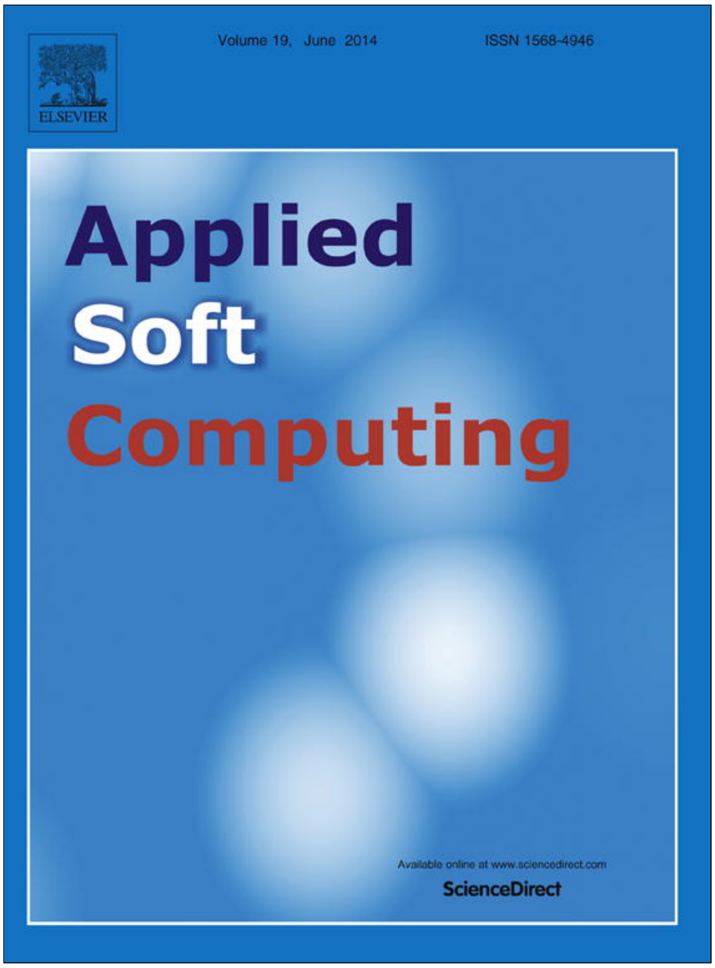
Volume 19, June 2014
ISSN 1568-4946
ELSEVIER
Applied
Soft
Computing
Available online at www.sciencedirect.com
ScienceDirect

Applied Soft Computing 19 (2014) 252–263

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Shuffled frog leaping algorithm and its application to 0/1 knapsack problem

Kaushik Kumar Bhattacharjee, S.P. Sarmah*

Department of Industrial Engineering and Management, Indian Institute of Technology, Kharagpur, WB 721302, India

ARTICLE INFO

Article history:
Received 9 February 2012
Received in revised form 5 November 2013
Accepted 9 February 2014
Available online 5 March 2014

Keywords:
Meta-heuristics
Discrete shuffled frog leaping algorithm
Knapsack problem
Genetic mutation

ABSTRACT

This paper proposes a modified discrete shuffled frog leaping algorithm (MDSFL) to solve 01 knapsack problems. The proposed algorithm includes two important operations: the local search of the 'particle swarm optimization' technique; and the competitiveness mixing of information of the 'shuffled complex evolution' technique. Different types of knapsack problem instances are generated to test the convergence property of MDSFLA and the result shows that it is very effective in solving small to medium sized knapsack problems. Further, computational experiments with a set of large-scale instances show that MDSFL can be an efficient alternative for solving tightly constrained 01 knapsack problems.

1. Introduction

The knapsack problem is one of the classical NP-hard optimization problem and the decision problem belongs to the class of NP-complete. It is thoroughly studied in the literature for last few decades. It offers many practical applications in vast field of different areas, such as project selection [1], resource distribution [2], network interdiction problem [3], investment decision-making [4] and so on. 01 knapsack problem is defined by: given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than a given limit and the total value is as large as possible. The most common formulation of the problem is the 01 knapsack problem, which restricts the number x_j of copies of each kind of item to zero or one.

$$\begin{aligned} &\textit{Maximize} \quad f(x_1, x_2, \ldots, x_n) = \sum_{j=1}^{n} c_j x_j \\ &\textit{Subject to} \quad g(x_1, x_2, \ldots, x_n) = \sum_{j=1}^{n} a_j x_j \le b, \\ &\qquad x_j \in \{0, 1\} \quad j = 1, 2, \ldots, n, \\ &\qquad c_j > 0, \quad a_j \ge 0, \quad b > 0. \end{aligned} \tag{1}$$

The binary decision variables x_j are used to indicate whether item j is included in the knapsack or not. It may be assumed that all profits and weights are positive, and that all weights are smaller than the capacity b.

In recent times, many heuristic and meta-heuristic algorithms have been employed to solve 01 knapsack problems: Zhao et al. [5] proposed genetic algorithm to solve 01 knapsack problem. Greedy strategy combining with traditional genetic algorithm proved to be much more effective to handle difficult instances. Lin [6] used genetic algorithm to solve knapsack problem with imprecise weight, and he investigated the possibility of using genetic algorithms in solving the fuzzy knapsack problem without defining membership functions for each imprecise weight coefficient. Liu and Liu [7] proposed a schema-guiding evolutionary algorithm (SGEA) to solve 01 knapsack problems. Wanga et al. [8] proposed quantum swarm evolutionary algorithm to solve 01 knapsack problems. Shi [9] modified the parameters of the ant colony optimization (ACO) model to adapt itself to 01 knapsack problems. The improved ACO has strong capability of escaping from the local optimum through artificial interference. Li and Li [10] proposed a binary particle swarm optimization based on multi-mutation strategy (MMBPSO) to solve knapsack problem. The MMBPSO can effectively escape from the local optima to avoid premature convergence due to the utilization of Multi-Mutation strategy. Zou et al. [11] proposed a novel global harmony search algorithm to solve 01 knapsack problems. They utilized position updating and discrete genetic mutation strategy to avoid the premature convergence.

Although many 01 knapsack problems have been solved successfully by these algorithms, but some new and more difficult 01

* Corresponding author. Tel.: +91 3222 283734.
E-mail addresses: bhattacharjee.kaushik@gmail.com (K.K. Bhattacharjee), spsarmah@iem.iitkgp.ernet.in, sp_sarmah@yahoo.com (S.P. Sarmah).

http://dx.doi.org/10.1016/j.asoc.2014.02.010
1568-4946/

knapsack problems hidden in the real world, so the research on this particular issue is still important. Many algorithms provide possible solutions for some 01 knapsack problems, but they may lose their efficiency on solving these difficult problems due to their own disadvantages and limitations. Most of these algorithm proposed recently are effective for solving 01 knapsack problem with very low dimension, but they may not be effective for 01 knapsack problems with high dimensional sizes.

The shuffled frog leaping algorithm (SFLA) is a meta-heuristic optimization method which is based on observing, imitating, and modeling the behavior of a group of frogs when searching for the location that has the maximum amount of available food [12]. SFLA, originally developed by Eusuff and Lansey in 2003, can be used to solve many complex optimization problems, which are nonlinear, non-differentiable, and multi-modal [13]. SFLA has been successfully applied to several engineering optimization problems such as water resource distribution [14], bridge deck repairs [15], job-shop scheduling arrangement [16], multi-mode resource-constrained project scheduling problem [17], unit commitment problem [18] and traveling salesman problem (TSP) [19]. The most distinguished benefit of SFLA is its fast convergence speed [20]. The SFLA combines the benefits of both the genetic-based memetic algorithm (MA) and the social behavior-based PSO algorithm [21].

2. Discrete shuffled frog leaping algorithm

In SFLA, the population consists of a set of frogs (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 defined number of memetic evolution steps, ideas are passed among memeplexes in a shuffling process [22]. The local search and the shuffling processes continue until defined convergence criteria are satisfied [14].

An initial population of P frogs is created randomly. For S-dimensional problems (S variables), a frog i is represented as $X_i = (x_{i1}, x_{i2}, \ldots, x_{iS})$. Afterwards, 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 back to the first memeplex, etc.

Within each memeplex, the frogs with the best and the worst fitnesses are identified as X_b and X_w, respectively. Also, the frog with the global best fitness is identified as X_g. Then, a process similar to PSO is applied to improve only the frog with the worst fitness (not all frogs) in each cycle. Accordingly, the position of the frog with the worst fitness is adjusted as follows:

$$D_i = Rand() \times (X_b - X_w), \tag{2}$$

where D_i is the change in ith frog position and new position is given by:

$$\begin{aligned} &X_w(new) = X_w + D_i, \\ &-D_{\max} \le D_i \le D_{\max}; \end{aligned} \tag{3}$$

where $Rand()$ is a random number($Rand() \sim U(0, 1)$); and D_{max} is the maximum allowed change in a frog's position. If this process produces a better solution, it replaces the worst frog. Otherwise, the calculations in Eqs. (2) and (3) are repeated but with respect to the global best frog (i.e. X_g replaces X_b). If no improvement possible in this case, then a new solution is randomly generated to replace that frog. The calculations then continue for a specific number of iterations [14].

