A Study of Self-Adaptive Semi-Asynchronous Evolutionary Algorithm on Multi-Objective Optimization Problem

Tomohiro Harada
College of Information Science and Engineering,
Ritsumeikan University
1-1-1, Noji-Higashi, Kusatsu
Shiga, Japan 525-8577
harada@ci.ritsumei.ac.jp

Keiki Takadama
Department of Informatics, The University of
Electro-Communications
1-5-1, Chofugaoka, Chofu
Tokyo, Japan 182-8585
keiki@inf.uec.ac.jp

ABSTRACT

This paper proposes a self-adaptive semi-asynchronous evolutionary algorithm, SA^{2}EA for short, and verifies its effectiveness on multi-objective optimization problems. SA^{2}EA is an extension of an asynchronous EA that continuously evolves solutions whenever one solution completes its evaluation in a parallel computation environment, unlike a conventional generation-based synchronous EA needs to wait for evaluations of all solutions in a population, which causes to waste much idle time of parallel computation nodes. In contrast to such asynchronous EA, SA^{2}EA adequately controls its asynchrony, which means the number of waited solutions, depends on the variance of evaluation time of solutions. To investigate the effectiveness of the proposed SA^{2}EA, this paper conducts the experiment on benchmark problems of multi-objective optimization where several variations of the variance of evaluation time are tested in pseudo-parallel computation environment. The experimental result reveals that the proposed SA^{2}EA outperforms the synchronous and the asynchronous EA with constant asynchrony not depends on the variance of evaluation time of solutions.

CCS CONCEPTS

•Mathematics of computing → Evolutionary algorithms; Bio-inspired optimization; •Computing methodologies → *Parallel algorithms;*

KEYWORDS

Asynchronous evolutionary algorithm; multi-objective optimization; parallelization; asynchrony; self-adaptation

ACM Reference format:
Tomohiro Harada and Keiki Takadama. 2017. A Study of Self-Adaptive Semi-Asynchronous Evolutionary Algorithm on Multi-Objective Optimization Problem. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3067695.3084221

GECCO '17 Companion, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3067695.3084221

1 INTRODUCTION

Evolutionary algorithms (EAs) have been applied to wide range of real world optimization problems [2, 18] because of their high optimization ability without any problem specific knowledge. When applying EAs to real world optimization problems, any solution evaluations may take much time due to physical simulation or measurement of actual consumption time. In such situation, a parallelization of fitness evaluations is a possible option to speed-up the optimization process. Parallelization techniques of EAs have been proposed [4, 5, 10, 24], however, if evaluation times of solutions differ from each other, conventional parallel EAs waste a lot of idle time to wait for completion of the longest fitness evaluation of solution. This is because conventional parallel EAs generate a next population after evaluating all solutions in a current population.

To overcome this problem, *asynchronous* evolution approach has been proposed, which continuously generates a new solution without waiting evaluations of other solutions, unlike the conventional *synchronous* approaches need to wait evaluations of all solutions in a population. Since the asynchronous EAs continuously evolve solutions in a parallel evaluation environment, it is possible to efficiently search solutions in the situation where evaluation times of solutions take time and differ from each other, such result has been presented in several previous researches [22, 23].

Most of the previous asynchronous EAs generate a next solution whenever each solution completes its evaluation. However, to improve search ability of a parallel EA, it can be considered more efficient that a parallel EA waits some, not only one, solution evaluations to utilize information of solution evaluations to generate new solutions. In concrete, if the variance in evaluation times is not large, it is more efficient to generate new solutions after waiting some or most evaluations in a population to utilize information of solution evaluation. While if the variance in evaluation times is large, an asynchronous approach is still better choice to reduce idle time. It is possible to improve search ability of parallel EAs by adjusting asynchrony depends on the trade-off between idle time to wait solution evaluations and search efficiency. From this fact, our recent paper introduces asynchrony in asynchronous EA and proposes a *semi-asynchronous* EA (SAEA) In addition to this, this paper explores a *self-adaptive* SAEA, SA^{2}EA for short, that adjusts its asynchrony depends on the tendency of the variance of evaluation times.

To investigate the effectiveness of SA^{2}EA, this paper conducts experiments to solve multi-objective optimization problems (MOPs) with several variances of evaluation times by using multi-objective EA (MOEA), in particular NSGA-II [8] is employed in this paper.

This paper employs a computational time model in a parallel environment proposed in [30], and compares four kind of parallelization, synchronous, complete asynchronous, semi-asynchronous, and self-adaptive semi-asynchronous MOEAs. We also test several variants of an adaptation of asynchrony in a self-adaptive semi-asynchronous MOEA.

The remaining of this paper is organized as follows. Section 2 shows some related works regarding asynchronous evolutionary algorithms and their analyses. Section 3 explains the concept of semi-asynchronous EA and shows the relation between the variance of evaluation times and asynchrony. Section 4 proposes the self-adaptive semi-asynchronous EA and shows the self-adaptive semi-asynchronous NSGA-II as an example. Section 5 describes the experimental settings where the synchronous, the semi-asynchronous, and the self-adaptive semi-asynchronous NSGA-IIs are compared, and Section 6 shows its results. Finally, Section 7 concludes this paper and presents future works.

2 RELATED WORKS

2.1 Asynchronous evolutionary algorithm

Several researches proposed asynchronous EAs for single- and multi-objective optimization problem. This paper focuses on the master-slave parallelization where a master computation node executes main process of EA like initialization, selection, solution generation through genetic operators and population maintenance, while many slave nodes execute evaluation process in parallel. This is because most of previous researches with respect to asynchronous evolution employ such master-slave parallelization.

As simple extensions of existing synchronous EAs, asynchronous steady-stage GP (ASSGP) [15], asynchronous particle swarm optimizaiton (APSO) [3, 13], asynchronous differential evolution (ADE) [16, 26] were proposed, and APSO was also extended to multi-objective APSO (MAPSO) for solving MOPs [14]. For asynchronous MOEAs, Depolli et al. proposed asynchronous master-slave parallelization of differential evolution for multi-objective optimization (AMS-DEMO) [9] that is an asynchronous extension of DEMO [19]. Some recent researches also proposed asynchronous MOEA, for instance, Santander-Jimenez and Vega-Rodriguez proposed asynchronous non-generational indicator-based multi-objective bat algorithm (ANIMOBA) [20] that is an asynchronous extension of bat algorithm for MOPs [21], while in [25], Wessing et al. compares synchronous and asynchronous variant of S-metric selection evolutionary multi-objective algorithm SMS-EMOA [1].

