Improving CPU utilization of interleaving generation parallel evolutionary algorithm with precedence evaluation of tentative solutions and their suspension

Hayato Noguchi\textsuperscript{a}, Akari Sonoda\textsuperscript{b}, Tomohiro Harada \textsuperscript{c} and Ruck Thawonmas\textsuperscript{b}

\textsuperscript{a}Graduate School of Information Science and Engineering, Ritsumeikan University, Shiga, Japan; \textsuperscript{b}College of Information Science and Engineering, Ritsumeikan University, Shiga, Japan; \textsuperscript{c}Faculty of Systems Design, Tokyo Metropolitan University, Tokyo, Japan

**ABSTRACT**

This paper proposes a new mechanism to improve the CPU efficiency of parallel evolutionary algorithms (PEAs). The proposed method is based on interleaving generation evolutionary algorithm (IGEA) that was proposed in a previous study. Whereas PEA generates offspring after all individuals are evaluated, IGEA generates offspring of which all parents have been determined before other evaluations are completed. The proposed method introduced a precedence evaluation of tentative offspring and their suspension mechanism into IGEA. In particular, while IGEA generates offspring of which all parents have been determined, the proposed method tentatively generates offspring when one of two parents has been determined and then begins their evaluations. The evaluation of unnecessary offspring is suspended when the other parent of tentative offspring is determined. We compare the proposed method with the original IGEA and a simple PEA to investigate the effectiveness of the proposed method. This paper considers two replacement schemes of PEAs, \((\lambda, \lambda)\)-PEA and \((\lambda + \lambda)\)-PEA. The experimental results reveal that the proposed method has higher CPU utilization than the original IGEA and the simple PEA on both schemes.

**ARTICLE HISTORY**

Received 30 October 2020
Revised 28 June 2021
Accepted 10 August 2021

**KEYWORDS**

Evolutionary algorithm; parallelization; interleaving generations; evaluation time difference; computational efficiency

1. Introduction

Evolutionary algorithms (EAs) typified by genetic algorithm (GA) [1] and genetic programming (GP) [2] have high search capability without any problem-specific knowledge. EAs are widely applied to real-world optimization problems [3,4]. When applying EAs to real-world optimization problems, solution evaluations need much computational time due to, for example, physical simulations to evaluate solutions or measurement of actual consumption time. In such a situation, parallel evolutionary algorithms (PEAs) [5,6] are a possible attempt for accelerating the optimization process by parallelizing fitness evaluations. Although PEAs have been widely studied for reducing the enormous computation time of evolutionary algorithms (EAs), the conventional PEA has a problem that the parallelization efficiency decreases when the difference in the evaluation time of solutions is significant [7–9]. This is because the conventional PEA is generally based on the generation-based approach where the new population is generated after all solutions are evaluated, and it is necessary to wait for the slowest evaluation.

For improving the computational efficiency of PEAs, asynchronous PEAs (APEAs) have been proposed [10]. Since APEAs generate a new individual immediately after one individual is evaluated, it can reduce idling time to wait for the evaluation of the slowest individual and utilize 100% of the computing resources regardless of the difference in the evaluation time of solutions. However, APEAs have a problem when the evaluation time is biased. Due to the evaluation time bias, APEAs suffer from converging to the search area with a shorter evaluation time [11]. Therefore, APEAs fall into local optima with a shorter evaluation time rather than the global optimum with a longer evaluation time [12].

To overcome such a problem, Pilát and Neruda proposed an interleaving generation evolutionary algorithm (IGEA) [13]. IGEA is based on a generation-based approach, but IGEA can reduce the computation time by generating new offspring of which parents have been determined even when all individuals have not been evaluated. Since IGEA is the generation-based scheme, it does not suffer from falling into local optimum with a shorter evaluation time. However, IGEA has a problem that decreases the computational efficiency while increasing the number of CPUs. This is because the number of unused processors increases due to increasing the probability that one of the two parents has not been evaluated.

In this work, we propose a method for improving the computational efficiency of IGEA. In particular, for increasing the CPU utilization of IGEA, the proposed
method introduces a mechanism of a precedence evaluation of tentative offspring that is generated one of two parents is determined. Additionally, when another parent of tentative offspring completes its evaluation, the proposed method suspends the evaluation of tentative offspring that need not be evaluated.

To investigate the effectiveness of the proposed method, we conduct two experiments to compare the performance of the proposed method with the original IGEA and a simple generation-based PEA. Each experiment uses several fitness functions and the different features of the evaluation time. We consider two variants of replacement schemes, \((\lambda, \lambda)\)-EA and \((\lambda + \lambda)\)-EA.

This paper is an extension of our conference paper [14]. The additional original contribution of this paper from the conference paper is as follows:

- This paper implements the proposed method to the \((\lambda + \lambda)\)-IGEA variant, though our conference paper implemented only the \((\lambda, \lambda)\)-IGEA variants.
- This paper compares not only the proposed method and the original PEA but also the simple generation-based PEA, though our conference paper only compares the proposed method with the original IGEA.
- This paper further analyses the behaviour of precedence evaluation of tentative solutions and the suspension mechanism.

The remainder of this paper is organized as follows. The following section describes the overview of PEAs. Section 3 shows the detail of IGEA. Then, Section 4 describes the detail of the proposed method based on the IGEA. Section 5 describes the experimental settings in which the proposed method is compared with a simple PEA and the original IGEA. Section 6 shows the experimental results and their analyses. Section 7 analyses the effectiveness of the suspension of precedence evaluation, and we discuss the applicability and limitation of the proposed method. Finally, Section 8 concludes this paper and presents future works.

2. Parallel evolutionary algorithm

Evolutionary algorithms (EAs) are meta-heuristics that search for optimal solutions by imitating the natural evolution of organisms. Since an EA needs a huge amount of computation time for solution evaluations, parallel EAs (PEAs) have been studied.

The master-slave model is a simple and general parallelization model that evaluates the fitness of individuals in parallel [15]. PEAs using the master-slave parallelization model perform the main procedure of EAs, e.g., initialization, genetic operators, selection, and population replacement, on a master node, while the fitness evaluation is performed on multiple processors, called slave nodes, in parallel [16]. An illustration of the master-slave model is shown in Figure 1. The master processor sends the information of the individual to each slave processor. Then, the slave processor calculates the fitness value of the individual based on the received information and returns the result to the master processor.

3. Interleaving generation evolutionary algorithm

This section describes previous research related to our proposed method, named interleaving generation evolutionary algorithm (IGEA) [13] proposed by Pilát and Neruda.

IGEA is based on a generation-based PEA and improves its CPU efficiency by reducing the waiting time of the slave nodes. In particular, IGEA generates individuals which parents complete their evaluations and starts their evaluations on the slave nodes. This is possible because, in general, a few parent individuals are needed to generate an offspring individual, and it is enough to wait for the evaluations of the parent individuals. The original IGEA is available in two population replacement schemes, \((\lambda, \lambda)\)-EA and \((\lambda + \lambda)\)-EA. \((\lambda, \lambda)\)-EA generates \(\lambda\) offspring from \(\lambda\) population and fully replaces the current population with newly generated offspring. \((\lambda + \lambda)\)-EA, on the other hand, generates \(\lambda\) offspring from \(\lambda\) population and selects the best \(\lambda\) individuals from a merged set of the \(\lambda\) population and the \(\lambda\) offspring.

