Competitive-Adaptive Algorithm-Tuning of Metaheuristics inspired by the Equilibrium Theory: A Case Study

Kei Nishihara
Collage of Engineering
Yokohama National University
Yokohama, Japan
nishihara-kei-jv@ynu.jp

Masaya Nakata
Collage of Engineering
Yokohama National University
Yokohama, Japan
nakata-masaya-tb@ynu.ac.jp

This paper proposes a competitive-adaptive algorithm-tuning framework for metaheuristic algorithms. Our proposed method, called CAT, is inspired by the Equilibrium Theory in economics, which explains competitors eventually converge to an equilibrium status, e.g. in terms of the price of products. In detail, our proposal runs multiple optimizers with different algorithmic configurations, e.g. mutation variants. Then, the configurations of inferior optimizers are adaptively tuned so that they can derive good solutions that superior ones have derived. This intends to boost the performance even with a limited number of fitness evaluations, by the following technical advantage. The CAT preliminarily validates a search capacity of tuned algorithmic configurations and then constructs an ensemble optimizer by utilizing multiple optimizers. As a case study, this paper applies the CAT to tune the differential evolution algorithms (DEs). Experimental results show that our proposal outperforms the standard DE and an alternative approach i.e. jDE, which adapts hyper-parameters of genetic operators.

***Index Terms*—self-adaptation, algorithm tuning, differential evolution**

I. Introduction

While many successful versions of metaheuristic algorithms have been proposed, we often face to a practical difficulty: how we should select algorithmic configurations suitable to specific optimization problems. In fact, as pointed out in the no-free-lunch theorem, we can expect that there is a proper optimization algorithm for a specific problem. This expectation raises a challenge to automatically adapt algorithmic configurations of metaheuristics (or to construct an optimization algorithm, like hyperheuristics [1]). Considering a practical use of such an adaptation technique in real world problems, which may be computationally-expensive problems, a key is how efficiently adaptation techniques can improve the performance with a limited number of fitness evaluations.

For this challenge, self-adaptation of algorithm configurations is a promising approach. A main idea is to optimize the algorithm configurations during exploration so that improving the search capacity of specific metaheuristics. Thus far, various self-adaptation techniques have been proposed and succeeded in improving the performance compared with standard (non-adaptive) metaheuristics [2]–[4]. A popular approach is to adaptively tune (or control) hyper-parameter settings of a specific metaheuristic algorithm, e.g. APSO-VI [5] and SHADE [6] for a given optimization problem. In addition, algorithmic variants, e.g. mutation variants, can also be considered as options to specify algorithms, such as SaDE [7] and a grammatical evolution approach [8]–[10].

While those approaches have different technical advantages, we can suppose that they are based on a common adaptation strategy, called an *experience*-based adaptation in this paper. In detail, the common principle of those approaches is to inherit a superior algorithmic configuration that succeeded in producing good solutions. For instance, on the individual-level adaptation framework, hyper-parameter values are typically sampled from a defined distribution in which its average is set to values assigned to a good solution. However, this requires an assumption that configurations similar to its superior one may produce better solutions. Thus, the quality of a newly-adjusted configuration is not evaluated with any certain criterion until a solution produced with its configuration is evaluated. Consequently, the number of fitness evaluations may increase in order to improve the performance while discovering proper algorithmic configurations; but this is not accepted in computationally-expensive optimization problems.

Accordingly, this paper proposes a self-adaptation technique with a novel concept, inspired by the Equilibrium Theory [11] discussed in economics. In brief, the Equilibrium Theory argues that, under a condition that we temporally stop dynamics of the market influenced by e.g. tax rate, competitors eventually converge to an equilibrium status, e.g. in terms of prices of products, when they competitively make efforts to reduce the price. Accordingly, we design our proposed method, named as a competitive-adaptive algorithm-tuning framework (or CAT), based on the following concept;

- as the main motivation of our proposed method, we shift to a *quality*-based adaptation from the *experience*-

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

based adaptation. We tune the algorithmic configuration so that it produces good solutions discovered thus far. Technically, this intends to identify a good algorithm configuration validated to have a good search capacity before a solution produced with its configuration is evaluated;
- we integrate a competition framework into the adaptation technique to implement the above framework. We run multiple metaheuristics with different configurations as competition, and we compare their performances at prescribed intervals of generations, i.e. we temporally stop the dynamics of exploration. Then, inferior algorithm configurations are updated in order to derive similar solutions discovered by superior configurations. This intends to construct an ensemble optimizer by utilizing multiple optimizers.

Besides, we are also motivated to construct a self-adaptation technique on computationally-expensive optimization problems since its practical advantage should be more highlighted in this problem domain. Hence, our CAT framework aims to boost the performance of metaheuristics even at early generation, by producing the ensemble optimizer having good configurations. In this paper, we present a case study to test our proposed method; we employ the differential evolution algorithm (DE) [12] and adaptively tune the algorithmic configurations of DE, i.e. mutation variants, crossover variants and their hyper-parameters (a scaling factor F and a crossover rate CR). Hence, we run and then adapt multiple DEs with our CAT framework. Note that our proposed framework is also different from alternative adaptation techniques of DE, which are based on the *experience*-based adaptation, e.g. jDE [13] and JADE [14] as well as SHADE and SaDE.

This paper is organized as follows. Section II gives a brief description of the standard DE framework and the mutation and crossover variants, which will be tuned as configurations in this paper. In Section III, we explain the detailed mechanism of our CAT framework. Section IV tests our proposed method on a set of single-objective benchmark functions. Then, Section V demonstrates an adaptation result obtained by the experimental results. Finally, Section VI summaries our contributions and future work. All figures shown in this paper are best viewed in color.

II. Differential Evolution

This paper focuses on single-objective minimization problems; $f : \mathbb{R}^D \to \mathbb{R}$, where D is a problem dimension. This section gives a brief description of DE and then introduces both mutation variants and crossover variants as well as their hyper-parameters, which are treated as configurations to be tuned in our CAT framework.