Scott et al. analyzed behavior of asynchronous EA on the evaluation time different problems [22, 23]. In [23], they compared a synchronous $(\mu + \lambda)$ EA and an asynchronous $(\mu + 1)$ EA from the viewpoint of the relationship between fitness value of solutions and their evaluation time, and indicated that the asynchronous EA outperforms the synchronous one in all fitness-time relationships. On the other hand, in [22], it was indicated that an asynchronous EA increases the possibility to converge to local optima that is worse fitness value than the global optima but can be quickly evaluated.

2.2 Computational time model in a parallel environment [30]

Zvoianu et al. modeled the computational time of synchronous and asynchronous EAs in a parallel computational environment, and compared synchronous $(\mu+\lambda)$ EA with asynchronous $(\mu+1)$ (steady-state) EA [30]. This comparison is conducted on MOPs and two typical MOEAs, NSGA-II [8] and SPEA2 [28], are employed. In their research, assuming the master-slave computational environment that consists of single master node and λ slave nodes. A master node executes the main process of EA by consuming time t_s, while each slave node evaluates a received solution by consuming time t_p.

Their previous research did not only model the computational time of synchronous and asynchronous EAs in a parallel environment, but also they conducted an experimental comparison of these EAs on MOPs with different variance of evaluation time of solutions. Concretely, they defined a parameter c_v that decides the variance of evaluation time, and each solution evaluation requires a certain computational time decided by the normal distribution with mean t_p and standard deviation $t_p \times c_v$. From the result of this experiment, it was indicated that large variance of evaluation time increases the efficiency of an asynchronous EA in comparison with a synchronous one.

3 SEMI-ASYNCHRONOUS EVOLUTIONARY ALGORITHM

3.1 Overview

Previous researches proposed a lot of variation of asynchronous single- and multi-objective EAs and revealed their effectiveness in computational time variant optimization problems. However, asynchronous EAs have a problem that their search ability becomes worse than synchronous EAs in terms of same computational time when the variance evaluation times of solutions is not large. This is because bad influence due to not waiting for evaluations of other solutions in asynchronous EAs cannot be ignored. While in synchronous EAs, even though they waste much idle time of slave nodes, they have higher search ability than asynchronous ones because synchronous EAs can use evaluation information of all solutions when generating next population. From this fact, this paper proposes a *semi-asynchronous* EA that waits for $n = \lceil \alpha \times \lambda \rceil (1/\lambda \leq \alpha \leq 1)$ evaluations ($\lceil x \rceil$ means the ceiling function that maps a real number x to the smallest next integer) of slave nodes and executes the master process, unlike (complete) asynchronous EAs waits for only one evaluation. In other words, this means that a semi-asynchronous $(\mu + \lceil \alpha\lambda \rceil)$ EA that selects μ better solutions from original μ solutions and newly evaluated $\lceil \alpha\lambda \rceil$ solutions.

Hereafter, this paper denotes the parameter α that decides the number of evaluations to be waited as an *asynchrony* parameter, and $\lceil \alpha\lambda \rceil$ is simply represented as $\alpha\lambda$ unless otherwise noted. High asynchrony means that a semi-asynchronous EA is closer to a synchronous EA, e.g., waits for λ solutions when $\alpha = 1$, which is same as a synchronous EA. While low asynchrony means that it is closer to a complete asynchronous EA, e.g., waits for only one

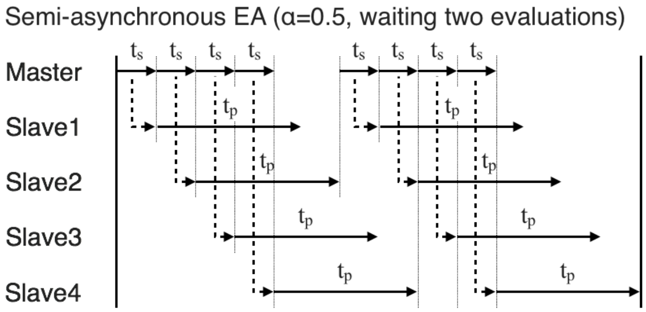


Figure 1: Semi-asynchronous ($\mu + \alpha\lambda$) EA where $\lambda = 4$ slave nodes and the asynchrony parameter $\alpha = 0.5$, i.e., $n = \alpha\lambda = 2$ evaluations out of four slave nodes are waited to execute the master process

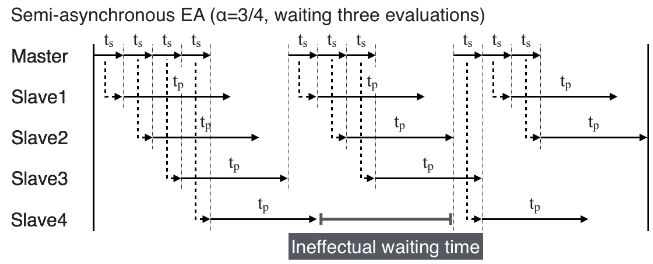


Figure 2: An example of ineffectual asynchrony of semi-asynchronous ($\mu + \alpha\lambda$) EA where $\lambda = 4$ slave nodes and the asynchrony parameter $\alpha = 3/4$, i.e., $n = \alpha\lambda = 3$ evaluations out of four slave nodes are waited to execute the master process

solution when $\alpha = 1/\lambda$, which is same as a complete asynchronous EA.

3.2 Computational time model generalized with consideration of asynchrony

According to the computational time model in [30], Fig. 1 depicts an illustration of a semi-asynchronous EA with the asynchrony parameter $\alpha = 2/\lambda$, i.e., $n = \lambda/2 = 2$ evaluations are waited. In this situation, as same as a synchronous and a complete asynchronous EAs, the initial generation takes $\lambda \times t_s + t_p$ computational time, while after that, a semi-asynchronous EA waits $\alpha\lambda$ evaluations (two evaluations in Fig. 1) and generates new $\alpha\lambda$ solutions. Accordingly, each generation, except for the initial generation, takes $\alpha \times \lambda \times t_s + t_p$ computational time after the last λth evaluation completes in each generation, and in total, the computational time of a semi-asynchronous EA to complete N generations, i.e., $N \times \lambda$ evaluations, is generalized and denoted as follows:

$$T(\alpha; N, \lambda) = (\lambda \times t_s + t_p) + (N-1) \times (\alpha \times \lambda \times t_s + t_p), \quad (1)$$

which is derived from sum of the initial computational time $\lambda t_s + t_p$ and $(N-1)$ generations of $\alpha\lambda t_s + t_p$. According to this generalized equation, a synchronous model is expressed as $T(1; N, \lambda)$, while a complete asynchronous model is expressed as $T(1/\lambda; N, \lambda)$.

