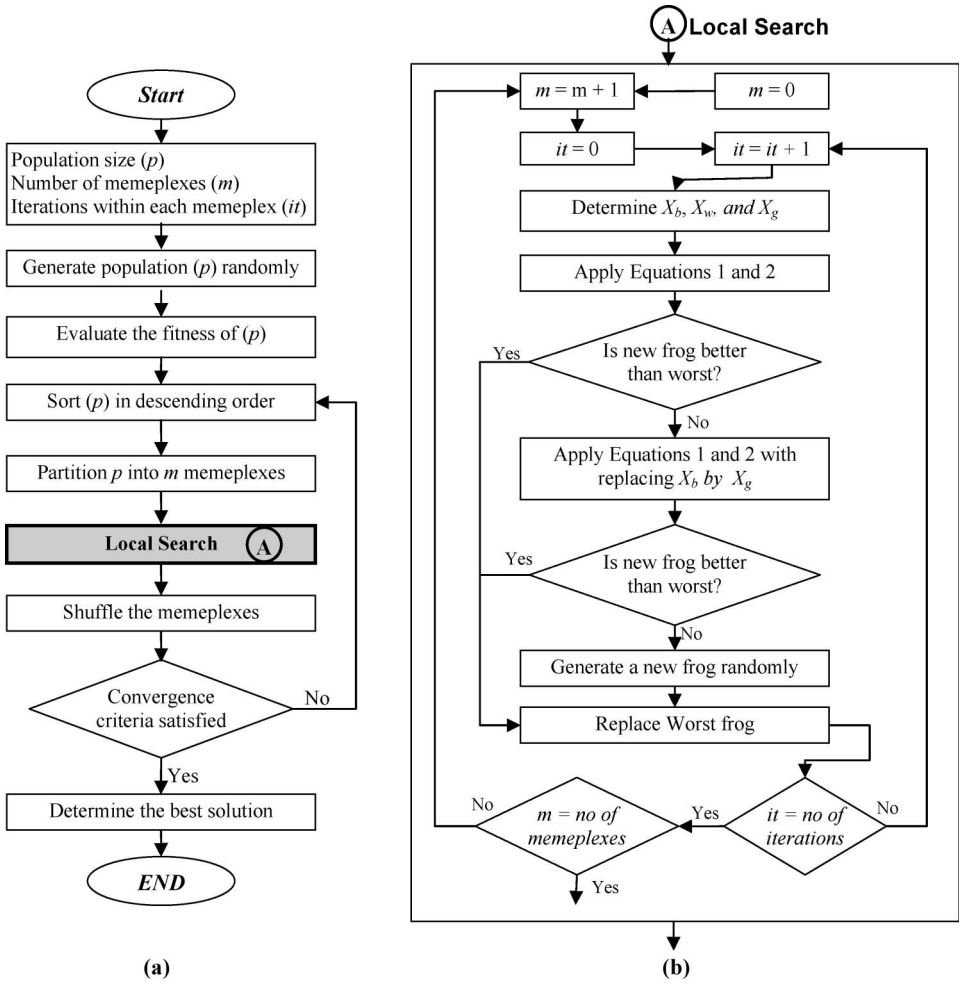

Figure 1. Flowchart of the shuffled frog-leaping algorithm.

change in frog X_w's position will be very small, and thus it might stagnate at a local optimum and lead to premature convergence. To overcome such an occurrence, this study proposes that the right-hand side of equation (1) be multiplied by a factor C called the 'search–acceleration factor', as follows:

$$\text{Change in frog position } (D_i) = rand() \cdot C \cdot (X_b - X_w) \quad (3)$$

Assigning a large value to the factor C at the beginning of the evolution process will accelerate the global search by allowing for a bigger change in the frog's position and accordingly will widen the global search area. Then, as the evolution process continues and a promising location is identified, the search–acceleration factor, C, will focus the process on a deeper local search as it will allow the frogs to change its positions. The search–acceleration factor, which can be a positive constant value, linear, or nonlinear function of time, provides the means to balance between global and local search.

