NEW POPULATION-BASED DESCENT ALGORITHM USING MEAN VALUE OF COORDINATE COMPONENTS

Tatsuya Kobayashi
LAC Co., Ltd.
JAPAN
tatsuya.kobayashi2@gmail.com

Hirotaka Shimizu
Class Technology Co., Ltd.
JAPAN
hirotaka.smz@gmail.com

Tsutomu Shohdohji
Nippon Institute of Technology
JAPAN
shodoji@nit.ac.jp

ABSTRACT

Supported by dramatic advances in computer processing, heuristic methods such as metaheuristics (also known as meta-strategies and meta-solutions) have found widespread use for solving complex combinatorial optimization problems and nonlinear optimization problems. Algorithm performance and ease of implementation are important factors in solving complex optimization problems in the real world. With the aim of developing an algorithm that excels in both performance and ease of implementation, we propose a mean search algorithm and report on its superior results in various performance-comparison tests.

Keywords: non-linear programming problem, metaheuristics, mean search algorithm, population-based descent algorithm, operational research.

INTRODUCTION

Generally speaking, it is extremely difficult to find a global optimum to complex and massive optimization problems that exist in the real world. Many such problems deal with high-dimension, multi-modal functions having multiple local optima, so finding a global optimum within a realistic period of time is difficult. In addition, techniques that can guarantee the finding of a global optimum within an acceptable computation time do not, at present, exist. On the other hand, it is sufficient in many high-dimension engineering design problems to find a solution that satisfies certain constraints (called an "approximate solution" in this paper) without being a global optimum. For this reason, attention has recently come to focus on metaheuristics as a method for solving complex optimization problems.

Metaheuristics constitute a framework for solving optimization problems by heuristic searching. They are problem-independent techniques that can be applied to a wide variety of problems. While there is no guarantee that a discovered solution is optimal, metaheuristics can be used for finding an approximate solution at high speed [2]. A variety of techniques have been proposed as metaheuristics such as the genetic algorithm (GA) [3], differential evolution (DE) [8], and particle swarm optimization

(PSO) [5, 6]. In particular, DE uses the difference between two solutions in solution updating to automatically update the variation width of solution components according to the degree of progression of the search. A feature of this algorithm is its effectiveness in dealing with complex multi-modal functions despite its simplicity. On the other hand, DE suffers from insufficient local searching (centralization) and slow speed of convergence.

Complex optimization problems that occur in the real world also require a high calculation cost for evaluating the solution. Minimizing the number of solution evaluations is therefore important in shortening computation time. In this study, we propose a mean search (MS) algorithm as a new algorithm using the mean value of coordinate components of solutions.

The MS algorithm enhances local search performance by using the mean value of solution coordinates and performs searching while achieving diversity through DE-inspired mutations. This method aims to reduce the number of evaluations by discovering an approximate solution at an early stage. We demonstrate the effectiveness of the MS algorithm through a numerical experiment applying several benchmark functions.

METAHEURISTICS

Metaheuristics is a framework for solving optimization problems by heuristic searching. They are problem-independent techniques that can be applied to a wide variety of problems. Although there is no guarantee that a discovered solution is optimal, metaheuristics can be used for finding an approximate solution (a solution that satisfies constraints without being a global optimum) at high speed. A variety of techniques have been proposed as metaheuristics—GA, DE, and PSO are typical of these.

Metaheuristics include various types of algorithms with different properties, such as techniques like GA and PSO that speed up solution improvement and techniques like the artificial bee colony (ABC) algorithm [1, 4] and harmony search (HS) [9] that excel in the accuracy of the final solution.

In this paper, we use the following six algorithms for performing a performance comparison with the MS algorithm.

(a) Artificial bee colony algorithm

Proposed by Dervis Karaboga in 2005, the ABC algorithm is based on the foraging behavior of honey bees [1, 4]. It performs three phases of operations corresponding to employed bees, onlookers, and scouts and searches for an optimal solution.

(b) Cuckoo search

Cuckoo search (CS) is an optimization algorithm proposed by Xin-She Yang and Suash Deb in 2009 [9]. It was inspired by the behavior of some cuckoos that lay their eggs in the nests of other birds to be nurtured by them (called "brood parasitism"). This algorithm enhances search performance by performing Lévy flights based on a Lévy distribution.

