Multi-Tasking Evolutionary Algorithm (MTEA)
for Single-Objective Continuous Optimization
Dongrui Wu and Xianfeng Tan

Abstract—Multi-task learning uses auxiliary data or knowledge from related tasks to facilitate the learning in a new task. Multi-task optimization applies multi-task learning to optimization to study how to effectively and efficiently tackle multiple optimization problems simultaneously. Evolutionary multi-tasking, or multi-factorial optimization, is an emerging subfield of multi-task optimization, which integrates evolutionary computation and multi-task learning. This paper proposes a novel easy-to-implement multi-tasking evolutionary algorithm (MTEA), which copes well with significantly different optimization tasks by estimating and using the bias among them. Comparative studies with several state-of-the-art single- and multi-task approaches in the literature on nine benchmarks demonstrated that on average the MTEA outperformed all of them, and has lower computational cost than six of them. Particularly, unlike other multi-task algorithms, the performance of the MTEA is consistently good whether the tasks are similar or significantly different, making it ideal for real-world applications.

Index Terms—Evolutionary algorithm, evolutionary multi-tasking, multi-task learning, multi-factorial optimization

I. INTRODUCTION

MULTI-task learning [3], [23] is a subfield of machine learning, particularly transfer learning [17], [22], [24], [25], which uses auxiliary data or knowledge from related/similar tasks to facilitate the learning in a new task. As a result, a learning model for the new task can be built with much less task-specific training data. Or, in other words, with the same amount of task-specific data, a much better model could be trained. In multi-task learning, multiple related learning tasks are performed simultaneously using a (partially) shared model representation. As a result, the common information contained in these related tasks can be exploited to improve the learning efficiency and generalization performance of each task-specific model.

Multi-task optimization (MTO) [6], [12], [16], [19] applies multi-task learning to optimization to study how to effectively and efficiently tackle multiple optimization problems simultaneously. Evolutionary multi-tasking [16], or multi-factorial optimization (MFO) [12], is an emerging subfield of MTO, which integrates evolutionary computation and multi-task learning. It assumes that each constitutive task has some (positive) influence on the evolutionary process of a single population of individuals, and hence evolving multiple populations from different tasks together simultaneously could be more efficient than evolving each individual task separately. A multi-factorial evolutionary algorithm (MFEA) has recently been proposed in [12] and demonstrated promising performance in synthetic and real-world MTO problems.

In this paper we consider single-objective multi-tasking optimization, where every point in the search space maps to a scalar objective value. There have been a few such approaches in the literature. Sagarna and Ong [18] applied multi-task evolutionary computation to concurrently searching branches in software tests generation. Yuan et al. [21] introduced two new improvements, a new unified representation and a new survivor selection procedure, to the MFEA and demonstrated their effectiveness. Cheng et al. [5] developed a particle swarm optimization based co-evolutionary multi-tasking approach for concurrent global optimization, and demonstrated its performance on synthetic functions and in real-world complex engineering design. Liew and Ting [15] proposed a general framework, the evolution of bioconiosis through symbiosis, for evolutionary algorithms to deal with many-tasking problems, and showed that its performance may be better than the MFEA. Tang, Gong and Zhang [20] employed evolutionary multi-tasking to evolving the modular topologies of extreme learning machine classifiers. Chen et al. [4] proposed an evolutionary multi-tasking single-objective optimization approach based on the cooperative co-evolutionary memetic algorithm. Local search based on the quasi-Newton approach was used to accelerate its convergence. Bali et al. [1] proposed a linearized domain adaptation strategy to improve the MFEA. It transforms the search space of a simple task to the search space similar to its constitutive complex task. This high order representative space resembles high correlation with its constitutive task and provides a platform for efficient knowledge transfer via crossover. The proposed LDA-MFEA demonstrated competitive performances against the MFEA. Wen and Ting [21] proposed two improvements to the MFEA (parting ways detection and resource reallocation), and showed that they can often result in better solutions, especially when the tasks share low similarity of landscapes. Ding et al. [9] proposed decision variable translation and shuffling strategies to facilitate knowledge transfer between optimization problems having different locations of the optima and different numbers of decision variables, and verified their effectiveness in multi-tasking optimization. Feng et al. [10] used autoencoding to explicitly transfer knowledge across tasks in evolutionary multi-tasking, and demonstrated its performance on both single- and multi-objective multi-task optimization problems. Hashimoto et al. [14] pointed out that the MFEA can be viewed as a special island model, and proposed a simple im-
plementation of evolutionary multi-tasking using the standard island model, which achieved promising performance.

This paper proposes a novel multi-tasking evolutionary algorithm (MTEA), which on average outperforms seven state-of-the-art approaches on nine multi-tasking benchmarks [6]. It possesses two important advantages, compared with existing MFO approaches in the literature:

1) The MTEA is robust to the differences among the tasks, i.e., it can still achieve superior performance when the differences among the tasks are so large that the performances of existing MTO algorithms degrade significantly. This is very beneficial in practice because usually we do not know in priori how large the differences between different tasks are, and hence the MTEA would be a much safer choice than other approaches.

2) The MTEA is very easy to implement, and has much lower computational cost than most existing MFO approaches.

The remainder of the paper is organized as follows: Section II introduces the MFEA approach. Section III proposes our MTEA approach. Section IV presents experimental studies to validate the performance of the MTEA. Section V draws conclusion and points out several future research directions.

II. MULTI-FACTORIAL EVOLUTIONARY ALGORITHM (MFEA)

This section introduces the MFEA, which is the earliest and most popular MTO approach, and also a baseline in this paper.

We use the same problem setting as the one in [12]. Without loss of generality, assume there are $M$ tasks, all of which are minimization problems. The $m$th task, $T_m$, has an objective function $f_m : X_m \rightarrow \mathbb{R}$ on a search space $X_m$. Each task may also be constrained by several equalities and/or inequalities that must be satisfied by a feasible solution. The goal of the MFEA is to find:

$$\{x_1^*, \ldots, x_M^*\} = \{\arg\min_{x_1} f_1(x_1), \ldots, \arg\min_{x_M} f_M(x_M)\}, \quad (1)$$

where $x_m^*$ is a feasible solution in $X_m$, $m = 1, \ldots, M$.