For handling integer programming problems the discrete version of the SFLA is used, called discrete shuffled frog leaping algorithm (DSFLA). The worst frog within each memeplex is updated [12] according to

$$\begin{aligned} &D_i = \begin{cases} \min\{\mathrm{int}[Rand \times (X_b - X_w)], D_{\max}\} & \text{for a positive step,} \\ \max\{\mathrm{int}[Rand \times (X_b - X_w)], -D_{\max}\} & \text{for a negative step;} \end{cases} \\ &X_w(new) = X_w + D_i. \end{aligned} \tag{4}$$

Like SFLA, DSFLA also follows same steps to replace the worst frog. If Eq. (4) does not produce a better solution, then X_b is replaced by the global best frog i.e. X_g; and if in this case also we replace the worst frog by a new randomly generated solution, if Eq. (4) does not produce a better solution. Accordingly, the main parameters of DSFLA are: number of frogs P; number of memeplexes m; number of generation for each memeplex before shuffling n; number of shuffling iterations it; and maximum number of iterations $iMax$. The pseudocode for the DSFLA is given in Algorithm 1.

Algorithm 1. Pseudocode for a DSFLA procedure

```
Generate random population of P solutions (frogs)
for each individual i ∈ P do
   calculate fitness(i)
end for
Sort the population P in descending order of their fitness
Divide P into m memeplexes
for each memeplex do
   Determine the best and worst frogs
   Improve the worst frog position using Eq. (4)
   Repeat for a specific number of iterations
end for
Combine the evolved memeplexes
Sort the population P in descending order of their fitness
if termination = true then
   Return best solution
end if
```

3. Modified DSFLA for 01 knapsack

01 knapsack problem cannot be handled directly by SFLA or DSFLA because of its particular structure. For this reason original DSFLA is modified and applied to solve 01 knapsack problems as discussed below. Shuffled frog leaping algorithm has the most advantageous property of fast convergence speed. But at the same time it looses the searching capability of divergent field and sometimes trapped within a local optima. To make a balance between the convergent and divergent property we further modified DSFLA by hybridizing which include the genetic mutation property of divergent category. The modified discrete shuffled frog leaping algorithm (MDSFLA) is discussed in this section in full details.

3.1. Construction of individual frog

01 knapsack problem is an integer programming problem. There are two possible values for the decision variable x_j, zero or one. The individual frog is represented by a n-bit binary string, where n is the dimension of the problem. The initial population is created randomly to achieve sufficient diversification.

3.2. Process for discrete variables

In this paper we have used three kinds of discretization to solve the 01 knapsack problems.

(i) Method 1: the worst frog X_w of each memeplex is replaced according to

$$\begin{aligned} t &= X_w + D; \\ X_w(new) &= \begin{cases} 0 & \text{if} \quad t \le 0, \\ round(t) & \text{if} \quad 0 < t < 1, \\ 1 & \text{if} \quad t \ge 1. \end{cases} \end{aligned} \tag{5}$$

(ii) Method 2: D is transformed to the interval [0, 1] by using sigmoid function. The worst frog is replaced according to

$$\begin{aligned} t &= 1/(1+\exp(-D)); \\ u &\sim U(0,1) \\ X_w(new) &= \begin{cases} 0 & \text{if} \quad t \le u, \\ 1 & \text{if} \quad t > u. \end{cases} \end{aligned} \tag{6}$$

(iii) Method 3: the updating formula for the worst frog is given by

$$\begin{aligned} t &= 1/(1+\exp(-D)); \\ X_w(new) &= \begin{cases} 0 & \text{if} \quad t \le \alpha, \\ X_w & \text{if} \quad \alpha < t \le \frac{1}{2}(1+\alpha), \\ 1 & \text{if} \quad t \ge \frac{1}{2}(1+\alpha). \end{cases} \end{aligned} \tag{7}$$

The parameter α is called static probability.

3.3. Constrained optimization

Constrained optimization problems are much more difficult to solve compared to the unconstrained part. Due to the presence of constraint, the global best solution of unconstrained problem is different from the constrained one. In the later case, we have to find the optimal balance between the constraints and objective function value. For this reason here two types of conventional algorithms are described and tested: algorithms based on penalty functions and algorithms based on repair methods [23]. Three types of penalty functions are used: logarithmic penalty, linear penalty, and quadratic penalty and they are represented as follows.

$$f_1(\mathbf{x}) = \mathbf{p} \times \mathbf{x}^T - \log_2(1 + \rho(\mathbf{w}\mathbf{x}^T - b)), \tag{8}$$

$$f_2(\mathbf{x}) = \mathbf{p} \times \mathbf{x}^T - \rho(\mathbf{w} \times \mathbf{x}^T - b), \tag{9}$$

$$f_3(\mathbf{x}) = \mathbf{p} \times \mathbf{x}^T - (\rho(\mathbf{w} \times \mathbf{x}^T - b))^2, \tag{10}$$

where $\rho = \max_{i=1,\ldots,n} (p_i/w_i)$.

In algorithms based on repair methods, the profit $f(\mathbf{x})$ of each vector $\mathbf{x}$ is determined as

$$f(\mathbf{x}) = \mathbf{p} \times \mathbf{x}_1^T, \tag{11}$$

where $\mathbf{x}_1$ is a repaired vector of the original solution $\mathbf{x}$. Two types of repair algorithms considered here. The only difference is the selection procedure among them, which chooses an item for removal from the knapsack:

- *Rep*1 (random repair): The selection procedure selects a random element from the knapsack.
- *Rep*2 (greedy repair): All items in the knapsack are sorted in the decreasing order of their profit to weight ratios. The selection procedure always chooses the last item for deletion.

Here we use these repair methods as well as penalty functions to handle the feasibility of the solution.

Table 1
The dimension and parameters of ten test problems.

f	Dimension	Parameter(**w**,**p**,b)
f_1	10	**w** ={95, 4, 60, 32, 23, 72, 80, 62, 65, 46}; **p** = {55, 10, 47, 5, 4, 50, 8, 61, 85, 87}; b = 269.
f_2	20	**w** = {92, 4, 43, 83, 84, 68, 92, 82, 6, 44, 32, 18, 56, 83, 25, 96, 70, 48, 14, 58}; **p** = {44, 46, 90, 72, 91, 40, 75, 35, 8, 54, 78, 40, 77, 15, 61, 17, 75, 29, 75, 63}; b = 878.
f_3	4	**w** = {6, 5, 9, 7}; **p** = {9, 11, 13, 15}; b = 20.
f_4	4	**w** = {2, 4, 6, 7}; **p** = {6, 10, 12, 13}; b = 11.
f_5	15	**w** = {56.358531, 80.87405, 47.987304, 89.59624, 74.660482, 85.894345, 51.353496, 1.498459, 36.445204, 16.589862, 44.569231, 0.466933, 37.788018, 57.118442, 60.716575}; **p** = {0.125126, 19.330424, 58.500931, 35.029145, 82.284005, 17.41081, 71.050142, 30.399487, 9.140294, 14.731285, 98.852504, 11.908322, 0.89114, 53.166295, 60.176397}; b = 375.
f_6	10	**w** = {30, 25, 20, 18, 17, 11, 5, 2, 1, 1}; **p** = {20, 18, 17, 15, 15, 10, 5, 3, 1, 1}; b = 60.
f_7	7	**w** = {31, 10, 20, 19, 4, 3, 6}; **p** = {70, 20, 39, 37, 7, 5, 10}; b = 50.
f_8	23	**w** = {983, 982, 981, 980, 979, 978, 488, 976, 972, 486, 486, 972, 972, 485, 485, 969, 966, 483, 964, 963, 961, 958, 959}; **p** = {81,980, 979, 978, 977, 976, 487, 974, 970, 485, 485, 970, 970, 484, 484, 976, 974, 482, 962, 961, 959, 958, 857}; b = 10000.
f_9	5	**w** = {15, 20, 17, 8, 31}; **p** = {33, 24, 36, 37, 12}; b = 80.
f_{10}	20	**w** = {84, 83, 43, 4, 44, 6, 82, 92, 25, 83, 56, 18, 58, 14, 48, 70, 96, 32, 68, 92}; **p** = {91, 72, 90, 46, 55, 8, 35, 75, 61, 15, 77, 40, 63, 75, 29, 75, 17, 78, 40, 44}; b = 879.

3.4. Genetic mutation

DSFLA sometimes trapped in a local optima. To avoid this situation we utilize the genetic mutation. After shuffling the memeplexes we use the genetic mutation operator. To avoid the premature convergence of DSFLA mutation operator with a small probability is applied to modify the original population, which serves as the minute diversification in original solution procedure. This will allow the modified discrete shuffled frog leaping algorithm (MDSFLA) to search for other optimal points with new basis.

