IEICE TRANSACTIONS

on Information and Systems

VOL. E105-D NO. 2
FEBRUARY 2022

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Novel Metaheuristic: Spy Algorithm

Dhidhi PAMBUDI[†,††], *Student Member and* **Masaki KAWAMURA**[†a)], *Senior Member*

SUMMARY We proposed a population-based metaheuristic called the spy algorithm for solving optimization problems and evaluated its performance. The design of our spy algorithm ensures the benefit of exploration and exploitation as well as cooperative and non-cooperative searches in each iteration. We compared the spy algorithm with genetic algorithm, improved harmony search, and particle swarm optimization on a set of non-convex functions that focus on accuracy, the ability of detecting many global optimum points, and computation time. From statistical analysis results, the spy algorithm outperformed the other algorithms. The spy algorithm had the best accuracy and detected more global optimum points within less computation time, indicating that our spy algorithm is more robust and faster then these other algorithms.

key words: *population, metaheuristic, optimization*

1. Introduction

Humans have long benefited from optimization. Optimization will always be needed in line with our desire to always obtain the best solutions by considering the constraints. We can find optimization cases in almost all fields. Optimization is common in the fields of engineering design, management, economics, physics, and biology. One of the optimization applications is the very large-scale integration (VLSI) design [1] for creating an integrated circuit (IC) that supports hardware and technology development. The demand to obtain a high-capability chip requires increasingly more components to be embedded. This is made even more complicated by demands that the chip size should be small, consume little power, yet work fast and be able to handle noise. All processes must be done in a short time if the manufacturer wants to launch a new chip by adjusting the time-to-market strategy.

Solving optimization cases commonly starts with modeling the problem to make it easier to understand. Unfortunately, the obtained models often tend to be complicated and hard to solve. As optimization is equal to a decision problems, a large-scale model often take too long to solve. Moreover, many problems are intractable and categorized as non-deterministic polynomial (NP)-complete problem [2]. VLSI design is just one real-world application of the graph-partitioning problem which is categorized as NP-complete. One of the basic partitioning problems is the maximum cut problem which is included in the original Karp's 21 NP-complete problems [3] and is equivalent to the Ising model [4], which is very influential in the fields of physics and mechanical statistics. Another example of an NP-complete problem that exists is complex job schedulling.

In optimization, it is common to pursue accurate solutions, but it is also practical to find a solution within a reasonable and acceptable time. We do not want to waste time in a highly competitive world. The demand for obtaining good solutions in a short time leads us to make a compromise and accept the solution obtained through an approximation approach. A well-known approximation methods are metaheuristic algorithms, which usually includes stochastic search. Compared with deterministic searches, metaheuristics have the advantages that they do not require any information about the function to be optimized. Many metaheuristics do not necessarily take into account the gradient of the function. Most are also flexible or problem-independent which means they can be applied to various kind of problems.

The advantages of metaheuristics has led to many such algorithms being proposed. One of the most well-known metaheuristic algorithms is the genetic algorithm (GA) which was inspired by Darwin's theory of evolution [5]. It mimics the natural evolution of a population by the process of solution reproductions, creates new solutions, and competes for survival [6] based on the operators of crossover, mutation, and sometimes elitism [7]. The GA has been used in various fields and is able to provide good solutions in many areas. Another well-known algorithm is particle swarm optimization (PSO), which was originally intended for simulating social behavior, as it mimics the movement of organisms such as a flock of bird or school of fish [8]. PSO is a more recent algorithm than GA and can be an alternative to GA since it is simpler and converges faster than GA [9], [10]. Another more recent metaheuristic algorithm that has attracted much attention is improved harmony search (IHS). IHS is a modification of HS, the process of which is inspired by a jazz musician finding a good harmony of musical notes [11]–[13].

Metaheuristic algorithms have an important role in optimization in academic research and in real-world practice. Many metaheuristic algorithms have been introduced, but the demand for better, more accurate and faster algorithms continues. Since there are various goals in the optimization

Manuscript received April 26, 2021.
Manuscript revised September 16, 2021.
Manuscript publicized November 1, 2021.

[†]The authors are GSTI, Yamaguchi University, Yamaguchi-shi, 753–8512 Japan.
[††]The author is Dept. of Mathematics Education, FKIP, Sebelas Maret University, Indonesia.
a) E-mail: m.kawamura@m.ieice.org

DOI: 10.1587/transinf.2021EDP7092

problems, it is difficult to develop a single algorithm that can solve a wide range of problems and always outperform other algorithms. On the basis of the 'no free lunch theorem' [14], it is challenging to derive a single metaheuristic algorithm that fits any problems, since a certain metaheuristic algorithm may be strong for certain problems but weak for others.

The major challenge in the field of optimization is to develop robust algorithm that can solve a wide range of problem types as accurately and efficiently as possible. A robust algorithm is one that ensures convergence even when starting from an arbitrary initial solution [15], [16]. Even after many runs, the obtained solutions should not be sensitive to parameters [17] and have low variation [18]. Certain fast algorithms are so focused on speed that they lack robustness [18].

In an effort to develop a robust metaheuristic algorithm, we propose an algorithm called the spy algorithm. The spy algorithm is a population-based metaheuristic algorithm that mimics the strategy of a group of spies, the spy ring. Within a spy ring, each spy agent considers three types of movements, which are: (i) movement within a small perimeter, (ii) movement by considering another spy agent, or (iii) just making a random move.

To demonstrate its robustness, it was tested to solve two types of problems; (i) optimization problem with multidimensional function, and (ii) multimodal optimization. The non-convex functions were used on the test as non-convex optimizations are NP-hard problem [19], so they are suitable for testing metaheuristic algorithms. A total of 12 standard benchmark functions were used to demonstrate the performance of the spy algorithm and the results were compared with the GA, IHS, and PSO.

2. Metaheuristic Algorithm

Many metaheuristic algorithms have been introduced. Some are categorized as population-based and others are categorized as single-trajectory-based [20], [21]. The searching with single-trajectory-based metaheuristic algorithms manipulates and modifies a single solution point in every iteration. The well-known simulated annealing (SA) [22] is an example of a successful single-trajectory-based metaheuristic algorithm. In contrast, the population-based metaheuristic algorithms combine a set of points to create new solutions in every iteration.

A metaheuristic algorithm usually consists of two components, i.e., exploration and exploitation. Exploration means searching for solutions in the global space while exploitation means searching for solutions by focusing in a small area or an area near an already known solution. The single-trajectory-based metaheuristic algorithm is exploitation oriented [20]. Population-based metaheuristic algorithm executes searching by using many points distributed on all search spaces, therefore it is exploration oriented [20]. Some metaheuristic algorithms use both exploration and exploitation orientation. To obtain a good result, exploration and exploitation should be balanced by choosing the right value of algorithm parameters. However, this is sometimes time consuming.