4. Experiments and discussions

In order to quantify the influence of the acceleration factor C on the performance of the shuffled frog-leaping algorithm, two common benchmark problems with known global optimum values are first investigated in the following. Then, a construction management (time–cost trade-off) problem is considered. To facilitate the experimentation, the two benchmark problems were coded using the Visual Basic (VB) programming language, while the VBA code of MS Project and MS Excel were used for the construction management problems. All experiments were conducted on a 1.8 GHz AMD Laptop machine.

4.1 *Benchmark test problems for continuous optimization*

Two well-known objective functions often used to test optimization algorithms are the so-called *F8* and *F10* functions described in the following.

F8 function (Griewank's function): This objective function is a scalable, non-linear and non-separable function that may have any number N of variables x_i, i.e.

$$f(x_{i|i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} (\cos(x_i/\sqrt{i})) \qquad (4)$$

The summation term of equation (4) defines a parabolic shape, over which the cosine function in the product term creates waves. These waves create local optima in the solution space (Whitley *et al*. 1995). The value of each of the N variables is constrained to a range of -512 to 511. The global optimum (minimum) of the *F8* function is known to be zero when all N variables are equal to zero (Whitley *et al*. 1995).

F10 function: This objective function is non-linear, non-separable and involves two variables, x and y, i.e.

$$F10\ (x, y) = (x^2 + y^2)^{0.25}[\sin^2(50\ (x^2 + y^2)^{0.1}) + 1] \qquad (5)$$

An expanded *EF10* function may be created by scaling equation (5) to any number N of variables as follows (Whitley *et al*. 1995):

$$EF10\ (x_{i|i=1,N}) = \sum_{i=1}^{N-1} (x_i^2 + x_{i+1}^2)^{0.25}[\sin^2(50\ (x_i^2 + x_{i+1}^2)^{0.1}) + 1] \qquad (6)$$

Similar to the *F8* function, the global optimum of the *EF10* function (equation (6)) is known to be zero when all N variables equal zero, and the variable values ranging from -100 to 100 (Whitley *et al*. 1995).

Benchmark results and discussion: For the purpose of comparison between the SFL and MSFL algorithms, and to study the effect of the acceleration factor C, all the experiments used the same basic parameter settings: a population size of 200, a set of 20 memeplexes and 10 evolutionary iterations within each memeplex were used. These values were found suitable to produce good solutions in terms of the processing time and the quality of the solution as reported in the literature (Elbeltagi *et al*. 2005). In all experiments, the solution process stopped when one of the two following termination criteria is satisfied: (1) the objective function reaches a target value of 0.05 (considered as the optimum); or (2) the objective function does not improve in 10 successive shuffling iterations. The *F8* function was solved using 10 and 50 variables, while the *EF10* test function (which significantly increases in complexity as the number of variables increases) was solved using 10 and 20 variables.

Different values for the acceleration factor C were used for the experimentation, including $C = 1.0$ to account for results produced by the original SFL algorithm. For each selected C value, 20 runs of the MSFL algorithm were performed and the values of two parameters were recorded: (1) the percentage of runs that reached the optimum solution (i.e. an indicator of success); and (2) the average number of iterations required to find the global optimum (an indicator of convergence speed). The results of solving the two benchmark problems using different search–acceleration factors C are summarized in tables 1 and 2, and figure 2. In tables 1 and 2, the empty cells (zero value or dash) indicate that the SFL algorithm failed to find the global optimum within 10 successive shuffling iterations in all 20 runs. The numbers of iterations required to find the

Table 1. Percentage of success for benchmark test problems.