The following important definitions are used in the MFEA [12]:

Definition 1: For a given task $T_m$, the factorial cost $\Psi_{nm}$ of $x_n$ is $\Psi_{nm} = f_m(x_n) + \lambda \cdot \delta_{nm}$, where $f_m(x_n)$ and $\delta_{nm}$ are the objective value and the total constraint violation, respectively, and $\lambda$ is a large penalizing multiplier.

Definition 2: The factorial rank $r_{nm}$ of $x_n$ on Task $T_m$ is the index of $x_n$ in the list of population members sorted in ascending order w.r.t. the factorial cost $\Psi_{nm}$.

Definition 3: The skill factor $\tau_n$ of $x_n$ is the particular task that $x_n$ is associated with, or, in other words, in which $x_n$ has the smallest factorial rank among all $M$ tasks, i.e., $\tau_n = \arg\min_m r_{nm}$.

Definition 4: The scalar fitness of $x_n$ in a multi-tasking environment is $\phi_n = 1/r_{n, \tau_n}$.

The MFEA uses a unified search space $[0, 1]$, which enables each chromosome in the population to be evaluated by any of the $M$ tasks. Let $d_m$ be the search space dimensionality of Task $T_m$. The dimensionality of the unified search space is then $d = \max_m d_m$. If we have some a priori knowledge about the correspondences among the variables in different search spaces, we may organize their order in such a way that such relationship is kept. Otherwise, Task $T_m$ just uses the first $d_m$ variables in the evaluation.

Each gene in an MFEA chromosome is coded as a random key in the unified search space $[0, 1]$, which is called a unified representation [12]. The MFEA also keeps a record of the range of each gene in each task. Assume the unified representation is $g$, and its range is $[l, u]$. Then, during fitness evaluation, $g$ is mapped to its real search space value $l + g(u - l)$, which is straightforward.

The pseudocode of the MFEA is given in Algorithm 1. It first randomly generates $N$ chromosomes in the unified search space to form the initial population $P$, assigns each chromosome a skill factor, ensuring all $M$ tasks are equally represented, and computes the scalar fitness for each chromosome. The remainder of the MFEA is similar to any standard evolutionary algorithm, except that for each offspring we also need to determine its skill factor.

In the MFEA, chromosomes with the same skill factor can be viewed as a subpopulation, and chromosome crossovers can exchange genetic information among different subpopulations. Similar to the practice of hybridization among different populations in the biological world, the MFEA can potentially borrow good traits of individuals in other subpopulations to improve the fitness of a current subpopulation, and hence may result in fitter chromosomes. This is particular likely when the tasks have the same optimal solutions, or similar fitness landscapes in the unified search space.

III. MULTI-TASKING EVOLUTIONARY ALGORITHM (MTEA)

This section introduces our proposed MTEA. For the ease of illustration, we only consider two tasks, i.e., $M = 2$. The extension to more than two tasks is straightforward.

The pseudocode of the MTEA is given in Algorithm 2. Its Matlab code will be available for download after this paper is published.

A. Motivation of the Proposed MTEA

The MFEA may offer no advantage over optimizing each single task separately, if in the unified search space the optimal solutions of different tasks, or their fitness landscapes, are significantly different. Unfortunately, in practice often we do not know in priori how similar the tasks are, and it is desirable to have a multi-tasking optimization algorithm that can achieve good performance even in the worst-case scenario that the tasks are significantly different.

The MTEA is proposed to cope with the above problem. Particularly, it addresses two important questions: 1) how to estimate the difference between the optimal solutions of the two tasks, and 2) how to effectively transfer the fittest chromosomes between the two populations (tasks). Its main idea is to estimate the bias between the two tasks and then remove it in chromosome transfer, so that the optimal solutions...
of the two tasks are close to each other. In this way, a promising chromosome from one task can also be transformed into a promising solution for the other task, expediting the convergence.

The MTEA’s approach to estimate the bias can be explained using the example in Fig. 1, where the two tasks are one-dimensional Ackley and Sphere functions. Ackley maintains a population \( P_1 \), and Sphere a population \( P_2 \), each of which has 10 chromosomes. Clearly, there is a large bias between their optimal solutions, and hence blindly transferring a promising solution from \( P_1 \) to \( P_2 \) (or the opposite) may not benefit the search. The MTEA first computes the mean of a few fittest chromosomes (four were used in our example) in each population, and then estimates the bias between their optimal solutions as the difference between these means. When transferring a promising chromosome from one population to the other, it adds (or subtracts, depending on the direction) this bias to make it more compatible with the new task. In the first a few generations, bias estimation may not be very

---

Algorithm 1: Pseudocode of the MFEA [6], [12].

| Input: | \( M \) tasks \( \{T_m\}_{m=1}^M \); |
| |
| \( N \), the population size; |
| \( K \), the maximum number of generations; |
| \( rmp \), the random mating probability. |
| Output: | \( x_m^* \), the best chromosome for each \( T_m \), \( m = 1, \ldots, M \). |
| |
| Randomly generate \( N \) chromosomes \( \{x_n\}_{n=1}^N \) to initialize the population \( P \); |
| for \( n = 1, \ldots, N \) do |
| Assign skill factor \( \tau_n = \text{mod}(n, M) + 1 \) to \( x_n \); |
| Evaluate \( x_n \) for Task \( T_{\tau n} \) only; |
| Compute the fitness for \( x_n \); |
| end |
| Set \( k = 1 \); |
| while \( k \leq K \) do |
| Initialize the offspring population \( O = \emptyset \); |
| for \( n = 1, \ldots, N/2 \) do |
| Randomly pick two chromosomes \( x_a \) and \( x_b \) from \( P \) as the parents; |
| Generate \( r \) as a random number in \([0, 1]\); |
| if \( \tau_a = \tau_b \) or \( r < rmp \) then |
| Crossover \( x_a \) and \( x_b \) to generate two offsprings \( x_e \) and \( x_f \); |
| Mutate \( x_e \); |
| \( x_e \) randomly inherits its skill factor from \( x_a \) or \( x_b \); |
| Mutate \( x_f \); |
| \( x_f \) randomly inherits its skill factor from \( x_a \) or \( x_b \); |
| else |
| Mutate \( x_a \) to generate an offspring \( x_e \); |
| \( x_e \) inherits its skill factor from \( x_a \); |
| Mutate \( x_b \) to generate an offspring \( x_f \); |
| \( x_f \) inherits its skill factor from \( x_b \); |
| end |
| Evaluate \( x_e \) and \( x_f \) according to their skill factors; |
| Compute the fitness for \( x_e \) and \( x_f \); |
| Add \( x_e \) and \( x_f \) to the offspring population \( O \); |
| end |
| Form the new population \( P \) using the \( N \) fittest chromosomes in \( P \cup O \); |
| \( k = k + 1 \); |
| end |
| Return The fittest chromosome for each \( T_m \), \( m = 1, \ldots, M \). |