3.5. Termination condition

Maximum number of iterations (*iMax*) determines the termination condition for original SFLA. Here we choose another alternative criteria along with the *iMax*. If it is not possible to improve the best solution for a large number of steps (Δ), then the algorithm will terminate. And in the mean time total number of iterations must be less than the maximum number of iterations. The parameter Δ is problem dependent, and determined by hit and trial. Generally if we choose $\lceil \frac{iMax}{20} \rceil \le \Delta \le \lceil \frac{iMax}{10} \rceil$, then the performance of the algorithm is found to be optimal. For $iMax > 500$, we use both these criteria to optimize the algorithmic performance. The flowchart of MDSFLA is given in Fig. 1.

4. Experiments and computational results

In this section, the performance of DSFLA and MDSFLA extensively investigated by a large number of experimental studies. Thirty five 01 knapsack programming problem instances are considered to testify the validity of the MDSFLA. All computational experiments are conducted with Matlab 7.6.0 in Intel(R) Core(TM)2 Duo CPU E7400 @2.80 GHz with 4 GB of RAM. Standard ten test problems are studied here and detailed information about these problems are given in Table 1. These problems are studied by many authors to test the performance of the different algorithms presented in the literature.

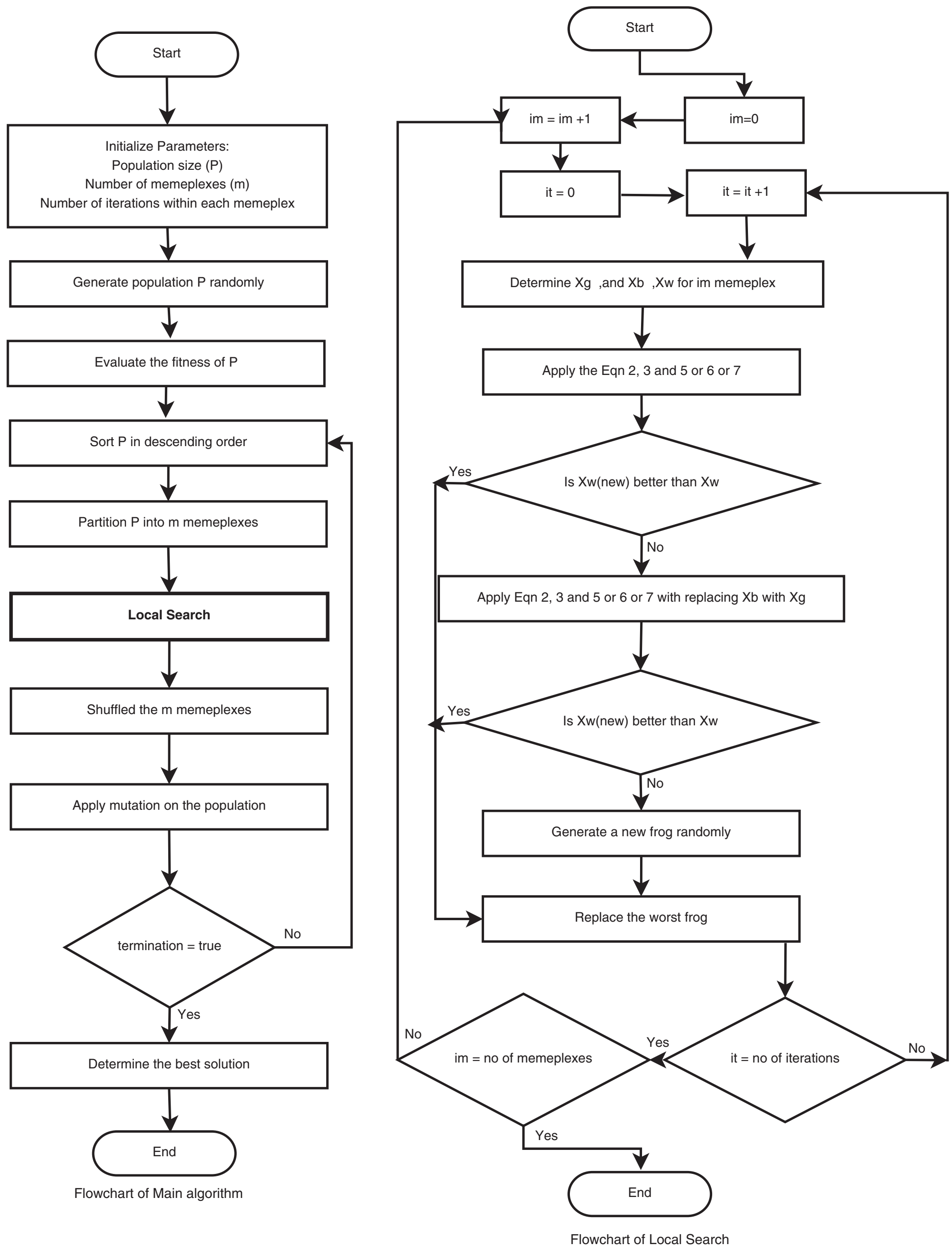

Fig. 1. Flowchart of MDSFLA.

Shi [9] introduced test problems 1 and 2 to test the performance of proposed improved ant colony algorithm. An and Fu [24] proposed a method called sequential combination tree algorithm to solve the test problem 3. You [25] used the test problem 4 for the performance analysis of greedy-policy-based algorithm. Yoshizawa and Hashimoto [26] used the information of search-space landscape to search the optimum of the test problem 5. Fayard and Plateau [27] employed a method to solve the test problem 6, and this method derives from the "shrinking boundary method". Zhao [28] proposed a method called nonlinear

Table 2
The detailed information of CPLEX solutions.

f	$\mathbf{x}^*$	$f(\mathbf{x}^*)$	$g(\mathbf{x}^*)$	b
f_1	{0,1,1,1,0,0,0,1,1,1}	295	269	269
f_2	{1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,1}	1024	871	878
f_3	{1,1,0,1}	35	18	20
f_4	{0,1,0,1}	23	11	11
f_5	{0,0,1,0,1,0,1,1,0,1,1,1,0,1,1}	481.0694	354.960784	375
f_6	{0,0,1,0,1,1,1,1,1,1}	52	57	60
f_7	{1,0,0,1,0,0,0}	107	50	50
f_8	{1,1,1,1,1,1,1,1,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0}	9767	9768	10000
f_9	{1,1,1,1,0}	130	60	80
f_{10}	{1,1,1,1,1,1,1,1,1,0,1,1,1,1,0,1,0,1,1,1}	1025	871	879

Table 3
Comparison between three discrete shuffled frog algorithms.

f	Criteria						f	Criteria					
	best	*worst*	*average*	*median*	*std*	*ATT*		*best*	*worst*	*average*	*median*	*std*	ATT
f_1	295	269	287.8	290	6.77	0.53	f_6	52	49	51.52	52	0.75	0.23
	295	253	287.24	288	8.64	0.5		52	49	51.62	52	0.73	0.17
	295	279	291.7	294	4.26	0.38		52	50	51.63	52	0.61	0.17
f_2	955	816	868	859	37.21	0.45	f_7	107	105	106.79	107	0.63	0.06
	958	815	868.77	864	38.15	0.46		107	105	106.72	107	0.70	0.08
	950	801	872.03	874.5	33.50	0.44		107	105	106.73	107	0.69	0.08
f_3	35	35	35	35	0	0.00	f_8	9755	9718	9736.13	9734	9.55	0.61
	35	35	35	35	0	0.00		9752	9712	9732.93	9734	9.32	0.59
	35	35	35	35	0	0.00		9759	9707	9733.47	9735	9.08	0.56
f_4	23	23	23	23	0	0.00	f_9	130	130	130	130	0	0.00
	23	23	23	23	0	0.00		130	130	130	130	0	0.00
	23	23	23	23	0	0.00		130	130	130	130	0	0.00
f_5	454.43	362.98	402.90	400.39	21.81	0.60	f_{10}	951	810	868.43	868	33.75	0.45
	466.34	356.76	406.41	410.21	27.11	0.58		952	822	874.07	872.5	33.54	0.46
	481.07	352.35	408.55	409.76	27.73	0.54		1010	811	879.9	870	44.51	0.45

dimensionality reduction to solve the test problem 7 and 8. Test problem 9 is from [29], in which the DNA algorithm is proposed to solve 01 knapsack problems. Test problem 10 is from literature [30], in which three algorithms are used to solve 01 knapsack problems. The detailed information of the ten test problems along with the best solution found by CPLEX V12.2.0 is given in the Table 2.

According to the structure of the test problems, mainly two different parameter sets are determined. These parameters are set based on the general guidelines given in the literature [12].

4.1. Effect of discretization

The effect of three methods of discretization discussed in earlier section, are given in Table 3.

For ten test problems best solution and worst solution among 30 independent runs are reported in Table 3. Also average, median and standard deviation (*std*) for all the solutions are given here along with average total time (ATT) to solve the problem. Maximum number of iterations is considered as *iMax* = 100, and population size is P = 200 along with m = 10 memeplexes. From Table 3, it is clear that, Method 3 is much more effective to find out best solution with respect to others. In most of the cases Method 3 performs better with respect to average, median, standard deviation and ATT (except for the function f_{10}, Method 2 performs better with respect to *median* and standard deviation (*std*)).