In [30], a criterion that indicates how many solutions can be evaluated by an asynchronous EA in the time interval required by a synchronous EA to evaluate N generations of λ solutions was referred as the structural improvement measurement (Δ_{struct}). As same as this, this paper refers the *generalized* structural improvement $\Delta_{struct}(\alpha)$ that contains the asynchrony parameter α, which is given by the following equation:

$$\begin{aligned}\Delta_{struct}(\alpha) &= \frac{T(1; N, \lambda) - T(\alpha; N, \lambda)}{T(\alpha; N, \lambda)} \\ &= \frac{(N-1) \times (1-\alpha) \times \lambda \times t_s}{N \times (\alpha \times \lambda \times t_s + t_p) + (1-\alpha) \times \lambda \times t_s}.\end{aligned} \quad (2)$$

From this equation, it is easily indicated that $\Delta_{struct}(\alpha)$ monotonically decreases by increasing the asynchrony parameter α, i.e., the computational efficiency of a semi-asynchronous EA decreases by increasing the number of waiting solutions. From this criterion, in order to achieve better performance by semi-asynchronous EAs, it is necessary to accomplish the improvement of search ability surpassing the disadvantage of decrease of the computational efficiency.

3.3 Effectual and ineffectual asynchrony

In a semi-asynchronous EA, all values of α within the range of $1/\alpha$ to 1 cannot be used. This is because incorrect setting of α causes ineffectual parallelization. Concretely, an effectual α should satisfy the following restriction:

$$\lambda \equiv 0 \pmod{\lceil \alpha\lambda \rceil}, \quad (3)$$

where $a \equiv b \pmod{n}$ expresses congruent modulo where the difference of integers a and b, $(a-b)$, is multiple of a positive integer n. This is because if α does not satisfy the restriction of Eq. (3) and there exists some k that satisfies $\lambda \equiv k \pmod{\lceil \alpha\lambda \rceil}$, k slave nodes cannot be synchronized with other slave nodes and ineffectual waiting time occurs.

An example of such ineffectual asynchrony setting is shown in Fig. 2, where $\lambda = 4$, $\alpha = 3/4$ and $\lambda \equiv 1 \pmod{\lceil \alpha\lambda \rceil}$. In that case, as shown in Fig. 2, the first three evaluations by Slaves 1 to 3 are synchronized, and next three solutions are generated with starting their evaluations in these slave nodes. Then next evaluation by Slave 4 completes at almost the same time, but since other slaves just start their evaluations, Slave 4 mast wait for evaluations of other two out of three slaves, which causes the ineffectual waiting time occurs. It is possible that such ineffectual situation is avoided if evaluation times of solutions differ from each other, but this paper supposes that the restriction in Eq. (3) should be preserved in a semi-asynchronous EA.

3.4 Relation between the variance of evaluation time and asynchrony

As noticed in [30], the number of evaluations in the asynchronous EA increases by increasing the variance of evaluation time of solutions within the same computation time of the synchronous one. This tendency can be applied to the semi-asynchronous EA. In particular, not only the number of evaluations in the semi-asynchronous EA increases by increasing the variance of evaluation

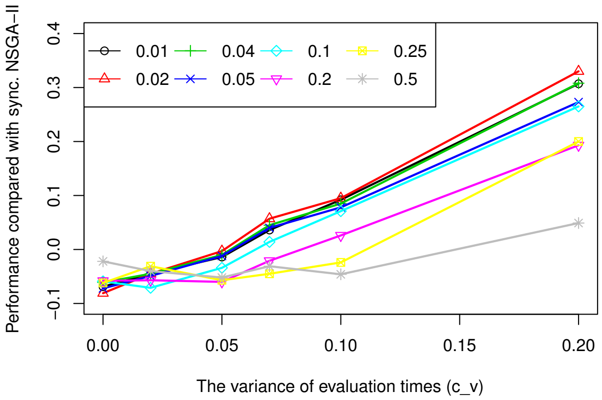


Figure 3: Relation between the variance of evaluation times and the performance of semi-asynchronous NSGA-II with several asynchrony

time of solutions within the same computation time of the synchronous one, but also it decreases by increasing the asynchrony parameter α [11].

Figure 3 shows the ratio how many evaluations are computed to achieve a certain performance by the semi-asynchronous EA with a certain asynchrony α in comparison with the synchronous EA. In this case NSGA-II [8] is employed, which is one of the most powerful MOEA method, and ZDT1 benchmark [27] that is two objective optimization problem is solved. The horizontal axis shows the variance of evaluation time of solutions, while the vertical axis shows the ratio of the number of evaluations to achieve 95% the Hypervolume [29], which is an indicator to assess the quality of the achieved solutions, between the semi-asynchronous and the synchronous EAs. The negative value indicates the semi-asynchronous NSGA-II is not better than the synchronous one, while the positive value indicates the semi-asynchronous NSGA-II outperforms the synchronous one. In this figure, each solution is evaluated in time determined from $\mathcal{N}(t_p, (c_v \times t_p)^2)$, where $\mathcal{N}(\mu, \sigma^2)$ denotes the normal distribution with the mean μ and the standard deviation σ, and the results of $c_v = \{0.0, 0.02, 0.05, 0.07, 0.10, 0.20\}$ are depicted. As shown in Fig. 3, the ratio increases by increasing the variance c_v, while the ratio decreases by increasing the asynchrony parameter α, i.e., by increasing the number of waited solutions. From this analysis, it is indicated that if the variance of the evaluation times is low, e.g., $c_v \leq 0.05$, the semi-asynchronous version is not efficient, while if the variance is large, e.g., $c_v > 0.05$, the semi-asynchronous one greatly outperforms the synchronous one and low asynchrony, e.g., $0.01 \leq \alpha \leq 0.05$ have better performance than high asynchrony.