A metaheuristic algorithm may be inspired by a unique phenomenon that results in different strategies. Some metaheuristic algorithms may have simple processes while others do not. Although many metaheuristic algorithms have different processes, regardless of whether it is population-based or single-trajectory-based, the framework in most consists of the following five main steps.

1. **Initialization**. Initialization is commonly started from random positions since there is no information about the solution space. However, an initialization method, such as the greedy procedure in GRASP [23], [24], can be used.
2. **Solution refinement**. Better solutions are generated in this step. Each metaheuristic algorithm has its own refinement strategy inspired by real-world natural phenomena, such as ant colony optimization [25] and cuckoo search [21]. The new solutions that are expected to be better than the previous solutions are produced. In exceptional cases, algorithms such as SA can temporarily result in worse solutions [26]. Based on its strategies, metaheuristic algorithms, such as the GA and HS, may apply sorting mechanisms [27].
3. **Solution update**. When a new solution was created from the previous solution by refinement strategy, the algorithm should update its solution by selecting a new one or old one on the basis of a certain rule. The updated solutions will be transferred to the next iteration.
4. **Termination**. As the solution needs to be obtained in a reasonable time, the algorithm should be stopped in a reasonable manner. Usually, the algorithm is terminated when the solutions are not expected to improve. In addition, a maximum number of iterations is set to avoid an infinite loop.
5. **Finalization**. After the algorithm is stopped, one of the possible solutions is reported as the optimal solution.

The most important steps are the refinement strategy and the updating mechanism since they make a metaheuristic algorithm unique from others. The other steps are almost similar in any metaheuristics algorithms. Refinement strategy focuses on the mechanism for obtaining new solutions. This step can be categorized into two strategies; cooperative, non-cooperative.

The cooperative strategy means the new solutions are obtained by involving and manipulating two or more previous solutions. An example of cooperative search is the crossover operator in the GA. The non-cooperative strategy requires only one previous solution for obtaining new solutions such as in SA [22] and tabu search [28], [29]. The previous solution is not necessary for random search.

3. Spy Algorithm

The spy algorithm was inspired by the strategy used by a

spy ring to locate an enemy base. A spy ring is a group of spies cooperating with each other by sharing intelligence. It should be noted that the strategy adopted in the spy algorithm is not the same as real-life espionage since a real spy agent, alone or in a ring, will make many considerations to determine movements.

3.1 Concepts

The spy algorithm is based on a scenario in which a country has been infiltrated by an enemy but its base is unknown so that the spy agency tries to find the enemy base. The information quality of the enemy base is evaluated by a given objective function. We consider the minimization problems. Therefore, the smaller the value of the function, the better the quality. The spy agents should follow the following steps for finding the enemy base.

1. *Initialization*. The agents are sent to random locations then evaluate the information quality of the locations by using the objective function.
2. *Refinement*
   - *Agent classification*. On the basis of the information quality, the agents are classified into three classes: high-rank, mid-rank, and low-rank. The agent which give smaller value of function is the criteria for the high-rank agent in the context of minimization. In the case of maximization, this selection can be easily switched or adjusted.
   - *Movement*. On the basis of agent class, each agent searches for a new location on the basis of the following rules.
     a. High-rank agents perform SwingMove, i.e., they move within a small perimeter on the basis of their own location.
     b. Mid-rank agents perform MoveToward, i.e., they move toward another agent location.
     c. Low-rank agents perform random search.

     After performing these movements, each agent evaluates the information quality of the new locations.
3. *Update*. If the new location has a better information quality, the agent adopts the location; otherwise goes back to the previous location.
4. *Termination*. Repeat main step 2 to 3 until stop command is issued.
5. *Finalization*. The location and its information obtained by the best agent are reported as the final answer.

In the context of optimization, a spy agent can be seen as a solution and the unknown enemy base is the optimum point. The location or position of an agent forms a solution vector and the information regarding its location is evaluated by the objective function. In this scenario, the cooperative strategy is implemented in *MoveToward* and a non-cooperative strategy is implemented into two movements, i.e., *SwingMove* and random search. To increase the convergence speed, the agent with higher rank can be chosen in the *MoveToward* movement. The movement for each agent is illustrated in Fig. 1. The updating mechanism is a one-to-one comparison between the newly generated solutions and previous ones.

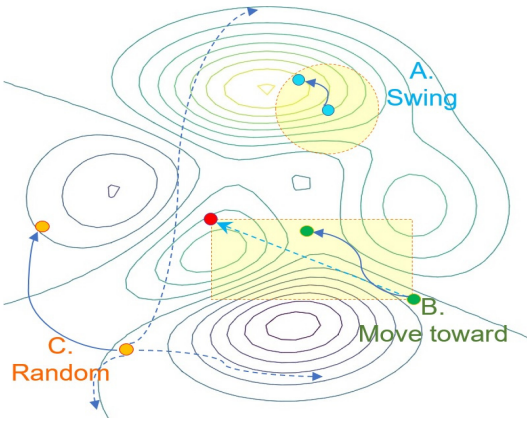


Fig. 1 Agent movements

The concept of the spy algorithm is based on that rational thinking that, on the high-rank agents, we should make a slight refinement to avoid the risk of a significant decrease in the quality of information. However, we need to make a progressive movement as well as take advantage of the benefit of the already known solutions. Implementing this movement for mid-rank agents is the right choice so that they can improve the solution obtained by considering other agents. However, the movements carried out by high and mid-rank agents may lead to the local optimum. Therefore, we need a mechanism that does not require taking into account the already obtained solutions so as not to be trapped at the local optimum. This is where the low-rank agents play the role to perform random searches so they can explore new locations. Since the random search sometimes leads to uncertain results, assigning this search to low-rank agents is appropriate. This design enables us to never lose the advantage of exploratory search even when other solutions start converging at certain points. While many traditional population-based-metaheuristic algorithms encounter decreasing benefits of exploration search as iterations increase and solutions tend to converge to certain points [30], [31], the py algorithm is able to maintain and improve the solutions that have been obtained while maintaining exploration.

The whole design of the spy algorithm is to provide assurance that each type of movement will always exist in every iteration. Each solution is only allowed to perform one type of movement and the consideration is determined on the basis of its solution quality that is converted into a ranking system. Although the spy algorithm applies a ranking system, its application differs from rank selection applied in other metaheuristics, e.g., the GA [32]. To fit the problem to be solved, the balance of occurrence of the three types of movements is set with the parameters used to

determine which solution is categorized as a high, medium, or low quality.