(c) Differential evolution

Proposed by Rainer Storn and Kenneth Price in 1997, the DE algorithm enables searches tailored to search conditions and problem properties by using a difference vector between search points in search-point update processing [8].

(d) Genetic algorithm

Proposed by John Henry Holland in 1975, GA is modeled on the evolutionary process of organisms [3]. It updates search points based on the three basic operations of selection, crossover, and mutation.

(e) Harmony search

The HS algorithm proposed by G.V. Loganathan, Joong Hoon Kim, and Zong Woo Geem in 2001 imitates the improvisation process of musicians [7]. It updates search points based on three operations: use existing elements unchanged, change some of the existing elements, and replace with new elements.

(f) Particle swarm optimization

Proposed by James Kennedy and Russell Eberhart in 1995, POS is an optimization method using swarm intelligence [5, 6]. It carries out searching while sharing information on best position as found by the group (swarm) and by an individual. In the event that a best position is found within the swarm, that information is passed on to the other individuals, which are then drawn to the individual with the best position.

MEAN SEARCH ALGORITHM

In general, the possibility is high that an even better solution can be found in the vicinity of existing candidate solutions (proximate optimality principle). The MS algorithm centralizes search points by performing a move operation to a position midway between two randomly selected search points. However, search-point centralization can also lead to a loss of search-point diversity. In metaheuristics, search-point diversity makes it difficult to become trapped in local optima. A loss of diversity can cause solution updating to stop. With this in mind, the MS algorithm uses the mean distance between search points to perform efficient mutations according to search conditions thereby preserving search-point diversity.

Specifically, the MS algorithm observes the following three rules and generates a new search point based on selected (existing) search points. These operations are performed on all elements of all individuals.

I. Crossover

A conceptual diagram of crossover processing is shown in Figure 1. This operation randomly selects two search points x_{aj}^{t}, x_{bj}^{t} and treats the position of the mean coordinate between those two search points as new search point x_{ij}^{new}.

II. Mutation

A conceptual diagram of mutation processing is also shown in Figure 1. This operation determines the mean distance between the currently selected search point x_{ij}^{t} and two randomly selected search points x_{aj}^{t}, x_{bj}^{t} and randomly generates new search point x_{ij}^{new} within the range of that mean distance.

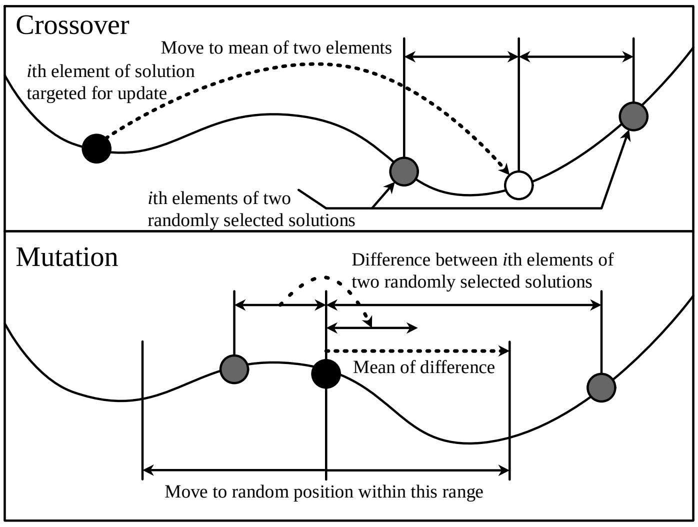
III. Copy

The copy operation treats the position of a randomly selected search point as new search point x_{ij}^{new}. This operation then compares the newly generated search point and the currently selected search point, and if the new search point is better, replaces the currently selected search point with the new search point.

The specific the MS algorithm execution procedure is as follows.

Step 0. Initialize

Read in the target function and generate initial population $\left\{x_i^t, i = 1, 2, \cdots, N\right\}$ consisting of N initial search points x_i^1 (time $t = 1$) each of D dimensions with randomly valued elements.


Crossover
Move to mean of two elements
ith element of solution targeted for update
ith elements of two randomly selected solutions
Mutation
Difference between ith elements of two randomly selected solutions
Mean of difference
Move to random position within this range
Figure 1. Crossover and mutation in mean search

Step 1. Evaluate initial population

Evaluate search points by the value of the target function $f(x_i^1)$ at each search point and calculate their fitness.