	% Success for various C values													
Test function	0.5	0.7	0.9	1.0	1.1	1.3	1.5	1.7	1.9	2.1	2.3	2.5	2.7	2.9
F8 – 10 variables	0	15	50	50	70	80	75	60	50	40	25	25	20	0
F8 – 50 variables	5	50	70	80	80	90	100	100	100	100	100	90	80	0
EF10 – 10 variables	0	15	50	80	80	90	100	100	100	95	90	90	50	0
EF10 – 20 variables	0	0	0	20	20	30	40	90	90	80	20	0	0	0

Table 2. Number of iterations to find global optimum for benchmark problems.

	No. of iterations for various C values													
Test function	0.5	0.7	0.9	1.0	1.1	1.3	1.5	1.7	1.9	2.1	2.3	2.5	2.7	2.9
F8 – 10 variables	–	325	165	158	151	29	86	113	152	177	244	404	1951	–
F8 – 50 variables	1678	1610	1358	1343	1278	891	96	79	120	216	456	935	2203	–
EF10 – 10 variables	–	635	590	590	580	70	50	68	75	146	263	372	1240	–
EF10 – 20 variables	–	–	–	612	543	442	407	265	221	244	704	1074	–	–

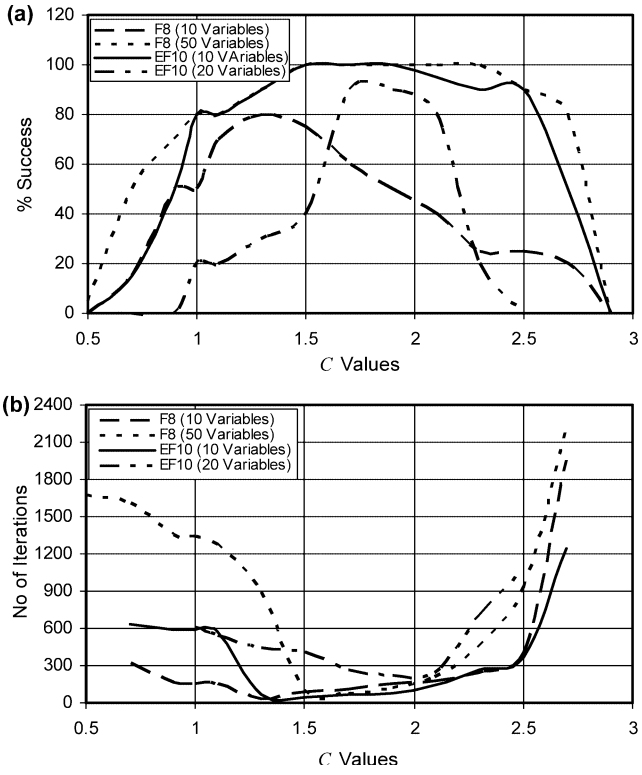

Figure 2. Relationship between C values and success rate and number of iterations.

global optimum for the different experiments are shown table 2.

As shown in figure 2a, the highest success rate was obtained when C varies from 1.1 to 2.3. From figure 2b, it is interesting to note that the MSFL was able to reach the optimum solution using the least number of iterations (i.e. processing time) when C varies from 1.3 to 2.1. An average value of $C = 1.6$ was able to give the highest success rate and the lowest number of iterations in all test problems (figures 2a and 2b).

It is noted that the search is mostly local when the C value is smaller than unity ($C < 1.0$) and thus, as shown in table 1, the success rate for finding the global optimum decreases. Also, for larger C values ($C > 2.5$), the frogs make large leaps in the search space without finding a promising location. Accordingly, for the benchmark problems studied, a range of acceleration values between 1.3 and 2.1 ($1.3 < C < 2.1$) provide the SFL algorithm with the best chance to find the global optimum while taking the minimum number of iterations.