4 SELF-ADAPTIVE SEMI-ASYNCHRONOUS EVOLUTIONARY ALGORITHM (SA2EA)

4.1 Adaptation of asynchrony parameter

From the analysis mentioned in the previous section, this paper proposes the self-adaptive semi-asynchronous EA that adapts its asynchrony depends on the measured variation of evaluation times during the evolution process. Concretely, we define the asynchrony function $\alpha(\hat{c_v})$ that is monotone decreasing function depends on the measured variance $\hat{c_v}$. We test four kind of the asynchrony functions as follows:

Step

$$\alpha_{step}(\hat{c_v}) = \begin{cases} 1.0 & \hat{c_v} \leq 0.05 \\ 0.01 & otherwise \end{cases} \tag{4}$$

Linear

$$\alpha_{linear}(\hat{c_v}) = 1.0 - \frac{\hat{c_v}}{0.2} \tag{5}$$

Inverted Sigmoid (ISig)

$$\alpha_{isig}(\hat{c_v}) = 1.0 - \frac{1.0}{1.0 + \exp(-\beta_{isig} \times (\hat{c_v} - 0.05))} \tag{6}$$

Exponentioal (Exp)

$$\alpha_{exp}(\hat{c_v}) = \exp(-\beta_{exp} \times \hat{c_v}) \tag{7}$$

Since the semi-asynchronous EA has the effectual asynchrony as mentioned in Section 3.3, the calculated asynchrony value is converted to the closest effectual asynchrony as follows:

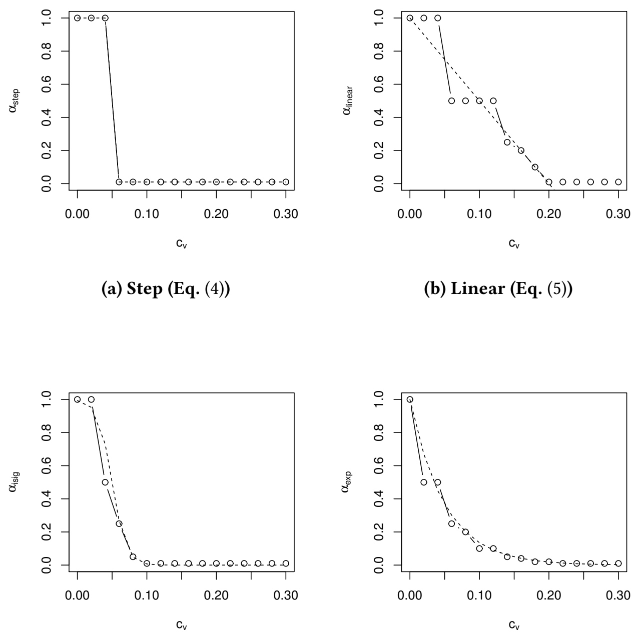
$$\alpha^{eff}(\alpha_{ada}, \hat{c_v}) = \underset{a \in \text{effectual } \alpha}{\arg\min} \ (|a - \alpha_{ada}(\hat{c_v})|), \tag{8}$$

where α_{ada} corresponds to any asynchrony function. The reason why Step function is separated at 0.05 is that the semi-asynchronous EA outperforms the synchronous one in the situation where the variance of evaluation time c_v is larger than 0.05 as shown in Fig. 3. As the same reason, Inverted Sigmoid function switches its inflection point at $c_v = 0.05$. In Equations (6) and (7), the parameters β_{isig} and β_{exp} control sensitivity of adaptation, and this paper employs $\beta_{isig} = 100.0$ and $\beta_{exp} = 20.0$ in both equations, which is decided to converge to $\alpha = 0.01$ when the variation c_v is larger than 0.2. Figure 4 shows adaptive asynchrony value calculated by each asynchrony function. In this figure, the horizontal axis indicates the variation of evaluation times c_v, while the vertical axis indicates the asynchrony value. The dashed line indicates the asynchrony value directly calculated by each function, while the solid line indicates the converted value by equation (8).

4.2 Self-adaptive semi-asynchronous NSGA-II: An example

To take an application of a SA^{2}EA into account, this section shows a concrete example of self-adaptive semi-asynchronous NSGA-II, SA^{2}NSGA-II for short. NSGA-II [8] is one of the most powerful MOEA method. The brief flow of the master node in the parallel synchronous NSGA-II is described as follows:

(1) Set generation counter $t = 0$.
(2) Initialization
    (a) Initialize population P_0.
    (b) Send all solutions to slave nodes.



(a) Step (Eq. (4)) **(b) Linear (Eq. (5))**

(c) Inverted Sigmoid (ISig) (Eq. (6)) **(d) Exponential (Exp) (Eq. (7))**

Figure 4: Adaptive asynchrony value calculated by each asynchrony function

   (c) Wait for evaluations of all solutions.
(3) Repeat until termination condition is satisfied
   (a) Generate offspring population, which size is equal to P_t, through binary tournament selection and genetic operator.
   (b) Send all offspring to slave nodes.
   (c) Wait for evaluations of all offspring $\rightarrow Q_t$.
   (d) $R_t = P_t \cup Q_t$.
   (e) Select next population P_{t+1} from merged population R_t according to the non-dominated sorting and the crowding distance metric.
   (f) $t = t + 1$.

In this flow, the role of slave nodes is evaluating a solution given from a master node and returning its evaluation to a master node.

Since NSGA-II is a special case of $(\mu + \lambda)$-EA where $\mu = \lambda = |P|$, it can be easily extended to a self-adaptive semi-asynchronous approach. Concretely, the parallel SA^{2}NSGA-II is executed as follows, where difference between synchronous and self-adaptive semi-asynchronous versions is denoted as *italic* style:

(1) Set generation counter $t = 0$.
(2) Initialization
   (a) Initialize population P_0.
   (b) Send all solutions to slave nodes.
   (c) Wait for evaluations of all solutions.
   (d) $\hat{c_v} = 0.0$ *(initialized by zero)*
   (e) $n = 0$
   (f) $\alpha = 1.0$ *(initialized by zero)*
(3) Repeat until termination condition is satisfied
   (a) Generate offspring population, which size is equal to *the number of idling slave nodes*, through binary tournament selection and genetic operator.
   (b) Send all offspring to *idling* slave nodes.
   (c) Wait for evaluations of $\alpha\lambda$ *offspring* $\rightarrow Q_t$.
   (d) $R_t = P_t \cup Q_t$.
   (e) Select next population P_{t+1} from merged population R_t according to the non-dominated sorting and the crowding distance metric.
   (f) $n = n + |Q_t|$
   (g) *if* n = *# of slaves*
      (i) $\hat{c_v} \leftarrow \dfrac{\text{standard deviation of evaluation time}}{\text{average evaluation time}}$
      (ii) $\alpha = \alpha^{eff}(\alpha_{ada}, \hat{c_v})$
      (iii) $n = 0$
   (h) $t = t + 1$.