4.2. The effect of penalty functions and repair operators on the performance of the DSFLA

For handling the constrained part we used the penalty functions and repair operators. The effect of different penalty functions and repair operators is shown in Table 4. For performance analysis we use standard ten test problems given in Table 1.

Here in this experimental setup, we find out the best solution and worst solution for 30 independent experiments for each case and average, standard deviation and ATT also reported in the Table 4. In case of the first penalty function only two functions (f_2 and f_{10}) are solved successfully. For other cases we get infeasible solutions. Also in the case of second penalty function we get infeasible solutions for three functions (f_3, f_4 and f_8). Here maximum number of iterations is considered as *iMax* = 100, and population size is P = 200 along with m = 10 memeplexes for all these experiments. Repair method 2 outperform others with respect to all the performance criteria as reported in Table 4.

4.3. The effect of static probability on the performance of the DSFLA

Among ten test problems f_{10} is much more difficult to solve. So we test the performance criteria for different static probabilities with respect to the test function f_{10}. The effect of different *alpha* values for the function f_{10} is given in Fig. 2. Average profit for 30 individual runs corresponding to different α values are plotted. Corresponding to the results we may choose α = 0.4.

4.4. The effect of mutation on the performance of the MDSFLA

In this subsection, the effect of genetic mutation probability p_m on the performance of the MDSFLA is investigated. For the standard ten test problems above, population size of MDSFLA is set to P = 200 along with m = 10 memeplexes, and the number of iterations in each memeplex is set to it = 10. Total 30 independent experiments are carried out in each case, and Table 5 gives the average of maximum number of iterations required to solve the particular problem with different values of p_m. In all the cases we find the best solution for each test problem. So best, worst, average and standard deviation of objective function is same for each case. Range of p_m is taken as [0.01, 0.1], as beyond this range the performance of MDSFLA degraded gradually.

The best results were obtained when p_m = 0.06 for most of the cases (except for the function f_5). From Table 5, it is clear that mutation probability with $0.01 \leq p_m \leq 0.1$ can be effective to solve all

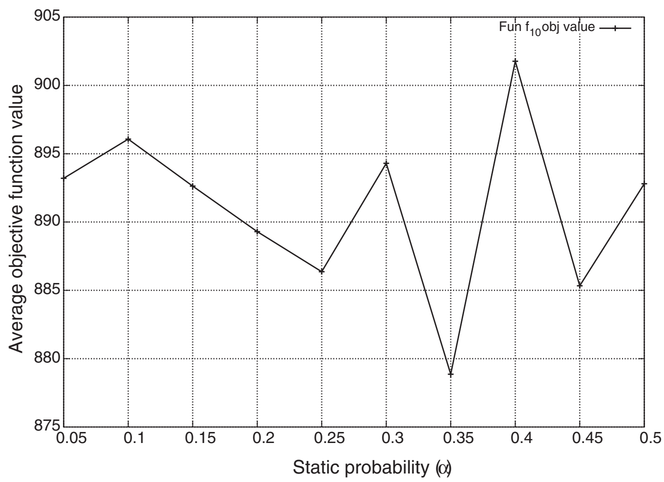

Fig. 2. Effect of static probability on the performance of DSFLA.

Table 4
Experimental results of knapsack problems with different penalty functions and repair operators.

f	Criteria	*Pen*1	*Pen*2	*Pen*3	*Rep*1	*Rep*2	f	Criteria	*Pen*1	*Pen*2	*Pen*3	*Rep*1	*Rep*2
f_1	*best*	–	295	295	295	295	f_6	*best*	–	52	52	52	52
	worst	–	280	274	279	293		*worst*	–	50	51	49	51
	average	–	290.35	290.93	290.83	294.17		*average*	–	51.48	51.6	51.7	51.97
	std	–	4.69	5.20	4.91	0.75		*std*	–	0.68	0.50	0.65	0.18
	ATT	–	0.47	0.42	0.30	0.28		ATT	–	0.26	0.23	0.12	0.02
f_2	*best*	984	1005	989	970	1005	f_7	*best*	–	107	107	107	107
	worst	780	800	809	788	811		*worst*	–	107	102	107	107
	average	885.32	869.76	877.2	885.03	889.27		*average*	–	107	106.34	107	107
	std	44.78	37.41	35.92	40.86	36.59		*std*	–	0	1.20	0	0
	ATT	0.45	0.45	0.45	0.45	0.40		ATT	–	0.00	0.16	0.01	0.00
f_3	*best*	–	–	35	35	35	f_8	*best*	–	–	9751	9755	9767
	worst	–	–	35	35	35		*worst*	–	–	9718	9731	9752
	average	–	–	35	35	35		*average*	–	–	9732.47	9742.97	9759.8
	std	–	–	0	0	0		*std*	–	–	7.41	6.34	4.60
	ATT	–	–	0.00	0.00	0.00		ATT	–	–	0.57	0.47	0.44
f_4	*best*	–	–	23	23	23	f_9	*best*	–	130	130	130	130
	worst	–	–	23	23	23		*worst*	–	130	130	130	130
	average	–	–	23	23	23		*average*	–	130	130	130	130
	std	–	–	0	0	0		*std*	–	0	0	0	0
	ATT	–	–	0.00	0.00	0.00		ATT	–	0.00	0	0.00	0.00
f_5	*best*	–	475.48	450.67	475.48	481.07	f_{10}	*best*	970	960	940	938	971
	worst	–	365.96	379.48	386.82	450.67		*worst*	843	791	816	806	799
	average	–	404.60	409.17	421.58	468.68		*average*	893.24	877.57	877.3	877.87	893.07
	std	–	23.58	18.82	23.71	10.23		*std*	38.83	39.91	31.22	33.03	38.44
	ATT	–	0.56	0.56	0.46	0.35		ATT	0.46	0.45	0.45	0.45	0.46

problems. But the complexity of the 01 knapsack problem increases with its size and the adaptivity of p_m to problems with higher dimension sizes may decrease more or less within this region. So for finding dynamic balance between problem size and p_m value, we fixed the value of $p_m = 2/n$, where n is the dimension of the 01 knapsack problem.

4.5. Comparison among DSFLA and MDSFLA

We consider standard ten test problems to compare the performance of DSFLA and MDSFLA. In the first case, we present the comparison between these two with respect to objective function value, and in the second case we only consider the maximum number of iterations. The parameter setting for the two algorithms for the first case are as follows.

For the DSFLA, population size $P = 200$, number of memeplexes $m = 10$, number of iterations within each memeplex $it = 10$ and static probability $\alpha = 0.4$. For the MDSFLA, population size $P = 200$, number of memeplexes $m = 10$, number of iterations within each memeplex $it = 10$, static probability $\alpha = 0.4$ and mutation probability $p_m = 0.06$. Three values of maximum number of iterations (*iMax*) 50, 100 and 150 respectively, are considered to test the performance analysis. Total 30 independent runs are made and corresponding results are given in Table 6.

As we can see in Table 6, the MDSFLA performs better than the DSFLA, and it can easily find the optimal solution for all the cases. Within the small value *iMax* = 50, MDSFLA is able to find the best solution in all cases. On the other hand, DSFLA is not able to find out the best solution within *iMax* = 150 for all the test problems (for functions: f_2, f_8 and f_{10}). When we consider *iMax* = 150, MDSFLA finds best solutions for every independent runs for every test function (*std* = 0 for each of the test functions).

In the second case, we consider the maximum number of iterations (*iMax*) for each algorithm to find the best solution. The parameter settings of the two algorithms are as follows. For DSFLA, population size $P = 200$, number of memeplexes $m = 10$, number of iterations within each memeplex $it = 10$, static probability $\alpha = 0.4$ and maximum number of iterations *iMax* = 500. For the MDSFLA, population size $P = 200$, number of memeplexes $m = 10$, number of iterations within each memeplex $it = 10$, static probability $\alpha = 0.4$, mutation probability $p_m = 0.06$ and maximum number of iterations *iMax* = 500. Here, 30 independent runs are considered and the corresponding results for ten standard test problems are reported in Table 7.