Algorithm 2: Pseudocode of the proposed MTEA.

| Input: | Two tasks \( T_1 \) and \( T_2 \); |
| |
| \( N \), the population size; |
| \( K \), the maximum number of generations; |
| \( rmp \), the random mating probability; |
| \( n_t \), the number of transferred chromosomes. |
| Output: | \( x_m^* \), the best chromosome for each \( T_m \), \( m = 1, 2 \). |
| |
| Randomly generate \( N \) chromosomes \( \{x_{m,n}\}_{n=1}^N \) to initialize the population \( P_m \) for Task \( T_m \); |
| Compute the fitness for each \( x_{m,n} \) in \( P_m \); |
| Sort all chromosomes in \( P_m \) in descending order according to their fitness; |
| Set \( k = 1 \); |
| while \( k \leq K \) do |
| Compute \( m_1 \) and \( m_2 \) in (2); |
| for \( m = 1, 2 \) do |
| Initialize a temporary population \( P_t = \emptyset \); |
| Construct the first \( n_t \) chromosomes of \( P_t \) from the \( n_t \) best chromosomes in \( P_3-m \), using (4); |
| Construct the remaining \( N - n_t \) chromosomes of \( P_t \) as the \( N - n_t \) best chromosomes in \( P_m \); |
| Construct an index vector \( S \) as a random permutation of \([1, \ldots, N]\); |
| Initialize the offspring population \( O = \emptyset \); |
| for \( n = 1, \ldots, N/2 \) do |
| Pick two parents \( x_{m,S(n)} \) and \( x_{m,S(N/2+n)} \); |
| Crossover to generate two offsprings \( x_e \) and \( x_f \) according to (6) and (7); |
| Mutate each of \( x_e \) and \( x_f \); |
| Add \( x_e \) and \( x_f \) to \( O \); |
| end |
| Evaluate the fitness of each chromosome in \( O \); |
| Set \( P = P_m \cup O \); |
| Sort the chromosomes in \( P \) in descending order according to their fitness; |
| Form the new population \( P_{m} \), using the \( N \) fittest chromosomes in \( P \); |
| \( k = k + 1 \); |
| end |
| Return The fittest chromosome for each \( T_m \), \( m = 1, 2 \). |
To effectively transfer the fittest chromosomes between the two populations, the MTEA uses sequential transfer in each generation, i.e., $P_1$ first transfers its fittest chromosomes to $P_2$ (after considering the bias), then $P_2$ performs crossover, mutation, fitness evaluation and reproduction to generate a new $P_2$. The fittest chromosomes in this new $P_2$ are then transferred to $P_1$ (after considering the bias), which next goes through crossover, mutation, fitness evaluation and reproduction to generate a new $P_1$. In this way, the updated fittest chromosomes in one population are immediately used by the other in the same generation, expediting the converge. On the contrary, the MFEA uses simultaneous chromosome transfer, i.e., the two tasks transfer their fittest chromosomes to the other simultaneously (without considering the bias), and then perform crossover, mutation, fitness evaluation and reproduction separately. As a result, the updated fittest chromosomes for one task cannot be shared by the other until the next generation, which is a waste of information. The difference is illustrated in Fig. 2.

The details of the MTEA are presented next.

**B. Population Initialization**

Unlike the MFEA, which keeps a single population and uses a skill factor to identity which chromosome belongs to which task, the MTEA keeps a separate population for each individual task.

Let $N$ be the population size of each task, and $P_m$ the population for $T_m$ ($m = 1, 2$). $P_m$ is randomly initialized in the first generation of the MTEA.

**C. Chromosome Transfer**

Chromosome transfer is considered in each subsequent iteration. We first compute the means of the best $n_t$ chromosomes in $P_1$ and $P_2$ respectively, and denote them as $\mathbf{m}_1$ and $\mathbf{m}_2$. The difference between $\mathbf{m}_1$ and $\mathbf{m}_2$ represents the bias between the two tasks, which can be used to make the transferred chromosomes more consistent with the new population.

Let’s focus on $T_1$ as an example. We transfer $n_t$ best chromosomes from $P_2$ to $P_1$, to replace the $n_t$ worst chromosomes in $P_1$. Assume the chromosomes in $P_1$ and $P_2$ have been sorted from the best to the worst according to their fitness, respectively. Denote the sorted chromosomes as \{$x_{1,m}$\}$_{m=1}^M$ and \{$x_{2,m}$\}$_{m=1}^M$, respectively. Then,

$$
\mathbf{m}_1 = \frac{1}{n_t} \sum_{m=1}^{n_t} x_{1,m}, \quad \mathbf{m}_2 = \frac{1}{n_t} \sum_{m=1}^{n_t} x_{2,m} \tag{2}
$$

We then construct a temporary population $P_t$ as:

$$
P_t = \{x'_{1,1},...,x'_{1,n_t},x_{1,1},...,x_{1,N-n_t}\} \tag{3}
$$

where $x'_{1,m}$ ($m = 1, ..., n_t$) is a transferred chromosome from $P_2$, computed as follows.

Let $d_m = |\mathbf{X}_{m}|$ be the dimensionality of the search space of Task $T_m$, or equivalently, the number of genes in a chromosome in $P_m$, $m = 1, 2$. When $d_1 \geq d_2$, we construct an index set $I$ by randomly sampling without replacement $d_2$ locations from the $d_1$ locations. When $d_1 < d_2$, we construct an index set $I$ by randomly sampling with replacement $d_2$ locations from the $d_1$ locations. Then,

$$
x'_{1,m}(i) = x_{2,m}(I(i)) - \mathbf{m}_2(I(i)) + \mathbf{m}_1(i), \quad m = 1, ..., n_t \tag{4}
$$

where $x'_{1,m}(i)$ is the $i$th gene in $x'_{1,m}$, $x_{2,m}(I(i))$ is the $I(i)$th gene in $x_{2,m}(I(i))$, $\mathbf{m}_2(I(i))$ is the $I(i)$th gene in $\mathbf{m}_2(I(i))$, and $\mathbf{m}_1(i)$ is the $i$th gene in $\mathbf{m}_1$.