Step 2. Update search-point population

Perform the following processing for all search points.

Step 2-1. Generate new search point

Generate new search point x_i^{new} according to Eq. (1).

$$x_{ij}^{new} = \begin{cases} \dfrac{x_{aj}^t + x_{bj}^t}{2}, & \text{if } r^i = j \text{ } or \text{ } rand(0,1) < CR, \\ x_{ij}^m, & \text{else if } rand(0,1) < MR, \\ x_{ij}, & \text{otherwise.} \end{cases} \tag{1}$$

Here, *CR* and *MR* denote crossover probability and mutation probability, respectively, while *rand* (0, 1) returns a real number from 0 to 1 as a uniformly distributed random number. In descending order, each row in the above equation signifies crossover, mutation, and inheritance. According to this equation, the mean value of the same elements of two other randomly selected search points x_{aj}^t and x_{bj}^t may be adopted depending on crossover probability *CR*. If not, the value x_{ij}^m may then be selected depending on mutation probability *MR*. If neither of the above is adopted, the original value x_{ij}^t is adopted as the new search point. The value x_{ij}^m is determined by Eq. (2) below.

$$x_{ij}^m = x_{ij}^t + \left[rand(-1,1) \left(\frac{\left| x_{ij}^t - x_{aj}^t \right| + \left| x_{ij}^t - x_{bj}^t \right|}{2} \right) \right]. \tag{2}$$

Step 2-2. Evaluate search point

Evaluate the newly generated search point x_i^{new}, and if this search point is better than the original search point x_i^t (minimization problem), set x_i^{new} as the search point after updating.

Step 3. Test for termination

Terminate the procedure if termination conditions are satisfied. Otherwise, return to Step 2.

NUMERICAL EXPERIMENT, RESULTS, AND DISCUSSION

We performed a numerical experiment to assess the performance of the MS algorithm. In this experiment, we executed multiple algorithms including the MS algorithm against multiple test functions and demonstrated the superior performance of the MS algorithm by comparing the test results of these algorithms.

The test functions used in this experiment are listed in Table 1. All of these 21 test functions represent a minimization problem; each was adjusted so that the minimum value that could be obtained would be 0.

For the sake of fairness, we adjusted population size and number of generations to keep computational complexity as uniform as possible and conducted the experiment under a variety of conditions. In this regard, computational complexity is proportional to the product NT of population size N and maximum number of generations T, so we varied the individual values of N and T without changing the value of their product. Specifically, we conducted the experiment with $N \geq 10, T \geq 10$, NT = 20,000, 40,000, 80,000, and number of dimensions D = 100, 200, 400. In addition, the experiment was independently executed 100 times for each set of conditions and the average of those trials was taken to be the final result.

The parameters for each of the algorithms used in the experiment are as follows. With N and D denoting population size and number of dimensions, respectively, we set mutation probability to 0.1 and crossover probability to 0.1 in the MS algorithm, colony size to 2N and limit to 0.1ND in the ABC algorithm, mutation probability to 0.25 and variation width to 0.01 in the CS algorithm, crossover probability to 0.05 and variation width to 1.0 in the DE algorithm, crossover probability to 0.6, mutation probability to 0.001, and tournament selection factor to 5 in the GA algorithm, selection probability to 0.9, adjustment probability to 0.3, and variation width to 0.001 in the HS algorithm, and inertia weight coefficient to 0.729, cognitive coefficient to 1.49445, and social coefficient to 1.49445 in the POS algorithm.