The essential difference between the synchronous and the self-adaptive semi-asynchronous NSGA-II is 3-c to 3-g in the above flow. SA^{2}NSGA-II waits for $\alpha\lambda$ evaluations in every step and generates the same number of offspring from the current population. Note that the population selection according to the non-dominated sorting and the crowding distance metric is applied to $(\lambda + \alpha\lambda)$ solutions in R_t, unlike the synchronous NSGA-II applies them to $(\lambda + \lambda)$ solutions in R_t, and this difference may affect to the difference of search ability in the synchronous and semi-asynchronous NSGA-II. Then, SA^{2}NSGA-II calculates the average and the standard deviation of evaluation times of all evaluated solutions and calculates adaptive the asynchrony value α by Eq. (8). Such adaptation is executed every when solutions as the same number of slave nodes are evaluated. This is because to avoid the ineffectual waiting time by adapting the asynchrony value α every step.

5 PERFORMANCE COMPARISON

5.1 Experimental settings

To verify the effectiveness of the proposed SA^{2}EA, this paper conducts the performance comparison using the self-adaptive semi-asynchronous NSGA-II (SA^{2}NSGA-II), described in the previous section. We employ the typical multi-objective benchmark problems, the ZDT series (except for ZDT5) and the WFG series [12]. The ZDT series consists six two objective optimization problems, in which 30 decision variables in ZDT1, 2, and 4, while 10 decision variables in ZDT3 and 6 are optimized in this paper. The WFG series consists of nine scalable, multi-objective test problems, and this paper employs two objectives and six decision variables ($k = 4, l = 2$) WFG2–7 and WFG9. WFG1 and WFG8 are not employed in this paper because it is hard for all variation of NSGA-II in this paper to achieve enough quality of HV (more than 80% of the maximum HV).

This experiment is conducted on the simulated parallel computational environment, in which single master node and $\lambda = 100$ slave nodes work. t_p is set as 1000, which is 1000 times longer than the master process, and several variances of $c_v = \{0.0, 0.02, 0.05, 0.07, 0.10, 0.20\}$ are tested.

We compare complete synchronous, semi-asynchronous, and self-adaptive semi-asynchronous NSGA-IIs. Hereafter, these variants of NSGA-II are represented as CSNSGA-II, SANSGA-II, and SA^{2}NSGA-II, respectively. For them, the following genetic operations and the parameter settings are commonly employed in this experiment:

- Population size $\lambda = 100$
- The maximum number of evaluations = 50000 (=the maximum number of generation = 500)
- Simulated Binary Crossover (SBX) [6] with $P_c = 0.9$ and $\eta_c = 20.0$
- Polynomial Mutation (PM) [7] with $P_m = 1/D$ and $\eta_m = 20.0$

All effectual asynchrony of SANSGA-II are tested in this experiment as described in Section 3.3. Concretely, since the population size λ is set as 100, the asynchrony parameters $\alpha = \{0.01, 0.02, 0.04, 0.05, 0.1, 0.2, 0.25, 0.5, 1.0\}$ are effectual, in which $\alpha = 0.01$ corresponds to the complete asynchronous NSGA-II, while $\alpha = 1.0$ corresponds to CSNSGA-II. In SA^{2}NSGA-II, four asynchrony functions are tested, and each combination of SA^{2}NSGA-II and an asynchrony function is represented as SA^{2}NSGA-II/*func*, e.g., SA^{2}NSGA-II/*step*.

5.2 Evaluation criteria

In this experiment, we compare the variants of NSGA-II from the viewpoint of the computational time to achieve a certain quality of Pareto optimal solutions (POS) and the finally achieved POS. In particular, this paper employs the Hypervolume (HV) [29] indicator to evaluate the quality of achieved POS. For each benchmark problem, we measure the computational time to achieve a certain percentage, 95% of HV with true POS in $T^{x\%}$, and compare it with one measured in CSNSGA-II. Concretely, the percentage of how CANSGA-II, SANSGA-II and SA^{2}NSGA-II shorten the computational time than CSNSGA-II is calculated as follows:

$$\Delta^{x\%}_{time} = 100 \times \left(1 - \frac{T^{x\%}_{method}}{T^{x\%}_{CSNSGA-II}}\right), \tag{9}$$

and we evaluate that the compared methods are effective if $\Delta^{x\%}_{method}$ is greater than 0, while if $\Delta^{x\%}_{method}$ is less than 0, it indicates that the compared methods need more computational time than CSNSGA-II. On the other hand, HV of the achieved POS after the maximum number of evaluations is also compared to evaluate the convergence ability of all variants of NSGA-II. In this experiment, the achieved HV is compared with HV of true POS and the remaining percentage of HV achieved by the semi-asynchronous NSGA-II is calculated as follows:

$$\Delta_{HV} = 100 \times \left(1 - \frac{HV_{method}}{HV_{true}}\right), \tag{10}$$

where HV_{method} and HV_{true} indicate HV of the achieved POS by the variants of NSGA-II and the true POS, respectively.

Our experiment and the examined methods are implemented by using jMetal framework [17], and 25 independent trials are conducted for each combination of the benchmark problem, the variance of c_v, and the variants of NSGA-IIs.

6 RESULT

Table 1 shows the percentage of how SANSGA-II and SA^{2}NSGA-II shorten the computational time than CSNSGA-II calculated by equation (9). In this table, each row indicates the variance of the evaluation time, i.e., c_v. Each column indicates the result of the best case of SANSGA-II and SA^{2}NSGA-II with different asynchrony functions. In SANSGA-II, the best case is shown and its asynchrony parameter is denoted within the parentheses. The negative values, i.e., the worse performance than CSNSGA-II, are colored by gray in this table, while the best case for each problem and each c_v value is denoted as bold style. The Wilcoxon rank sum test is conducted as the statistical test. If the results of SA^{2}NSGA-IIs and the best case of SANSGA-II are significantly better than CSNSGA-II with 5% significance level, it is marked with "△", while if they are significantly worse, it is marked with "▼".

From Table 1, it is confirmed that SANSGA-II, which uses constant asynchrony value, has worse performance than the synchronous NSGA-II when the variance c_v is small, though it is effective when the variance c_v is large. SA^{2}NSGA-IIs with Step or ISig functions achieve better or equivalent performant in comparison with the synchronous NSGA-II when the variance c_v is low, while they greatly outperform the synchronous one when the variance c_v is high, though their performance are a little worse than SANSGA-II. On the other hand, SA^{2}NSGA-IIs with Linear or Exp functions do not achieve better performance regardless the variance c_v. This is because these two functions has possibility to adapt the asynchrony α to the middle value like 0.5 or 0.25, but such values are not effective in any variance of the evaluation time. In SA^{2}NSGA-II with ISig function, it is indicated that when the variance $c_v = 0.05$, SA^{2}NSGA-II has worse performance than SANSGA-II and CSNSGA-II. This is caused by the same reason of SA^{2}NSGA-IIs with Linear or Exp functions, which means ISig function adapts α value to the middle value when c_v is close to 0.05. Therefore, its performance decreases when the variance $c_v = 0.05$.