For any optimization search, it is generally a good idea for the algorithm to explore the whole search space initially to find a good seed, and then have the algorithm intensely search the local area around the seed (Shi and Eberhart 1998). At the beginning of the evolution process, the frogs are scattered and small C values (greater than 1.0) accelerate the global search. Then, as the solution improves, a larger C value is necessary to maintain the evolution process from stagnating (i.e. similar to the effect of the mutation operator in genetic algorithms). Using this approach, a time-variant linear increase from 1.0 to 2.5 for the value of C was implemented as the number of iterations increased for the benchmark test problems. When solving the *F8* test function with 10 variables, the solution quality improved and the success rate changed from 80% to 90%. Also, the success rate of the *EF10* function with 20 variables was improved from 90% to 100%.

4.2 *Time–cost trade-off problem*

In order to examine the applicability of the MSFL algorithm in the civil engineering domain, the algorithm is here applied to a discrete optimization problem concerning construction time–cost tradeoff (TCT) management. The problem relates to an 18-activity construction project that was previously described in Feng *et al.* (1997). The activities, their predecessors and durations are presented in table 3, along with five optional methods of construction that vary from cheap and slow (option 5) to fast and expensive (option1). The 18 activities were input to a commercial project management software system (Microsoft Project) with activity durations being set to least costs and longest durations (option 5). The total direct cost of the project in this case is \$99,740 (sum of all activities' costs for option 5), with the project duration being 169 days. An indirect cost of \$500/day is then added to obtain a total project cost of \$184,240.

The initial schedule exceeds a desired deadline of 110 days. It is required to search for the optimum set of construction options that meet the deadline for minimum total cost. In this problem, the decision variables are the methods of construction for the different activities (each activity has five discrete methods, 1 to 5, with associated durations and costs). The objective is to minimize the total project cost (direct and indirect) as follows:

$$Min\left(T \cdot I + \sum_{i=1}^{n} Direct\ \ Cost_{ij}\right) \tag{7}$$

where n = number of activities; $Direct\ Cost_{ij}$ = direct cost of activity i when using its jth method of construction; T = total project duration; and I = daily indirect cost. The SFL evolutionary process selects an index to one of the five construction options to set the duration and cost for each activity. The project's total cost (objective function) and duration changes accordingly. The SFL process then continues its evolutionary cycles as it seeks to optimize (minimize) the objective function. To facilitate the optimization using the SFL algorithm within the Microsoft

Table 3. Data for the time–cost trade-off test problem.

Activity No.	Preceding activities	Method 1		Method 2		Method 3		Method 4		Method 5	
		Duration (days)	Cost ($)	Duration (days)	Cost ($)	Duration (days)	Cost ($)	Duration (days)	Cost ($)	Duration (days)	Cost ($)
1	–	14	2 400	15	2 150	16	1 900	21	1 500	24	1 200
2	–	15	3 000	18	2 400	20	1 800	23	1 500	25	1 000
3	–	15	4 500	22	4 000	33	3 200	–	–	–	–
4	–	12	45 000	16	35 000	20	30 000	–	–	–	–
5	1	22	20 000	24	17 500	28	15 000	30	10 000	–	–
6	1	14	40 000	18	32 000	24	18 000	–	–	–	–
7	5	9	30 000	15	24 000	18	22 000	–	–	–	–
8	6	14	220	15	215	16	200	21	208	24	120
9	6	15	300	18	240	20	180	23	150	25	100
10	2, 6	15	450	22	400	33	320	–	–	–	–
11	7, 8	12	450	16	350	20	300	–	–	–	–
12	5, 9, 10	22	2 000	24	1 750	28	1 500	30	1 000	–	–
13	3	14	4 000	18	3 200	24	1 800	–	–	–	–
14	4, 10	9	3 000	15	2 400	18	2 200	–	–	–	–
15	12	12	4 500	16	3 500	–	–	–	–	–	–
16	13, 14	20	3 000	22	2 000	24	1 750	28	1 500	30	1 000
17	11, 14, 15	14	4 000	18	3 200	24	1 800	–	–	–	–
18	16, 17	9	3 000	15	2 400	18	2 200	–	–	–	–

Project software, a macro program written using the Visual Basic language (VBA) inherent to the software was used to capture and manipulate the data in table 3. The construction management problem was then ready for experimentation and comparative analysis of the results found by the original and modified forms of the SFL algorithm. In this test problem, a search–acceleration factor of 1.6 was used (based on the results reported in the previous section).