The overall framework of DE is described in Algorithm 1. Firstly, as noted in *Initialization*, DE generates N solutions $\boldsymbol{x}_i$ ($i = 1, 2, \cdots, N$) and inserts them to a population $\mathcal{P}$ at the initial generation $t = 0$. In particular, for $t = 0$, j-th element $x_{i,j}$ ($j = 1, 2, \cdots, D$) of $\boldsymbol{x}_i$ is initialized as;

$$x_{i,j} = (x_{\max,j} - x_{\min,j})\textit{rand}[0, 1) + x_{\min,j}, \tag{1}$$

Algorithm 1 Differential Evolution

```
t = 0
for i = 1 to N do
    x_i ← Initialization
    Add x_i to P
while t < t_max do
    for i = 1 to N do
        v_i ← Generate mutant individual from P
        u_i ← Generate solution via crossover
    for i = 1 to N do
        if f(u_i) < f(x_i) then
            x_i ← u_i
    t = t + 1
```

TABLE I
Mutation variants

index	variant	definition
1	*rand/1*	$\boldsymbol{v}_i = \boldsymbol{x}_{r_1} + F(\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3})$
2	*rand/2*	$\boldsymbol{v}_i = \boldsymbol{x}_{r_1} + F(\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3}) + F(\boldsymbol{x}_{r_4} - \boldsymbol{x}_{r_5})$
3	*best/1*	$\boldsymbol{v}_i = \boldsymbol{x}_{best} + F(\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2})$
4	*best/2*	$\boldsymbol{v}_i = \boldsymbol{x}_{best} + F(\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2}) + F(\boldsymbol{x}_{r_3} - \boldsymbol{x}_{r_4})$
5	*current-to-rand/1*	$\boldsymbol{v}_i = \boldsymbol{x}_i + F(\boldsymbol{x}_{r_1} - \boldsymbol{x}_i) + F(\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3})$
6	*current-to-best/1*	$\boldsymbol{v}_i = \boldsymbol{x}_i + F(\boldsymbol{x}_{best} - \boldsymbol{x}_i) + F(\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2})$
7	*current-to-pbest/1*	$\boldsymbol{v}_i = \boldsymbol{x}_i + F(\boldsymbol{x}_{pbest} - \boldsymbol{x}_i) + F(\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2})$

where $x_{\max,j}$ and $x_{\min,j}$ are the upper and lower bounds of $x_{,j}$, respectively, and a function *rand*$[0, 1)$ returns a random real value between 0 and 1 sampled from the uniform distribution. Next, DE generates a new solution $\boldsymbol{u}_i$ for $\boldsymbol{x}_i$; it generates a mutant individual $\boldsymbol{v}_i$ for $\boldsymbol{x}_i$; and then it further generates $\boldsymbol{u}_i$ by a crossover operator, i.e. replacing elements of $\boldsymbol{x}_i$ with that of $\boldsymbol{v}_i$. Thus far, various mutation and crossover variants have been proposed. In detail, this paper employs seven popular mutation variants summarized in TABLE I. All the variants commonly use the hyper-parameter F. We can roughly classify the mutation variants as: *rand/1*, *rand/2* and *current-to-rand/1*, which heavily depend on randomly-selected solutions and tend to enhance a pressure of global search; other variants, i.e. *best/1*, *best/2*, *current-to-best/1* and *current-to-pbest/1* tend to enhance a pressure of global search but near to the current-best solution. Since *rand/2* and *best/2* employ two additional (randomly-selected) vectors compared with *rand/1* and *best/1*, respectively, these variants may improve a diversity of solutions. For instance, a *rand/1* variant generates $\boldsymbol{v}_i$ with the following equation;

$$\boldsymbol{v}_i = \boldsymbol{x}_{r_1} + F(\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3}), \tag{2}$$

where $\boldsymbol{x}_{r_1}$, $\boldsymbol{x}_{r_2}$ and $\boldsymbol{x}_{r_3}$ are solutions randomly selected from the current population $\mathcal{P}$; a scaling factor $F \in [0, 1]$ is a hyper-parameter. Then, the crossover operator with a hyper-parameter $CR \in [0, 1]$, i.e. a crossover rate, is applied. In this paper we employ two popular crossover variants, i.e. the binomial crossover and the exponential crossover; the detailed algorithms of these crossover variants are described

Algorithm 2 Binomial Crossover

$j_{rand} \leftarrow$ *randomly select an index of dimension from* $[1, D]$
for $j = 1$ **to** D **do**
  if *rand*$[0, 1) < CR$ **or** $j == j_{rand}$ **then**
    $u_{i,j} = v_{i,j}$
  else
    $u_{i,j} = x_{i,j}$

Algorithm 3 Exponential Crossover

$j \leftarrow$ *randomly select an index of dimension from* $[1, D]$
$k = 1$
$\boldsymbol{u}_i \leftarrow \boldsymbol{x}_i$
repeat
  $u_{i,j} = v_{i,j}$
  $j = (j + 1) \bmod D$
  $k = k + 1$
until *rand* $[0, 1) \geq CR$ **or** $k \geq D$

in Algorithms 2 and 3, respectively. Finally, The fitness of $\boldsymbol{u}_i$ is calculated, and $\boldsymbol{x}_i$ is replaced with $\boldsymbol{u}_i$ if $f(\boldsymbol{u}_i) < f(\boldsymbol{x}_i)$; otherwise $\boldsymbol{x}_i$ is used for the next generation. Those processes except for the initialization are repeated until the termination criterion is met, defined as $t < t_{\max}$ in this paper, where $t_{\max}$ is the maximum generation.

III. Competitive-Adaptive Algorithm-Tuning

This section describes our conceptual model inspired by the Equilibrium Theory in economics and then the detailed mechanism of the CAT.

A. Conceptual model

The Equilibrium Theory explains when competitors eventually converge to an equilibrium status under an ideal condition. In detail, as shown in Fig. 1 (top), three competitors *A*, *B* and *C* seek to maximize their profit by selling products, under complex dynamics of the market influenced by e.g. tax rate; they typically make their own efforts (e.g. cost-cutting) to reduce the price of their products to be competitive to other competitors, i.e. price competition. The Equilibrium Theory supposes an ideal condition where we temporally pause the dynamics of the market, that is, a static market's environment. Then, the Equilibrium Theory argues that the competitors eventually reach an equilibrium status, that is, they derive almost the same price of their products, if the competitors intensively make efforts to reduce the price under the ideal condition. Interestingly, this theory does not suppose that the competitors take similar strategies to reduce the price but argue that they eventually derive almost the same price by a pressure of price-competition in which each competitor can take its own strategy to reduce the price.