Table 1. List of test functions

Number	Function	Bounds	Continuity	Modality
F01	Ackley	$-32.768 < x_i < 32.768$	Continuous	Multimodal
F02	Alpine01	$-10 < x_i < 10$	Continuous	Multimodal
F03	CosineMixture	$-1 < x_i < 1$	Continuous	Multimodal
F04	Csendes	$-1 < x_i < 1$	Continuous	Unimodal
F05	Dixon&Price	$-10 < x_i < 10$	Continuous	Unimodal
F06	Griewank	$-600 < x_i < 600$	Continuous	Multimodal
F07	Holzman2	$-10 < x_i < 10$	Continuous	Unimodal
F08	Lévy	$-10 < x_i < 10$	Continuous	Multimodal
F09	Mishra11	$-10 < x_i < 10$	Continuous	Unimodal
F10	Penalty01	$-50 < x_i < 50$	Continuous	Multimodal
F11	Penalty02	$-50 < x_i < 50$	Continuous	Multimodal
F12	Rastrign	$-5.12 < x_i < 5.12$	Continuous	Multimodal
F13	Rosenbrock	$-2.048 < x_i < 2.048$	Continuous	Unimodal
F14	Salomon	$-100 < x_i < 100$	Continuous	Multimodal
F15	Schwefel1.2	$-64 < x_i < 64$	Continuous	Unimodal
F16	Schwefel2.21	$-10 < x_i < 10$	Continuous	Unimodal
F17	Schwefel2.22	$-10 < x_i < 10$	Continuous	Unimodal
F18	Schwefel2.26	$-512 < x_i < 512$	Continuous	Multimodal
F19	Sphere	$-5.12 < x_i < 5.12$	Continuous	Unimodal
F20	Step	$-5.12 < x_i < 5.12$	Discontinuous	Unimodal
F21	Zakharov	$-5 < x_i < 10$	Continuous	Unimodal

As an example of experimental results, Tables 2 and 3 list results for the conditions N=100, T=200, and D=100.

The results listed in Tables 2 and 3 show that the MS algorithm is superior for many test functions for the conditions described above.

Figures 2 to 5 show convergence graphs for various test functions. For functions whose optimal solution lies in the center such as F1, F3, and F12, the MS algorithm converges fast compared with the other algorithms, but for the Schwefel2.26 function of Figure 5 whose optimal solution lies near a boundary value, the MS algorithm converges relatively slowly.

Next, Figures 6 to 8 each show the results of varying the number of search points and number of generations while keeping the product of the number of search points (population size) and number of generations unchanged. It can be seen from these results that changing the number of dimensions or the value of *NT* affects algorithm performance but hardly affects the manner in which algorithm performance changes according to the *N*:*T* ratio. The way in which algorithm performance changes is particularly easy to understand from the experimental graphs of Figure 6 for the Ackley function.

Table 2. Experimental results for N=100, T=200, and D=100

	MS	ABC	CS	DE	GA	HS	PSO
F01	**2.74**	19.78	17.19	18.90	10.15	14.93	7.89
F02	**0.20**	127.46	89.30	113.99	17.58	40.09	30.65
F03	**0.75**	25.96	45.32	24.42	5.89	9.53	33.36
F04	2.54E-04	5.53	0.39	1.44	4.95E-03	0.36	**8.38E-05**
F05	**1.44E+03**	1.03E+07	1.26E+06	3.53E+06	5.69E+04	1.07E+06	5.99E+03
F06	**3.37**	1.22E+03	380.96	763.77	61.54	259.27	23.96
F07	**452.71**	2.57E+06	3.12E+05	8.54E+05	1.39E+04	2.68E+05	1.13E+03
F08	**1.23**	392.95	158.20	270.14	16.87	89.89	14.22
F09	6.05E-07	0.17	**0.00**	0.06	3.44E-04	7.97E-03	2.99E-05
F10	3.09E+04	1.87E+09	1.94E+08	6.39E+08	2.12E+06	1.63E+08	**7.25E+03**
F11	9.90E+07	5.38E+09	7.03E+13	3.24E+09	**1.28E+07**	1.59E+08	3.34E+13
F12	**126.55**	992.48	840.10	918.32	228.83	326.79	485.61
F13	**193.15**	1.30E+04	3.21E+03	7.55E+03	840.84	2.92E+03	210.22
F14	**2.51**	46.40	24.90	35.24	16.20	18.18	6.83
F15	1.25E+05	1.43E+05	6.97E+04	1.19E+05	1.11E+05	1.82E+05	**6.81E+03**
F16	2.93	9.22	7.04	8.26	7.27	6.90	**1.94**
F17	**1.63**	7.40E+06	158.40	637.71	45.25	70.98	37.93
F18	**5.88E+03**	2.42E+04	2.99E+04	2.36E+04	6.54E+03	7.21E+03	3.28E+04
F19	**0.62**	352.78	112.02	223.89	17.71	75.92	6.60
F20	33.87	341.02	**31.69**	274.79	91.08	118.43	387.02
F21	895.19	1.59E+03	**660.56**	1.29E+03	1.57E+03	1.40E+03	4.79E+09
Time	228.90	**119.65**	451.90	156.50	147.88	273.86	130.85
Total	**13**	1	3	0	1	0	4