Exploitation is implemented by *SwingMove* while *random movement* is for exploration. The *MoveToward* tends to be exploratory at the beginning but gradually turns into exploitation as the iteration increases and the solutions get closer to each other. The ability of the spy algorithm to converge relies on *SwingMove* and *MoveToward*. This concept enables the spy algorithm to take advantage of exploration and exploitation as well as use the cooperative and non-cooperative strategies on solution refinement. The spy algorithm organizes all these aspects to occur separately while guaranteeing its presence in each iteration.

The spy algorithm can be simply implemented using sorting to arrange the solutions. There are four main parameters that affect the performance, i.e., the number of solution (*NSol*) that represents the number of spy agents, the maximum index for high-rank (*HMI*), the maximum index for mid-rank (*MMI*), and the swing factor (*SF*) to bound the perimeter. Another parameter is the number of iterations (*NI*) as a stopping criterion commonly used in metaheuristic algorithms.

3.2 Implementation

The objective function f should be defined by using variable X. We assume the dimension of variable X is $D \in \mathbb{N}$ so that $X = (x_1, x_2, \ldots, x_D)$. This variable represents the location of an agent in D dimensional space. Since there are *NSol* agents, *NSol* values of the objective function are obtained from *NSol* locations. We denote each location and its value by X^μ and $f(X^\mu)$, respectively, where $\mu = 1, 2, \ldots, NSol$. The main concepts of the spy algorithm are very simple and can be implemented in many various ways. One of its implementations can be seen on the pseudocode in Fig. 2. The ascending sorting can be used to simply determine the rank of each agent. The best agent that gives the smallest value should be placed at the first position and the largest at the last position. It should be noted that refinements can be carried out starting from the agent with the lowest to the highest rank. This strategy is to maintain the position of the higher rank agent to ensure that the rank, as well as its location, does not change before it is used as a reference by other lower rank agents. For clarity, we propose simple movements for *SwingMove* and *MoveToward* on the basis of the assumption that the algorithm works in discrete time t related to the iteration. We denote the location of the μ-th agent $X^\mu = (x_1^\mu, x_2^\mu, \ldots, x_D^\mu)$ at time t by $X^\mu(t)$, for $t = 1, 2, .., (NI - 1)$.

* SwingMove

$$X^\mu(t+1) := X^\mu(t) + rand(-1, 1)(SF/t) \tag{1}$$

* MoveToward (assume that agent X^μ move toward X^ν)

$$\nu := randint(1, \mu - 1) \tag{2}$$

$$X^\mu(t+1) := X^\mu(t) + rand(-1, 1)(X^\nu(t) - X^\mu(t)) \tag{3}$$

```
//set up the problem
objective function: a D-dimensional function f
search space I= <I_i>, where I_i=[a_i,b_i] for i ∈ D

//set up the algorithm parameter value
NSol                //number of solutions
HMI,MMI ∈ (0,1]     //as the ratio of the number of solutions
SF ∈ ℝ+             //Swing Factor
NI                  //number of iterations

HMI = int(HMI*NSol) //High.MaxIndex
MMI = int(MMI*NSol) //Mid.MaxIndex

//init
Sol = random_search(I) //create the initial solutions by global search
fSol = f(Sol)          //get the objective values
sort(Sol, fSol)        //best solution get smallest index

//refinement
for iter: 1 to NI
    for i: NSol downto (MMI-1)                  //low-rank agents
        NewSol[i] = random_search(I)
    for i: MMI downto (HMI-1)                   //mid-rank agents
        j= randint(1, i-1)         //select other solution j, 1≤j<i
        NewSol[i] = MoveToward(Sol[i], Sol[j])
    for i: HMI downto 1                         //high-rank agents
        NewSol[i] = SwingMove(Sol[i], SF)

    fNewSol=f(NewSol)                   //get the objective values

    //updating
    for i:1 to NSol
        if fNewSol[i] <= fSol[i]
            Sol[i] = NewSol[i]
            fSol[i] = fNewSol[i]
    sort(Sol,fSol)

//finalization
FinalSol=Sol[1]      //the best solution
```

Fig. 2 Pseudocode of the spy algorithm

The function $rand(-1, 1)$ is for generating a vector of real random numbers within $[-1, 1]$ that allows movement in any directions. In *MoveToward*, the considered agent is a randomly selected better agent X^ν. The function $randint(1, \mu - 1)$ is used for generating an integer random number ν within $[1, \mu - 1]$ where μ is the index of the agent that performs *MoveToward* movement. Based on the pseudocode in Fig. 2, the process of selecting agent X^ν can be done before going to the *MoveToward* function.

Considering that the new obtained solutions must be within the search space, there needs to be an additional simple mechanism to control the *SwingMove* and *MoveToward*. This mechanism returns the value of each element of the solution vector that is outside the search space to the nearest bound of the search space.

Unlike most metaheuristic algorithms, the spy algorithm implements a concept that each solution is only allowed to perform one type of movement at the refinement step, which is (*SwingMove*), (*MoveToward*), or random search. Nevertheless, the spy algorithm assures all these movements will always occur in each iteration. Therefore, a proper arrangement is needed to guarantee the occurrence of each movement. One of the strengths of population-based metaheuristic algorithms is the cooperative search which combines previously obtained solutions to create new better solutions. Considering this benefit, it is necessary to set that there is an adequate number of agents in the category of mid-rank agent to perform *MoveToward*. As *SwingMove* and *random search* are non-cooperative searches that do not take a benefit from other solutions, it is reasonable to set the

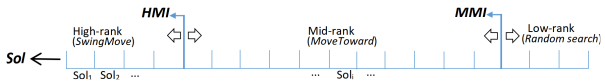


Fig. 3 Solution distribution in each category

number of high-rank and low-rank agents small. Figure 3 shows how the parameters affect the number of agents in each category.

It is common that, in all population-based algorithms, cooperative search dominates the search processes. In the case of GA, new solutions are created more through crossover than through mutation. The spy algorithm accommodates the cooperative search by *MoveToward* performed by mid-rank agents. To get the benefits of population-based algorithms, it is necessary to arrange for the spy algorithm to have a sufficient number of mid-rank agents. From our experiments, it is suggested that the number of the mid-rank agent is around 75%-95% of the total solution (*NSol*) with the rest is for either high or low-rank agents.

The value of *HMI* parameter that regulates the number of high-rank agents will affect the performance of the spy algorithm. A small *HMI* value will make the spy algorithm run longer than using a large *HMI*. A large *HMI* value will reduce the number of cooperative searches so that the algorithm can run faster but also risks reducing accuracy. In the tuning process, the accuracy decreases quite a lot when the *HMI* value is more than 0.15 which means the number of high-quality agents is 15% of the total solution (*NSol*). Note that in order to still get the benefits of exploratory search, the spy algorithm needs to maintain the presence of the low-rank agents category.