DSFLA fails to find best solutions for f_2, f_8 and f_{10}, and it successfully solves function f_1 only for 12 cases out of 30 and only 5 out of 30 cases for function f_5. Whereas MDSFLA needs *iMax* on an average 5 (for function f_1), 28 (for function f_2), 1 (for function f_3), 1 (for function f_4), 3 (for function f_5), 2 (for function f_6), 1 (for function f_7), 10

Table 5
The effect of p_m on the performance of MDSFLA.

f	p_m									
	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
f_1	2.73	2.8	2.8	2.33	2.4	2.2	2.6	2.1	2.23	2.4
f_2	78.1	66.87	50.5	60.6	62.67	49.13	57.77	57.43	72.57	65.63
f_3	1	1	1	1	1	1	1	1	1	1
f_4	1	1	1	1	1	1	1	1	1	1
f_5	2.43	2.07	2.53	2.17	2.43	2.3	2.5	2.2	2.67	2.4
f_6	1	1	1.03	1.03	1	1	1.1	1	1	1.03
f_7	1	1	1	1	1	1	1	1	1	1
f_8	11.43	21.2	16.3	14.47	18.17	16.07	14.8	16.43	18	16.87
f_9	1	1	1	1	1	1	1	1	1	1
f_{10}	76.37	73.1	51.87	44.77	71.2	43.83	53.83	55.77	56.93	58.63

Table 6
Comparison between DSFLA and MDSFLA.

iMax	*f*	DSFLA						MDSFLA					
		best	*worst*	*average*	*median*	*std*	ATT	*best*	*worst*	*average*	*median*	*std*	ATT
iMax = 50	f_1	295	290	293.97	294	1.10	0.10	295	295	295	295	0.00	0.02
	f_2	985	817	888.03	878	41.76	0.16	1024	1018	1022.8	1024	2.44	0.19
	f_3	35	35	35.00	35	0.00	0.01	35	35	35	35	0.00	0.00
	f_4	23	23	23.00	23	0.00	0.00	23	23	23	23	0.00	0.01
	f_5	481.07	413.17	463.22	466.34	17.87	0.11	481.07	481.07	481.07	481.07	0.00	0.02
	f_6	52	51	51.97	52	0.18	0.01	52	52	52	52	0.00	0.00
	f_7	107	107	107.00	107	0.00	0.00	107	107	107	107	0.00	0.01
	f_8	9765	9752	9,759.13	9758.5	3.69	0.16	9767	9767	9767	9767	0.00	0.10
	f_9	130	130	130.00	130	0.00	0.00	130	130	130	130	0.00	0.00
	f_{10}	941	819	880.97	884.5	35.52	0.15	1025	1019	1024.8	1025	1.10	0.16
iMax = 100	f_1	295	293	294.33	294	0.71	0.16	295	295	295	295	0.00	0.02
	f_2	970	814	886.87	879.5	41.51	0.31	1024	1018	1023.6	1024	1.52	0.24
	f_3	35	35	35.00	35	0.00	0.00	35	35	35	35	0.00	0.00
	f_4	23	23	23.00	23	0.00	0.00	23	23	23	23	0.00	0.00
	f_5	481.07	433.17	465.89	469.16	16.22	0.19	481.07	481.07	481.07	481.07	0.00	0.02
	f_6	52	51	51.93	52	0.25	0.02	52	52	52	52	0.00	0.01
	f_7	107	107	107.00	107	0.00	0.00	107	107	107	107	0.00	0.01
	f_8	9767	9750	9,757.50	9757	4.08	0.30	9767	9767	9767	9767	0.00	0.08
	f_9	130	130	130.00	130	0.00	0.00	130	130	130	130	0.00	0.00
	f_{10}	947	797	888.33	889	36.00	0.30	1025	1019	1024.8	1025	1.10	0.22
iMax = 150	f_1	295	293	294.03	294	0.85	0.28	295	295	295	295	0.00	0.02
	f_2	962	835	888.17	884	30.89	0.45	1024	1024	1024	1024	0.00	0.16
	f_3	35	35	35.00	35	0.00	0.00	35	35	35	35	0.00	0.00
	f_4	23	23	23.00	23	0.00	0.00	23	23	23	23	0.00	0.00
	f_5	481.07	422.57	465.70	469.16	17.30	0.26	481.07	481.07	481.07	481.07	0.00	0.01
	f_6	52	51	51.97	52	0.18	0.02	52	52	52	52	0.00	0.00
	f_7	107	107	107.00	107	0.00	0.01	107	107	107	107	0.00	0.00
	f_8	9765	9750	9,757.13	9757	3.17	0.46	9767	9767	9767	9767	0.00	0.08
	f_9	130	130	130.00	130	0.00	0.00	130	130	130	130	0.00	0.00
	f_{10}	963	819	882.43	876.5	35.19	0.45	1025	1025	1025	1025	0.00	0.15

(for function f_8), 1 (for function f_9) and 29 (for function f_{10}) respectively to solve the test functions. The fewer iterations shows that the MDSFLA has higher efficiency than DSFLA on finding best solutions of 01 knapsack problems. In general, suitable mutation operation can increase the diversity of candidate solutions and improve the capability of space exploration for the MDSFLA.

The convergence process of DSFLA and MDSFLA for the standard test functions are shown in Fig. 3. Parameter setting is same as discussed above, only difference is *iMax* is set to 200. And objective function values are plotted against the iteration numbers for a single instance as long as both the algorithms reach their best solutions.

For test functions f_3, f_4, f_6, f_7 and f_9, both the algorithms find the best solutions with the first iteration itself. For test functions f_1, f_2, f_8 and f_{10} DSFLA trapped in a local optimum point. For both the test functions f_8 and f_{10}, MDSFLA traps within a local optimum point for some time. But due to the divergence category introduced by genetic mutation in MDSFLA, it is able to get out from the local optimum point and easily find out the global optimum point. This eventually proves that the searching capability and algorithmic efficiency of MDSFLA is much more better than DSFLA. It is observed from Fig. 3, for other test functions also MDSFLA performs well better than DSFLA. For test function f_1 as well as f_{10}, MDSFLA starts with a lower basis than DSFLA though it finds the global optimum point really faster than DSFLA.

4.6. Knapsack instances with medium dimension sizes

In order to investigate effectiveness of the algorithm for different instance types we analyze the experimental behavior of algorithm on several sets of randomly generated test problems. Since the difficulty of such problems is greatly affected by the correlation between profits and weights [31], three randomly generated sets of data are considered [23]:

- uncorrelated: $w_i \sim U[1, 2, \ldots, v]$, and $p_i \sim U[1, 2, \ldots, v]$.
- weakly correlated: $w_i \sim U[1, 2, \ldots, v]$, $t_i \sim U[-r, -r+1, \ldots, r-1, r]$ and $p_i := w_i + t_i$ (if, for some i, $p_i \leq 0$, such profit value is ignored and the calculations are repeated until $p_i > 0$).
- strongly correlated: $w_i \sim U[1, 2, \ldots, v]$, and $p_i := w_i + r$.

Table 7
Comparison between DSFLA and MDSFLA with respect to iteration number.

f	DSFLA						MDSFLA					
	Min.iter	Max.iter	Mean.iter	Median.iter	Std.iter	ATT	Min.iter	Max.iter	Mean.iter	Median.iter	Std.iter	ATT
f_1	1	500	300.4	500	248.64	0.88	1	12	4.23	3	3.10	0.03
f_2	500	500	500	500	0	1.49	5	83	27.13	22	22.96	0.17
f_3	1	1	1	1	0	0.00	1	1	1	1	0	0.00
f_4	1	1	1	1	0	0.00	1	1	1	1	0	0.00
f_5	1	500	416.83	500	189.15	1.24	1	9	2.83	3	1.90	0.02
f_6	1	1	1	1	0	0.01	1	3	1.07	1	0.37	0.00
f_7	1	1	1	1	0	0.01	1	1	1	1	0	0.01
f_8	500	500	500	500	0	1.53	1	21	9.3	9	4.51	0.06
f_9	1	1	1	1	0	0.00	1	1	1	1	0	0.00
f_{10}	500	500	500	500	0	1.50	5	115	28.43	18.5	26.59	0.18