Accordingly, as shown in Fig. 1 (bottom), we design our proposed model as follows. Firstly, we run multiple optimization algorithms (or agents) in parallel as competitors; and each algorithm is designed to have a different algorithmic configuration from the other optimizers. As usual, each algorithm seeks to minimize an objective function under dynamics of solution exploration, like the price of the products. Then, we temporally

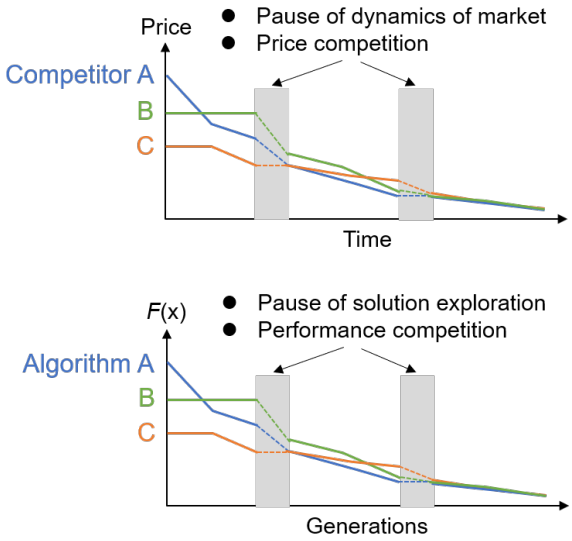


Fig. 1. Concept of the competitive-adaptive algorithm-tuning framework.

pause the solution exploration of all the algorithms and compare the current-best solutions derived by all the algorithms. Superior algorithms, which derived better solutions, are inherited for the restart of solution exploration to further explore the solution-space. However, for the other inferior algorithms, their algorithmic configurations are adjusted so that they derive almost the same solutions derived by the superior algorithms. Then, the inferior algorithms with adjusted configurations are used for the restart of the solution exploration.

B. Preliminaries

As a case study, this paper applies our CAT algorithm to the DE framework; and it tunes the variants of mutation and crossover strategies as well as their hyper-parameters during exploration. Thus, we run multiple DEs as competitors. Different from alternative methods, e.g. jDE, JADE, SHADE and SaDE, which can be classified as an individual-level tuning approach, our proposal is an algorithm-level tuning approach; each DE has its algorithm configuration and then it generates solutions with the same procedure as in the standard DE framework.

To specify the algorithm configuration of DE, we define a *configuration vector* θ, which will be tuned by our CAT algorithm. In detail, θ consists of the following four variables; an index of mutation variant $x_v = \{1, 2, \cdots 7\}$, which is a categorical value corresponding to the index of mutation variant noted in TABLE I; a scaling factor $x_F \in [0, 1]$, which is a real-value used as F for the x_v-th mutation variant; an index of crossover variant $x_u = \{0, 1\}$, which is a binary value and 0 and 1 indicate the binomial crossover and the exponential crossover, respectively; and a crossover rate $x_{CR} \in [0, 1]$, which is a real-value used as CR in a determined crossover variant. Note that we set the parameter $p = 1/2$ for *current-to-pbest/1*.

We also define a set of algorithms (i.e. optimizers) $\mathcal{A} = \{A^1, A^2, \cdots A^n\}$, where n is the number of algorithms. In our

case study, each algorithm A^i is set to a DE optimizer with a corresponding configuration vector $\theta^i = \{x^i_v, x^i_F, x^i_u, x^i_{CR}\}$, denoted by $A^i \leftarrow DE(\theta^i)$. Each algorithm A^i i.e. $DE(\theta^i)$ runs Algorithm 1 with an algorithmic configuration specified by θ^i. Here, we define that $\boldsymbol{x}^i_j \in \mathcal{P}^i$ is a j-th solution generated by i-th algorithm A^i and belongs to a population $\mathcal{P}^i$ of A^i. Hence, as in the standard DE, each $DE(\theta^i)$ has its own population $\mathcal{P}^i$, which consists of N solutions.

Note that the configuration vector can be extended dependent on metaheuristic algorithms; and thus CAT should be also applicable to different metaheuristics. Note also that as explained in the next subsection, each algorithm A^i can employ a different optimizer from other algorithms (forming an ensemble optimizer with heterogeneous optimizers), since each algorithm is tuned independently of the other algorithms; however, this paper leaves those possible extensions as future work.

C. Mechanism

Our CAT framework consists of the following four components; 1) *Initialization*, which sets an initial algorithmic configuration for each algorithm A^i, 2) *Validation*, which runs multiple algorithms and obtains their optimization results, 3) *Competition*, which identifies superior/inferior algorithms by comparing the obtained optimization results and 4) *Tuning*, which tunes algorithmic configurations of inferior algorithms so that they produce better solutions discovered by superior algorithms.

1) Initialization: CAT starts with an initialization process of $\mathcal{A}$. For each algorithm A^i, its configuration vector θ^i is initialized as follows. For x^i_F and x^i_{CR}, those values are commonly set as $x^i_F = 0.5$ and $x^i_{CR} = 0.9$, which are the initial values of jDE. However, for x^i_v, x^i_u, to improve the diversity of the configurations, a pair of mutation and crossover variants, i.e. $\{x^i_v, x^i_u\}$ is set to a different pair for each algorithm. After the initialization of $\mathcal{A}$ is conducted, N initial solutions of each algorithms are generated with the same procedure as in the standard DE.

2) Validation: This component runs n algorithms to explore the solution space as usual and to identify superior/inferior algorithms in the next component. Each algorithm A_i is independently executed by evolving solutions in $\mathcal{P}^i$; the maximum generations of one validation is defined by a hyper-parameter I named a *validation-interval*. Then, CAT collects the final optimization results of all the n algorithms, i.e. population $\mathcal{P}^i$ at the I-th generation. Thus, to complete validations for all the n algorithms, CAT takes $N \times I \times n$ fitness evaluations. However, this validation process is corresponding to the solution exploration in the standard DE; and thus the fitness evaluations are used only to discover better solutions.

3) Competition: This component aims to identify superior/inferior algorithms by comparing of the optimization results. Firstly, CAT combines all the n populations $\mathcal{P}^i$ $(i = 1, 2, \cdots, n)$ into a whole population denoted by $\mathcal{P}$. Thus, $\mathcal{P}$ includes $N \times n$ solutions. Then, CAT sorts solutions of $\mathcal{P}$ in ascending order (for minimization problems); and then it identifies top K solutions, i.e. $\boldsymbol{x}^*_1, \boldsymbol{x}^*_2, \cdots, \boldsymbol{x}^*_K$, where K $(0 \leq K \leq n)$ is a hyper-parameter defined in CAT. From those top K solutions, we identify superior/inferior algorithms. In detail, we define that an algorithm A^i is an inferior algorithm if any solution $\boldsymbol{x}^i_j \in \mathcal{P}^i$ is not involved in a set of top K solutions; otherwise, i.e. if at least one solution $\boldsymbol{x}^i_j$ is involved in the set of top K solutions, A^i is a superior algorithm. Let $\mathcal{A}'$ and $\mathcal{A}^*$ be a set of inferior algorithms and a set of superior algorithms, respectively. Then, we can define $\mathcal{A}^*$ and $\mathcal{A}'$ as;

$$\mathcal{A}' = \{A^i \in \mathcal{A} \mid \boldsymbol{x}^*_k \notin P^i \ \forall k = 1, 2, \cdots, K\}, \quad (3)$$

$$\mathcal{A}^* = \{A^i \in \mathcal{A} \mid A^i \notin \mathcal{A}'\}. \quad (4)$$

The superior algorithms in $\mathcal{A}^*$ (without any modification of θ^i) are re-used for the next validation process in order to further explore the search space, as they have good algorithmic configurations at the current generation. However, the inferior algorithms in $\mathcal{A}'$ will be updated in the next *Tuning* component. Note that if a superior algorithm A^* has generated more than one solution involved in the set of top K solutions, we allow CAT to duplicate and then add A^*s to $\mathcal{A}^*$ in order to promote A^*'s exploration by increasing its population size. Thus, the sizes of $\mathcal{A}'$ and $\mathcal{A}^*$ are always $|\mathcal{A}'| = n - K$ and $|\mathcal{A}^*| = K$, respectively.