Note that $I$ is randomly constructed for each transferred chromosome, i.e., the $i$th gene in $x'_{1,m}$ is randomly matched to a gene (not necessarily the $i$th gene) in $x_{2,m}$, and the matching is also different for different $m$. We use a random matching instead of a fixed matching because usually in practice we do not know which gene in $T_2$ can best benefit a gene in $T_1$, and randomizing the matching for different genes and different
m offers more diversity and higher likelihood to find a good matching than a blind fixed matching.

Once a matching between $x_{1:m}(i)$ and $x_{2:m}(I(i))$ is established, $m_1(i) - m_2(I(i))$ represents the bias between the two genes in the two tasks, and subtracting this bias from $x_{2:m}(I(i))$ makes the genes in the two tasks more consistent, and hence facilitates the knowledge transfer.

D. Crossover and Mutation

Next, we perform crossover, and make sure the $n_t$ transferred chromosomes from $P_2$ are all used in the crossover.

We define an index vector $S$ as a random permutation of $[1, ..., N]$. Each time, we pick two parents $x_{S(i)}$ and $x_{S(i+N/2)}$ ($i = 1, ..., N/2$), and use the simulated binary crossover (SBX) and polynomial mutation operators, the same as those in [8].

Let the two parents be $x_{S(i)} = [x^d_1, ..., x^d_a]$ and $x_{S(i+N/2)} = [x^d_1, ..., x^d_b]$, where $d$ is the dimensionality of the search space. Then, in SBX, we first compute:

$$c^i = \begin{cases} (2r)^{1/(\beta + 1)}, & r \leq 0.5 \\ [2(1-r)]^{-1/(\beta+1)}, & r > 0.5 \end{cases}, \quad i = 1, ..., d$$  \hspace{1cm} (5)

where $\beta$ is a user-specified parameter, and $r$ is a random number in $[0, 1]$, which is regenerated for each $i$. The two offsprings, $x_e = [x^e_1, ..., x^e_a]$ and $x_f = [x^f_1, ..., x^f_b]$, obtained from the SBX are:

$$x^e_i = \left[ (1 + c^i) x^d_i + (1 - c^i) x^d_i \right]/2, \quad i = 1, ..., d$$ \hspace{1cm} (6)

$$x^f_i = \left[ (1 + c^i) x^d_i + (1 - c^i) x^d_i \right]/2, \quad i = 1, ..., d$$ \hspace{1cm} (7)

Clearly, $x_e + x_f = x_{S(i)} + x_{S(i+N/2)}$. Additionally, it’s easy to observe that $x_e$ is closer to $x_{S(i)}$ than to $x_{S(i+N/2)}$, and $x_f$ is closer to $x_{S(i+N/2)}$ than to $x_{S(i)}$, because $c^i > 0$.

Let $\eta$ be a user-specified parameter, and $r$ be a random number in $[0, 1]$. Then, the polynomial mutation of a gene $g$ with range $[l, u]$ is computed as [7]:

$$g' = \begin{cases} g + [(2r)^{1/(1+\eta)} - 1](g - l), & r \leq 0.5 \\ g + [1 - (2(1-r))^{1/(1+\eta)}](u - g), & r > 0.5 \end{cases}$$ \hspace{1cm} (8)

After mutation, $x_e$ and $x_f$ are added to the offspring population $O$. The above crossover and mutation operations are repeated $N/2$ times so that $O$ has $N$ chromosomes.

E. Reproduction

We then evaluate the fitness of each chromosome in $O$, combine the chromosomes in $O$ with those in $P_m$, and sort the $2N$ chromosomes from the best to the worst according to their fitness. We use an elitist selection mechanism to prorogate the first $N$ best chromosomes to the next generation for $T_m$.

IV. EXPERIMENTS

This section compares the MTEA with eight state-of-the-art single- and multi-task evolutionary algorithms on nine benchmarks introduced in [6], which are also summarized in Table I.

| No. | Category | $d_1$ | $[l, u]$ | $d_2$ | $[l, u]$ | Intersection Similarity |
|-----|----------|-------|----------|-------|----------|-----------------------|
| 1   | CH+HS   | 50    | [−100, 100] | 50    | [−50, 50] | CI 1.0000  |
| 2   | CH+MS   | 50    | [−50, 50]   | 50    | [−50, 50] | CI 0.2261  |
| 3   | CH+LS   | 50    | [−50, 50]   | 50    | [−500, 500] | CI 0.0002  |
| 4   | Pb+HS   | 50    | [−50, 50]   | 50    | [−100, 100] | PI 0.8670  |
| 5   | Pb+MS   | 50    | [−50, 50]   | 50    | [−50, 50] | PI 0.2154  |
| 6   | Pb+LS   | 50    | [−50, 50]   | 25    | [−0.5, 0.5] | PI 0.0725  |
| 7   | Ni+HS   | 50    | [−50, 50]   | 50    | [−50, 50] | NI 0.9434  |
| 8   | Ni+MS   | 50    | [−100, 100] | 50    | [−50, 50] | NI 0.3669  |
| 9   | Ni+LS   | 50    | [−50, 50]   | 50    | [−500, 500] | NI 0.0016  |

1. CI (complete intersection): The unified representations of the global optima of the two tasks are identical w.r.t. all variables.
2. PI (partial intersection): The unified representations of the global optima of the two tasks are different w.r.t. all variables.
3. HS: High similarity; MS: Moderate similarity; LS: Low similarity.

The inter-task similarity is computed as the Spearman’s rank correlation between the solutions of the two tasks [6].

A. Performance Measures

In addition to the fitness in each task, the simple performance metric proposed in [6] is also used to quantify the performance of different algorithms.