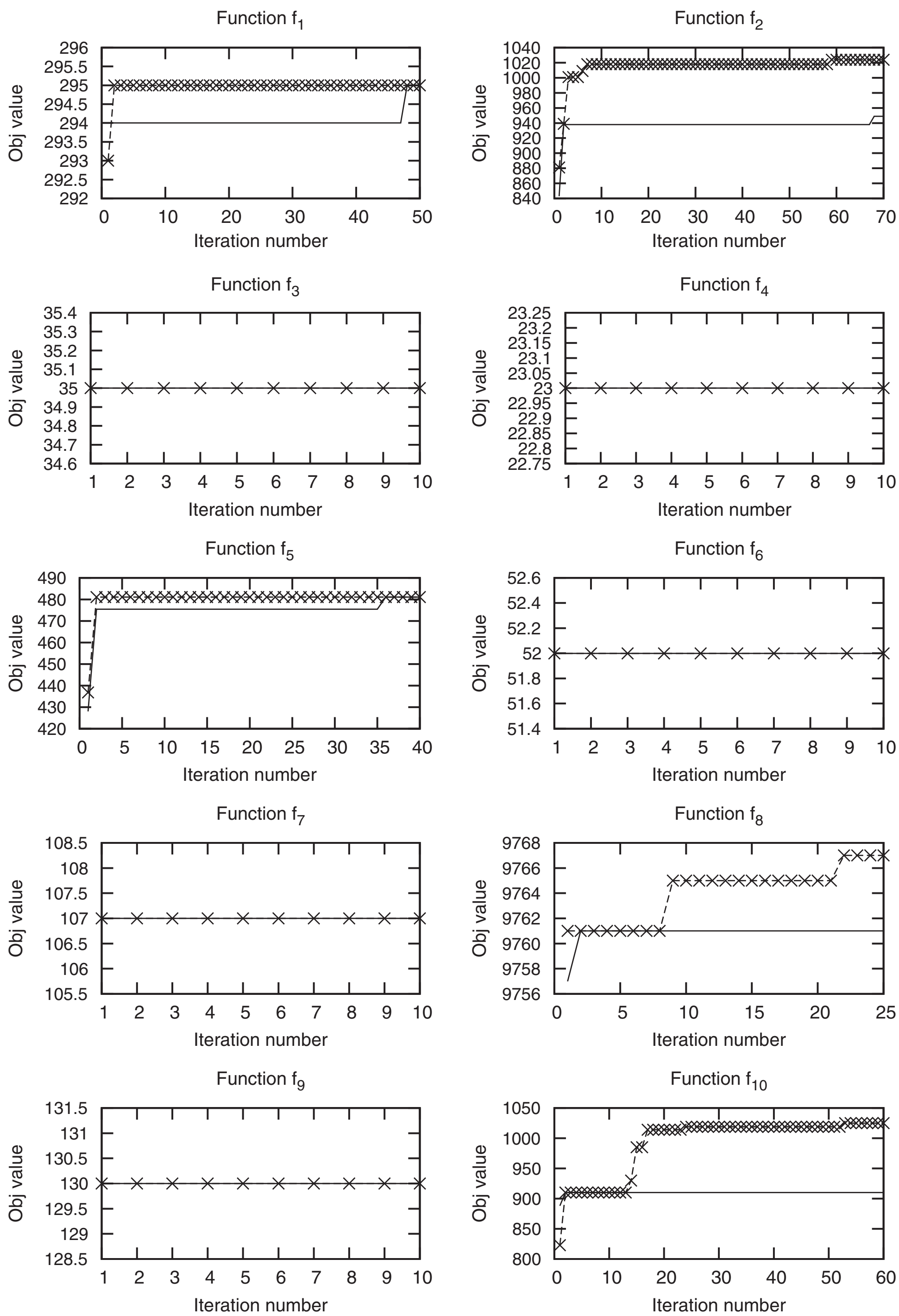

Fig. 3. Convergence process of DSFLA(solid line) and MDSFLA(dotted cross line) for standard test functions.

Data have been generated with the following parameter settings: $v = 100$ and $r = 10$. For the tests we used four data sets of each type containing $n = 25, 50, 75$ and 100 items, respectively. Again, following a suggestion from [31], we have taken under consideration four knapsack types:

1. highly restrictive knapsack capacity: A knapsack with the capacity of $b_1 = v$. In this case the optimal solution contains very few items.
2. restrictive knapsack capacity: A knapsack with the capacity of $b_2 = 2v$. In this case the optimal solution contains few items. An area, for which conditions are not fulfilled, occupies almost the whole domain.
3. average knapsack capacity: A knapsack with the capacity $b_3 = 0.5\sum_{i=1}^{n} w_i$. In this case about half of the items are in the optimal solution.
4. trimmed knapsack capacity: A knapsack with the capacity $b_4 = R\sum_{i=1}^{n} w_i$, where $R \sim U(0.25, 0.75)$. In this case $E(R) = 0.5$ and $Var(R) = 0.25/12$.

As reported by Martello and Toth [31], further increasing the value of capacity does not significantly increase the computation times of the classical algorithms. Problem instances corresponding to

Table 8
Solutions of generated knapsack instances by DSFLA.

Group	No. of items	*b*	*op*	*best*	*worst*	*mean*	*median*	*SD*	*dev*(%)	*A.dev*(%)
Uncorrelated	25	100	231	231	231	231.00	231	0.00	0.00	0.00
		200	340	340	335	336.00	336	0.83	0.00	1.18
		566	1082	1075	961	1000.07	999	26.39	0.65	7.57
		644	985	977	880	916.10	914	22.10	0.81	6.99
	50	100	442	442	442	442.00	442	0.00	0.00	0.00
		200	862	862	788	823.17	818	24.64	0.00	4.51
		1223	2307	2100	1918	2006.30	2006	48.00	8.97	13.03
		1180	2449	2277	1901	2050.07	2026	81.88	7.02	16.29
	75	100	795	791	720	753.93	755.5	18.05	0.50	5.17
		200	1239	1162	1047	1102.43	1094	31.40	6.21	11.02
		1802	3018	2557	2351	2466.87	2464.5	50.89	15.28	18.26
		1942	2813	2526	2262	2336.33	2318.5	61.49	10.20	16.95
	100	100	1100	1080	1004	1044.00	1041	20.36	1.82	5.09
		200	1201	1160	1060	1105.50	1100	26.65	3.41	7.95
		2593	4076	3490	3222	3365.30	3385	70.83	14.38	17.44
		2617	3947	3299	3014	3133.13	3122.5	66.77	16.42	20.62
Weakly correlated	25	100	125	125	123	123.13	123	0.51	0.00	1.49
		200	224	224	224	224.00	224	0.00	0.00	0.00
		630	697	692	671	682.53	684	5.52	0.72	2.08
		556	610	610	590	598.13	598	5.35	0.00	1.95
	50	100	122	122	122	122.00	122	0.00	0.00	0.00
		200	249	249	244	247.23	248	1.41	0.00	0.71
		1401	1585	1546	1505	1528.20	1528.5	9.44	2.46	3.58
		1445	1596	1553	1516	1535.37	1535.5	9.49	2.69	3.80
	75	100	158	158	157	157.53	158	0.51	0.00	0.30
		200	289	289	274	280.40	280	4.45	0.00	2.98
		2049	2223	2165	2124	2142.80	2142.5	9.08	2.61	3.61
		1084	1242	1230	1213	1220.10	1220	4.29	0.97	1.76
	100	100	171	171	163	166.27	166	1.66	0.00	2.77
		200	285	282	272	276.70	277	2.42	1.05	2.91
		2570	2864	2782	2731	2748.27	2747.5	12.55	2.86	4.04
		1790	2057	2007	1969	1983.20	1983.5	9.52	2.43	3.59
Strongly correlated	25	100	150	150	148	149.27	150	0.98	0.00	0.49
		200	300	300	298	299.67	300	0.61	0.00	0.11
		638	808	807	777	790.03	788	6.19	0.12	2.22
		344	474	474	458	462.60	462.5	3.39	0.00	2.41
	50	100	180	180	175	178.83	179	1.56	0.00	0.65
		200	340	337	326	329.27	329	2.56	0.88	3.16
		1295	1645	1599	1571	1583.03	1582	7.29	2.80	3.77
		1106	1446	1405	1373	1388.23	1389.5	9.49	2.84	3.99
	75	100	220	220	208	214.13	215.5	4.37	0.00	2.67
		200	360	360	344	352.73	350	4.83	0.00	2.02
		1505	2055	1985	1918	1940.93	1938	15.31	3.41	5.55
		1565	2045	1983	1944	1961.47	1961.5	9.32	3.03	4.08
	100	100	220	220	209	217.37	219	3.44	0.00	1.20
		200	380	380	354	364.97	365.5	5.93	0.00	3.96
		2338	3058	2925	2881	2902.47	2903	10.14	4.35	5.09
		2139	2779	2674	2628	2648.83	2646	10.94	3.78	4.68

each class are solved by DSFLA as well as MDSFLA. The results for DSFLA are shown in Table 8 and results of MDSFLA are given in Table 9. Here, *b* represents the capacity of the knapsack; *op* represents best solution finds by CPLEX V12.2.0. For DSFLA, the best, worst solutions among 30 independent runs are given along with average, median and standard deviation of the results. Also the deviations ($dev = \frac{op-sol}{op} \times 100\%$) from the best known solution with best solution and average solution is reported. On the other hand MDSFLA finds the best known solution for each individual run, so the number of iterations required to find the solution is reported in Table 8; each number in the column *b.iter* is the minimal number of iterations to reach best solution among 30 independent runs of the proposed algorithm; each number in the column *w.iter* is the maximal number of iterations to reach best solution among 30 independent runs of the proposed algorithm; each number in the column *a.iter* is the average number of iterations to reach best solution among 30 independent runs of the proposed algorithm; each number in the column *me.iter* is the median of the number of iterations to reach best solution among 30 independent runs of the proposed algorithm; and each number in the column *SD.iter* represents standard deviation. ATT is the average total time to solve the particular problem.