In addition, we update the algorithm set $\mathcal{A}$ for the next validation. Firstly, we set $\mathcal{A}$ to an empty set. Then, each superior algorithm $A^* \in \mathcal{A}^*$ with its population produced at the last validation is added to $\mathcal{A}$. Note that, A^* generates N new solutions, which are evaluated for the beginning of next validation; and then we add A^* to $\mathcal{A}$. Hence, the superior algorithms continue to explore the search space without any modification of their algorithmic configurations. The inferior algorithms will be added to $\mathcal{A}$ after the *Tuning* component is conducted (see the next subsubsection).

4) Tuning: We tune the configuration vector θ^i of $A' \in \mathcal{A}'$, as the main component of CAT. A basic idea of our algorithm-tuning is to improve a diversity of algorithms in terms of their search capacity. This idea intends to employ a divide-and-conquer strategy. In detail, considering that one of the advantages of CAT is to construct an ensemble optimizer with multiple optimizers, each algorithm will be tuned to have a search capacity that explores a specific region of the solution space.

Technically, we here assume that top n solutions in $\mathcal{P}$, i.e. $\boldsymbol{x}^*_1$, $\boldsymbol{x}^*_2$, $\cdots$, $\boldsymbol{x}^*_K$, $\boldsymbol{x}^*_{K+1}$, $\cdots$, $\boldsymbol{x}^*_n$ are representative solutions existed in specific regions of the solution space; and each of those specific regions, i.e. a subspace around each representative solution, is worth to be further explored. However, we can consider that CAT has already produced algorithms having the search capacities that explores around the top K solutions $(0 \leq K \leq n)$, i.e. the superior algorithms $\mathcal{A}^*$. Thus, in this *Tuning* component, CAT tunes the inferior algorithms with the rest of top n solutions $\boldsymbol{x}^*_{K+1}$, $\cdots$, $\boldsymbol{x}^*_n$.

In detail, for $A'^i \in \mathcal{A}'$ $(i = 1, 2, \cdots, n - K)$, A'^i is tuned so that it derives a solution existed near to a representative solution $\boldsymbol{x}^*_{K+i}$. For instance, A'^1 will be tuned with $K + 1$-th solution i.e. $\boldsymbol{x}^*_{K+1}$. Next, as described in Algorithm 4, for

Algorithm 4 Algorithm Tuning()

while $|\mathcal{P}^i_{\epsilon_d}| < \theta_N$ **do**
  $\theta_i \leftarrow$ *set random values of* x^i_v, x^i_F, x^i_u, *and* x^i_{CR}
  for $j = 1$ **to** N **do**
    $\boldsymbol{v}^i_j \leftarrow$ *Mutation*$(x^i_v, x^i_F, \mathcal{P})$
    $\boldsymbol{u}^i_j \leftarrow$ *Crossover*(x^i_u, x^i_{CR})
    Add $\boldsymbol{u}^i_j$ *to* $\mathcal{P}^i$
  $\mathcal{P}^i_{\epsilon_d} \leftarrow \{\boldsymbol{u}^i_j \in \mathcal{P}^i \mid ||\boldsymbol{u}^i_j - \boldsymbol{x}^*_{K+i}|| < \epsilon_d\}$
return *DE*(θ^i), $\mathcal{P}^i$

the inferior algorithm A'^i, CAT initializes its configuration vector θ^i of A'^i; each variables, i.e. x^i_v, x^i_F, x^i_u and x^i_{CR} are randomly selected. Then, CAT generates N solutions $\boldsymbol{u}^i_j$ $(j = 1, 2, \cdots, N)$ with the whole population $\mathcal{P}$; as in DE, it generates a mutant individual $\boldsymbol{v}^i_j$ with a mutation variant (decided by x_v and x_F) based on selected individuals from $\mathcal{P}$, e.g. $\boldsymbol{x}_{r_1}$, $\boldsymbol{x}_{r_2}$ and $\boldsymbol{x}_{r_3}$ for the *rand/1* variant; then it generates a new solution $\boldsymbol{u}^i_j$ with a crossover variant decided by x_u and x_{CR}. Each solution $\boldsymbol{u}^i_j$ is added to a population $\mathcal{P}^i$ of A'^i. Next, $\mathcal{P}^i$ is used to validate whether A'^i with a tuned θ^i can have a proven search capacity to explore a subspace around $\boldsymbol{x}^*_{K+i}$. In detail, CAT counts the number of $\boldsymbol{u}^i_j$ near $\boldsymbol{x}^*_{K+i}$. Technically, it builds a subset of $\mathcal{P}^i$, denoted by $\mathcal{P}^i_{\epsilon_d}$, which consists of those solutions $\boldsymbol{u}^i_j$, given by;

$$\mathcal{P}^i_{\epsilon_d} = \{\boldsymbol{u}^i_j \in \mathcal{P}^i \mid ||\boldsymbol{u}^i_j - \boldsymbol{x}^*_{K+i}|| < \epsilon_d\}, \tag{5}$$

where a hyper-parameter $\epsilon_d \in \mathbb{R}$ controls a tolerance of gap between $\boldsymbol{u}^i_j$ and $\boldsymbol{x}^*_{K+i}$. Then, we define that A'^i with the tuned θ^i can have a proven search capacity to explore a subspace around $\boldsymbol{x}^*_{K+i}$, if the size of $\mathcal{P}^i_{\epsilon_d}$, i.e. $|\mathcal{P}^i_{\epsilon_d}|$ is greater than a hyper-parameter $\theta_N \in \mathbb{N}_0$ $(0 \leq \theta_N \leq N)$; otherwise CAT re-initializes θ^i and generates $\mathcal{P}^i$ until $\mathcal{P}^i$ satisfies the condition $|\mathcal{P}^i_{\epsilon_d}| > \theta_N$. After the tuning for each $A'^i \in \mathcal{A}'$ is completed, we employ θ^i with $\mathcal{P}^i$ as an initial population for the next validation; N solutions in $\mathcal{P}^i$ will be evaluated at the beginning of the next validation. Then, we set $A'^i \leftarrow$ *DE*(θ^i) and add A'^i with $\mathcal{P}^i$ to $\mathcal{A}$. Finally, CAT re-conducts the *Validation* process with the update algorithm set. Note that this *Tuning* process is called every $N \times n \times I + N \times n$ fitness evaluations; $N \times n \times I$ fitness evaluations to complete the *Validation* process; and $N \times n$ ones at the beginning of the validation to evaluate $N \times n$ initial solutions.