Assume there are $K$ algorithms, $A_1, ..., A_K$ for a problem with $M$ minimization tasks $T_1, ..., T_M$, and each algorithm has been run for $L$ repetitions. Let $I(k, m)_l$ denote the best obtained result on the $l$th repetition by Algorithm $A_k$ on Task $T_m$, and $\mu_m$ and $\sigma_m$ be the mean and standard deviation of $I(k, m)_l$, $k = 1, ..., K$, $l = 1, ..., L$. Then, the normalized performance $I'(k, m)_l$ is computed as:

$$I'(k, m)_l = \frac{I(k, m)_l - \mu_m}{\sigma_m}$$ \hspace{1cm} (9)

and the performance score of Algorithm $A_m$ is:

$$s_m = \sum_{k=1}^{K} \sum_{l=1}^{L} I'(k, m)_l$$ \hspace{1cm} (10)

A smaller performance score indicates a better overall performance.

B. Algorithms

We compare the performance of our proposed MTEA with a classic single-task evolutionary algorithm, and seven state-of-the-art multi-task algorithms:

1) The single-objective evolutionary algorithm (SOEA), which considers each task independently. We used the SOEA code (in Matlab) provided in the WCCI2018 competition on evolutionary multi-task optimization. Essentially, each SOEA implements a genetic algorithm [11], [26], with SBX crossover and polynomial mutation.

2) The MFEA, which was introduced in Section [1]. We also used the MFEA code (in Matlab) provided in the WCCI2018 competition on evolutionary multi-task optimization.

3) The evolution of biocenosism through symbiosis (EBS) algorithm [15], which can deal with many-tasking problems. The basic evolutionary algorithm used in the EBS
was identical to the genetic algorithm in the SOEA and the MFEA.

4) The MFEA-LBS \cite{27}, which employs a permutation based unified representation and level-based selection (LBS) to enhance the original MFEA.

5) The multi-factorial evolutionary algorithm with resource reallocation (MFEARR) \cite{21}, which adds a resource allocation mechanism to facilitate the discovery and utilization of synergy among tasks.

6) The linearized domain adaptation multi-factorial evolutionary algorithm (LDA-MFEA) \cite{1}, which uses LDA to transform the search space of a simple task to a new one similar to its constitutive complex task for efficient problem solving.

7) The generalized multi-factorial evolutionary algorithm (G-MFEA) \cite{9}, which uses decision variable translation and shuffling strategies to facilitate knowledge transfer between optimization problems.

8) The evolutionary multi-tasking via explicit autoencoding (EMEA) algorithm \cite{10}, which uses autoencoding to explicitly transfer knowledge across tasks in evolutionary multi-tasking.

We used a population size of 200 in the MFEA and its variants. For the SOEA, the G-MTEA, the EMEA and the MTEA, each task had a population size of 100. The maximum number of function evaluations was 100,000 for all algorithms, i.e., all algorithms terminated after 500 iterations. \( rmp = 0.3, \beta = 2 \) in \( 5 \) of the SBX, and \( \eta = 5 \) in \( 3 \) of polynomial mutation, were used in all algorithms. Additionally, \( n_t = 10 \) was used in the EMEA (as in \( 10 \)), and \( n_t = 40 \) in the MTEA.

To cope with the randomness, each algorithm was run 20 times, each time with a randomly initialized population. Then several statistics, such as the mean and standard deviation of the objectives in the two tasks, were computed.

C. Experimental Results

Individual experimental results on the nine benchmarks are shown in Fig. 3 and the average performance scores of the nine algorithms across the nine benchmarks are shown in Fig. 4. Observe that:

1) On average all multi-task algorithms outperformed the SOEA, which suggests that the transfer of information between the tasks can indeed improve the overall optimization performance.

2) On average the EBS and the MFEARR performed worse than the MFEA, and the MFEA-LBS had comparable performance as the MFEA.

3) The LDA-MFEA achieved the best performance when the number of function evaluations was small, but gradually degraded when the number of function evaluations increased.

4) The G-MFEA slightly outperformed the MFEA when the number of function evaluations was large.

5) Among the eight existing algorithms, on average EMEA achieved the second best performance (worse only than the LDA-MFEA) when the number of function evaluations was small, and the best performance when the number of function evaluations was large.

6) Among all nine algorithms, on average our proposed MTEA achieved the second best performance (worse only than the LDA-MFEA) when the number of function evaluations was small, and the best performance when the number of function evaluations was large.

The average performances (mean and standard deviation) of different algorithms on different tasks, after 100,000 function evaluations, are given in Table II. Observe that the MTEA achieved the best performance score in eight out of the nine benchmarks. Overall it dominated the other eight algorithms. The normalized mean and standard deviation of different algorithms after 100,000 function evaluations, w.r.t. the SOEA, are shown in Fig. 5. Clearly, the MTEA achieved on average the smallest mean and standard deviation, suggesting that the MTEA consistently outperformed other approaches.

It is also interesting to compare the computational cost of different algorithms. For each benchmark, we normalized the computation time of the eight multi-task algorithms w.r.t. the SOEA, and plot the results in Fig. 6. On average the SOEA, the EMEA and the MTEA had comparable computational cost, and they were the fastest among the nine. The MFEA-LBS and the EBS also had comparable computational cost, all of which were much smaller than the MFEA, the G-MFEA and the MFEARR. The computational cost of the LDA-MFEA was almost three times higher than the MTEA.

In summary, we can conclude that the proposed MTEA is effective, efficient, and stable.

D. Parameter Sensitivity of the MTEA

In addition to the standard parameters considered in an evolutionary algorithm, e.g., the population size, the maximum number of generations, and the random mating probability, the MTEA also has a parameter \( n_t \), the number of transferred chromosomes between the two tasks. Next, we study the sensitivity of the MTEA with respect to \( n_t \).

To this end, we compared the performances of the SOEA, six versions of the MTEA with \( n_t = \{10, 20, 30, 40, 50, 60\} \), and the MTEA-NB (no bias), which uses \( n_t = 40 \) but does not consider the bias, i.e., the MTEA-NB transfers 40 fittest chromosomes between the two populations directly. The average performances of the eight algorithms on the nine benchmarks are shown in Fig. 7. As expected, all MTEA with different \( n_t \) outperformed the SOEA and the MTEA-NB, suggesting that it is critical to consider the bias. The MTEA with different \( n_t \) had similar performance, indicating its robustness to \( n_t \); however, a closer look reveals that the MTEA with a smaller \( n_t \) performed better at the early stage of the evolution, but worse at the later stage. This is because at the beginning of the evolution the fittest chromosomes are usually far away from the global optimum, so the estimated bias may have a large error, and hence the transferred chromosomes based on this bias do not perform well. As a result, transferring too many chromosomes deteriorates the performance at this stage. However, as the evolution goes on, the estimated bias becomes more accurate, and the transferred chromosomes also perform better, so more chromosomes should be transferred.