Focusing on the benchmarks ZDT4, WFG4, WFG7, and WFG9, it is indicated that SANSGA-II outperforms the synchronous NSGA-II even when the variance c_v is low. In such case, it is hard for SA^{2}NSGA-IIs in this paper to achieve such performance because they are designed to adapt the asynchrony α to high, mostly 1.0, when the variance c_v is low. To achieve the best performance of SANSGA-II in SA^{2}NSGA-II, it is revealed that the adaptation should be designed by considering not only the variance of evaluation time of solutions, but also the balance between the idling time and the search performance.

From these results, it is indicated the effectiveness of SA^{2}NSGA-II with the asynchrony functions like Step and ISig, which rapidly change the asynchrony value depending on the variance of evaluation time of solutions. In addition, further improvement of the asynchrony adaptation should be tackled by consider other indicator excluding the variance.

7 CONCLUSION

This paper proposed the concept of the self-adaptive semi-asynchronous EA (SA^{2}EA) that adapts the asynchrony parameter, which decides how many evaluations are waited to generate new solutions, depending on the variance of evaluation time of solutions in

Table 1: The percentage of how the semi-asynchronous, and the self-adaptive semi-asynchronous NSGA-IIs shorten the computational time than the synchronous one in each benchmark problems ($\Delta^{x\%}_{time}$). The best improvement greater than 0 is denoted as bold style for each variance of the evaluation time, i.e, c_v. If SA^{2}NSGA-IIs and the best case of SANSGA-II are significantly better than CSNSGA-II with 5% significance level, it is marked with "△", while if they are significantly worse, it is marked with "▼".

	ZDT1 ($\Delta^{95\%}_{time}$)					ZDT2 ($\Delta^{95\%}_{time}$)				
c_v	SA best (α)	step	linear	isig	exp	SA best (α)	step	linear	isig	exp
0.0	-2.5% (0.5)▼	1.1%	0.7%	**2.5%**	-3.5%▼	-4.9% (0.05)▼	**1.8%**	-1.5%	-0.7%	-3.5%▼
0.02	-2.7% (0.02)▼	0.7%	-0.2%	**0.8%**	-4.1%▼	-1.9% (0.25)	0.7%	**1.7%**	0.9%	-0.5%
0.05	-0.5% (0.05)	-1.2%	-2.8%▼	-3.1%▼	-4.3%▼	2.7% (0.01)	**3.3%**	-4.8%▼	-9.2%▼	-4.6%▼
0.07	**3.1%** (0.01)△	1.3%	-4.9%▼	-3.3%▼	-6.9%▼	**9.5%** (0.04)△	8.5%△	1.3%	4.2%	-2.3%
0.1	**8.9%** (0.04)△	8.4%△	-4.6%▼	6.7%△	4.3%△	**13.4%** (0.01)△	11.4%△	-3.8%▼	11.2%△	7.4%△
0.2	**23.2%** (0.01)△	22.7%△	21.5%△	21.5%△	22.5%△	**29.0%** (0.02)△	24.2%△	25.8%△	26.5%△	23.5%△

	ZDT3 ($\Delta^{95\%}_{time}$)					ZDT4 ($\Delta^{95\%}_{time}$)				
c_v	SA best (α)	step	linear	isig	exp	SA best (α)	step	linear	isig	exp
0.0	-2.3% (0.5)▼	-1.3%	-2.8%	**3.0%**	-2.1%	3.2% (0.05)	-0.8%	1.4%	2.1%	**3.4%**
0.02	-1.6% (0.04)▼	5.8%△	-0.7%	**7.4%**△	3.3%	**6.1%** (0.04)△	-3.4%	-2.6%	-4.0%	-2.8%
0.05	**4.2%** (0.05)	-0.4%	-0.5%	-1.9%	-5.3%▼	**6.8%** (0.04)△	5.2%	-0.1%	3.8%	3.9%
0.07	**3.3%** (0.01)	1.1%	-3.4%	-0.1%	-3.9%	**14.1%** (0.01)△	13.3%△	6.0%△	8.6%△	3.5%
0.1	6.3% (0.05)△	**6.6%**△	-7.7%▼	4.5%△	-16.6%	**18.4%** (0.04)△	17.5%△	5.3%△	17.9%△	10.5%△
0.2	**26.9%** (0.01)△	21.4%△	19.9%△	21.8%△	22.3%△	32.0% (0.01)△	29.7%△	**32.2%**△	28.6%△	29.0%△

	ZDT6 ($\Delta^{95\%}_{time}$)					WFG2 ($\Delta^{95\%}_{time}$)				
c_v	SA best (α)	step	linear	isig	exp	SA best (α)	step	linear	isig	exp
0.0	-4.0% (0.5)▼	0.5%	**1.2%**	0.4%	-2.9%▼	-2.8% (0.25)	**2.2%**	0.4%	-2.8%	-8.8%▼
0.02	-5.1% (0.01)▼	-0.7%	-0.2%	-1.4%	-6.6%▼	-2.9% (0.02)	1.2%	-1.7%	-7.4%	**3.8%**
0.05	-0.1% (0.01)	**0.7%**	-5.8%▼	-5.8%▼	-5.9%▼	**4.8%** (0.25)	1.0%	-4.7%	-1.3%	0.5%
0.07	**5.1%** (0.01)△	4.6%△	-5.1%▼	1.1%	-4.2%▼	**1.4%** (0.04)	-3.6%	-9.8%	-2.6%	-10.8%
0.1	9.9% (0.05)△	8.9%△	-2.1%	**10.0%**△	7.0%△	**10.6%** (0.05)△	9.1%△	-1.7%	10.5%△	1.6%
0.2	**24.8%** (0.01)△	24.0%△	24.5%△	24.4%△	23.2%△	**29.0%** (0.04)△	18.7%△	18.3%△	22.8%△	17.2%△