Table 3. Normalized figures for results in Table 2

	MS	ABC	CS	DE	GA	HS	PSO
F01	**1.00**	7.22	6.27	6.90	3.71	5.45	2.88
F02	**1.00**	632.49	443.15	565.65	87.23	198.93	152.11
F03	**1.00**	34.40	60.05	32.36	7.80	12.63	44.20
F04	3.03	6.60E+04	4.63E+03	1.72E+04	59.14	4.33E+03	**1.00**
F05	**1.00**	7.14E+03	874.68	2.45E+03	39.39	741.55	4.15
F06	**1.00**	361.37	113.07	226.68	18.26	76.95	7.11
F07	**1.00**	5.69E+03	689.68	1.89E+03	30.81	592.81	2.49
F08	**1.00**	318.99	128.42	219.29	13.70	72.97	11.54
F09	1.00	2.80E+05	**0.00**	9.11E+04	568.26	1.32E+04	49.37
F10	4.27	2.58E+05	2.68E+04	8.81E+04	292.30	2.25E+04	**1.00**
F11	7.75	421.28	5.51E+06	253.78	**1.00**	12.43	2.61E+06
F12	**1.00**	7.84	6.64	7.26	1.81	2.58	3.84
F13	**1.00**	67.21	16.64	39.10	4.35	15.12	1.09
F14	**1.00**	18.48	9.92	14.04	6.45	7.24	2.72
F15	18.35	21.05	10.23	17.48	16.27	26.72	**1.00**
F16	1.51	4.74	3.62	4.25	3.74	3.55	**1.00**
F17	**1.00**	4.53E+06	97.07	390.81	27.73	43.50	23.24
F18	**1.00**	4.11	5.08	4.01	1.11	1.23	5.58
F19	**1.00**	571.33	181.41	362.59	28.69	122.95	10.69
F20	1.07	10.76	**1.00**	8.67	2.87	3.74	12.21
F21	1.36	2.40	**1.00**	1.95	2.38	2.13	7.26E+06
Time	1.91	**1.00**	3.78	1.31	1.24	2.29	1.09
Total	**13**	1	3	0	1	0	4

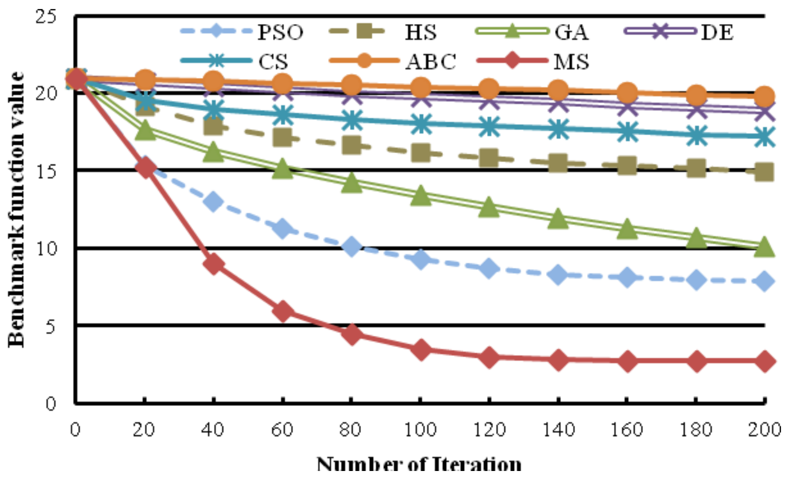


Figure 2. Graphs for F1: Ackley function (N=100, T=200, D=100)