These results also suggest that maybe an adaptive \( n_t \) can be used to further improve the MTEA, by starting with a small \( n_t \)
Fig. 3. Experimental results on the nine *original* benchmarks. (a) Benchmark 1; (b) Benchmark 2; (c) Benchmark 3; (d) Benchmark 4; (e) Benchmark 5; (f) Benchmark 6; (g) Benchmark 7; (h) Benchmark 8; (i) Benchmark 9.
TABLE II
AVERAGE PERFORMANCES (MEAN AND BRACKETED STANDARD DEVIATION) OF DIFFERENT ALGORITHMS ON THE NINE ORIGINAL BENCHMARKS, AFTER 100,000 FUNCTION EVALUATIONS. BEST PERFORMANCES ARE MARKED IN BOLD.

| Benchmark  | 1       | 2       | 3       | 4       | 5       | 6       | 7       | 8       | 9       |
|------------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| SOEA       | T1 mean | 0.8996  | 3.0690  | 21.2084 | 417.8177 | 5.2415  | 5.9080  | 2934.5108 | 0.9121  | 42.8271 |
|            | T2 mean | 433.9618| 446.0967| 4350.4544| 81.9612  | 30652.1236| 12.5031 | 431.0790  | 37.4071 | 4261.6665|
|            | Score   | 0.8029  | 1.6159  | 0.0000  | 2.1983  | -0.2145 | 2.0068  | 1.3676   | -0.0417 |
| MFEA       | T1 mean | 0.9312  | 4.8780  | 20.2536 | 587.5578 | 4.8739  | 17.8681 | 10833.5260| 0.9763  | 568.3542|
|            | T2 mean | 328.0565| 341.3237| 4833.4306| 86.2004  | 16927.1056| 15.5357 | 419.0696  | 36.7019 | 4286.2454|
|            | Score   | 0.8979  | 4.8817  | 20.2536 | 439.1479 | 5.2223  | 8.2290  | 1331.3190 | 0.9241  | 448.3269|
| MFEARR     | T1 mean | 0.6213  | 6.6270  | 21.1512 | 634.4187 | 3.5334  | 4.4146  | 4250.2864 | 1.0077  | 1742.9144|
|            | T2 mean | 304.3237| 319.2107| 4044.1627| 111.6610 | 9756.4585| 19.3068 | 393.8628  | 25.1179 | 4424.1310|
|            | Score   | 0.9149  | 6.5781  | 20.5802 | -9.3290 | -1.3942 | -1.9062 | -9.3290   | -1.3942 | -1.9062 |
| LDA-MFEA   | T1 mean | 0.5933  | 4.8460  | 20.2975 | 677.2061 | 5.0222  | 20.1171 | 11263.0957| 0.8969  | 545.5597|
|            | T2 mean | 304.3237| 319.2107| 4044.1627| 111.6610 | 9756.4585| 19.3068 | 393.8628  | 25.1179 | 4424.1310|
|            | Score   | 0.6213  | 6.6270  | 21.1512 | 634.4187 | 3.5334  | 4.4146  | 4250.2864 | 1.0077  | 1742.9144|
| MFEA-LBS   | T1 mean | 0.8979  | 4.8817  | 20.2536 | 439.1479 | 5.2223  | 8.2290  | 1331.3190 | 0.9241  | 448.3269|
|            | T2 mean | 304.3237| 319.2107| 4044.1627| 111.6610 | 9756.4585| 19.3068 | 393.8628  | 25.1179 | 4424.1310|
|            | Score   | 0.9149  | 6.5781  | 20.5802 | -9.3290 | -1.3942 | -1.9062 | -9.3290   | -1.3942 | -1.9062 |
| EBS        | T1 mean | 0.8979  | 4.8817  | 20.2536 | 439.1479 | 5.2223  | 8.2290  | 1331.3190 | 0.9241  | 448.3269|
|            | T2 mean | 304.3237| 319.2107| 4044.1627| 111.6610 | 9756.4585| 19.3068 | 393.8628  | 25.1179 | 4424.1310|
|            | Score   | 0.9149  | 6.5781  | 20.5802 | -9.3290 | -1.3942 | -1.9062 | -9.3290   | -1.3942 | -1.9062 |
| G-MFEA     | T1 mean | 0.9016  | 4.8973  | 20.2362 | 538.3213 | 4.9707  | 13.7719 | 12065.2897| 0.9645  | 579.1136|
|            | T2 mean | 298.4173| 325.5007| 3790.2887| 10.2819 | 9254.1989| 10.8155 | 395.5002  | 24.3334 | 3972.5676|
|            | Score   | 0.9016  | 4.8973  | 20.2362 | 538.3213 | 4.9707  | 13.7719 | 12065.2897| 0.9645  | 579.1136|
| EMAE       | T1 mean | 0.8029  | 1.6159  | 0.3841  | 1.8285  | -0.0017 | 1.6851  | 0.9490    | -0.0017 |
|            | T2 mean | 304.3237| 319.2107| 4044.1627| 111.6610 | 9756.4585| 19.3068 | 393.8628  | 25.1179 | 4424.1310|
|            | Score   | 0.9149  | 6.5781  | 20.5802 | -9.3290 | -1.3942 | -1.9062 | -9.3290   | -1.3942 | -1.9062 |
| MTEA       | T1 mean | 0.9312  | 4.8780  | 20.2536 | 587.5578 | 4.8739  | 17.8681 | 10833.5260| 0.9763  | 568.3542|
|            | T2 mean | 304.3237| 319.2107| 4044.1627| 111.6610 | 9756.4585| 19.3068 | 393.8628  | 25.1179 | 4424.1310|
|            | Score   | 0.9149  | 6.5781  | 20.5802 | -9.3290 | -1.3942 | -1.9062 | -9.3290   | -1.3942 | -1.9062 |

E. Practical Considerations

Many of the nine benchmarks considered above are special, in that the search spaces for both $x_1$ and $x_2$ are symmetric, and the unified representations of $x_1^*$ and $x_2^*$ are completely or partially overlap, and hence it is expected that sharing information between them would speed up the optimization. However, in practice the search spaces for $x_1$ and $x_2$ are not necessarily symmetric, and we may not always have a priori knowledge on whether the unified representations of $x_1^*$ and $x_2^*$ overlap. This subsection investigates this more practical and also more challenging problem, and tries to answer two questions:

1. **Q1**: Can multi-tasking still facilitate the optimization when the unified representations of $x_1^*$ and $x_2^*$ do not overlap at all?
2. **Q2**: Is multi-tasking search easier when the search spaces become smaller?