We propose two variants of the spy algorithm. The first variant uses only a single agent in the high-rank category. Since the GA variant may use elitism where the algorithm preserves the best individual and pass it over to the new generation, this idea became a motivation to be loosely adapted into the spy algorithm. However, GA and the spy algorithm apply a different strategy. The best solution in GA may not be modified as the changes only apply to the selected solutions, but the spy algorithm always tries to improve each solution including the best one. Another inspiration came from PSO where the swarm move by considering the global-best location. This situation also inspired the global-best HS which is a variant of IHS [13]. As the best solution should be only a single solution, we designed the first variant of the spy algorithm where the high-rank category consists of a single solution only. This case is also common in spy agencies where each agent competes to be the best one. Applying a single high-rank agent will result in increasing the progressive search implemented by *MoveToward* on mid-rank agents. This design will increase the occurrence of referrals to the best agent in the solution refinement step. The first variant can be obtained by setting parameter $HMI = 1/NSol$ or by setting in the code so the number of the high-rank agent is directly set to 1. To make it simple, the first variant is referred as Spy1. The second variant is the case where the high-rank category consists of more than one agents, and it is referred as Spy2.

The Spy2 allows the algorithm to have several distinct solutions that are considered high quality. In the case of a problem having many solutions, it is advantageous to have many agents to exploit many basins. In the context of spy algorithm, exploitation is accommodated by *SwingMove* which is performed by high-rank agents. When some agents have more information than others, they have a greater chance of finding the best solution. Therefore, a slight movement is recommended for those agents rather than always moving progressively and exploring large areas. Considering the whole process of the spy algorithm, increasing the number of high-rank agents will reduce the number of mid-rank agents assuming that the low-rank agents are constant. The result is an increase in the occurrence of *SwingMove* and a decrease in *MoveToward*. Since the *SwingMove* process is simpler than *MoveToward*, the algorithm may run faster.

4. Evaluation

4.1 Test Condition

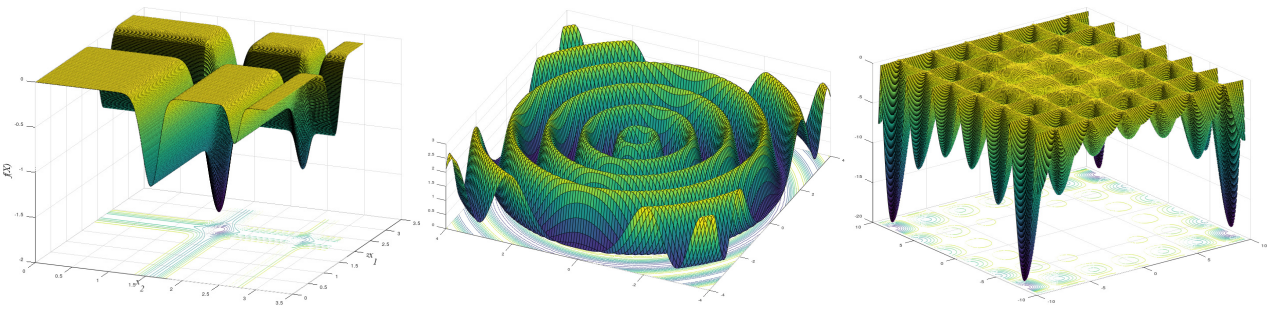
Two tests were conducted for investigating the performance of the spy algorithm, i.e., (i) optimization on multi-dimensional function, and (ii) multimodal optimization. Test (i) also involved multimodal functions, but the focus was for finding the global optimum. Test (ii) focused on finding all global optimums, which means how many global optimum points can be detected with a certain algorithm. For comparison, Spy1 and Spy2 were tested with the three population-based metaheuristic algorithms of the GA [33], IHS [12], and PSO [34]. A real-coded GA with simple arithmetic operators was used to give an equal condition as the other tested algorithms used simple arithmetic operators as well. Tournament selection was used in the GA because of its stability compared with roulette wheel selection. The elitism was applied to GA. All tested algorithms were in their basic version and did not use any specific approach enabling us to investigate the original potential of each algorithm.

The optimum points for each test function were known so that the performance for each algorithm could be properly measured. We made all tested functions have an optimum value of 0 by normalizing several functions. On test (i), we set the dimension size on 30 to gain adequate insight into how the algorithm works on large dimensional problems. The dimension size was set to 2 for test (ii) so we could get visual results for better understanding. All test functions are non-convex functions and listed in Table 1. The plot of some 2-dimensional version of the test functions are shown in Fig. 4. Unlike optimizations on convex functions, which can be solved in polynomial time, optimizations on non-convex functions are more difficult to solve since there are many local optimums, valleys, or plateaus that can trap the algorithm so that it fails to find a global optimum.

Table 1 Test functions

Test	Name	Test function	Dim (D)	Interval I_i	GOP
(i)	Michalewicz*	$f_1 = -\sum_{i=1}^{n} \sin(x_i) sin^{20}(\frac{ix_i^2}{\pi})$	30	$[0, \pi]$	1
	Rosenbrock	$f_2 = \sum_{i=1}^{n} (100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2)$	30	[0, 10]	1
	Alpine01	$f_3 = \sum_{i=1}^{n} \lvert x_i \sin(x_i) + 0.1x_i \rvert$	30	[-10, 10]	1
	Ackley	$f_4 = -20e^{\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)} - e^{\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)} + 20 + e$	30	[-30, 30]	1
	Salomon	$f_5 = 1 - \cos\left(2\pi\sqrt{\sum_{i-1}^{n} x_i^2}\right) + 0.1\sqrt{\sum_{i-1}^{n} x_i^2}$	30	[-100, 100]	1
	Griewank	$f_6 = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(x_i/\sqrt{i})$	30	[-600, 600]	1
(ii)	Bird*	$f_7 = \left(\sin x_1 e^{(1-\cos x_2)^2} + \cos x_2 e^{(1-\sin x_1)^2} + (x_1 - x_2)^2\right)$	2	$[-2\pi, 2\pi]$	2
	Cross in Tray*	$f_8 = -0.0001\left(\left\lvert \sin x_1 \sin x_2 e^{\left\lvert 100 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert}\right\rvert + 1\right)^{0.1}$	2	[-10, 10]	4
	Holder Table*	$f_9 = -\left\lvert \sin x_1 \cos x_2 e^{\left\lvert 1 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert}\right\rvert$	2	[-9.7, 9.7]	4
	Himmelblau	$f_{10} = \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + x_2^2 - 7\right)^2$	2	[-6, 6]	4
	Shubert*	$f_{11} = \prod_{i=1}^{n}\left(\sum_{j=1}^{5} j\cos((j+1)x_i + j)\right)$	2	[-10, 10]	18
	Inv. Vincent	$f_{12} = \frac{1}{n}\sum_{i=1}^{n} \sin(10\log x_i)$	2	[0.2, 10]	36

*: normalized by subtracting it with the optimum value



(a) f_1, Michalewicz (b) f_5, Salomon (c) f_9, Holder Table

Fig. 4 2-dimensional version of the test functions

The optimization of the non-convex function is categorized as NP-hard [19], so it is suitable for testing metaheuristic algorithms.