3.1. \((\lambda, \lambda)\)-IGEA

\((\lambda, \lambda)\)-IGEA is composed of the main function and the propagation function. The main function and the propagation function are shown in Algorithms 1 and 2, respectively.

In the main function in Algorithm 1, first, an initial population \(G_0\) is randomly generated. \(G_0\) contains the aspirant pair for tournament selection. Then, individuals in the aspirant list are submitted to the slave nodes and evaluated. The obtained fitness is assigned
Algorithm 1 ($\lambda$, $\lambda$)-IGEA.

1: $G_0 \leftarrow$ initial population
2: for all $p$ in $G_0$ do
3: \hspace{1em} if $p$ is aspirants in $G_0$ then
4: \hspace{2em} Submit parent to a slave node for the evaluation
5: \hspace{1em} end if
6: end for
7: while not termination condition do
8: $i,f \leftarrow$ receive an evaluation result from a slave node
9: Assign fitness $f$ to individual $i$ in $G_{i,gen}$
10: Comma-Propagate($i, G$)
11: end while

to the individual completing the evaluation, and then the propagation function in Algorithm 2 is performed.

In the propagation function in Algorithm 2, first, all aspirant pairs in the current generation $g$ are examined. If both individuals in an aspirant pair ($a$ and $b$ in Algorithm 2) have already been evaluated, the tournament selection is performed (line 6 in Algorithm 2). An individual selected by the tournament selection is placed at the same index as the aspirant pair index in the array of $g_{sel}$. Both parents ($p_1$ and $p_2$) have been already evaluated, their offspring ($g_{off} [p]$ and $g_{off} [p']$) are generated by performing genetic operators. The generated offspring are placed in the offspring array $g_{off}$ and copied to the parent population of the next generation ($g'$). If the new offspring are not evaluated and are needed to generate the next offspring, they are submitted to the slave nodes for evaluation. Otherwise, the propagation function is called recursively for the evaluated offspring. Since new offspring are submitted for evaluation in line 18 in Algorithm 2 before other individuals in the same generation have not been evaluated, ($\lambda$, $\lambda$)-IGEA can improve the efficiency of the slave processors compared to the generation-based PEA.

Figure 2 shows an example of the ($\lambda$, $\lambda$)-IGEA flow at the tournament selection and genetic operators (lines 13–23 in Algorithm 2). In this example, the offspring $x_3'$ and $x_4'$ (corresponding to $g_{off} [p]$ and $g_{off} [p']$ in Algorithm 2) are generated by the winners ($p_1$ and $p_2$ in Algorithm 2) from two tournament selections; one is a winner of $x_1$ and $x_5$, while another is a winner of $x_3$ and $x_6$. In such a case, the tournament selection can be performed when individuals $x_1, x_3, x_5$, and $x_6$ have been evaluated, even if other individuals have not been evaluated.

3.2. ($\lambda + \lambda$)-IGEA

($\lambda + \lambda$)-IGEA is also composed of the main function and the propagation function as with ($\lambda$, $\lambda$)-IGEA. The main function of ($\lambda + \lambda$)-IGEA is almost the same as

Algorithm 2 Comma-Propagate($ind, G$).

1: $g \leftarrow G_{ind,gen}$
2: $g' \leftarrow G_{ind,gen+1}$
3: for all aspirant pairs ($a, b$) in $g$ that contain $ind$ do
4: \hspace{1em} if both $a$ and $b$ are already in $g$ and evaluated then
5: \hspace{2em} $i \leftarrow$ index of the aspirant pair
6: \hspace{2em} $g_{sel}[i] \leftarrow$ better of $a$ and $b$
7: \hspace{1em} end if
8: \hspace{1em} $i' \leftarrow i + 1$
9: else
10: \hspace{1em} $i' \leftarrow i - 1$
11: end if
12: $p_1, p_2 \leftarrow g_{off}[i], g_{off}[i']$
13: if both $p_1$ and $p_2$ in $g_{sel}$ then
14: \hspace{1em} $g_{off}[p], g_{off}[p'] \leftarrow$ create offspring from $p_1$ and $p_2$
15: $g'[p], g'[p'] \leftarrow g_{off}[p], g_{off}[p']$
16: for all new offspring $o$ in $g'$ do
17: \hspace{1em} if $o$ not evaluated and in aspirants in $g'$ then
18: \hspace{2em} Submit $o$ for evaluation
19: else
20: \hspace{1em} Comma-Propagate($o, G$)
21: end if
22: end for
23: end if
24: end if
25: end for
Algorithm 3 Plus-Propagate(ind, G).

1: \( g \leftarrow G_{\text{ind,gen}} \)  
2: \( g' \leftarrow G_{\text{ind,gen}+1} \)  
3: \( P \leftarrow \text{evaluates parents from } g \)  
4: \( O \leftarrow \text{evaluates offspring from } g \)  
5: \( S' \leftarrow \text{best } | P \cup U | - \lambda \text{ individuals from } P \cup U \)  
6: \( S \leftarrow \text{individuals from } S' \text{ which are not among parents of } g' \)  
7: \( \text{for all individuals } \text{ind from } S \text{ do} \)
   8: \( \text{Add ind to parents of } g' \)  
9: \( \text{for all aspirant pairs } (a, b) \text{ in } g' \text{ that contain } \text{ind do} \)
   10: \( \text{if both } a \text{ and } b \text{ are already in } g' \text{ and evaluated then} \)
   11: \( i \leftarrow \text{index of the aspirant pair} \)  
   12: \( g'_s[i] \leftarrow \text{better of } a \text{ and } b \)  
   13: \( \text{if } i \text{ even then} \)
   14: \( i' \leftarrow i + 1 \)  
   15: \( \text{else} \)
   16: \( i' \leftarrow i - 1 \)  
   17: \( \text{end if} \)  
18: \( p_1, p_2 \leftarrow g'_s[i], g'_s[i'] \)  
19: \( \text{if both } p_1 \text{ and } p_2 \text{ in } g'_s \text{ then} \)
20: \( g'_s[p], g'_s[p'] \leftarrow \text{create offspring from } p_1 \text{ and } p_2 \)  
21: \( \text{Create next generation after } g \text{ if needed} \)  
22: \( \text{for all new offspring } o \text{ in } g' \text{ do} \)
   23: \( \text{Submit } o \text{ for evaluation} \)  
24: \( \text{end for} \)  
25: \( \text{end if} \)  
26: \( \text{end if} \)  
27: \( \text{Plus-Propagate}(o, G) \)  
28: \( \text{end for} \)

Figure 2. An example of the \((\lambda, \lambda)\)-IGEA flow at the tournament selection and genetic operators.

the one of \((\lambda, \lambda)\)-IGEA, but calls the propagation function shown in Algorithm 3 at line 10 in Algorithm 1.

In the propagation function in Algorithm 3, when the \( k > \lambda \) individuals in a merged set of parents and offspring of the previous generation have been evaluated, the best \( k - \lambda \) individuals are determined to be selected in the next generation. Thus, offspring of which parents are determined can be generated as similar to \((\lambda, \lambda)\)-IGEA.

Figure 3 shows the example of the \((\lambda + \lambda)\)-IGEA flow at the tournament selection and genetic operators.