![Fig. 4. Average performance scores across the nine original benchmarks.](image_url)
For these purposes, we slightly modified at least one search space in each benchmark, making it non-symmetric, but still ensuring the original global optimum lies within it, as shown in Table III.

This simple modification enables us to answer both questions above:

1) It makes the unified representations of $x_1^*$ and $x_2^*$ completely different for all nine benchmarks, so we can study $Q_1$. For example, for the original Benchmark 1 in Table I the unified representations of the global optima are $[0, ..., 0] \in \mathbb{R}^{50}$, and hence better multi-tasking learning performance can be expected. For the modified Benchmark 1 in Table III the global optimum of Task 1 is still $[0, ..., 0] \in \mathbb{R}^{50}$, but its unified representation becomes $[1/3, ..., 1/3] \in \mathbb{R}^{50}$, and the unified representation of the global optimum of Task 2 remains as $[0.5, ..., 0.5] \in \mathbb{R}^{50}$. The two unified representations are different, indicating two less-relevant tasks.

2) The modified search spaces are always smaller than the original search spaces, which enables us to study $Q_2$. Intuitively, we’d expect that smaller search spaces make it easier to find the global optima.

It should be noted that the nine original benchmarks also
include a few cases in which the two tasks share low similarity. So, by focusing on these benchmarks, we can also investigate Q1; however, they do not allow the study of Q2.

Due to the page limit, we only present the average performance scores of the nine algorithms across the nine modified benchmarks in Fig. 8. Observe that:

1) On average the EMEA and the MTEA still outperformed the SOEA, but the performance improvements were smaller for the EMEA.
2) When the two tasks are completely different, the MFEA, the G-MFEA and the MFEA-LBS had comparable performances, and all three of them performed worse than the SOEA.
3) The MFEARR and the EBS, which performed worse than the MFEA on the original benchmarks, both outperformed the MFEA on the modified benchmarks. This is mainly because the performance of the MFEA degraded, instead of the performances of the MFEARR and the EBS increased. The SOEA, the MFEARR and the EBS had comparable performances.
4) The LDA-MFEA still achieved the best performance when the number of function evaluations was very small, but degraded when the number of function evaluations increased.
5) Among the eight existing algorithms, on average the EMEA achieved the second best performance (worse only than the LDA-MFEA) when the number of function evaluations was small, and the best performance when the number of function evaluations was large, as on the original benchmarks.
6) Among the nine algorithms, our proposed MTEA again achieved the best overall performance, and outperformed the other eight algorithms by a large margin, suggesting that it can effectively perform multi-tasking, no matter whether the tasks are similar or not.

Fig. 8. Average performance scores across the nine modified benchmarks.

Table IV shows the final optimization results after 100,000 function evaluations for all nine modified benchmarks. The MTEA still dominated the other eight algorithms: it achieved the best performance score on all nine modified benchmarks. Because the original Benchmark $i$ and the modified Benchmark $i$ ($i = 1, ..., 9$) have the same global optimum, the performance scores on them can be directly compared.

Fig. 9 shows that:

1) The SOEA on the modified benchmarks always outperformed the SOEA on the original benchmarks, which is reasonable, as the modified benchmarks had smaller search spaces, and hence the SOEA can search them more extensively.
2) The MTEA on the modified benchmarks always outperformed the MTEA on the original benchmarks, again because the modified benchmarks had smaller search spaces, and hence the MTEA can search them more extensively.
3) Although on average all multi-task algorithms outperformed the SOEA on the original benchmarks, most of them (except the EBS, the EMEA and the MTEA) performed worse than the SOEA on the modified benchmarks, suggesting that they were not able to cope well with more realistic, and also more challenging, optimization problems.
4) The MTEA outperformed the other eight algorithms by a large margin, both on the original benchmarks and the modified benchmarks, suggesting that the MTEA is very robust, and much safer to use in real-world applications, in which we often do not know how similar the tasks are.

In summary, although the search spaces were smaller on the modified benchmarks, the increased differences between the tasks made the modified benchmarks more challenging for multi-tasking. Nevertheless, a carefully designed multi-task algorithm like the MTEA can still benefit from learning multiple tasks together. It is very robust, regardless of whether the tasks are similar or significantly different. So, the MTEA should be preferred in real-world applications.

V. CONCLUSION AND FUTURE RESEARCH

Evolutionary multi-tasking, or multi-factorial optimization, is an emerging subfield of multi-task optimization, which integrates evolutionary computation and multi-task learning to optimize multiple optimization tasks simultaneously. This paper has proposed a novel easy-to-implement multi-tasking evolutionary algorithm, which copes well with significantly different tasks by estimating and using the bias among them. Comparative studies with eight existing approaches in the literature on nine benchmarks demonstrated that on average our algorithm outperformed all of them, and has lower computational cost than six of them. Particularly, unlike other multi-task algorithms, the performance of the MTEA is consistently good whether the tasks are similar or significantly different, making it ideal for real-world applications.

Our future research will:

1) Further improve our algorithm by borrowing promising ideas from other approaches, e.g., from the LDA-MFEA when the number of function evaluation is small.
2) Investigate whether our algorithm can also be extended to other evolutionary computation algorithms such as genetic algorithms and particle swarm optimization.

3) Extend our idea to multi-objective multi-factorial optimization [13].

REFERENCES

[1] K. B. Kil, A. Gupta, L. Feng, Y. S. Ong, and T. P. Siew, “Linearized domain adaptation in evolutionary multitasking,” in Proc. IEEE Int’l Conf. on Evolutionary Computation, San Sebastian, Spain, Jun. 2017, pp. 1295-1302.