To adjust to the test and the characteristics of the problem, a different set of algorithm parameter values was tuned. The solution (population) size was set to be equal for each algorithm. The *NI* was set at 50 times the size of the problem dimension. Since IHS only generates one new solution per iteration, the *NI* for IHS was set at 50 times the dimension size times the harmony memory size (HMS) to make it equal. Other algorithm parameter values are listed in Table 2. These values were tuned to obtain an equal condition that takes into account the number of function evaluations and the search balance. All algorithms were implemented in Python code. The code was run using Python 3.9.4 on a Windows 10 PC powered by Intel i7-9750H and 16-GB of RAM.

Each algorithm was run in 100 independent repetitions to obtain sufficient data to observe its behavior. The performance criteria were mainly based on the average error and its standard deviation, but because test (ii) is also for investigating the potential for finding all global optimum points (GOPs), it has an additional criterion, which is the maximum peak ratio (MPR).

Table 2 Algorithm parameters

GA	IHS	PSO	Spy1	Spy2
Pop= 40	HMS=40	NSwarm=40	NSol=40	NSol=40
Tourn= 10	HMCR= 0.85	c1= 0.9	HMI= 1/NSol$^{(*)}$	HMI= 0.1
Parent= 10	minPAR= 0.7	c2= 1	MMI=0.9	MMI= 0.9
pc= 0.9	maxPAR= 0.85		SF=1	SF=1
pm= 0.2	minBW= 0.5			
StepSize=1	maxBW= 1.5			
NI= 50D	NI= 50D*HMS	NI= 50D	NI= 50D	NI= 50D

D: dimension size
(*): the high-rank agent can also directly be set fix at 1

Table 3 Average error ± Standard deviation

f	GA	IHS	PSO	Spy1	Spy2
f_1	**3.501** ±0.564	18.556 ±0.423	18.181 ±0.776	9.311 ±3.378	12.387 ±1.461
f_2	21.674 ±10.316	98.344 ±60.779	477.124 ±1429.226	**17.157** ±29.856	22.856 ±28.084
f_3	0.284 ±0.141	1.922 ±0.267	1.229 ±1.681	**0.019** 0.079	0.250 0.999
f_4	0.431 ±1.742	2.396 ±0.115	1.434 ±0.776	**7.617e-6** ±.972e-6	4.213e-4 ±1.504e-4
f_5	10.909 ±2.170	**0.285** ±0.031	0.856 ±0.336	0.426 ±0.056	0.62 ±0.077
f_6	24.640 ±20.167	0.203 ±0.115	1.013 ±0.195	0.004 ±0.011	**0.002** ±0.008
f_7	1.559 ±7.155	3.595 ±7.521	0.053 ±0.140	1.041e-6 ±5.553e-6	**1.415e-7** ±1.045e-6
f_8	2.855e-7 ±4.595e-7	3.973e-4 ±0.003	5.553e-5 ±7.386e-5	3.592e-9 ±2.251e-8	**5.826e-10** ±3.681e-9
f_9	1.368 ±3.318	0.199 ±1.364	0.011 ±0.004	2.186e-6 ±6.678e-6	**2.983e-7** ±1.442e-6
f_{10}	2.286e-4 ±4.968e-4	0.014 ±0.014	0.007 ±0.008	8.502e-7 ±4.269e-6	**6.126e-7** ±2.536e-6
f_{11}	3.207 ±13.836	1.083 ±1.057	0.961 ±2.216	0.003 ±0.004	**7.885e-4** ±0.001
f_{12}	0.001 ±0.010	0.002 ±0.007	0.007 ±0.021	7.759e-7 ±2.800e-6	**4.342e-7** ±2.139e-6

***in bold**: smallest among others

$$MPR = \frac{Number\ of\ detected\ GOPs}{Number\ of\ all\ actual\ GOPs}. \quad (4)$$

When a solution falls near a certain GOP location which means the error E is less than ε, that is,

$$E = \|X_{opt} - \hat{X}\| < \varepsilon, \quad \varepsilon \in \mathbb{R}^+, \quad (5)$$

and falls within the same narrow basin of the GOP, we count it as able to detect the GOP even though the obtained solution might differ from the exact one. We used $\varepsilon = 0.1$ for test (ii) considering that all tested algorithms were set to use a small number of solutions and iterations. The main purpose of test (ii) is to investigate the ability of the tested algorithms to distribute their set of solutions to reach many distinct GOPs. The last criteria for each test is the computation time taken by the algorithm.

4.2 Experimental Result

We tested the spy algorithm, GA, IHS, and PSO on various problems having various characteristics to investigate their accuracy, speed, and robustness. The descriptive statistical results are listed in Table 3. The spy algorithm provided the most accurate results in most test functions. Even though the spy algorithm performed poorly in two test functions (f_1 and f_5), Spy1 and Spy2 did not perform the worst. The charts of the average error are shown in Fig. 5. From these results, the spy algorithm tended to have small errors and small standard deviation. These results differed from GA, IHS, and PSO whose results tended to vary markedly at different test functions. The results indicate that the spy algorithm was more robust than the other tested algorithms.

Of all tests, the spy algorithm performed worse in accuracy for the Michalewicz (f_1) function. This algorithm was worse than the GA, although it was better than IHS and PSO. As the representation of the characteristic, the 2-dimensional Michalewicz function has a contour in the form of valleys as well as plateaus which can trap the algorithm so that it fails to approach the global optimum point. Such a characteristic did not exist in other tested functions. On the Michalewicz function, GA showed its superiority over the others. For functions similar to the Michalewicz function, the spy algorithm may perform worse than the GA. Considering that the spy algorithm was better than IHS and PSO in this case, it is possible to improve the performance of the spy algorithm by adjusting the parameters.

To provide a more in-depth investigation in the differences of each algorithm, we conducted a Kruskal-Wallis H test using $\alpha = 0.05$ on each test function. Our test results on each test function indicate that there were significant differences among these algorithms. To know the detail, we conducted the Gomes-Howell post hoc comparison. The P-values of this comparison are listed in Table 4. Considering the average error values and these P-values, the spy algorithm significantly gave the best results or was always in the best group. These results also showed that the two variants of the spy algorithm had significant differences on several test functions. Table 3 shows that Spy1 tended to be better than Spy2 on the high-dimensional test functions while Spy2 was slightly better than Spy1 on the 2-dimensional test functions.