As shown in table 4, the results found when using the modified SFL algorithm are significantly improved over those for the original SFL algorithm, with a success rate of 100% (i.e., the MSFL was able to reach the optimum solution in all 20 trial runs) compared to 0% for the original SFL algorithm. The project cost reached its minimum value of $161, 270. Also, the average processing time was greatly improved from 15 seconds for the original SFL algorithm to 8 seconds for the MSFL algorithm.

The results in table 4 demonstrate the effectiveness and efficiency of the MSFL algorithm for solving discrete optimization problems, which are common in the civil engineering domain. Further experiments were also conducted on larger projects to examine the performance of the SFL algorithm on large scale projects. Projects with 90 and 180 activities were constructed by copying the 18-activity project several times on Microsoft Project as subprojects. Every sub-project was given a finish-to-start relationship with the previous one. Accordingly, the overall project duration becomes multiples of the original project duration (169 days using all normal durations). In all experiments, deadline duration was used as 110 days times the numbers of subprojects (110×5 and 110×10 days respectively) and indirect cost of $500/day was used. The MSFL algorithm was able to reach the optimum solution or near optimum in most experiments; e.g. in the case of the 90-activity project, a 550-day project duration was achieved with a minimum total cost of $810,550, for a recorded processing time of only 9 minutes. These results suggest potential for solving large-scale optimization problems using the MSFL algorithm.

Table 4. Results of the time–cost trade-off optimization.

Comparison criteria	Algorithm	
	SFL	MSFL
Minimum duration (days)	112	110
Average duration (days)	123	110
Minimum cost ($)	162 020	161 270
Average cost ($)	166 045	161 270
% Success Rate	0	100
Processing Time (second)	15	8

4.3 *Rehabilitation of bridge–deck infrastructure problem*

Discrete optimization is a computational tool often used in the project management domain, and a realistic large-scale discrete problem is here used to test the MSFL algorithm. The problem concerns the difficult decisions to be made by a municipality concerning a cost effective rehabilitation strategy for a network of bridge decks. In this problem, the

MSFL algorithm is applied to determine the optimum repair decision (i.e. 0 = do nothing; 1 = light repair, 2 = medium repair; and 3 = extensive repair) for each bridge–deck so that the total life cycle cost is minimized. A detailed description of this problem is presented in Hegazy *et al*. (2004).

The objective is to minimize the present value of the total life cycle cost (TLCC) of repairing all selected bridges throughout the planning horizon (5 years), while maintaining an acceptable condition for each bridge, as follows:

$$Min \quad TLCC = \sum_{t=1}^{T} \sum_{i=1}^{N} \frac{C_{ti}}{(1+r)^t} \tag{8}$$

where C_{ti} = repair cost of bridge–deck i at time t; r = discount rate; T = number of years; and N = number of bridges. The objective function is subjected to the following constraints: yearly LCC should be $\leqslant$ yearly budget limits; condition rating of individual facilities $\geqslant$ minimum acceptable level; and the overall condition rating for the whole network is $\geqslant$ pre-defined value. Figure 3 shows the formulation and the variables involved (Hegazy *et al*. 2004). Each bridge is arranged in a separate row and five columns are set to hold the values for the problem variables. These values represent indices to one of the four repair options (0, 1, 2 or 3). In this representation, the number of variables involved is $N \times T$.

To facilitate the optimization using the MSFL algorithm, the model was implemented on MS Excel software and a macro program written using the VBA. Then the bridge–deck rehabilitation problem (case of 50 bridges) was solved using both the original SFL and the MSFL with a search-acceleration factor of 2. For comparison purposes, the same problem was also solved using genetic algorithm (GA). The performance of each of the three methods was measured using two criteria: (1) percentage of success (i.e. how many trials out of 10 experiments were able to provide a solution without violating the constraints on both individual bridges and the whole network); and (2) best solution obtained (i.e. min TLCC).