	WFG3 ($\Delta^{85\%}_{time}$)					WFG4 ($\Delta^{95\%}_{time}$)				
c_v	SA best (α)	step	linear	isig	exp	SA best (α)	step	linear	isig	exp
0.0	-6.0% (0.04)	-2.3%	-2.5%	**1.7%**	-2.7%	5.0% (0.05)	1.1%	2.3%	**5.3%**△	4.6%
0.02	-0.7% (0.25)	-1.2%	-0.6%	-2.6%	**2.6%**	7.3% (0.01)△	4.3%	2.9%	4.6%	**8.6%**△
0.05	**6.4%** (0.05)	4.4%	2.2%	-0.3%	-1.7%	**6.7%** (0.02)	6.3%	-2.3%	-7.4%	-0.1%
0.07	**1.8%** (0.25)	-0.7%	-5.0%	-0.4%	-11.6%▼	**12.0%** (0.01)△	**12.0%**△	0.8%	7.5%△	11.1%△
0.1	**11.4%** (0.01)△	10.8%△	-5.1%	7.0%△	6.3%△	12.4% (0.02)△	9.8%△	-1.1%	**13.3%**△	9.1%△
0.2	**24.4%** (0.01)△	16.9%△	19.1%△	22.7%△	18.0%△	22.9% (0.02)△	21.8%△	**24.1%**△	20.5%△	23.8%△

	WFG5 ($\Delta^{80\%}_{time}$)					WFG6 ($\Delta^{80\%}_{time}$)				
c_v	SA best (α)	step	linear	isig	exp	SA best (α)	step	linear	isig	exp
0.0	-2.9% (0.5)	0.6%	-0.4%	**3.2%**	-9.4%▼	-5.4% (0.2)	-35.1%	-17.4%	-47.8%▼	-22.8%
0.02	-5.2% (0.5)▼	**0.8%**	-0.2%	-1.1%	-5.0%▼	**33.4%** (0.02)	0.1%	18.9%	24.4%	22.3%
0.05	-2.5% (0.01)	-3.1%	-4.8%	-8.0%▼	-3.9%	-0.8% (0.25)	-1.8%	-39.0%	-14.3%	-11.5%
0.07	**6.3%** (0.02)△	-0.6%	-3.2%	0.8%	-1.1%	33.8% (0.02)	**36.5%**△	35.5%	26.4%	-22.2%
0.1	**7.1%** (0.02)△	6.1%△	-5.5%▼	6.9%△	4.9%△	**32.5%** (0.04)△	14.5%△	14.3%	26.6%△	**32.5%**△
0.2	**23.0%** (0.05)△	17.0%△	20.3%△	20.4%△	22.8%△	35.1% (0.2)△	**36.2%**△	30.7%△	13.3%△	25.9%△

	WFG7 ($\Delta^{95\%}_{time}$)					WFG9 ($\Delta^{95\%}_{time}$)				
c_v	SA best (α)	step	linear	isig	exp	SA best (α)	step	linear	isig	exp
0.0	**1.1%** (0.05)	0.3%	-0.8%	-3.1%	-1.4%	**79.4%** (0.01)△	-12.4%	55.9%	58.6%	13.6%
0.02	**3.1%** (0.01)	-3.4%	-8.7%▼	-3.0%	0.8%	3.7% (0.01)△	1.6%	-19.0%△	-4.3%△	**32.1%**△
0.05	**4.8%** (0.02)	1.9%	-0.3%	1.3%	-1.7%	72.7% (0.02)△	69.8%△	70.0%	73.7%	**75.4%**
0.07	**12.1%** (0.01)△	10.4%△	-3.7%	2.9%	3.6%	40.5% (0.02)△	**50.3%**△	-17.1%△	11.4%△	17.4%△
0.1	**15.9%**△ (0.04)	8.1%△	-1.4%	10.3%△	12.0%△	58.5%△ (0.25)	8.1%△	39.6%	60.4%△	**61.8%**
0.2	**29.7%** (0.02)△	26.2%△	24.3%△	27.9%△	24.3%△	67.1% (0.5)△	45.4%△	33.5%△	**68.2%**△	32.3%△

the parallel computational environment. This paper also discussed SA^{2}NSGA-II as an application of SA^{2}EA, and designed four types of adaptation functions, Step, Linear, Inverted Sigmoid, and Exponential. To investigate the effectiveness of SA^{2}NSGA-II, this paper conducted the experiment on multi-objective optimization problems, and compare it with the synchronous and the semi-asynchronous NSGA-II. The experimental result revealed that SA^{2}NSGA-II with the asynchrony functions of Step and Inverted Sigmoid achieves better performance not depends on the variance of evaluation time of solutions, though the asynchrony functions of Linear and Exponential are not work well when the variance is small.

What should be noted here is that since the experiment in this paper was conducted on the pseudo-parallel environment, the verification of the results on the actual parallel environment should be tackled soon. In addition, as mentioned above, we will design more effective asynchrony function or adaptation method that considers not only the variance of evaluation time of solutions but also other indicator like improvement of quality of achieved solutions.

REFERENCES

[1] Nicola Beume, Boris Naujoks, and Michael Emmerich. 2007. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research* 181, 3 (2007), 1653 – 1669. DOI:http://dx.doi.org/10.1016/j.ejor.2006.08.008

[2] Christina Bonnington. 2011. Teen's iOS App Uses Complex Algorithms to Summarize the Web. http://www.wired.com/2011/12/summly-app-summarization/. (Dec 2011).

[3] A. Carlisle and G. Dozier. 2001. An Off-The-Shelf PSO. In *PSO Workshop*. Indianapolis, IN. http://antho.huntingdon.edu/publications/Off-The-Shelf_PSO.pdf

[4] Jui-Fang Chang, Shu-Chuan Chu, John F. Roddick, and Jeng-Shyang Pan. 2005. A parallel particle swarm optimization algorithm with communication strategies. *Journal of Information Science and Engineering* (2005), 809–818.

[5] A Chipperfield and P Fleming. 1996. Parallel genetic algorithms. *Parallel and distributed computing handbook* (1996), 1118–1143.

[6] Kalyanmoy Deb and Ram B. Agrawal. 1995. Simulated Binary Crossover for Continuous Search Space. *Complex Systems* 9 (1995), 115–148. citeseer.ist.psu.edu/deb95simulated.html

[7] Kalyanmoy Deb and Mayank Goyal. 1996. A Combined Genetic Adaptive Search (GeneAS) for Engineering Design. *Computer Science and Informatics* 26 (1996), 30–45.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on* 6, 2 (apr 2002), 182–197. DOI:http://dx.doi.org/10.1109/4235.996017

[9] M Depolli, R Trobec, and B Filipic. 2013. Asynchronous Master-Slave Parallelization of Differential Evolution for Multi-Objective Optimization. *EVOLUTIONARY COMPUTATION* 21, 2 (2013), 261 – 291.