Without changing the algorithm parameter values, on test (ii), we tested all algorithms to detect as many GOPs as possible. Combined with test (i), test (ii) will provide a more deep investigation of the robustness of the algorithm. The visual results of detecting GOPs on f_{11} are shown in Fig. 6. The diamonds denote that there are solutions that fall near the location of the global optimum, triangles indicate that no solution lies near the considered location of the global optimum. The averages of all maximum peak ratios (MPRs) are listed in Table 5. In this test, the GA and IHS performed poorly with low MPR results. These results indicate that the GA and IHS were weak in detecting many GOPs and tended to converge to a few. PSO provided a good enough MPR, so that it was suitable to use to find many GOPs. From these results, the spy algorithm was outperformed PSO, as seen from the MPR, which was greater than that of PSO. Both Spy1 and Spy2 had similar accuracy in detecting many GOPs.

In terms of time performance, we computed the aggregate computation times regardless of the test function. Although the complexity of the test function affected computation time, the differences were not very large. As the time difference is more affected by the dimensions of the problem, we separated test (i) that applied the dimension size of 30 and test (ii) that applied the dimension size of 2. The computation times are shown in the boxplot chart in Fig. 7. We again performed Kruskal-Wallis H test using $\alpha = 0.05$ to compare computation times. As the output