The results of using the MSFL algorithm show significant improvement over those of the original SFL algorithm and the GA, with a success rate of 100% compared to 10% for the original SFL algorithm, and 60% for the GA. Also, the total cost reached its minimum value of \$33,146,667 using the MSFL algorithm, compared with \$44,866,000 using the GA.

5. Summary and concluding remarks

In this paper, the shuffled frog-leaping algorithm is presented along with a pseudocode and flow chart to facilitate its implementation. A new parameter called a search–acceleration factor, C, is introduced to the original formulation of the shuffled frog-leaping algorithm. This search–acceleration factor balances the global and local search by widening the global search at the beginning and then searching deeply around promising solutions. Experiments were performed on two well-known benchmark test problems (*F8* and *EF10*) to illustrate the impact of this factor on the performance of the SFL algorithm. Based on the experiments that were performed on the two

8	General Information		Age	cond	Repair Option					Condition					Repair cost				
9	No.	Desc.	Now		1	2	3	4	5	1	2	3	4	5	Year 1	Year 2	Year 3	Year 4	Year 5
10	1	Bridge1	19	4.50	2	0	1	0	1	6.46	6.43	7.39	7.35	8.31	200,000	0	100,000	0	100,000
11	2	Bridge2	6	7.08	0	0	0	0	0	6.95	6.82	6.71	6.60	6.50	0	0	0	0	0
12	3	Bridge3	14	5.25	0	1	1	0	1	5.20	6.15	7.09	7.03	7.98	0	150,000	150,000	0	150,000
13	4	Bridge4	13	6.25	0	0	0	0	0	6.19	6.13	6.08	6.03	5.98	0	0	0	0	0
14	5	Bridge5	24	5.00	1	1	0	0	0	5.94	6.89	6.85	6.81	6.77	250,000	250,000	0	0	0
15	6	Bridge6	15	6.00	1	0	1	0	1	6.94	6.88	7.82	7.77	8.72	140,000	0	140,000	0	140,000
16	7	Bridge7	29	4.00	1	0	1	0	0	4.98	4.85	5.67	5.47	5.25	1,250,000	0	1,250,000	0	0
17	8	Bridge8	17	6.18	1	0	0	0	0	7.12	7.08	7.04	7.00	6.96	800,000	0	0	0	0
18	9	Bridge9	24	4.50	1	1	0	0	0	5.45	6.40	6.36	6.32	6.29	145,000	145,000	0	0	0
19	10	Bridge10	3	8.42	0	0	0	0	0	8.13	7.82	7.50	7.36	7.23	0	0	0	0	0
49	40	Bridge40	12	7.00	1	0	0	0	0	7.92	7.84	7.77	7.71	7.64	500,000	0	0	0	0
50	41	Bridge41	18	6.30	1	0	0	0	0	7.25	7.21	7.17	7.13	7.10	500,000	0	0	0	0
51	42	Bridge42	8	6.30	0	0	0	0	0	6.19	6.09	6.00	5.91	5.85	0	0	0	0	0
52	43	Bridge43	31	4.50	1	0	0	0	0	5.33	5.14	4.93	4.73	4.53	500,000	0	0	0	0
53	44	Bridge44	17	5.50	0	1	1	0	0	5.46	6.42	7.37	7.33	7.29	0	500,000	500,000	0	0
54	45	Bridge45	13	6.60	0	0	0	0	0	6.54	6.48	6.42	6.37	6.32	0	0	0	0	0
55	46	Bridge46	15	6.00	0	1	0	0	0	5.95	6.89	6.84	6.79	6.75	0	500,000	0	0	0
56	47	Bridge47	4	8.00	0	1	0	0	0	7.70	8.34	8.19	8.04	7.90	0	500,000	0	0	0
57	48	Bridge48	24	5.00	0	1	0	0	0	4.95	5.91	5.87	5.84	5.81	0	500,000	0	0	0
58	49	Bridge49	29	4.00	1	0	0	0	1	4.98	4.85	4.70	4.53	5.31	500,000	0	0	0	500,000
59	50	Bridge50	27	6.00	0	0	0	0	0	5.97	5.94	5.91	5.76	5.58	0	0	0	0	0