Figure 3 shows the example of the \((\lambda + \lambda)\)-IGEA flow at the tournament selection and genetic operators (lines 19–25 in Algorithm 3). In this example, since 13 individuals out of 16 individuals (eight parents and eight offspring) are evaluated, the best five (= \( 16 - 13 \)) individuals are determined to be selected for the next generation. Then, if new offspring \( o'_3 \) and \( o'_4 \) (corresponding to \( g'_s[p] \) and \( g'_s[p'] \) in Algorithm 3) are
generated by the winners from two tournament selections of \( p_1', p_2', p_3', \) and \( p_4' \) (\( p_1 \) and \( p_2 \) in Algorithm 3). In such a case, the tournament selection can be performed when the top 5 individuals are determined.

### 4. Proposed method

In this paper, we propose a modified version of IGEA that introduces the precedence evaluation of tentative offspring for improving the efficiency of the slave processors. This section firstly describes the problem of the original IGEA. Then, we propose the modification of IGEA and show its detailed algorithm.

#### 4.1. Problem of IGEA

Since the original IGEA is based on a generation-based EA, it has a problem that the efficiency of the slave processors decreases as the number of used processors increases. This is because the number of unused slave processors increases due to an increase in the waiting time of evaluations of the individuals by the evaluation time bias. For example, in Figure 2, when an individual \( x_5 \) has not been evaluated yet, the tournament selection using \( x_8 \) cannot be performed and the waiting time occurs. A similar problem arises even in \( (\lambda + \lambda) \)-IGEA. For example, in Figure 3, since the sixth parent individual cannot be determined, the tournament selection using \( p_6' \) cannot be performed, and the generation procedure stagnates.

#### 4.2. Two modifications

For the problems of the original IGEA described in Section 4.1, we propose a method to improve the efficiency of the slave processors in IGEA by introducing two modifications, the precedence evaluation of tentative offspring and the suspension of unnecessary precedence evaluation.

##### 4.2.1. Precedence evaluation of tentative offspring

In order to reduce the waiting time in IGEA, we propose a method to execute the precedence evaluation of tentative offspring.

In the original \( (\lambda, \lambda) \)-IGEA, when two parents are determined, new offspring are generated, and their evaluations start. However, even when one of two parents is evaluated, and another parent is not determined, there are only two possibilities of offspring, that is, either of the corresponding aspirant pair wins. For this fact, our proposed method performs the tournament selection and genetic operations when one of two parents is determined in advance, and it starts evaluations of newly generated tentative offspring.

Figure 4 shows an example of the tentative offspring generation. In this figure, three of four parents, i.e. \( x_2, x_4, \) and \( x_7 \), complete their evaluations, and one parent \( x_4 \) is determined, which is the winner of \( x_4 \) and \( x_7 \). On the other hand, since the evaluation of \( x_4 \) has not been completed, another parent is not determined. However, in such a case, another parent is either \( x_2 \) or \( x_8 \). Then, the proposed method temporarily generates offspring for the case that \( x_2 \) wins, and for the other case that \( x_8 \) wins as shown in Figure 4. As a result, four tentative offspring are generated, and the proposed method starts their evaluations immediately, even though the evaluation of \( x_8 \) has not been completed yet.

A similar mechanism can be considered in \( (\lambda + \lambda) \)-IGEA. In particular, when three of four parent candidates have been evaluated, one parent is determined, while another parent is either an individual that has already been evaluated or has not been evaluated. In such a case, as similar to the mechanism of \( (\lambda, \lambda) \) scheme, tentative offspring can be generated. Figure 5 shows an example of the tentative offspring generation in \( (\lambda + \lambda) \)-IGEA. In this figure, \( p_7' \) and \( p_8' \) are generated from the parent candidates of \( p_2', p_5', p_6', \) and \( p_4', p_5', p_6' \) are determined. In this case, one parent is determined from the winner of \( p_4' \) and \( p_5' \), while another parent is either \( p_2' \) or the other newly evaluated solution. Then, the proposed method temporarily generates offspring for the case that \( p_2' \) wins and starts their evaluations.

##### 4.2.2. Suspension of precedence evaluation

The precedence evaluations of the tentative offspring in the two schemes are continued until their evaluations are completed, or the remaining parent is determined. If the evaluation of unnecessary tentative offspring continues after the remaining parent is determined, the slave processor will waste the computing resource.
Figure 5. Precedence evaluation of tentative offspring in $(\lambda + \lambda)$-IGEA.

Figure 6. Suspension of unnecessary evaluations in $(\lambda, \lambda)$-IGEA.

Therefore, it needs to suspend and delete the unnecessary evaluation of the tentative offspring. Thus, the proposed method introduces a mechanism to suspend the evaluations of unnecessary individuals.

Figure 6 shows an example of the suspension procedure in $(\lambda, \lambda)$-IGEA. In Figure 4, the evaluation of $x_8$ was not completed, while later in Figure 6, its evaluation is completed, and $x_2$ is determined as the winner of the tournament selection. In such a case, the tentative offspring generated from $x_8$ and $x_4$ is no longer necessary. Thus, the proposed method suspends the evaluations of these unnecessary offspring.

A similar mechanism is introduced in the $(\lambda + \lambda)$ scheme. Figure 7 shows an example of the suspension procedure of $(\lambda + \lambda)$-IGEA. In this figure, when $p'_6$ is determined, either of $p'_6$ and $p'_2$ win. If $p'_2$ wins, tentative offspring can be left. However, if $p'_6$, a newly evaluated individual, wins, the precedence evaluation is suspended, and tentative offspring is deleted. Then, new offspring generated from $p'_6$ and $p'_5$ are generated and start their evaluations. By this procedure, the proposed method eliminates wasted computing resources.

5. Experiments

To investigate the effectiveness of the proposed method, we conduct experiments to compare the performance of the proposed method with the original IGEA and a simple generation-based PEA. EA used in our experiments is the real-coded GA that evolves continuous (real-value) design variables. We consider two schemes of $(\lambda, \lambda)$-EA and $(\lambda + \lambda)$-EA. Hereafter, we denoted the proposed IGEA with $(\lambda, \lambda)$ scheme as P-$(\lambda, \lambda)$-IGEA, while the one with $(\lambda + \lambda)$ scheme as P-$(\lambda + \lambda)$-IGEA.

The experiments use various fitness functions and the different features of the evaluation time. The experiments are implemented based on the Python code provided by the authors of [13]. The master-slave parallel processing is implemented in the simulation environment.

5.1. Experimental cases

This paper conducts two experiments: the first one uses a constant fitness function, while the second one uses the Rastrigin function.

5.1.1. Experiment 1

Experiment 1 uses a constant fitness function and three different features of the evaluation time. Concretely, we use the following three evaluation time functions:

- **Fixed**: the evaluation time is fixed to 1 s.
- **Uniform**: the evaluation time is randomly sampled from the uniform distribution of 1 to 100 s.
- **Exponential**: the evaluation time is sampled from the exponential distribution of 1 s on average.
The maximum number of evaluations is 10,000 in each method, and each method is performed in 20 trials. We change the number of utilized CPUs as 1, 10, 20, · · · , 90, 100, and compare the evaluation time on the simulation.

5.1.2. Experiment 2
Experiment 2 uses the Rastrigin function that is well-known multimodal benchmark problem and is defined as:

\[ f_R(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right), \]  

where \( n \) is the dimension of the individual. This experiment set \( n = 5 \), which is the same setting as the original work.