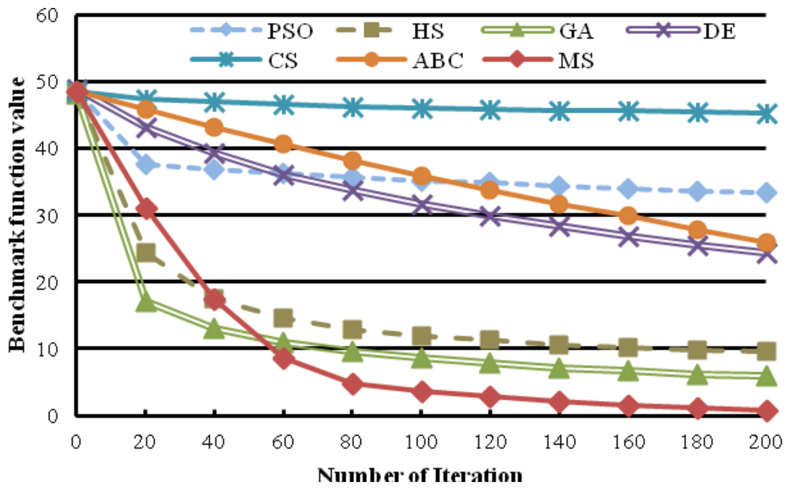


Figure 3. Graphs for F3: CosineMixture function (N=100, T=200, D=100)

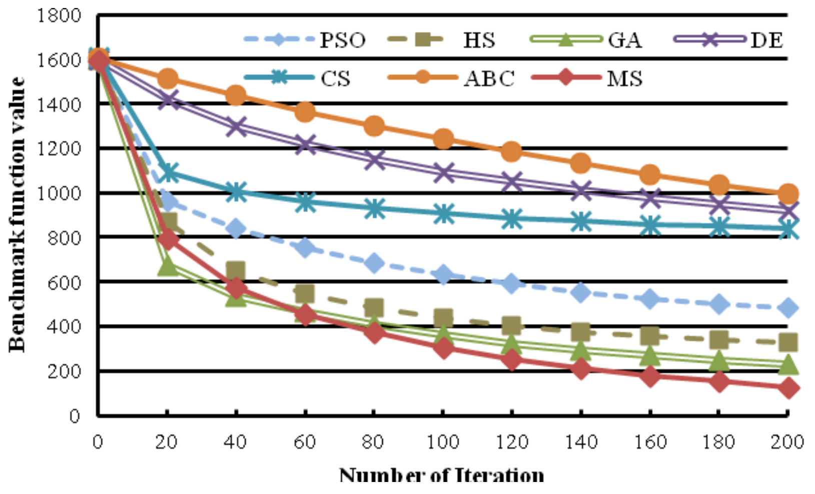


Figure 4. Graphs for F12: Rastrigin function (N=100, T=200, D=100)

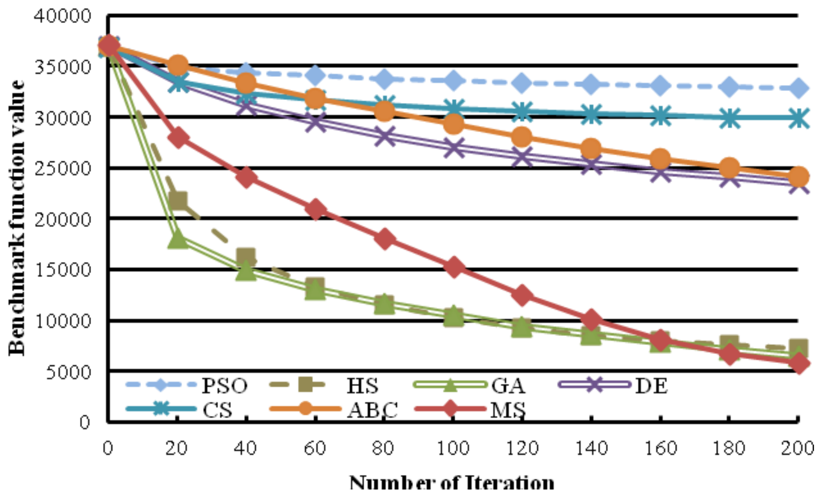


Figure 5. Graphs for F17: Schwefel2.26 function (N=100, T=200, D=100)

In Figure 6, the number of generations increases and the population size decreases moving from right to left, and conversely, the number of generations decreases and the population size increases moving from left to right. Since the value of *NT* is the same at any point in the graph, computation time is essentially unchanged, but as the number of search points increases, the amount of required memory increases. It can therefore be said that an algorithm with good performance at the left end of the graph is desirable. Moreover, from the viewpoint of ease of parameter setting, good performance at the left end is likewise desirable.