D. Summary

The CAT framework aims to tune the algorithmic configurations in terms of their search capacities, rather than inheriting previously-good configurations. Hence, without additional fitness evaluations, CAT identifies good configurations defied as having proven search capacities to produce solutions similar to good solutions discovered. Thus, we do not argue how configurations are set or inherited from previous configurations but CAT focuses on the quality of tuned configurations.

In addition, CAT requires the following five hyper-parameters; 1) the number of algorithms n, which controls the parallel number of algorithms to be executed and tuned; 2) the validation-interval I, which defines the maximum number of generations to validate tuned algorithms while exploring the search space; 3) the number of superior algorithms K, which defines how many algorithms are inherited to the next validation, 4) a threshold θ_N and 5) a tolerance ϵ_d, which control the quality of algorithm tuning in terms of the algorithm-level and the individual-level, respectively. Those additional parameters may be a drawback if we tune a single metaheuristic algorithm; in this case study, CAT can adaptively tune the four hyper-parameters of DEs but requires the five hyper-parameters of CAT. In fact, this drawback will be relaxed when we tune various types of metaheuristic algorithms together, as the number of hyper-parameters can increase in proportion to the number of type of metaheuristics; however we leave this extension as future work.

IV. Experiment

This section tests our CAT framework on single-objective benchmark problems with different problem dimensions.

A. Experimental design

As summarized in TABLE II, we use eight single-objective benchmark functions [13], [15]. For each benchmark function, we set the problem dimension $D = \{10, 20, 30\}$ and thus we here test our proposal on the 24 experimental cases. We add the CAT framework to multiple DEs, simply denoted by *ours* and compare our proposal with the standard DE and jDE as an alternative approach; jDE employs an individual-level adaptation strategy and it tunes the hyper-parameters F and CR of each individual during exploration. Note that our comparison with the three versions is minimally designed but enough to investigate the impact of our proposal, i.e. the competition framework with the *quality*-based adaptation strategy. We set the hyper-parameter settings of DE and jDE as; $N = 100$, $F = 0.5$, $CR = 0.9$, the *rand/1* variant and the binomial crossover are employed, where $F = 0.5$ and $CR = 0.9$ are used as their initial values in jDE. For both two versions, we set the maximum number of generations $t_{\max} = 1,000$ and thus the maximum fitness evaluations is $100,000$. For our proposal, we set $N = 10$, $n = 10$, $K = 3$, $I = 5$, $\theta_N = 5$, $\epsilon_d = (x_{\max} - x_{\min}) \times 0.1$ (see TABLE II). Note that we change values of ϵ_d dependent on the solution space of each benchmark function. Hence, in our *Validation* process, each DE runs for 5 generations, i.e. 50 fitness evaluations. Consequently, CAT requires 500 fitness evaluations to complete the validations of all 10 algorithms; and thus CAT executes algorithm-tuning, i.e. the *Tuning* component for every 600 fitness evaluations; additonal 100 evaluations are reqired for the initial solutions. We forcedly terminate our algorithm when the total number of fitness evaluations reaches $100,000$. Hence, all the three versions can be compared with the same fitness evaluations. The performance of each version will be evaluated as the best fitness discovered and reported as the median value of 30 trials. We report the performances measured at 1,200-th, 3,000-th, 30,000-th, and 100,000-th fitness evaluations, since we are motivated to investigate how the CAT algorithm improve the performance of alternative

TABLE II
A SUMMARY OF BENCHMARK FUNCTIONS EMPLOYED IN THIS PAPER

Function	Name	$[x_{\min}, x_{\max}]^D$	Definition
F1	Sphere	$[-100, 100]^D$	$F_1(x) = \sum_{i=1}^{D} x_i^2$
F2	Rosenbrock	$[-50, 50]^D$	$F_2(x) = \sum_{i=1}^{D-1} \left(100 \left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2\right)$
F3	Ackley	$[-50, 50]^D$	$F_3(x) = -20 \exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}) - \exp\left(\frac{1}{D}\sum_{i=1}^{D} \cos(2\pi x_i)\right) + 20 + e$
F4	Rastrgin	$[-50, 50]^D$	$F_4(x) = \sum_{i=1}^{D} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$
F5	Griewank	$[-100, 100]^D$	$F_5(x) = 1 + \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right)$
F6	Weierstrass	$[-0.5, 0.5]^D$	$F_6(x) = \sum_{i=1}^{D} \left(\sum_{k=0}^{k_{\max}} \left[a^k \cos\left(2\pi b^k (x_i + 0.5)\right)\right]\right) - D \sum_{k=0}^{k_{\max}} \left[a^k \cos\left(2\pi b^k \cdot 0.5\right)\right]$, $a = 0.5, b = 3, k_{\max} = 20$
F7	Schwefel	$[-500, 500]^D$	$F_7(x) = 418.9829 \times D - \sum_{i=1}^{D} x_i \sin(\sqrt{\lvert x_i \rvert})$
F8	Schwefel 1.2	$[-100, 100]^D$	$F_8(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j\right)^2$

approaches under a limited number of fitness evaluations, as noted in Section I; previous works, e.g. [13], [16] set more than 100,000 fitness evaluations. In addition, to find the significant difference, we apply the multiple-test for overall results of the 24 experimental cases for each fitness evaluation; we use the Friedman test and then we continue to apply the Holm method with the Wilcoxon Signed rank test as a post-hoc test if we find the significant difference from results of the Friedman test. A boundary of the significant probability is set to 0.05 in this paper.