We use the evaluation time function that is negatively or positively correlated to the fitness value. The evaluation time function of the positive correlation is formulated as:

\[ T_p(x) = 1 + f_R(x), \]  

where \( x \) is an individual to be determined its evaluation time, while \( f_R(x) \) indicates the fitness value of \( x \) calculated with (1). On the other hand, the evaluation time function of the negative correlation is formulated as:

\[ T_n(x) = 1 + \max\left(100 - f_R(x), 0\right), \]  

where the minimum evaluation time is limited to 1 s, even if the fitness value is greater than 100.

The maximum number of evaluations is 10,000 in each method, and each method is performed in 20 trials. We change the number of utilized CPUs as 1, 10, 20, · · · , 90, 100, and compare the evaluation time on the simulation and the transition of the fitness value.

Note that the search performance of the competitive methods (both in the \((\lambda, \lambda)\) and \((\lambda + \lambda)\) replacement schemes) is identical because there is no change in the optimization result with respect to the same generation. Therefore, the purpose of this experiment is to investigate whether the proposed parallelization scheme can reduce the execution time, and we do not discuss the search performance.

5.2. Parameter settings
We use the same experimental parameter settings as the original work. The population size is set to 100. We use the two-point crossover with the probability of 0.8 and the Gaussian mutation (a standard deviation is 1.0) with the probability of 0.1.

5.3. Evaluation criteria
To compare the performance of the competitive methods, we first calculate the execution time to complete 10,000 evaluations. Then, we calculate the CPU utilization, which is calculated as \( u = t_e/Ntw \). \( N \) is the number of CPUs used, \( t_w \) is the execution time to complete 10,000 evaluations in the simulation of the experiment, while \( t_e \) is the sum of all evaluation time.

6. Result
This section describes the experimental results described in Section 5. First, we show the results when using a constant fitness function, and then, the results with the Rastrigin function are presented.

6.1. Experiment 1
6.1.1. CPU utilization
Figure 8 shows the CPU utilization when using the fixed evaluation time, the uniform evaluation time, and the exponentially distributed evaluation time. The vertical axis shows the CPU utilization, while the horizontal axis shows the number of CPUs. This result shows the median of 20 trials. The blue lines with circles indicate \((\lambda, \lambda)\)-EA and \((\lambda + \lambda)\)-EA. The orange lines with squares indicate \((\lambda, \lambda)\)-IGEA and \((\lambda + \lambda)\)-IGEA, while the red lines with triangles indicate \(P(\lambda, \lambda)\)-IGEA and \(P(\lambda + \lambda)\)-IGEA. As to the line type, the solid lines indicate the \((\lambda, \lambda)\) scheme, while the dashed lines indicate the \((\lambda + \lambda)\) one.

From these results, we can first find that the CPU utilization of a simple PEA decreases as the number of utilized CPUs increases in all evaluation time functions. In particular, in the exponential evaluation time shown in Figure 8(c), the CPU utilization decreases to less than 20%. This indicates that the simple generation-based PEA wastes much computation resources when the variance of evaluation time is large.

Focusing on the original IGEA and the proposed method, we can generally find that the CPU utilization is 100% up to 30 or 40 CPUs, but it decreases when using more than 50 CPUs. However, the proposed method shows higher CPU utilization than the original IGEA in both schemes of \((\lambda, \lambda)\) and \((\lambda + \lambda)\) in all features of the evaluation time. In particular, for the fixed evaluation time, as shown in Figure 8(a), the CPU utilization of \(P(\lambda, \lambda)\)-IGEA is about 74% when the number of CPUs is 100, while that of \((\lambda, \lambda)\)-IGEA is about 71%. Meanwhile, the CPU utilization of \(P(\lambda + \lambda)\)-IGEA is about 83%, while that of \((\lambda + \lambda)\)-IGEA is about 82%.

On the other hand, when the evaluation times are different, i.e. the uniform evaluation time and the exponential distributed evaluation time, the difference between the proposed method and the original IGEA is larger on both the replacement schemes. For the uniform evaluation time, when the number of CPU is 100, the CPU utilization of \(P(\lambda, \lambda)\)-IGEA is about 55%, while that of \((\lambda, \lambda)\)-IGEA is about 42%. In addition, the
Figure 8. The CPU utilization using a constant fitness function. (a) Fixed, (b) Uniform, and (c) Exponential.

CPU utilization of P-$(\lambda + \lambda)$-IGEA is about 70%, while that of $(\lambda + \lambda)$-IGEA is about 49%. For the exponential distributed evaluation time, when the number of CPU is 100, the CPU utilization of P-$(\lambda + \lambda)$-IGEA is about 32%, while that of $(\lambda, \lambda)$-IGEA is about 26%. In addition, the CPU utilization of P-$(\lambda + \lambda)$-IGEA is about 44%, while that of $(\lambda + \lambda)$-IGEA is about 30%.

These results reveal that the proposed method improves the CPU utilization in PEA by generating precedence offspring. In particular, the proposed method improves the CPU utilization at most 21% compared with the original IGEA when 100 CPUs are utilized.

6.1.2. Execution time

Tables 1–3 show the execution times to complete the maximum number of evaluations and the occurrence percentage of precedence evaluations for each number of CPUs in the fixed evaluation time, the uniform evaluation time, and the exponential evaluation time, respectively. In these tables, $\Delta T_{PEA}$ indicates the reduction ratio of the computational time of the proposed method compared with the simple PEA, while $\Delta T_{IGEA}$ indicates that compared with the original IGEA. These ratios are calculated as:

$$\Delta T_{PEA/IGEA} = \frac{T_{PEA/IGEA} - T_{Proposed}}{T_{PEA/IGEA}} \times 100\%,$$

where $T_{PEA/IGEA}$ is the execution time of the simple PEA and the original IGEA, while $T_{Proposed}$ is the execution time of the proposed IGEA. If $\Delta T > 0$, the proposed IGEA reduces the execution time from PEA and IGEA, while if $\Delta T < 0$, the execution time of the proposed method increases. On the other hand, $\%PE$

| Table 1. The elapsed execution time until the maximum number of the evaluations and the occurrence percentage of precedence evaluations (Fixed) ($\times 10^2$ s). |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| #CPU | PEA | IGEA | P-IGEA | $\Delta T_{PEA}$ | $\Delta T_{IGEA}$ | %PE |
|------|-----|-----|-------|----------------|----------------|-----|
| 1 | 100.00 | 100.00 | 100.00 | 0.00 | 0.00 | 100.00 |
| 10 | 10.55 | 10.00 | 10.00 | 5.21 | 0.00 | 10.54 |
| 20 | 5.65 | 5.00 | 5.00 | 11.43 | 0.00 | 5.63 |
| 30 | 3.72 | 3.34 | 3.34 | 10.09 | 0.00 | 3.72 |
| 40 | 3.18 | 2.50 | 2.50 | 21.26 | 0.00 | 3.20 |
| 50 | 2.45 | 2.06 | 2.05 | 16.16 | 0.49 | 2.43 |
| 60 | 2.44 | 1.87 | 1.80 | 26.43 | 9.14 | 2.45 |
| 70 | 2.41 | 1.70 | 1.60 | 33.61 | 17.43 | 2.42 |
| 80 | 1.99 | 1.46 | 1.41 | 29.15 | 26.41 | 1.99 |
| 90 | 1.29 | 1.41 | 1.36 | 5.04 | 39.41 | 1.28 |
| 100 | 1.22 | 1.41 | 1.34 | 9.84 | 41.87 | 1.23 |