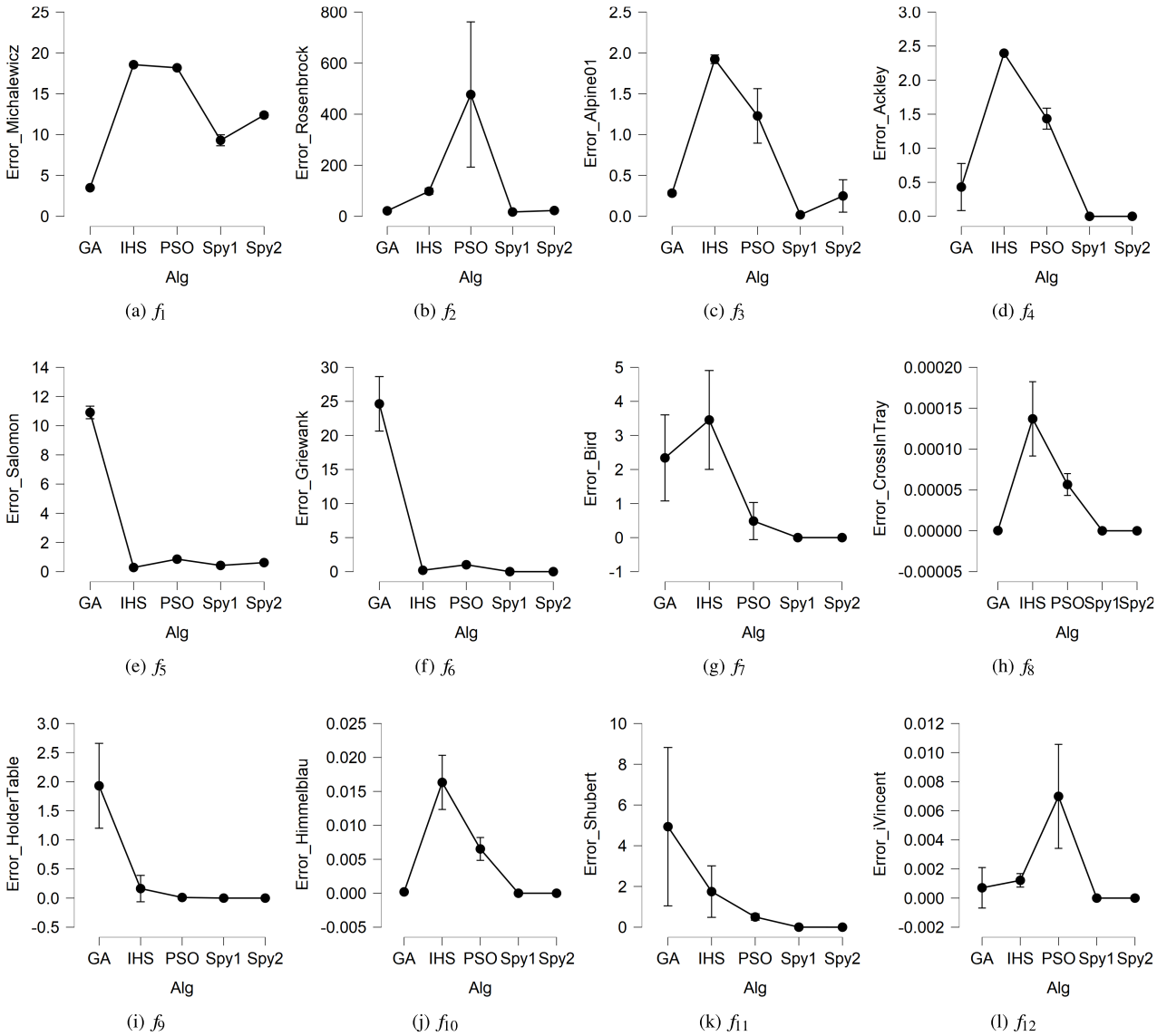


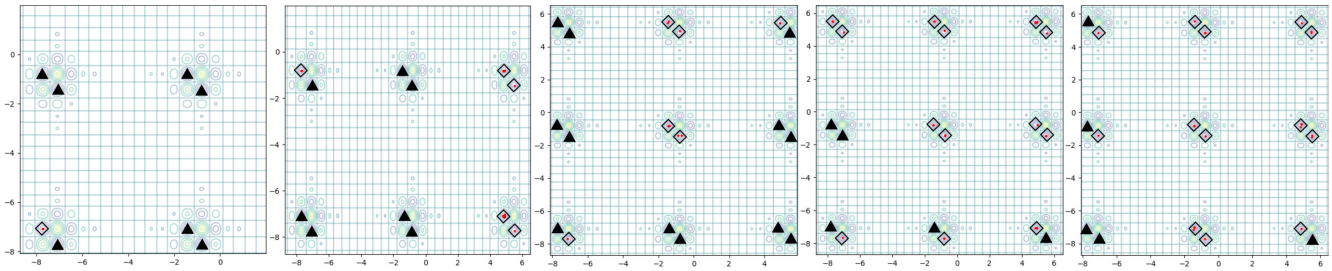
Fig. 5 Error charts

Table 4 P-values of Games-Howell post hoc error comparison

Comparison	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}
GA - IHS	<.001	<.001	<.001	<.001	<.001	<.001	0.780	<.001	<.001	<.001	0.534	0.958
GA - PSO	<.001	0.017	<.001	<.001	<.001	<.001	0.063	<.001	<.001	<.001	0.167	0.013
GA - Spy1	<.001	0.609	<.001	0.106	<.001	<.001	0.003	<.001	<.001	<.001	0.095	0.851
GA - Spy2	<.001	0.995	0.997	0.106	<.001	<.001	0.003	<.001	<.001	<.001	0.095	0.851
IHS - PSO	<.001	0.071	<.001	<.001	<.001	<.001	0.002	0.009	0.669	<.001	0.303	0.016
IHS - Spy1	<.001	<.001	<.001	<.001	<.001	<.001	<.001	<.001	0.609	<.001	0.055	<.001
IHS - Spy2	<.001	<.001	<.001	<.001	<.001	<.001	<.001	<.001	0.609	<.001	0.055	<.001
PSO - Spy1	<.001	0.015	<.001	<.001	<.001	<.001	0.398	<.001	<.001	<.001	<.001	0.002
PSO - Spy2	<.001	0.017	<.001	<.001	<.001	<.001	0.398	<.001	<.001	<.001	<.001	0.002
Spy1 - Spy2	<.001	0.634	0.154	<.001	<.001	0.471	0.613	0.383	0.058	1.000	0.012	0.951

The difference is significant if $P - value < \alpha$, ($\alpha = 0.05$)

of Kruskal-Wallis H test showed that there were significant differences, we followed it up with a post hoc comparison. The results are listed in Table 6. From these P-values, the only one insignificant difference was between IHS and Spy1 for the dimension size of 2. From the results in Fig. 7 and Table 6, the spy algorithm was the fastest among the algorithms. Another important result was that Spy2 was faster than Spy1. The shortest computation time of Spy2 could be easily understood because Spy2 used more *SwingMove* and less *MoveToward* than Spy1. *SwingMove* took less



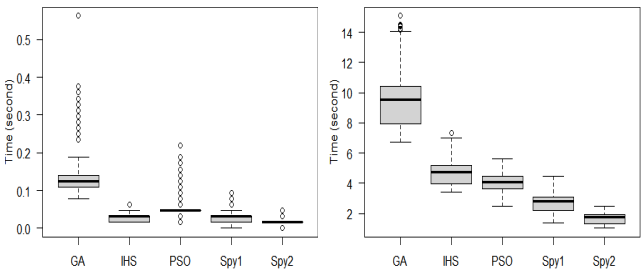
(a) GA, detected=1 (b) IHS, detected=5 (c) PSO, detected=6 (d) Spy1, detected=13 (e) Spy2, detected=13

Fig. 6 Detecting global optimums on Shubert function having 18 GOPs (red dots are solutions, diamonds means the solutions lie near a certain GOP while triangles are missed GOPs)

Table 5 Average MPR

f	GA	IHS	PSO	Spy1	Spy2
f_7	0.47	0.49	0.495	0.99	0.96
f_8	0.25	0.2575	0.5675	0.925	0.9075
f_9	0.1425	0.2375	0.2525	0.9875	1
f_{10}	0.25	0.255	0.305	0.7525	0.78
f_{11}	0.054	0.0961	0.18	0.4556	0.4828
f_{12}	0.0272	0.0464	0.1228	0.2925	0.2903

*larger is better, maximum is 1



(a) $D = 2$ (b) $D = 30$

Fig. 7 Boxplots of computation time

Table 6 P-values of post hoc time comparison

Comparison	$D = 2$	$D = 30$
GA - IHS	<.001	<.001
GA - PSO	<.001	<.001
GA - Spy1	<.001	<.001
GA - Spy2	<.001	<.001
IHS - PSO	<.001	<.001
IHS - Spy1	0.891	<.001
IHS - Spy2	<.001	<.001
PSO - Spy1	<.001	<.001
PSO - Spy2	<.001	<.001
Spy1 - Spy2	<.001	<.001

The difference is significant if $P - value < \alpha$, ($\alpha = 0.05$)

computation time because the *SwingMove* has simpler operation than *MoveToward*.

The spy algorithm, GA, IHS, and PSO were tested on two tests without changing the algorithm parameters. The tests applied an equal condition for all tested algorithms and all algorithm parameters were tuned up to suit the set of test functions as a whole. Based on the descriptive results and statistical analysis, overall spy algorithm showed the best performances in accuracy, MPR, and computation time. These results indicate that the spy algorithm was more robust than the GA, IHS, and PSO. Changes in parameter values in the spy algorithm have a direct effect on the number of occurrences of *SwingMove*, *MoveToward*, and *random search*. These changes may have an impact on the performance. Spy2 was substantially faster than Spy1 but Spy1 tended to have better accuracy on high-dimensional test functions. Both Spy1 and Spy2 showed almost equally good ability for detecting many GOPs.

The spy algorithm performed well because its rule and all processes are very simple so it can achieve a low computational cost. The spy algorithm is accurate because it uses two types of solution refinements. *SwingMove* is a slight refinement to avoid a sudden drop in solution quality. However, the *SF* has an important role to manage its change. A small *SF* tends to provide a new solution that is not much different from the previous solution. While the *SwingMove* preserve solution quality, the *MoveToward* performs a progressive search by benefiting the previously obtained location. These two strategies are well managed so that the spy algorithm can achieve high accuracy while reducing the computational costs.

5. Conclusion

We proposed the spy algorithm, which is a population-based metaheuristic algorithm that ensures the benefit of exploration and exploitation as well as cooperative and non-cooperative searches in each iteration. Unlike many traditional population-based metaheuristic algorithms that loses exploration as the iteration increase, the spy algorithm is able to maintain exploitation as well as exploration search to improve the solutions.

We tested the GA, IHS, PSO, and spy algorithm on a set of non-convex functions that focus on accuracy, the ability to detect many global optimum points, and computation time. We conducted a statistical analysis to gain insight into accuracy and computational cost. As a result, the spy algorithm outperformed GA, IHS, and PSO. It resulted in less errors and higher maximum peak ratios within less computation time, indicating that the spy algorithm was more robust than other tested algorithm.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 20K11973.