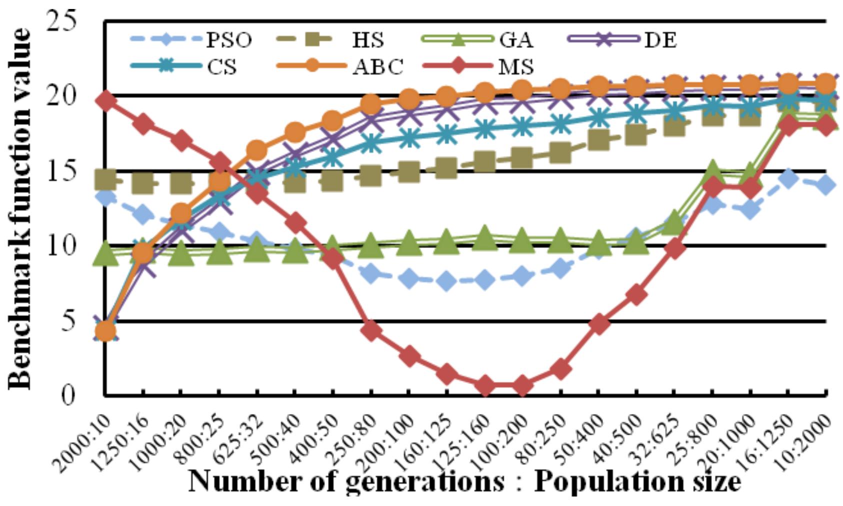


Figure 6. Graphs for F1: Ackley function (NT=20,000, D=100)

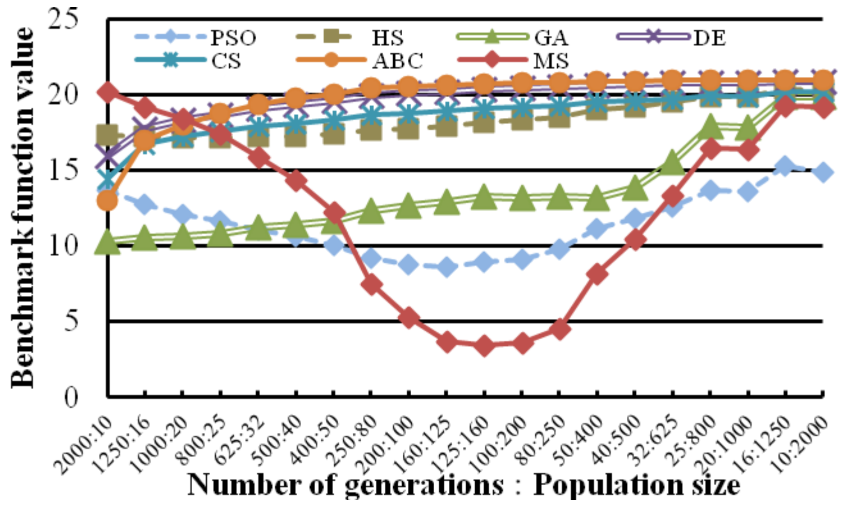


Figure 7. Graphs for F1: Ackley function (NT=20,000, D=400)

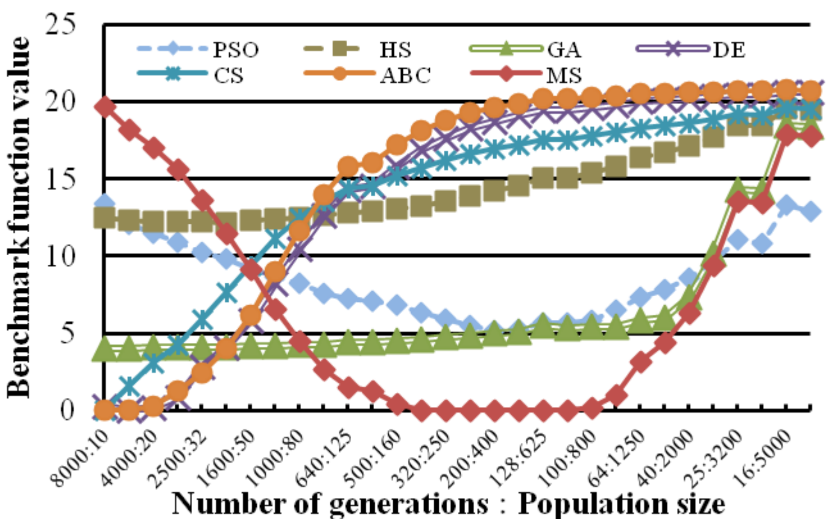


Figure 8. Graphs for F1: Ackley function (NT=80,000, D=100)

Table 4 summarizes the performance of each algorithm as obtained from the results shown in Figures 6 to 8.