| Table 2. The elapsed execution time until the maximum number of the evaluations and the occurrence percentage of precedence evaluations (Uniform) ($\times 10^3$ s). |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| #CPU | PEA | IGEA | P-IGEA | $\Delta T_{PEA}$ | $\Delta T_{IGEA}$ | %PE |
|------|-----|-----|-------|----------------|----------------|-----|
| 1 | 50.51 | 50.48 | 50.43 | 0.16 | 0.10 | 0.00 |
| 10 | 5.56 | 5.06 | 5.07 | 8.86 | 0.24 | 0.00 |
| 20 | 3.11 | 2.53 | 2.53 | 18.78 | 0.00 | 0.00 |
| 30 | 2.31 | 1.69 | 1.68 | 26.93 | 0.19 | 0.00 |
| 40 | 1.94 | 1.29 | 1.28 | 34.02 | 0.69 | 1.57 |
| 50 | 1.67 | 1.21 | 1.10 | 33.82 | 8.55 | 16.24 |
| 60 | 1.45 | 1.20 | 0.99 | 32.02 | 17.74 | 33.65 |
| 70 | 1.29 | 1.20 | 0.91 | 29.54 | 24.12 | 46.45 |
| 80 | 1.21 | 1.20 | 0.88 | 27.36 | 26.80 | 53.36 |
| 90 | 1.20 | 1.20 | 0.84 | 29.97 | 29.67 | 56.31 |
| 100 | 1.20 | 1.20 | 0.80 | 30.54 | 30.36 | 57.43 |

| Table 3. The elapsed execution time until the maximum number of the evaluations and the occurrence percentage of precedence evaluations (Exponential) ($\times 10^3$ s). |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| #CPU | PEA | IGEA | P-IGEA | $\Delta T_{PEA}$ | $\Delta T_{IGEA}$ | %PE |
|------|-----|-----|-------|----------------|----------------|-----|
| 1 | 50.51 | 50.48 | 50.43 | 0.16 | 0.10 | 0.00 |
| 10 | 5.56 | 5.06 | 5.07 | 8.86 | 0.24 | 0.00 |
| 20 | 3.11 | 2.53 | 2.53 | 18.78 | 0.00 | 0.00 |
| 30 | 2.31 | 1.69 | 1.68 | 26.93 | 0.19 | 0.00 |
| 40 | 1.94 | 1.29 | 1.28 | 34.02 | 0.69 | 1.57 |
| 50 | 1.67 | 1.21 | 1.10 | 33.82 | 8.55 | 16.24 |
| 60 | 1.45 | 1.20 | 0.99 | 32.02 | 17.74 | 33.65 |
| 70 | 1.29 | 1.20 | 0.91 | 29.54 | 24.12 | 46.45 |
| 80 | 1.21 | 1.20 | 0.88 | 27.36 | 26.80 | 53.36 |
| 90 | 1.20 | 1.20 | 0.84 | 29.97 | 29.67 | 56.31 |
| 100 | 1.20 | 1.20 | 0.80 | 30.54 | 30.36 | 57.43 |
Table 3. The elapsed execution time until the maximum number of the evaluations and the occurrence percentage of precedence evaluations (Exponential) ($\times 10^2$ s).

| #CPU | PEA | IGEA | P-IGEA | $\Delta T_{PEA}$ | $\Delta T_{IGEA}$ | %PE |
|------|-----|------|--------|------------------|------------------|-----|
| 1    | 100.23 | 99.66 | 99.93  | 0.30             | -0.26            | 0.00 |
| 10   | 12.36  | 9.99  | 10.03  | 18.90            | -0.34            | 0.00 |
| 20   | 8.07   | 4.99  | 5.03   | 37.69            | -0.80            | 0.19 |
| 30   | 6.96   | 3.91  | 3.73   | 46.35            | 4.57             | 12.87|
| 40   | 6.52   | 3.92  | 3.29   | 49.64            | 16.24            | 36.43|
| 50   | 6.23   | 3.85  | 2.97   | 52.23            | 22.34            | 49.67|
| 60   | 6.15   | 3.88  | 2.93   | 52.37            | 24.46            | 55.71|
| 70   | 6.09   | 3.85  | 2.78   | 54.30            | 27.76            | 57.17|
| 80   | 6.09   | 3.85  | 2.84   | 53.37            | 26.23            | 57.63|
| 90   | 6.09   | 3.84  | 2.79   | 54.11            | 27.27            | 57.76|
| 100  | 6.03   | 3.82  | 2.86   | 52.68            | 25.35            | 57.69|

indicates the occurrence percentage of precedence evaluations in P-($\lambda$, $\lambda$)-IGEA and P-($\lambda$, $\lambda$)-IGEA, which is calculated as:

$$\%PE = \frac{\text{#precedence evaluations}}{\text{#all evaluations}} \times 100[\%],$$  

where #precedence evaluations means the number of precedence evaluations during the evolution, while #all evaluations means the number of all evaluations, i.e. 10,000 evaluations in our experiments.

As a result in Tables 1–3, when the number of CPUs is small, the evaluation time of the proposed IGEA is almost the same as the one of the original IGEA. In these cases, since no precedence evaluation is executed, the proposed method shows the same behaviour as the original IGEA. On the other hand, it is shown that the evaluation time of the proposed IGEA is shorter than the one of the original IGEA as the number of CPUs increases, and when 100 CPU are utilized, the proposed method reduces the execution time at most 62.44% from the simple PEA and 32.65% from the original IGEA. When a large number of cores is used, the occurrence percentage of precedence evaluations increases. In particular, more than 50% of newly generated individuals are tentative offspring for the precedence evaluations when using 100 CPUs. This contributes to improving the computational efficiency of IGEA. This result reveals that the proposed precedence evaluation of tentative offspring and their suspensions enable to shorten the execution time of PEA.

6.2. Experiment 2

6.2.1. CPU utilization

Figure 9 shows the CPU utilization of the evaluation time with negative and positive correlation when solving the Rastrigin function. The axes and the lines in these figures indicate the same meaning as Figure 8.

As shown in Figure 9(a), the CPU utilization decreases from 50 CPUs in both the proposed method and the original IGEA using the evaluation time with a negative correlation. The proposed method performs higher CPU utilization than the original IGEA in both replacement schemes. When the number of CPUs is 100, the CPU utilization of ($\lambda$, $\lambda$)-IGEA is about 68%, while the one of P-($\lambda$, $\lambda$)-IGEA is about 73%. In addition, the CPU utilization of ($\lambda$, $\lambda$)-IGEA is about 79%, while the one of P-($\lambda$, $\lambda$)-IGEA is about 80%.