In this case the parameter settings of the algorithms are as follows. population size $P = 200$, number of memeplexes $m = 10$, number of iterations within each memeplex $it = 10$, static probability $\alpha = 0.4$, maximum number of iterations $iMax = 2000$ and for MDSFLA mutation probability is $p_m = 2/n$. In the performance basis, DSFLA is only applicable for small item sized problem instances with highly restrictive knapsack capacity and restrictive type knapsack capacity. The average deviation from mean solution of 30 independent runs is 9.50% in case of uncorrelated problem type, 2.22% for weakly correlated problem type and 2.88% for strongly correlated problem type. So we may conclude that weakly correlated and strongly correlated problem instances are easier to solve than uncorrelated problem type in the case of DSFLA. The average deviation from best solution find by DSFLA is 5.35% in case of uncorrelated problem type, 0.99% for weakly correlated problem type and 1.33% for strongly correlated problem type which also supports our observation. From Table 9 it is observed that the standard deviation of the number of iterations to solve a particular problem instance

Table 9
Solutions of generated knapsack instances by MDSFLA.

Group	No. of items	b	op	$b.iter$	$w.iter$	$a.iter$	$me.iter$	$SD.iter$	ATT
Uncorrelated	25	100	231	2	2	2	2	0	0.04
		200	340	4	37	13.6	10.5	8.45	0.26
		566	1082	2	72	24.7	22.5	19.5	0.39
		644	985	6	271	26.9	13	53.61	0.42
	50	100	442	2	6	3.3	3	1.21	0.1
		200	862	6	53	17.2	15	10.52	0.52
		1223	2307	20	1356	433.93	227.5	467.26	8.98
		1180	2449	31	1456	664.17	639	429.33	13.1
	75	100	795	9	133	45.7	27.5	36.33	2.17
		200	1239	7	44	19.4	19.5	8.16	0.82
		1802	3018	36	1434	478.6	439	333.02	11.88
		1942	2813	23	704	194.07	117	196.06	4.82
	100	100	1100	8	103	29.53	25.5	18.91	1.99
		200	1201	12	152	52.5	39	37.96	3.34
		2593	4076	41	1561	597.73	501	373.54	18.73
		2617	3947	136	1210	645.27	634.5	320.63	20.17
Weakly correlated	25	100	125	2	10	5.43	5	2.27	0.11
		200	224	2	3	2.1	2	0.31	0.04
		630	697	40	767	179.8	130	168.19	2.89
		556	610	3	127	38.73	32.5	28.55	0.63
	50	100	122	2	3	2.03	2	0.18	0.06
		200	249	2	81	18.9	14.5	16.08	0.6
		1401	1585	25	1331	509.77	430.5	397.33	9.89
		1445	1596	72	1465	730.03	750.5	442.27	13.74
	75	100	158	2	19	8.03	8	3.56	0.38
		200	289	7	32	14.43	13	5.44	0.67
		2049	2223	120	1049	476.4	480	250.41	11.81
		1084	1242	36	1401	437.63	339.5	383.07	15.1
	100	100	171	6	48	19.23	14	12.67	1.35
		200	285	6	218	41.13	23	42.27	2.77
		2570	2864	82	1598	916.53	946.5	307.74	28.93
		1790	2057	64	1290	491.67	441.5	294.95	19.76
Strongly correlated	25	100	150	1	6	2.57	2	1.07	0.05
		200	300	3	15	7.3	6.5	3.31	0.12
		638	808	6	294	60.6	33.5	77.88	0.94
		344	474	5	39	18.1	18	8.29	0.29
	50	100	180	3	23	9.53	8.5	4.89	0.3
		200	340	7	36	19.37	21	7.44	0.53
		1295	1645	74	1483	594.93	311.5	504.96	11.11
		1106	1446	91	1583	599	563	382.56	11.23
	75	100	220	5	35	16.77	14.5	8.65	0.79
		200	360	9	27	14.77	13.5	4.78	0.63
		1505	2055	102	1083	581.67	569	280.59	13.95
		1565	2045	144	1359	651.73	557.5	362.31	16.28
	100	100	220	2	17	9.37	9	3.86	0.61
		200	380	6	60	23.97	24	12.62	1.51
		2338	3058	424	1559	953.47	944	322.5	29.11
		2139	2779	592	1573	1080.2	1133	246.4	13.39

within 30 independent runs is very high in most of the cases. So we prefer $me.iter$ over $a.iter$ to represents the average behavior of the algorithm. On an average number of iterations required to solve the instances are 171, 227 and 264 for uncorrelated, weakly correlated and strongly correlated medium size problems, respectively. Also results do not show any specific effect of correlation structure on the solving process by MDSFLA, though the solution process is much more dependent upon knapsack type.

From Table 9, it is observed that type 3 and type 4 of knapsack capacity problem instances are difficult to solve, and depending upon the problem size number of iterations increases gradually. Depending upon the knapsack capacity and problem size, MDSFLA effectively solve all the problems within 1600 iterations and average number of iterations within 1000 range, which is much smaller.

4.7. Knapsack instances with large dimension sizes

Nine 01 knapsack problems with large scales are devised to testify the performance of the modified shuffled frog leaping algorithm. The number of items n is set to 200, 500 and 1000, respectively. Only type 1 knapsack problems are generated and corresponding results are reported in Table 10. The uncorrelated, weakly correlated and strongly correlated knapsack instances of 200 items are graphically illustrated in Figs. 4–6, respectively.

The parameter settings for both the algorithms are as follows: population size $P = 400$, number of memeplexes $m = 20$, number of iterations within each memeplex $it = 10$, static probability $\alpha = 0.4$, mutation probability $p_m = 2/n$ and maximum number of iterations $iMax = 2000$. Total 30 independent runs are considered and the corresponding results for all test problems are given in Table 10.

From Table 10 it is seen that, MDSFLA finds the best solutions in all the cases though only for two cases DSFLA able to find out the best solution. Also the number of successful runs to find out the optimal solution out of 30 independent experiments are also reported here. MDSFLA performs much better than DSFLA in all the cases, the average deviation from mean solution is 6.25% for DSFLA and 0.03% for MDSFLA. In worst case maximum number of iterations required to solve the problem is within 1500 range and for average case it is within 1000 range. Strongly correlated and weakly

Table 10
Solution of large knapsack instances.

Group	No. of items	*op*	*best*	*worst*	*average*	*median*	*Std*	ATT	No. of cases
DSFLA solutions									
Uncorrelated	200	1097	1089	1021	1058.30	1059.50	21.96	135.30	0
	500	1572	1510	1403	1455.00	1458.00	26.36	285.63	0
	1000	2638	2381	2272	2304.46	2293.00	31.59	658.22	0
Weakly correlated	200	172	172	168	170.43	171.00	1.43	132.76	10
	500	231	229	218	222.13	222.00	3.08	283.93	0
	1000	276	265	258	261.63	261.50	2.20	553.43	0
Strongly correlated	200	300	300	280	286.03	287.00	4.80	57.95	1
	500	380	358	336	346.17	348.00	5.80	128.80	0
	1000	500	470	439	451.63	450.00	6.33	272.44	0
MDSFLA solutions									
Uncorrelated	200	1097	1097	1097	1097.00	1097.00	0.00	9.55	30
	500	1572	1572	1567	1570.70	1570.00	1.41	698.45	14
	1000	2638	2638	2624	2633.00	2634.00	4.60	1683.30	10
Weakly correlated	200	172	172	172	172.00	172.00	0.00	1.62	30
	500	231	231	231	231.00	231.00	0.00	61.84	30
	1000	276	276	276	276.00	276.00	0.00	531.35	30
Strongly correlated	200	300	300	300	300.00	300.00	0.00	4.44	30
	500	380	380	370	379.67	380.00	1.83	135.79	29
	1000	500	500	500	500.00	500.00	0.00	522.75	30

correlated problem instances are easier to solve than the uncorrelated problem type. Also depending upon the problem size number of iterations and average total time to solve the problem increase gradually. From the performance analysis it is clear that MDSFLA performs well to solve 01 knapsack problem in every cases of knapsack type and dimension. It finds best solution of knapsack problem within very small number iterations, which gives it preference over other algorithms.

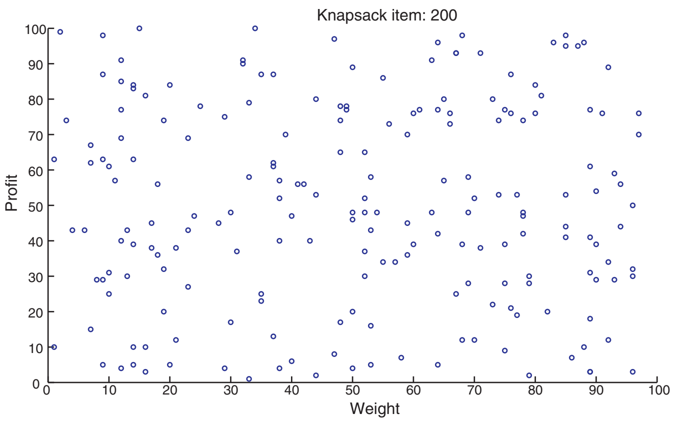

Fig. 4. 200-Item knapsack. Uncorrelated situation.