B. Result

TABLE III summarizes the performances of DE, jDE and our proposal. For the results at 1,200-th fitness evaluation, we find the significant differences for all possible pairs of versions except for the pair of DE and jDE ($p < 0.05$) and their average ranks are 2.38 for DE, 2.31 for jDE and 1.31 for our proposal, respectively. For 3,000-th fitness evaluation, we also obtained a statistical result similar to this result, as well as results of 30,000-th fitness evaluation[1]. Thus, we can statistically confirm that our proposal is a powerful optimizer compared with DE and jDE on our experimental cases. This empirically validates that our CAT framework succeeds in improving the performance with a limited number of fitness evaluations since CAT identifies good algorithmic configurations without additional fitness evaluations. In addition, a complexity of algorithm-tuning in CAT should be increasing more than jDE, since CAT tunes mutation and crossover variants as well as their hyper-parameters. However, as our proposal succeeds in deriving the best performances on 21 experimental cases (see TABLE III), we can suppose our CAT may be well-scalable for the increase of the algorithmic configurations; but, this insight should be further investigated in future work.

However, for 100,000-th fitness evaluation, we cannot find any significant difference ($p > 0.05$) while the average ranks are 2.33 for DE, 1.92 for jDE and 1.75 for our proposal, respectively. Hence, while our proposal is still competitive to DE and jDE, we can suppose the performance of our proposal gradually degrades with the increase of fitness evaluations, which may be a drawback of the divide-and-conquer strategy employed in this paper; each algorithm is tuned to explore a specific region but this design may fall to a local solution. This supposition is further investigated in Section V.

In summary, the experimental result empirically confirmed our hypothesis that a *quality*-based adaptation strategy boosts the performance as compared with the *experience*-based adaptation strategy in this paper. However, a main drawback of our CAT framework is the increase of the computational cost. The CAT framework approximately requires fourfold computational times of jDE due to the time-consuming *Tuning* component. Hence, the current CAT framework may be suitable for computationally-expensive optimization problems, as it outperforms DE and jDE by reducing fitness evaluations, i.e. 1,200 and 3,000.

V. ANALYSIS

We conduct an empirical analysis to investigate how the algorithm-tuning affects the performance of our proposal. In detail, we here show examples of algorithm-tunings obtained by CAT. Fig. 2 shows an example of our tuning results of x_v, x_F, x_u and x_{CR} for 20,000 fitness evaluations; figures a)-d) are sampled from one trial of the experimental result of *F1* with $D = 10$. Note that x_v and x_u are reported with stacked curves; each curve indicates the number of algorithms that employ a corresponding mutation (or crossover) variant. Note also that our proposal derived the best performance as compared with DE and jDE on *F1* with $D = 10$ (see TABLE III).

From the figure, we can firstly identify that CAT intensively executed the algorithm-tuning till 4,200 fitness evaluations, like training of algorithms; after 4,200 fitness evaluations, the algorithmic configurations of all algorithms $(A_1, A_2, \cdots, A_{10})$ are not changed even when the *Tuning* component are conducted. This suggests that CAT completed to tune all of the algorithm configurations since each corresponding algorithm has a search capacity to produce a target solution $\boldsymbol{x}^*_{K+i}$. Note

[1]There are significant differences for all possible pairs of versions ($p < 0.05$) and their average ranks are 2.60 for DE, 2.04 for jDE and 1.35 for our proposal, respectively.

TABLE III
COMPARISON OF THE BEST FITNESS DISCOVERED AT 1,200-TH, 3,000-TH, 30,000-TH AND 100,000-TH FITNESS EVALUATIONS, RESPECTIVELY. THE MEDIAN VALUE OF ALL THE 30 TRIALS ARE REPORTED.

		1,200-th			3,000-th			30,000-th			100,000-th		
id	D	DE	jDE	ours	DE	jDE	ours	DE	jDE	ours	DE	jDE	ours
F1	10	4.32E+03	3.53E+03	**1.96E+03**	7.26E+02	5.83E+02	**2.77E+02**	1.05E+00	6.91E-01	**4.52E-04**	1.02E-36	8.66E-39	**1.41E-84**
	20	2.00E+04	2.03E+04	**1.09E+04**	9.86E+03	8.16E+03	**3.96E+03**	5.50E+02	2.20E+02	**2.85E+00**	3.53E-14	4.14E-18	**4.24E-40**
	30	4.08E+04	4.29E+04	**2.46E+04**	2.30E+04	2.22E+04	**1.03E+04**	3.31E+03	1.95E+03	**1.75E+02**	6.17E-08	5.75E-11	**2.92E-19**
F2	10	1.74E+07	1.52E+07	**4.35E+06**	5.99E+05	4.47E+05	**8.32E+04**	1.76E+02	3.19E+02	**9.24E+00**	1.14E-11	6.25E-03	**9.51E-18**
	20	2.78E+08	2.98E+08	**1.08E+08**	7.65E+07	6.37E+07	**1.32E+07**	3.52E+05	1.62E+05	**8.48E+02**	7.58E+00	1.15E+01	**3.15E+00**
	30	9.84E+08	9.54E+08	**2.94E+08**	3.22E+08	2.88E+08	**5.63E+07**	1.09E+07	6.10E+06	**5.00E+04**	2.33E+01	2.42E+01	**2.09E+01**
F3	10	2.01E+01	2.01E+01	**1.93E+01**	1.99E+01	2.00E+01	**1.63E+01**	1.87E+01	1.99E+01	**3.08E-01**	4.49E-14	1.11E-14	**4.00E-15**
	20	2.04E+01	2.04E+01	**2.04E+01**	**2.01E+01**	**2.01E+01**	**2.01E+01**	**1.99E+01**	2.00E+01	2.00E+01	**1.96E+01**	1.99E+01	2.00E+01
	30	2.06E+01	2.05E+01	**2.05E+01**	**2.02E+01**	2.03E+01	**2.02E+01**	**2.00E+01**	**2.00E+01**	**2.00E+01**	**1.99E+01**	2.00E+01	2.00E+01
F4	10	1.04E+03	1.01E+03	**5.85E+02**	2.90E+02	2.56E+02	**1.65E+02**	5.13E+01	3.24E+01	**2.73E+01**	1.93E+01	**0.00E+00**	6.20E-01
	20	5.13E+03	5.27E+03	**3.27E+03**	2.56E+03	2.12E+03	**1.22E+03**	3.11E+02	2.28E+02	**1.47E+02**	9.68E+01	1.76E+01	**1.64E+01**
	30	1.10E+04	1.08E+04	**6.24E+03**	6.77E+03	5.74E+03	**2.70E+03**	1.09E+03	7.63E+02	**3.37E+02**	1.86E+02	6.14E+01	**4.57E+01**
F5	10	1.91E+00	1.90E+00	**1.48E+00**	1.18E+00	1.12E+00	**1.07E+00**	5.71E-01	4.07E-01	**3.99E-01**	2.38E-01	**1.41E-13**	2.83E-02
	20	6.03E+00	6.10E+00	**3.75E+00**	3.30E+00	2.94E+00	**1.84E+00**	1.14E+00	1.06E+00	**3.49E-01**	2.46E-14	**0.00E+00**	1.97E-02
	30	1.12E+01	1.17E+01	**7.15E+00**	6.75E+00	6.56E+00	**3.57E+00**	1.78E+00	1.49E+00	**1.04E+00**	5.01E-09	**4.36E-12**	7.40E-03
F6	10	1.05E+01	1.01E+01	**8.91E+00**	6.99E+00	**5.69E+00**	5.70E+00	1.54E+00	5.65E-01	**1.51E-01**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	20	2.66E+01	2.66E+01	**2.32E+01**	2.23E+01	2.01E+01	**1.85E+01**	1.27E+01	6.76E+00	**4.20E+00**	1.27E-03	**3.27E-10**	2.52E-01
	30	4.34E+01	4.48E+01	**3.87E+01**	3.87E+01	3.64E+01	**3.22E+01**	2.66E+01	1.85E+01	**1.19E+01**	6.41E-02	**4.40E-04**	9.08E-01
F7	10	2.27E+03	**2.12E+03**	2.39E+03	2.07E+03	**1.71E+03**	1.97E+03	1.72E+03	**9.83E+02**	1.19E+03	1.17E-01	**1.27E-04**	**1.27E-04**
	20	5.55E+03	**5.40E+03**	5.71E+03	5.23E+03	**4.82E+03**	5.34E+03	4.91E+03	**3.71E+03**	4.26E+03	3.66E+03	**2.15E-02**	4.83E+02
	30	9.12E+03	**8.96E+03**	9.33E+03	8.81E+03	**8.17E+03**	8.95E+03	8.21E+03	**6.82E+03**	7.58E+03	6.86E+03	**2.85E+03**	3.06E+03
F8	10	5.14E+03	6.39E+03	**3.70E+03**	**1.73E+03**	2.96E+03	1.80E+03	**4.72E+01**	8.90E+02	4.74E+01	2.97E-19	1.92E-04	**1.14E-19**
	20	6.56E+04	7.10E+04	**5.94E+04**	3.76E+04	4.58E+04	**3.19E+04**	1.36E+04	2.84E+04	**6.24E+03**	**8.95E+00**	6.01E+02	2.00E+01
	30	3.86E+05	3.94E+05	**2.58E+05**	2.20E+05	2.57E+05	**1.76E+05**	9.38E+04	1.46E+05	**4.35E+04**	2.68E+03	7.51E+03	**1.86E+03**