![Figure 9. The CPU utilization using the Rastrigin function. (a) Negative correlation, and (b) Positive correlation.](image-url)
Table 4. The elapsed execution time until the maximum number of the evaluations and the occurrence percentage of precedence evaluations (Negative) ($\times 10^4$ s).

| #CPU | PEA | IGEA | P-IGEA | $\Delta T_{PEA}$ | $\Delta T_{IGEA}$ | %PE |
|------|-----|------|--------|-----------------|-----------------|-----|
| 1    | 93.28 | 94.06 | 93.99  | $-0.77$        | 0.07            | 0.00|
| 10   | 9.80  | 9.37  | 9.38   | $-0.09$        | 0.07            | 0.00|
| 20   | 5.24  | 4.68  | 4.68   | $10.80$        | 0.12            | 6.36|
| 30   | 3.50  | 3.15  | 3.13   | $10.56$        | 0.37            | 20.41|
| 40   | 2.93  | 2.35  | 2.35   | $19.60$        | $-0.1$          | 35.55|
| 50   | 2.33  | 1.94  | 1.92   | $17.46$        | 0.72            | 43.89|
| 60   | 2.31  | 1.77  | 1.68   | $27.26$        | 5.01            | 50.72|
| 70   | 2.26  | 1.61  | 1.50   | $33.77$        | 6.81            | 54.09|
| 80   | 1.81  | 1.40  | 1.34   | $26.23$        | 4.42            | 55.90|
| 90   | 1.23  | 1.38  | 1.28   | $-4.20$        | 7.06            | 56.96|
| 100  | 1.19  | 1.37  | 1.27   | $-6.18$        | 7.61            | 57.30|

On the other hand, as shown in Figure 9(b), the CPU utilization decrease from 10 CPUs in both the proposed method and the original IGEA when using the evaluation time with a positive correlation. Same as the case of a negative correlation, the proposed method performs higher CPU utilization than the original IGEA in both replacement schemes. When the number of CPUs is 100, the CPU utilization of $(\lambda, \lambda)$-IGEA is about 27%, while the one of P-(\lambda, \lambda)-IGEA is about 33%. In addition, the CPU utilization of $(\lambda + \lambda)$-IGEA is about 29%, while the one of P-(\lambda + \lambda)-IGEA is about 43%.

These results indicate that the proposed method can improve the CPU utilization from the simple PEA and the original IGEA even when there exists correlation between the fitness value and the evaluation time.

6.2.2. Execution time

Tables 4 and 5 show the execution time to complete 10,000 evaluations and the occurrence percentage of precedence evaluations for each number of CPUs in the simple PEA, the original IGEA, and the proposed method. A similar tendency to the results of the constant fitness function can be found. As a result, when the number of CPUs is small, the execution time of the proposed method is slightly longer than the one of IGEA. In these cases, since no precedence evaluations are executed, the difference in the execution time can be negligible. On the other hand, it is shown that the reduction ratio of the proposed method increases as the number of CPUs increases. As increasing the number of used CPUs, the percentage of precedence evaluations increases, and almost a half of newly generated individuals are tentative offspring for the precedence evaluation when using 100 CPUs. For this fact, it is indicated that the proposed method contributes to decreasing the execution time when using a large number of CPUs.

6.2.3. Fitness transition

Finally, the fitness transitions with the negative and the positive correlation evaluation times are shown in Figure 10. The vertical axis shows the logarithm of fitness, while the horizontal axis shows the elapsed simulation times. This result shows the median of 20 trials. Each line indicates the same algorithm as Figure 9. These figures show the result when 100 CPUs are utilized.

From Figure 10(a), when we compare all methods at the same elapsed time, the proposed IGEA, both in $(\lambda, \lambda)$ and $(\lambda + \lambda)$ schemes obtains the equivalent fitness function value to the simple PEA and the original
Table 5. The elapsed execution time until the maximum number of the evaluations and the occurrence percentage of precedence evaluations (Positive) ($\times 10^3$ s).

| #CPU | PEA | IGEA | P-IGEA | $\Delta T_{PEA}$ | $\Delta T_{IGEA}$ | %PE |
|------|-----|------|--------|------------------|------------------|-----|
| 1    | 95.75 | 84.18 | 85.26 | 10.95 | $-1.29$ | 0.00 |
| 10   | 11.66 | 8.21  | 8.53  | 26.85 | $-3.87$ | 0.00 |
| 20   | 7.44  | 4.67  | 4.57  | 38.56 | 2.12  | 0.00 |
| 30   | 6.41  | 3.68  | 3.46  | 46.07 | 6.14  | 0.00 |
| 40   | 6.02  | 3.28  | 3.05  | 49.40 | 7.14  | 0.16 |
| 50   | 5.56  | 3.17  | 2.90  | 47.93 | 8.61  | 2.61 |
| 60   | 5.42  | 3.09  | 2.71  | 49.97 | 12.22 | 10.21|
| 70   | 5.29  | 3.04  | 2.66  | 49.78 | 12.75 | 18.50|
| 80   | 5.09  | 3.12  | 2.64  | 48.06 | 15.22 | 27.57|
| 90   | 5.02  | 2.99  | 2.62  | 47.88 | 12.42 | 35.92|
| 100  | 4.91  | 3.07  | 2.62  | 46.55 | 14.53 | 42.75|

| #CPU | PEA | IGEA | P-IGEA | $\Delta T_{PEA}$ | $\Delta T_{IGEA}$ | %PE |
|------|-----|------|--------|------------------|------------------|-----|
| 1    | 73.71 | 73.94 | 74.23 | $-0.70$ | $-0.39$ | 0.00 |
| 10   | 9.93  | 7.08  | 7.32  | 26.27 | $-3.43$ | 0.00 |
| 20   | 6.59  | 3.89  | 4.07  | 38.28 | $-4.53$ | 0.00 |
| 30   | 5.81  | 3.15  | 3.02  | 48.12 | 4.25  | 0.00 |
| 40   | 5.38  | 2.85  | 2.65  | 50.82 | 7.11  | 0.05 |
| 50   | 5.07  | 2.67  | 2.30  | 54.66 | 13.85 | 1.79 |
| 60   | 4.98  | 2.64  | 2.17  | 56.38 | 17.62 | 9.46 |
| 70   | 4.92  | 2.59  | 2.07  | 57.92 | 20.16 | 20.94|
| 80   | 4.77  | 2.51  | 1.98  | 58.52 | 21.33 | 34.98|
| 90   | 4.74  | 2.46  | 1.93  | 59.30 | 21.42 | 47.36|
| 100  | 4.71  | 2.47  | 1.95  | 58.53 | 21.10 | 52.45|

IGEA. This indicates that the proposed IGEA maintains the search efficiency of the simple PEA and the original IGEA when the evaluation value negatively correlates to the fitness value. On the other hand, from Figure 10(b), when comparing each method at the same elapsed time on the evaluation time positively correlates to the fitness value, the proposed IGEA obtains a better fitness function value than the original IGEA. This indicates that the proposed IGEA has a search efficiency equal to or better than the simple PEA and the original IGEA when there is a positive correlation in the evaluation time.

These results reveal that the proposed method can improve the CPU utilization regardless of the feature of the evaluation time and can achieve the equivalent search capability to the original IGEA. The previous research has shown that the original IGEA does not affect the search performance even when there is a bias in the evaluation time. The proposed method inherits the search performance of the original IGEA while improving CPU utilization, which is not affected by the evaluation time bias. Overall, these results show that P-$(\lambda + \lambda)$-IGEA performs best.

### 7. Discussion

This section further analyses and discusses the proposed method. First, the following subsection analyses the effectiveness of the evaluation suspension in the proposed method. Then, we consider the influence of the population size and the tournament size on the proposed method. Finally, we discuss the range of applications in the proposed method.

#### 7.1. Effectiveness of the suspension of precedence evaluation

Although our proposed method introduces the evaluation suspension that suspends and deletes tentative offspring of which evaluations are no longer necessary because of the decision of their parents. In this subsection, we analyse the effectiveness of the evaluation suspension.

Figure 11 shows the CPU utilization of the proposed method with and without the evaluation suspension. The horizontal axis shows the number of used CPUs, while the vertical axis shows the CPU utilization. The red lines with triangles indicate the result of the proposed method with the evaluation suspension, while the blue ones with inverse triangles indicate the result of the proposed method without the suspension.