Although the MS algorithm shows superior performance near the center of the graph (*N*:*T* close to 1:1), its performance deteriorates as the value of *N*:*T* becomes further removed from 1:1. A key property of the MS algorithm is early convergence of search points, and as a result, it can easily lose diversity in the search point population. It is therefore considered that a certain number of search points are needed to demonstrate good performance. However, excessively increasing the number of search points increases the number of generations needed to converge to an approximate solution, but in this experiment, the maximum number of generations drops off on approaching the right end of the graph, which explains the degraded performance in this area. When executing the MS algorithm under conditions of *N*:*T* near 1:1, the computational complexity can be decided beforehand, and from that, the specific number of search points and maximum number of generations can be determined. It can therefore be seen that this approach would not apply to the case of "continue calculations until obtaining a solution for a certain evaluation value."

Table 4. Algorithm performance trends

Algorithm	Location in graph of best performance	Note
MS	Near center	Large change compared with other algorithms
ABC	Left end	Few functions for which performance deteriorates
CS	Left end	Performance improves near graph center depending on the function
DE	Left end	Has properties similar to ABC but performance greatly deteriorates for some functions
GA	Slightly right of center	Nearly flat from left end to points near center
HS	Left end	Slight deterioration in performance at right of graph
PSO	Near center	Change is small compared with other algorithms

CONCLUSIONS

We proposed an algorithm called the MS (mean search) algorithm that excels in fast convergence at an early stage. However, while we have shown that the MS algorithm is superior to other algorithms in terms of speed of convergence, its key property of using mean values between search points in update processing presents a problem. Specifically, for the situation in which optimal solutions exist near boundary values with no nearby semi-optimal solutions, the MS algorithm can fall into local optima without doing a sufficiently extensive search. Additionally, as mutation processing in the MS algorithm makes use of the distance between search points, the variation width of search-point coordinates depends heavily on this distance between search points. Consequently, if the centralization of search points progresses too fast at an early stage, it will be difficult for search points to escape from local optima. To solve this problem, we can consider various methods, such as rearranging search points after search points have been somewhat centralized and solution updating has been terminated, or switching to another algorithm on termination of solution updating. Given that the MS algorithm is superior to other algorithms form the viewpoint of convergence speed, we consider that it would be applicable to a hybrid method that includes switching to another algorithm.

As a future research topic, we aim to develop a more robust and versatile algorithm by rearranging search points or using the MS algorithm in combination with another algorithm in a hybrid approach and thereby overcome the problem associated with termination of solution updating.

REFERENCES

[1] Basturk, B. & Karaboga, D. (2006) An artificial bee colony (ABC) algorithm for numeric function optimization, *IEEE Swarm Intelligence Sympousium, May 12-14, Indianapolis, Indiana, USA.*

[2] Blum, C., & Roli, A. (2003) Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, *ACM Computing Surveys*, **35**-3, September, 268-308.

[3] Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Annarbor, MI, USA.

[4] Karaboga, D. (2005) An idea based on honey bee swarm for numerical optimization, *Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, Turkey.*

[5] Kennedy, J. & Eberhart, R. C. (1995) Particle Swarm Optimization, *Proceedings of the IEEE International Conference on Neural Networks*, 1942-1948.

[6] Kennedy, J. & Eberhart, R. C. (2001) *SWARM INTELLIGENCE*, Academic Press.

[7] Loganathan, G.V., Kim, J. H., & Geem, Z. W. (2001) A New Heuristic Optimization Algorithm: Harmony Search, *SIMULATION*, *76*, 60-68.

[8] Storn, R. M., & Price, K. V. (1997) Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Space, *Journal of Global Optimization*, *11*, 341-359.

[9] Xin-She Yang & Suash Deb (2009) Cuckoo Search via Levy Flights, *Nature & Biologically Inspired Computing*, 210-214.