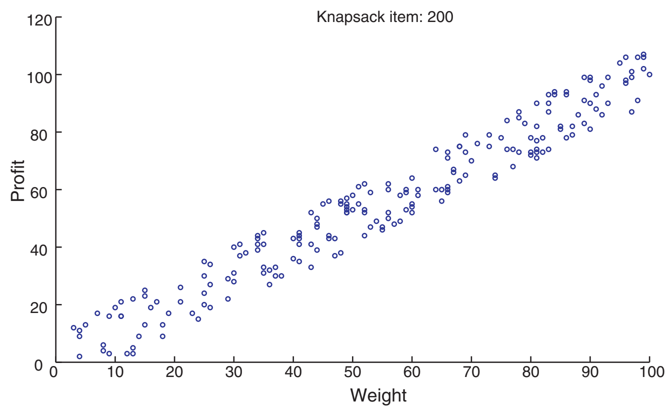

Fig. 5. 200-Item knapsack. Weakly correlated situation.

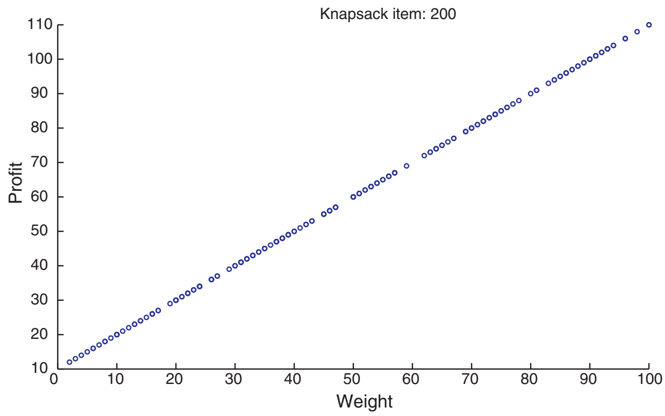

Fig. 6. 200-Item knapsack. Strongly correlated situation.

5. Conclusions

In this work, the performance of the shuffled frog leaping algorithm has been extensively investigated by using a large number of experimental studies. We have shown, first, that each of the three discrete SFLAs obtained good quality of solutions by utilizing both the local search procedure based on swarm intelligence and the competitiveness mixing of information of the genetic based memetic algorithms. The experimental results show that the DSFLA has demonstrated strong convergence and stability for 01 knapsack problems due to the utilization of the discretization effect of static probability.

However for solving unknown problems, DSFLA may trapped in some suboptimal point and modification of the searching space is required for such cases. On the other hand computational results reveal that the MDSFLA has strong capability of preventing premature convergence of the DSFLA throughout the whole iteration due to the utilization of the genetic mutation.

Also different types of knapsack problem instances are generated to test the convergence property of MDSFLA. Results show that MDSFLA is found very effective in solving small to medium sized knapsack problems. Actually, the proposed algorithm easily found all of the best solutions for smaller size problems. Further in this study, we have investigated different correlated problems for the performance analysis of MDSFLA and corresponding results do not support any specific advantage over correlation structure of the problem. The proposed algorithm is also found very effective for solving larger size and tightly constrained 01 knapsack problem. These problems are very complex, therefore their solution can be considered as a very good indicator for the potential of the shuffled frog leaping algorithm.

Based on our computational study we have observed that the proposed MDSFLA has a potential for solving any 01 knapsack problem. It may also be used for solving multiple knapsack problems or multi-objective knapsack problems and other combinatorial optimization problems like generalized assignment problems, set covering problems, etc. and is scheduled as a future work.

References

[1] G. Mavrotas, D. Diakoulaki, A. Kourentzis, Selection among ranked projects under segmentation, policy and logical constraints, European Journal of Operational Research 187 (2008) 177–192.

[2] D.C. Vanderster, N. Dimopoulos, R. Hernandez, R.J. Sobie, Resource allocation on computational grids using a utility model and the knapsack problem, Future Generation Computer Systems 25 (2009) 35–50.

[3] J. Yates, K. Lakshmanan, A constrained binary knapsack approximation for shortest path network interdiction, Computers & Industrial Engineering 61 (2011) 981–992.

[4] S. Peeta, D. Salman, K. Gunnec, Viswanath, Pre-disaster investment decisions for strengthening a highway network, Computers and Operations Research 37 (2010) 1708–1719.

[5] J.F. Zhao, T. Huang, F. Pang, Y. Liu, Genetic algorithm based on greedy strategy in the 0-1 knapsack problem, in: 3rd International Conference on Genetic and Evolutionary Computing, WGEC '09, 2009, pp. 105–107.

[6] F.T. Lin, Solving the knapsack problem with imprecise weight coefficients using genetic algorithms, European Journal of Operational Research 185 (2008) 133–145.

[7] Y. Liu, C. Liu, A schema-guiding evolutionary algorithm for 0-1 knapsack problem, in: International Association of Computer Science and Information Technology-Spring Conference, 2009, pp. 160–164.

[8] Y. Wanga, X. Feng, Y. Huang, D. Pub, W. Zhoua, Y. Liang, C. Zhou, A novel quantum swarm evolutionary algorithm and its applications, Neurocomputing 70 (2007) 633–640.

[9] H.X. Shi, Solution to 0/1 knapsack problem based on improved ant colony algorithm, in: International Conference on Information Acquisition, 2006, pp. 1062–1066.

[10] Z.K. Li, N. Li, A novel multi-mutation binary particle swarm optimization for 0/1 knapsack problem, in: Control and Decision Conference, 2009, pp. 3042–3047.

[11] D. Zou, L. Gao, S. Li, J. Wu, Solving 0-1 knapsack problem by a novel global harmony search algorithm, Applied Soft Computing 11 (2011) 1556–1564.

[12] M.M. Eusuff, K. Lansey, F. Pasha, Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization, Engineering Optimization 38 (2006) 129–154.

[13] X. Zhang, X. Hu, G. Cui, Y. Wang, Y. Niu, An improved shuffled frog leaping algorithm with cognitive behavior, in: Proc. 7th World Congr. Intelligent Control and Automation, 2008, pp. 6197–6202.

[14] M. Eusuff, K. Lansey, Optimization of water distribution network design using the shuffled frog leaping algorithm, Journal of Water Resource Plan Management 129 (2003) 210–225.

[15] H. Elbehairy, E. Elbeltagi, T. Hegazy, Comparison of two evolutionary algorithms for optimization of bridge deck repairs, Computer-Aided Civil and Infrastructure Engineering 21 (2006) 561–572.

[16] A. Rahimi-Vahed, A. Mirzaei, Solving a bi-criteria permutation flow-shop problem using shuffled frog-leaping algorithm, in: Soft Computing, Springer-Verlag, New York, 2007.

[17] L. Wang, C. Fang, An effective shuffled frog-leaping algorithm for multi-mod resource-constrained project scheduling problem, Information Sciences 181 (2011) 4804–4822.

[18] J. Ebrahimi, S.H. Hosseinian, G.B. Gharehpetian, Unit commitment problem solution using shuffled frog leaping algorithm, IEEE Transaction on Power Systems 26 (2011) 573–581.

[19] X.H. Luo, Y. Yang, X. Li, Solving tsp with shuffled frog-leaping algorithm, in: Proc. ISDA, vol. 3, 2008, pp. 228–232.

[20] E. Elbeltagi, T. Hegazy, D. Grierson, Comparison among five evolutionary-based optimization algorithms, Advanced Engineering Informatics 19 (2005) 43–53.

[21] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proc. IEEE Conf. Neural Networks, vol. 4, 1995, pp. 1942–1948.

[22] S. Liong, M. Atiquzzaman, Optimal design of water distribution network using shuffled complex evolution, Journal of Instrumentation Engineering 44 (2004) 93–107.

[23] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, vol. 3, Springer-Verlag, revised and extended edition, 1999.

[24] C. An, Y.J. Fu, On the sequential combination tree algorithm for 0-1 knapsack problem, Journal of Wenzhou University (Natural Sciences) 29 (2008) 10–14.

[25] W. You, Study of greedy-policy-based algorithm for 0/1 knapsack problem, Computer and Modernization 4 (2007) 10–16.

[26] H. Yoshizawa, S. Hashimoto, Landscape analyses and global search of knapsack problems, IEEE Systems, Man, and Cybernetics 3 (2000) 2311–2315.

[27] D. Fayard, G. Plateau, Resolution of the 0-1 knapsack problem comparison of methods, Mathematical Programming 8 (1975) 272–307.

[28] J.Y. Zhao, Nonlinear reductive dimension approximate algorithm for 0-1 knapsack problem, Journal of Inner Mongolia Normal University (Natural Science Edition) 36 (2007) 25–29.

[29] Y. Zhu, L.H. Ren, Y. Ding, DNA ligation design and biological realization of knapsack problem, Chinese Journal of Computers 31 (2008) 2207–2214.

[30] B.D. Li, Research on the algorithm for 0/1 knapsack problem, Computer and Digital Engineering 5 (2008) 23–26.

[31] S. Martello, P. Toth, Knapsack Problems, John Wiley, Chichester, UK, 1990.