From this figure, in the $(\lambda, \lambda)$ scheme, the CPU utilization of no evaluation suspension is constantly higher than that of the proposed method using the evaluation suspension. On the other hand, in the $(\lambda + \lambda)$ scheme, there is no difference between the proposed method with and without evaluation suspension in the constant fitness function shown in Figure 11(a)-(c). This is because tentative offspring are always left for the evaluations when the fitness function is constant. Meanwhile, in Figure 11(d, e), in which the Rastrigin function is used, the CPU utilization increases if the suspension of precedence evaluation is not performed. These results indicate that the evaluation suspension successfully reduces the unnecessary use of CPUs for both of $(\lambda, \lambda)$ and $(\lambda + \lambda)$ schemes.

Figure 12 shows the fitness transition of the proposed method with and without evaluation suspension. The horizontal axis shows the elapsed execution time, while the vertical axis shows the fitness value. From these figures, there is no large difference between the proposed method with and without evaluation suspension in both the $(\lambda, \lambda)$ and $(\lambda + \lambda)$ schemes. This indicates that the proposed method without the suspension just increases the CPU utilization without any improvement of the search efficiency. For this fact, the suspension of precedence evaluation is an important mechanism to reduce the execution time with the minimum CPU utilization significantly.

#### 7.2. Influence of population size and tournament size

We consider the influence of population size and tournament size on the efficiency of the proposed method. First, for the population size, as the population size

| #CPU | PEA | IGEA | P-IGEA | $\Delta T_{PEA}$ | $\Delta T_{IGEA}$ | %PE |
|------|-----|------|--------|------------------|------------------|-----|
| 1    | 195.75 | 84.18 | 85.26 | 10.95 | $-1.29$ | 0.00 |
| 10   | 11.66 | 8.21  | 8.53  | 26.85 | $-3.87$ | 0.00 |
| 20   | 7.44  | 4.67  | 4.57  | 38.56 | 2.12  | 0.00 |
| 30   | 6.41  | 3.68  | 3.46  | 46.07 | 6.14  | 0.00 |
| 40   | 6.02  | 3.28  | 3.05  | 49.40 | 7.14  | 0.16 |
| 50   | 5.56  | 3.17  | 2.90  | 47.93 | 8.61  | 2.61 |
| 60   | 5.42  | 3.09  | 2.71  | 49.97 | 12.22 | 10.21|
| 70   | 5.29  | 3.04  | 2.66  | 49.78 | 12.75 | 18.50|
| 80   | 5.09  | 3.12  | 2.64  | 48.06 | 15.22 | 27.57|
| 90   | 5.02  | 2.99  | 2.62  | 47.88 | 12.42 | 35.92|
| 100  | 4.91  | 3.07  | 2.62  | 46.55 | 14.53 | 42.75|
increases, the number of combinations of aspirant pairs increases, and therefore, the percentage of tentative offspring that can be generated may decrease. In this case, it is expected to decrease the CPU utilization close to that of IGEA without the precedence evaluation because the occurrence percentage of precedence evaluations decreases. On the other hand, if the tournament size increases, it is also expected that the efficiency of the proposed method decreases as well as the population size because the number of individuals required to generate new (tentative) offspring increases. From the above point of view, we are concerned that the proposed method will be adversely affected by increasing population size and tournament size.

To overcome such problems, we can improve the CPU utilization by changing the condition of tentative offspring generation and actively performing the precedence evaluation when the population size or tournament size are large. Specifically, in the proposed method, when three out of the four individuals necessary for generating offspring are determined, four tentative offspring are generated in $P-(\lambda, \lambda)$-IGEA. However, by relaxing this condition, for example, generating eight tentative offspring when two out of the four individuals are determined, CPU efficiency can be improved. This relaxation method reduces the waiting time when the population size or tournament size increases, and it allows us to utilize the available computing resources as much as possible. This analysis will be tackled in future research.

Figure 11. The CPU utilization of the proposed method with and without the evaluation suspension. (a) Fixed, (b) Uniform, (c) Exponential, (d) Negative correlation, and (e) Positive correlation.

7.3. Applicability of the proposed method

Finally, this subsection discusses the applicability of the proposed method. Although this paper used a real-coded GA in the experiments, the application scope of the proposed method is not limited to it. However, it should be noticed that the proposed method cannot be applied to all existing EA methods. In particular, the following conditions must be met to apply the proposed method.

**Condition 1**: Offspring can be generated with only a few individuals in the population.

**Condition 2**: Part of individuals in the new population can be determined without evaluating all individuals.

The first condition is necessary because the proposed method can achieve the interleaving generation by generating offspring without waiting for all population evaluations. For example, the roulette-wheel selection does not satisfy Condition 1 because it requires evaluating all individuals in the population. Another example is particle swarm optimization (PSO) [17]. Since PSO uses the global best to calculate the next position and velocity, it requires the evaluations of the entire population and does not satisfy Condition 1.

The second condition is also necessary. If the new population is selected depending on the evaluations of the entire population, no aspirant is determined without waiting for all evaluations. For example, IBEA [18], a well-known multi-objective EA, selects the next
population according to the indicator calculated from the entire population, and it cannot be performed until all evaluations have been obtained.

On the other hand, if these conditions are satisfied, the proposed method can be applied not only for the real-coded GA, but also other EA methods for combinatorial or structural optimization. One example applicable to the proposed method is differential evolution (DE) [19]. DE satisfies both of the two conditions. Precisely, DE satisfies Condition 1 because the offspring can be generated only with the parent individual and three randomly selected individuals. Also, DE can determine the next population by comparing the parent and offspring individuals, so Condition 2 is also satisfied. This detail is discussed in our another work [20].

Another example is crossover methods with multiple parents, such as SPX [21] and UNDX [22]. Since these crossover methods can generate offspring with some, but not all, individuals in the population, they can be applied to the proposed method. However, when using crossover requiring multiple parents, the efficiency of precedence evaluation by the proposed method decreases as the number of required parental individuals increases because it needs to wait for the completion of evaluation of those parent individuals.

8. Conclusion

This paper proposed an improved method of IGEA that introduces a mechanism of the precedence evaluation of tentative offspring and its suspension. To investigate the effectiveness of the proposed method, we compared the performance of the proposed IGEA with the original IGEA and a simple generation-based PEA. In the experiment, we considered two replacement schemes. One is \((\lambda, \lambda)\)-EA that replaces the population with newly generated offspring. Another is \((\lambda + \lambda)\)-EA that selects the best \(\lambda\) individuals from the current \(\lambda\) individuals and \(\lambda\) offspring.

In these experiments, we used the constant fitness function and three evaluation time functions fixed to 1 s, randomly sampled from the uniform distribution of 1 to 100 s, and sampled from the exponential distribution of 1 s on average. In addition, we used the Rastrigin function and the evaluation time function that negatively or positively correlated to the fitness value. The experimental results showed that the proposed \((\lambda, \lambda)\)-IGEA has higher CPU utilization and shorter execution time than the original \((\lambda, \lambda)\)-IGEA and the \((\lambda, \lambda)\)-PEA regardless of the features of the fitness function and the evaluation time. For the \((\lambda + \lambda)\) version, the experimental results were a similar tendency to those of \((\lambda, \lambda)\) version. Besides, the experimental results showed that the search capability of the proposed method is almost the same as the original IGEA on both the IGEA versions. Overall, the proposed IGEA with \((\lambda + \lambda)\) replacement scheme showed the best performance.