References

[1] F. Barahona, M. Grötschel, M. Jünger, and G. Reinel, "An application of combinatorial optimization to statistical physics and circuit layout design," Operations Research, vol.36, no.3, pp.493–513, June 1988.
[2] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, New York, 1979.
[3] R.M. Karp, Reducibility Among Combinatorial Problems, pp.219–241, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
[4] J.C.A. d'Auriac, M. Preissmann, and A. Sebö, "Optimal cuts in graphs and statistical mechanics," Mathematical and Computer Modelling, vol.26, no.8-10, pp.1–11, 1997.
[5] P.C. Jennings, S. Lysgaard, J.S. Hummelshøj, T. Vegge, and T. Bligaard, "Genetic algorithms for computational materials discovery accelerated by machine learning," npj Computational Materials, vol.5, April 2019.
[6] S. Chatterjee, M. Laudato, and L.A. Lynch, "Genetic algorithms and their statistical applications: an introduction," Computational Statistics & Data Analysis, vol.22, no.6, pp.633–651, Oct. 1996.
[7] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, Cambridge, MA, USA, 1996.
[8] J. Kennedy and R. Eberhart, "Particle swarm optimization," Proc. ICNN'95 - International Conference on Neural Networks, pp.1942–1948, 1995.
[9] S. Shabir and R. Singla, "A comparative study of genetic algorithm and the particle swarm optimization," International Journal of Electrical Engineering, vol.9, no.2, pp.215–223, 2016.
[10] N.R. Tayebi, F.M. Nejad, and M. Mola, "Comparison between GA and PSO in analyzing pavement management activities," Journal of Transportation Engineering, vol.140, no.1, pp.99–104, 2014.
[11] Z.W. Geem, J.H. Kim, and G. Loganathan, "A new heuristic optimization algorithm: Harmony search," SIMULATION, vol.76, no.2, pp.60–68, Feb. 2001.
[12] M. Mahdavi, M. Fesanghary, and E. Damangir, "An improved harmony search algorithm for solving optimization problems," Applied Mathematics and Computation, vol.188, no.2, pp.1567–1579, May 2007.
[13] M.G. Omran and M. Mahdavi, "Global-best harmony search," Applied Mathematics and Computation, vol.198, no.2, pp.643–656, 2008.
[14] D.H. Wolpert and W.G. Macready, "No free lunch theorems for optimization," IEEE Trans. Evol. Comput., vol.1, no.1, pp.67–82, April 1997.
[15] J.S. Arora, "Chapter 14 - practical applications of optimization," in Introduction to Optimum Design (Third Edition), ed. J.S. Arora, pp.575–617, Academic Press, Boston, 2012.
[16] P.B. Thanedar, J.S. Arora, G.Y. Li, and T.C. Lin, "Robustness, generality and efficiency of optimization algorithms for practical applications," Structural optimization, vol.2, no.4, pp.203–212, Dec. 1990.
[17] M. Komorowski, M.J. Costa, D.A. Rand, and M.P.H. Stumpf, "Sensitivity, robustness, and identifiability in stochastic chemical kinetics models," Proc. National Academy of Sciences, vol.108, no.21, pp.8645–8650, May 2011.
[18] N.D. Lagaros and D.C. Charmpis, "Efficiency and robustness of three metaheuristics in the framework of structural optimization," Artificial Intelligence Applications and Innovations, ed. H. Papadopoulos, A.S. Andreou, and M. Bramer, Berlin, Heidelberg, pp.104–111, Springer Berlin Heidelberg, 2010.
[19] K. Murty and S. Kabadi, "Some NP-complete problems in quadratic and nonlinear programming," Mathematical Programming, vol.39, pp.117–129, June 1987.
[20] E.G. Talbi, Metaheuristics From Design to Implementation, John Wiley & Sons, New Jersey, USA, 2009.
[21] X.S. Yang, Nature-Inspired Metaheuristic Algorithms, 2nd ed., Luniver Press, 2010.
[22] C. Koulamas, S. Antony, and R. Jaen, "A survey of simulated annealing applications to operations research problems," Omega, vol.22, no.1, pp.41–56, Jan. 1994.
[23] T. Feo and M. Resende, "Greedy randomized adaptive search procedures," Journal of Global Optimization, vol.6, pp.109–134, March 1995.
[24] M.G. Resende and C.C. Ribeiro, "Greedy randomized adaptive search procedures," in Handbook of Metaheuristics, ed. F. Glover and G.A. Kochenberger, International Series in Operations Research & Management Science, ch. 8, pp.219–249, Kluwer Academic, Dordrecht, 2003.
[25] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," IEEE Comput. Intell. Mag., vol.1, no.4, pp.28–39, Nov. 2006.
[26] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing," Science, vol.220, no.4598, pp.671–680, May 1983.
[27] R. Khosravanian, V. Mansouri, D. Wood, and M. Alipour, "A comparative study of several metaheuristic algorithms for optimizing complex 3-d well-path designs," Journal of Petroleum Exploration and Production Technology, vol.8, pp.1487–1503, Dec. 2018.
[28] F. Glover, "Tabu search - Part I," ORSA Journal on Computing, vol.1, no.3, pp.190–206, Aug. 1989.
[29] F. Glover, "Tabu search - Part II," ORSA Journal on Computing, vol.2, no.1, pp.4–32, Feb. 1990.
[30] K. Hussain, M.N.M. Saleh, S. Cheng, and Y. Shi, "On the exploration and exploitation in popular swarm-based metaheuristic algorithms," Neural Computing and Applications, vol.31, pp.7665–7683, 2019.
[31] B. Morales-Castañeda, D. Zaldívar, E. Cuevas, F. Fausto, and A. Rodríguez, "A better balance in metaheuristic algorithms: Does it exist?," Swarm and Evolutionary Computation, vol.54, p.100671, 2020.
[32] J.E. Baker, "Adaptive selection methods for genetic algorithms," Proc. an International Conference on Genetic Algorithms and their applications, p.111, Hillsdale, New Jersey, 1985.
[33] F. Herrera, M. Lozano, and J. Verdegay, "Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis," Artificial Intelligence Review, vol.12, no.4, pp.265–319, Aug. 1998.
[34] B.S.G. de Almeida and V.C. Leite, "Particle swarm optimization: A powerful technique for solving engineering problems," in Swarm Intelligence, ed. J.D. Ser, E. Villar, and E. Osaba, ch. 3, IntechOpen, Rijeka, 2019.

Dhidhi Pambudi received a B.S. in mathematical analysis and M.S. in computer science from Gadjah Mada University (Indonesia) in 2004 and 2008. He is a junior lecturer at Sebelas Maret University (Indonesia) and is currently taking doctoral courses at Yamaguchi University.

Masaki Kawamura received an B.E., M.E., and Ph.D. from the University of Tsukuba in 1994, 1996, and 1999. He joined Yamaguchi University as a research associate in 1999, where he is currently a professor. His research interests include optimization, associative memory models and information hiding. He is a senior member of IEICE and a member of JNNS, JPS, and IEEE.