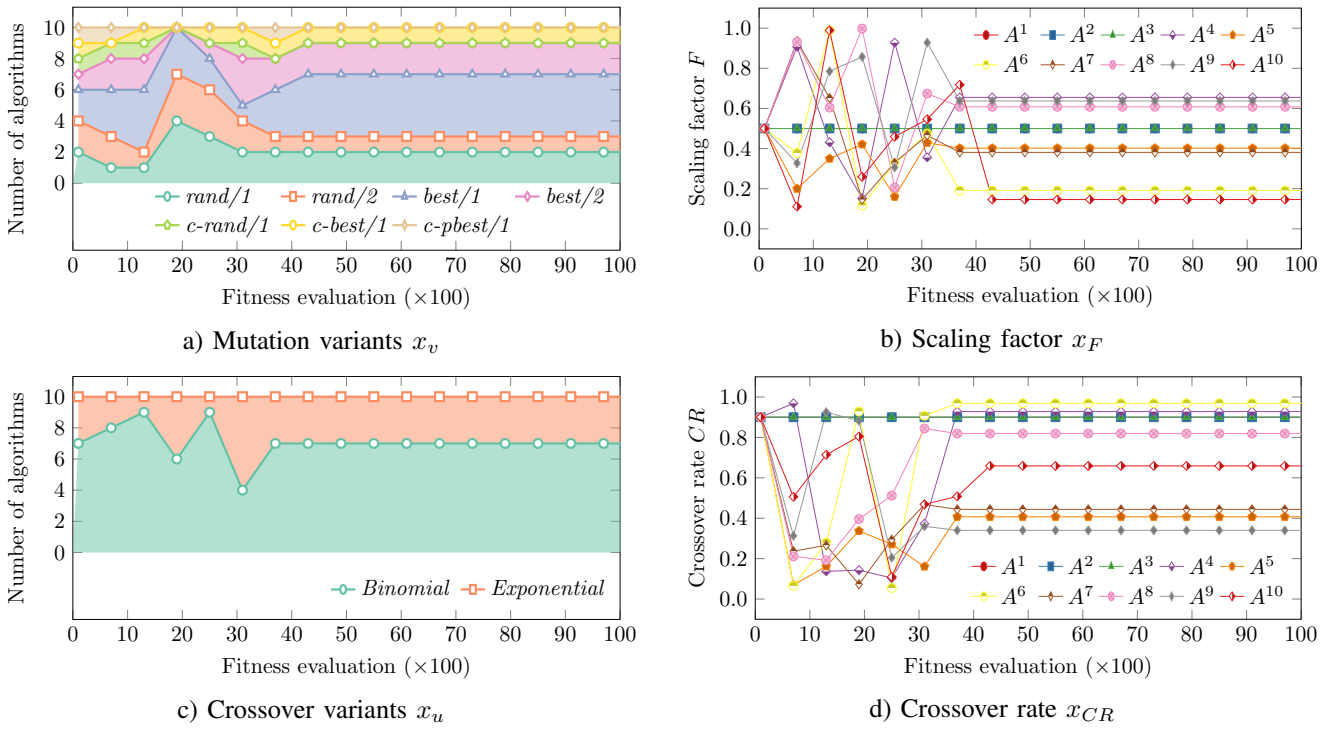
that as shown in Fig. 3, we obtained different tendencies from Fig. 2; the configurations are more frequently tuned on *F7* and *F8* with $D = 10$ as difficult problems than *F1*. Thus, after 4,200 fitness evaluations, we can suppose that CAT succeeded in discovering the algorithm configurations that continuously track a specific region around the corresponding target solution. However, this tendency would strongly depend on ϵ_d. In detail, since DE tends to intensively explore a specific region of the solution space with the improvement of the fitness score, the top n solutions may exist near to each other. Consequently, CAT may produce algorithms that explore similar specific regions for each other and thus CAT may fall to a local solution. While this paper sets ϵ_d to a fixed value, it may be suitable to adaptively tune a value of ϵ_d depending on the diversity of the top n solutions.