[2] J. C. Bean, “Genetic algorithms and random keys for sequencing and optimization,” ORSA Journal on computing, vol. 6, no. 2, pp. 154-160, 1994.

[3] R. Caruana, “Multitask learning,” Machine Learning, vol. 28, no. 1, pp. 41-75, 1997.

[4] Q. Chen, X. Ma, Z. Zha, and Y. Sun, “Evolutionary multi-tasking single-objective optimization based on cooperative co-evolutionary memetic algorithm,” in Proc. 13th Int’l Conf. on Computational Intelligence and Security, Hong Kong, Dec. 2017, pp. 197-201.

[5] M.-Y. Cheng, A. Gupta, Y.-S. Ong, and Z.-W. Ni, “Coevolutionary multitasking for concurrent global optimization: With case studies in complex engineering design,” Environmental Applications of Artificial Intelligence, vol. 64, pp. 13-24, 2017.

[6] B. Da, Y. Ong, L. Feng, A. K Qin, A. Gupta, Z. Zhu, C. Ting, K. Tang, and X. Yao, “Evolutionary multitasking for single-objective continuous optimization: Benchmark problems, performance metrics, and baseline results,” CoRR, vol. abs/1706.03470, 2017. [Online]. Available: http://arxiv.org/abs/1706.03470.

[7] K. Deb and B. Agrawal, “A niched-penalty approach for constraint handling in genetic algorithms,” in Artificial Neural Nets and Genetic Algorithms. Vienna: Springer, 1999, pp. 235-243.

[8] K. Deb and R. B. Agrawal, “Simulated binary crossover for continuous search space,” Complex Systems, vol. 9, no. 2, pp. 1-15, 1994.

[9] J. Ding, C. Yang, Y. Jin, and T. Chai, “Generalized multi-tasking for evolutionary optimization of expensive problems,” IEEE Trans. on Evolutionary Computation, 2018, in press.

[10] L. Feng, L. Zhou, J. Zhong, A. Gupta, Y. Ong, K. Tan, and A. K Qin, “Evolutionary multitasking via explicit autoencoding.” IEEE Trans. on Cybernetics, 2018, in press.

[11] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. Boston, MA: Addison-Wesley, 1989.

[12] A. Gupta, Y. S. Ong, and L. Feng, “Multifactorial evolution: Toward
Fig. 9. Performance scores (the smaller the better) of the nine algorithms on the nine original (color bars) and modified (black bars) benchmarks.

evolutionary multitasking.” IEEE Trans. on Evolutionary Computation, vol. 20, no. 3, pp. 343–357, 2016.

[13] A. Gupta, Y. S. Ong, L. Feng, and K. C. Tan, “Multi-objective multifactorial optimization in evolutionary multitasking;” IEEE Trans. on Cybernetics, vol. 47, no. 7, pp. 1652–1665, 2017.

[14] R. Hashimoto, H. Ishibuchi, N. Masuyama, and Y. Nojima, “Analysis of evolutionary multi-tasking as an island model,” in Proc. of the Genetic and Evolutionary Computation Conf. Companion, Kyoto, Japan, Jul. 2018, pp. 1894–1897.

[15] R.-T. Liaw and C.-K. Ting, “Evolutionary many-tasking based on bioconiosis through symbiosis: A framework and benchmark problems,” in Proc. IEEE Int’l Conf. on Evolutionary Computation, San Sebastian, Spain, Jun. 2017, pp. 2266–2273.

[16] Y. S. Ong and A. Gupta, “Evolutionary multitasking: A computer science view of cognitive multitasking,” Cognitive Computation, vol. 8, no. 2, pp. 125–142, 2016.

[17] S. J. Pan and Q. Yang, “A survey on transfer learning,” IEEE Trans. on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345–1359, 2010.

[18] R. Sagarna and Y.-S. Ong, “Concurrently searching branches in software tests generation through multitask evolution,” in Proc. IEEE Symposium Series on Computational Intelligence, Athens, Greece, Dec. 2016, pp. 1–8.

[19] K. Swersky, J. Snoek, and R. P. Adams, “Multi-task bayesian optimization,” in Proc. 26th Int’l Conf. on Neural Information Processing Systems (NIPS), Lake Tahoe, Nevada, Dec. 2013, pp. 2004–2012.

[20] Z. Tang, M. Gong, and M. Zhang, “Evolutionary multi-task learning for modular extremal learning machine,” in Proc. IEEE Int’l Conf. on Evolutionary Computation, San Sebastian, Spain, Jun. 2017, pp. 474–479.

[21] Y.-W. Wen and C.-K. Ting, “Parting ways and reallocating resources in evolutionary multitasking,” in Proc. IEEE Int’l Conf. on Evolutionary Computation, San Sebastian, Spain, Jun. 2017, pp. 2404–2411.

[22] D. Wu, “Online and offline domain adaptation for reducing BCI calibration effort,” IEEE Trans. on Human-Machine Systems, vol. 47, no. 4, pp. 550–563, 2017.

[23] D. Wu and J. Huang, “Affect estimation in 3D space using multi-task active learning for regression,” IEEE Trans. on Affective Computing, 2018, in press.

[24] D. Wu, V. J. Lawhern, S. Gordon, B. J. Lance, and C.-T. Lin, “Driver drowsiness estimation from EEG signals using online weighted adaptation regularization for regression (OwARR),” IEEE Trans. on Fuzzy Systems, vol. 25, no. 6, pp. 1522–1535, 2017.

[25] D. Wu, V. J. Lawhern, W. D. Hairston, and B. J. Lance, “Switching EEG headsets made easy: Reducing offline calibration effort using active weighted adaptation regularization,” IEEE Trans. on Neural Systems and Rehabilitation Engineering, vol. 24, no. 11, pp. 1125–1137, 2016.

[26] D. Wu and W. W. Tan, “Genetic learning and performance evaluation of type-2 fuzzy logic controllers,” Engineering Applications of Artificial Intelligence, vol. 19, no. 8, pp. 829–841, 2006.

[27] Y. Yuan, Y.-S. Ong, A. Gupta, P. S. Tan, and H. Xu, “Evolutionary multitasking in permutation-based combinatorial optimization problems: Realization with TSP QAP LOP and JSP,” in Proc. IEEE Region 10 Conf., Singapore, Nov. 2016, pp. 3157–3164.