[10] Juan J. Durillo, Qingfu Zhang, Antonio J. Nebro, and Enrique Alba. 2011. Distribution of Computational Effort in Parallel MOEA/D. In *Learning and Intelligent Optimization*, Carlos A. Coello Coello (Ed.). Lecture Notes in Computer Science, Vol. 6683. Springer Berlin Heidelberg, Berlin, Heidelberg, 488–502. DOI:http://dx.doi.org/10.1007/978-3-642-25566-3_38

[11] Tomohiro Harada and Keiki Takadama. 2017. Performance Comparison of Parallel Asynchronous Multi-Objective Evolutionary Algorithm with Different Asynchrony. In *Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC 2017)*. to appear.

[12] Simon Huband, Luigi Barone, Lyndon While, and Phil Hingston. 2005. *A Scalable Multi-objective Test Problem Toolkit*. Springer Berlin Heidelberg, Berlin, Heidelberg, 280–295. DOI:http://dx.doi.org/10.1007/978-3-540-31880-4_20

[13] Byung-Il Koh, Alan D George, Raphael T Haftka, and Benjamin J Fregly. 2006. Parallel asynchronous particle swarm optimization. *International journal for numerical methods in engineering* 67, 4 (07 2006), 578–595. DOI:http://dx.doi.org/10.1002/nme.1646

[14] Andrew Lewis, Sanaz Mostaghim, and Ian Scriven. 2009. Asynchronous Multi-Objective Optimisation in Unreliable Distributed Environments. In *Biologically-Inspired Optimisation Methods*, Andrew Lewis, Sanaz Mostaghim, and Marcus Randall (Eds.). Studies in Computational Intelligence, Vol. 210. Springer Berlin Heidelberg, 51–78. DOI:http://dx.doi.org/10.1007/978-3-642-01262-4_3

[15] Sidney R. Maxwell III. 1994. Experiments with a Coroutine Model for Genetic Programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, Vol. 1. IEEE Press, Orlando, Florida, USA, 413–417a. DOI:http://dx.doi.org/doi:10.1109/ICEC.1994.349915

[16] A. Milani and V. Santucci. 2010. Asynchronous Differential Evolution. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010)*. 1–7. DOI:http://dx.doi.org/10.1109/CEC.2010.5586107

[17] Antonio J. Nebro, Juan J. Durillo, and Matthieu Vergne. 2015. Redesigning the jMetal Multi-Objective Optimization Framework. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15)*. ACM, New York, NY, USA, 1093–1100. DOI:http://dx.doi.org/10.1145/2739482.2768462

[18] Shigeru Obayashi, Shinkyu Jeong, Koji Shimoyama, Kazuhisa Chiba, and Hiroyuki Morino. 2010. Multi-Objective Design Exploration and its Applications. *International Journal of Aeronautical and Space Sciences* 4, 4 (Dec 2010). DOI:http://dx.doi.org/10.5139/IJASS.2010.11.4.247

[19] Tea Robič and Bogdan Filipič. 2005. *DEMO: Differential Evolution for Multiobjective Optimization*. Springer Berlin Heidelberg, Berlin, Heidelberg, 520–533. DOI:http://dx.doi.org/10.1007/978-3-540-31880-4_36

[20] Sergio Santander-Jiménez and Miguel A. Vega-Rodríguez. 2016. Asynchronous Non-Generational Model to Parallelize Metaheuristics: A Bioinformatics Case Study. *IEEE Transactions on Parallel and Distributed Systems* PP, 99 (2016), 1–1. DOI:http://dx.doi.org/10.1109/TPDS.2016.2645764

[21] Sergio Santander-Jiménez and Miguel A. Vega-Rodríguez. 2016. Performance Evaluation of Dominance-based and Indicator-based Multiobjective Approaches for Phylogenetic Inference. *Inf. Sci.* 330, C (Feb. 2016), 293–314. DOI:http://dx.doi.org/10.1016/j.ins.2015.10.021

[22] Eric O. Scott and Kenneth A. De Jong. 2015. 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)*. ACM, New York, NY, USA, 1209–1212. DOI:http://dx.doi.org/10.1145/2739482.2768482

[23] Eric O. Scott and Kenneth A. De Jong. 2015. Understanding Simple Asynchronous Evolutionary Algorithms. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII (FOGA '15)*. ACM, New York, NY, USA, 85–98. DOI:http://dx.doi.org/10.1145/2725494.2725509

[24] D.K. Tasoulis, N.G. Pavlidis, V.P. Plagianakos, and M.N. Vrahatis. 2004. Parallel differential evolution. In *Evolutionary Computation, 2004. CEC2004. Congress on*, Vol. 2. 2023 – 2029 Vol.2. DOI:http://dx.doi.org/10.1109/CEC.2004.1331145

[25] Simon Wessing, Günter Rudolph, and Dino A. Menges. 2016. *Comparing Asynchronous and Synchronous Parallelization of the SMS-EMOA*. Springer International Publishing, Cham, 558–567. DOI:http://dx.doi.org/10.1007/978-3-319-45823-6_52

[26] Evgeniya Zhabitskaya and Mikhail Zhabitsky. 2013. Asynchronous Differential Evolution with Restart. In *Numerical Analysis and Its Applications*, Ivan Dimov, Istvn Farag, and Lubin Vulkov (Eds.). Lecture Notes in Computer Science, Vol. 8236. Springer Berlin Heidelberg, 555–561. DOI:http://dx.doi.org/10.1007/978-3-642-41515-9_64

[27] E. Zitzler, K. Deb, and L. Thiele. 2000. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* 8, 2 (2000), 173–195.

[28] E. Zitzler, M. Laumanns, and L. Thiele. 2001. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. TIK Report 103. Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland.

[29] Eckart Zitzler and Lothar Thiele. 1998. *Multiobjective optimization using evolutionary algorithms — A comparative case study*. Springer Berlin Heidelberg, Berlin, Heidelberg, 292–301. DOI:http://dx.doi.org/10.1007/BFb0056872

[30] Alexandru-Ciprian Zflvoianu, Edwin Lughofer, Werner Koppelstter, Gnther Weidenholzer, Wolfgang Amrhein, and Erich Peter Klement. 2015. Performance comparison of generational and steady-state asynchronous multi-objective evolutionary algorithms for computationally-intensive problems. *Knowledge-Based Systems* 87 (2015), 47 – 60. DOI:http://dx.doi.org/10.1016/j.knosys.2015.05.029 Computational Intelligence Applications for Data Science.