In the near future, we will propose a method to improve CPU utilization further. We will also analyse the influence of the parameters (population size and tournament size) in the proposed method. In addition to them, we will address to implement some concrete EA algorithms by using the proposed method and compare them with the original IGEA.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by Japan Society for the Promotion of Science Grant-in-Aid for Young Scientists [grant number JP19K20362].
Notes on contributors

Hayato Noguchi received the B.S. degree in engineering from Ritsumeikan University, Japan, in 2019. He has currently been a master student in Graduate School of Information Science and Engineering, Ritsumeikan University, Japan, since April 2019. His research interests include evolutionary computation, artificial intelligence, and entertainment computing. He won the 3rd Evolutionary Computation Competition (multi-objective optimization category) organized by the Japanese Society of Evolutionary Computation in 2019.

Akari Sonoda received the B.S. degree in engineering from Ritsumeikan University, Japan, in 2020. Her research interests include evolutionary computation and artificial intelligence.

Tomohiro Harada received the B.S., M.E., and Ph.D. degrees from the University of Electro-Communications, Japan, in 2010, 2012, and 2015, respectively, all in engineering. He was a Research Fellow of the Japan Society of the Promotion of Science DC1 from 2012 to 2015. He had been an assistant professor with College of Information Science and Engineering, Ritsumeikan University, Japan, from 2015 to 2019 and was a visiting researcher at University of Málaga, Spain, in 2018. He has currently been an assistant professor with Faculty of System Design, Tokyo Metropolitan University, Japan, since October 2019. His research interests include evolutionary computation, machine learning, and game AI. He is a member of IEEE, ACM, and major AI- and informatics-related academic societies in Japan. He was an editorial committee member of the Information Processing Society of Japan Journal from 2017 to 2021. He works for organizing major international conferences such as IEEE CoG 2020, and ACM GECCO 2018. Dr. Harada received the Journal Paper Award from System and Information, the Society of Instrument and Control Engineering in 2016, and IEEE Computational Intelligence Society Japan Chapter Young Researcher Award in 2018.

Ruck Thawonmas received the B.Eng. degree in electrical engineering from Chulalongkorn University, Bangkok, Thailand, in 1987, the M.Eng. degree in information science from Ibaraki University, Hitachi, Japan, in 1990, and the D.Eng. degree in information engineering from Tohoku University, Sendai, Japan, in 1994. He is currently a Professor with the College of Information Science and Engineering, Ritsumeikan University, Kusatsu, Japan, where he is leading the Intelligent Computer Entertainment Laboratory with around 40 of the lab graduates working in the game industry. He has authored/coauthored more than 200 peer-reviewed papers in both Japanese and English. In addition, his students won a number of game AI competitions, such as the first AIBIRDS Level Generation Champion at the IEEE Conference on Computational Intelligence and Games (IEEE CIG) 2016, the IEEE CIG 2014 StarCraft AI Competition, and the AIIDE 2014 StarCraft AI Competition. Prof. Thawonmas is currently an Associate Editor for the IEEE TRANSACTIONS ON GAMES and Games for Health Journal.

ORCID

Tomohiro Harada http://orcid.org/0000-0002-0704-4351

References

[1] Goldenberg DE. Genetic algorithms in search, optimization and machine learning; 1989.
[2] Koza JR, Koza JR. Genetic programming: on the programming of computers by means of natural selection. Vol. 1, MIT Press; 1992.
[3] Obayashi S, Jeong S, Shimoyama K, et al. Multi-objective design exploration and its applications. Int J Aeronaut Space Sci. 2010;11(4):247–265.
[4] Oyama A, Kohira T, Kemmotsu H, et al. Simultaneous structure design optimization of multiple car models using the k computer. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI); 2017 Nov. p. 1–4.
[5] Alba E, Luque G, Nesmachnow S. Parallel metaheuristics: recent advances and new trends. Int Trans Oper Res. 2013;20(1):1–48.
[6] Harada T, Alba E. Parallel genetic algorithms: a useful survey. ACM Comput Surv. 2020;53(4):1–39.
[7] Harada T, Takadama K. Asynchronous evaluation based genetic programming: comparison of asynchronous and synchronous evaluation and its analysis. In: Krawiec K, Moraglio A, Hu T, et al., editors. Genetic programming. Berlin, Heidelberg: Springer; 2013. p. 241–252.
[8] Harada T, Takadama K. Asynchronously evolving solutions with excessively different evaluation time by reference-based evaluation. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO ’14; New York, NY, USA: Association for Computing Machinery; 2014. p. 911–918.
[9] Závoianu A-C, Lughoffer E, Koppelstätter W, et al. Performance comparison of generational and steady-state asynchronous multi-objective evolutionary algorithms for computationally intensive problems. Knowl Based Syst. 2015;87:47–60. Computational Intelligence Applications for Data Science.
[10] Harada T, Takadama K. Analysis of semi-asynchronous multi-objective evolutionary algorithm with different asynchronies. Soft Comput. 2020;24:2917–2939.
[11] Harada T. Mathematical model of asynchronous parallel evolutionary algorithm to analyse influence of evaluation time bias. In 2019 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), 2019. p. 1–8.
[12] Scott EO, De Jong KA. Evaluation-time bias in asynchronous evolutionary algorithms. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO Companion 15; New York, NY, USA: Association for Computing Machinery; 2015. p. 1209–1312.
[13] Pilat M, Neruda R. Parallel evolutionary algorithm with interleaving generations. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 17; New York, NY, USA: Association for Computing Machinery; 2017. p. 865–872.
[14] Noguchi H, Sonoda A, Harada T, et al. Interleaving generation evolutionary algorithm with precedence evaluation of tentative offspring. In: 59th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE); 2020.

[15] Durillo JJ, Nebro AJ, Luna F, et al. A study of master-slave approaches to parallelize nsga-ii. In 2008 IEEE international symposium on parallel and distributed processing; IEEE; 2008. p. 1–8.

[16] Cantú-Paz E. Designing efficient and accurate parallel genetic algorithms (parallel algorithms) [PhD thesis]. USA: University of Illinois at Urbana-Champaign; 1999. AAI9952979.

[17] Kennedy J, Eberhart R. Particle swarm optimization. In: Proceedings of ICNN’95-International Conference on Neural Networks, Vol. 4; IEEE; 1995. p. 1942–1948.

[18] Zitzler E, Künzli S. Indicator-based selection in multi-objective search. In Yao X, Burke EK, Lozano JA, editors, Parallel Problem Solving from Nature – PPSN VIII. Berlin, Heidelberg: Springer; 2004. p. 832–842.

[19] Storn R, Price K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J Global Opt. 1997;11(4):341–359.

[20] Noguchi H, Harada T, Thawonmas R. Parallel differential evolution applied to interleaving generation with precedence evaluation of tentative solutions. In: Proceedings of the Genetic and Evolutionary Computation Conference; 2021. to appear.

[21] Tsutsui S, Yamamura M, Higuchi T. Multi-parent recombination with simplex crossover in real coded genetic algorithms. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1, GECCO’99; San Francisco, CA, USA: Morgan Kaufmann Publishers Inc; 1999. p. 657–664.

[22] Ono I, Kita H, Kobayashi S. A real-coded genetic algorithm using the unimodal normal distribution crossover. In: Ghosh A and Tsutsui S, editors, Advances in Evolutionary Computing: Theory and Applications. Berlin, Heidelberg: Springer; 2003. p. 213–237.