For a reasonability of the tuning results, while it is difficult to identify the best configuration on *F1* as well as other functions, we can roughly provide the following insight. Since *F1* is a globally-unimodal function, we can expect that an adequate strategy is to generate solutions by inheriting the best solutions. From Fig. 2 a) and b), we can identify that CAT eventually employs mutation variants using the best solution with lower values of F ($F \leq 0.5$), i.e. *best/1*, *best/2 current-to-best/1* and *current-to-pbest/1*. This tends to produce mutant individuals $\boldsymbol{v}$ near to the best solution. Besides, as reported in Fig. 2 c) and d), many algorithms employ the binomial crossover with high value of CR. Accordingly, those are designed to generate solutions $\boldsymbol{u}$ by inheriting the mutant individuals using the best solution since the binomial crossover with high value of CR tends to frequently replace elements of $\boldsymbol{u}$ with that of $\boldsymbol{v}$.

VI. CONCLUSION

In this paper, we proposed a competitive-adaptive algorithm-tuning framework, called CAT. Inspired by the Equilibrium Theory in economics, we tune an algorithmic configuration of a metaheuristic so that it produces solutions similar to the current-best solutions. Thus, CAT adaptively tunes the configurations while validating a search capacity (i.e. quality) of its configurations. With this framework, we intended to boost the performance with a limited number of fitness evaluations by the following technical advantages. Firstly, CAT can identify good algorithmic configurations without additional fitness evaluations. Secondly, like a divide-and-conquer strategy and an ensemble optimizer, CAT tunes multiple algorithms so that each algorithm explores a specific region of the solution space. Accordingly, as a case study, this paper used CAT to tune the differential evolution algorithm. The experimental result statistically confirmed that DE with CAT successfully outperforms the standard DE and jDE especially for the limited number of fitness evaluations, i.e. 1,200 and 3,000.

As our future work, we should further investigate how hyper-parameters of CAT affect the performance. Since our analysis revealed that the frequency of algorithm-tuning process degrades with the increase of fitness evaluations due to a fixed value of tolerance ϵ_d, we should further study the setting methodology for ϵ_d, e.g. an adaptive tuning dependent on a diversity of the solutions. Furthermore, in the CAT framework, each algorithm can be set to a different type of metaheuristics from other algorithms, e.g. PSO, we can also extend the CAT framework to construct a self-adaptive ensemble (heterogeneous) optimizer.



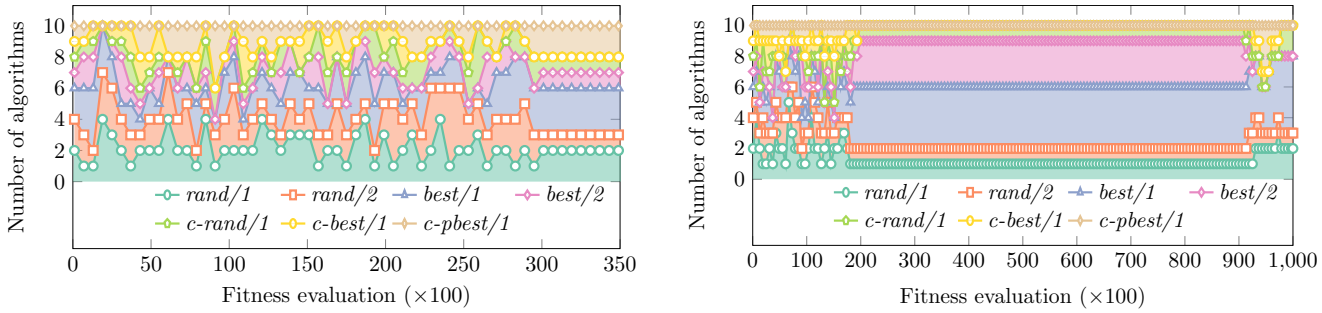
a) Mutation variants x_v

b) Scaling factor x_F

c) Crossover variants x_u

d) Crossover rate x_{CR}

Fig. 2. Examples of algorithm-tuning obtained by CAT on the *F1* with $D = 10$. The figures a)-d) report the tuning results of x_v, x_F, x_u and x_{CR}, which are sampled from one trial, respectively. Note that x_v and x_u are reported with stacked curves; each curve indicates the number of algorithms that employ a corresponding mutation (or crossover) variant.



a) *F7* with $D = 10$

b) *F8* with $D = 10$

Fig. 3. Examples of algorithm-tuning of mutation variants obtained by CAT on the *F7* and *F8* with $D = 10$.

REFERENCES

[1] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches: Revisited," in *Handbook of Metaheuristics*. Springer, 2019, pp. 453–477.

[2] Á. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on evolutionary computation*, vol. 3, no. 2, pp. 124–141, 1999.

[3] F. Lobo, C. F. Lima, and Z. Michalewicz, *Parameter setting in evolutionary algorithms*. Springer Science & Business Media, 2007, vol. 54.

[4] G. Karafotias, M. Hoogendoorn, and Á. E. Eiben, "Parameter control in evolutionary algorithms: Trends and challenges," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 167–187, 2014.

[5] G. Xu, "An adaptive parameter tuning of particle swarm optimization algorithm," *App. Math. Comp.*, vol. 219, no. 9, pp. 4560–4569, 2013.

[6] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *CEC2013*. IEEE, 2013, pp. 71–78.

[7] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. on Evolutionary Comp.*, vol. 13, no. 2, pp. 398–417, 2008.

[8] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[9] P. B. Miranda and R. B. Prudêncio, "A novel context-free grammar for the generation of pso algorithms," *Natural Computing*, pp. 1–19, 2018.

[10] A. Bogdanova, J. Pereira Junior, and C. Aranha, "Franken-swarm: grammatical evolution for the automatic generation of swarm-like meta-heuristics," in *Genetic and Evo. Comp. Conf. Comp.*. pp. 411–412,2019.

[11] L. Walras, *Elements of pure economics*. Routledge, 2013.

[12] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[13] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE transactions on evolutionary computation*, vol. 10, no. 6, pp. 646–657, 2006.

[14] J. Zhang and A. C. Sanderson, "Jade: adaptive differential evolution with optional external archive," *IEEE Transactions on evolutionary computation*, vol. 13, no. 5, pp. 945–958, 2009.

[15] B. Da, Y.-S. Ong, L. Feng, A. K. Qin, A. Gupta, Z. Zhu, C.-K. Ting, K. Tang, and X. Yao, "Evolutionary multitasking for single-objective continuous optimization: Benchmark problems, performance metric, and baseline results," *arXiv preprint arXiv:1706.03470*, 2017.

[16] X. Li and M. Yin, "Modified differential evolution with self-adaptive parameters method," *J. Com. Opt.*, vol. 31, no. 2, pp. 546–576, 2016.