Decision variables: Selected bridges and repair type

Yearly repair cost

Figure 3. Bridge–deck rehabilitation problem formulation.

benchmark test problems, it is concluded that the MSFL algorithm with an acceleration factor in the range of 1.3 to 2.1, on average, has the best chance of finding the global optimum with the least number of evolutionary iterations (i.e. less processing time). Also, another discrete optimization time–cost trade-off construction management problem was used to test the performance of the algorithm on a civil engineering problem. To verify the findings and to make sure that the results represent a fairly wide class of problems, the modified SFL algorithm was applied for a bridge–deck rehabilitation infrastructure problem and the corresponding results demonstrated significantly improved performance over both the SFL algorithm and a genetic algorithm.

References

Duan, Q.Y., Gupta, V.K. and Sorooshian, S., Shuffled complex evolution approach for effective and efficient global minimization. *J. Optimization Theory Appns*, 1993, **76**, 502–521.

Elbeltagi, E., Hegazy, T. and Grierson, D., Comparison among five evolutionary-based optimization algorithms. *J. Adv. Engng. Informatics*, 2005, **19**, 43–53.

Eusuff, M.M. and Lansey, K.E., Optimization of water distribution network design using the shuffled frog leaping algorithm. *J. Water Resour. Planning Mgmt*, 2003, **129**, 210–225.

Feng, C., Liu, L. and Burns, S., Using genetic algorithms to solve construction time–cost trade-off problems. *J. Comput. Civil Engng*, 1997, **11**, 184–189.

Hegazy, T., Elbeltagi, E. and Elbehairy, H., Bridge deck management system with integrated life cycle cost optimization. *Transportation Research Record: J. Transportation Research Board*, 2004, **No. 1866**, TRB, National Research Council, Washington, D.C., 44–50.

Holland, J., *Adaptation in Natural and Artificial Systems*, 1975 (University of Michigan Press: Ann Arbor, MI).

Kennedy, J. and Eberhart, R., Particle swarm optimization, in *Proceedings IEEE International Conference on Neural Networks*, IEEE Service Center, Piscataway, NJ, pp. 1942–1948, 1995.

Liong, S.-Y. and Atiquzzaman, Md., Optimal design of water distribution network using shuffled complex evolution. *J. Instn. Engrs*, 2004, **44**, 93–107.

Lovbjerg, M., Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality. MSc thesis, Aarhus University, Denmark, 2002.

Shi, Y. and Eberhart, R., A modified particle swarm optimizer, in *Proceedings of the IEEE International Conference on Evolutionary Computation*, IEEE Press, Piscataway, NJ, pp. 69–73, 1998.

Whitley, D., Beveridge, R., Graves, C. and Mathias, K., Test driving three 1995 genetic algorithms: new test functions and geometric matching. *J. Heuristics*, 1995, **1**, 77–104.

Appendix 1. Pseudocode for SFL algorithm

```
Begin;
  Generate random population of P solutions (individuals);
  For each individual i ϵ P: calculate fitness (i);
  Sort the whole population P in descending order of their
    fitness;
    Divide the population P into m memeplexes;
    For each memeplex;
        Determine the best and worst individuals;
        Improve the worst individual position using Eqs. 1
        and 2;
        Repeat for a specific number of iterations;
    End;
    Combine the evolved memeplexes;
    Sort the population P in descending order of their
    fitness;
  Check if termination = true;
End;
```