This is the Pre-Published Version.

The following publication Z. Liang, H. Li, N. Yu, K. Sun and R. Cheng, "Bridging Evolutionary Multiobjective Optimization and GPU Acceleration via Tensorization," in IEEE Transactions on Evolutionary Computation, vol. 30, no. 1, pp. 420-434, Feb. 2026 is available at https://doi.org/10.1109/TEVC.2025.3555605.

Bridging Evolutionary Multiobjective Optimization and GPU Acceleration via Tensorization

Zhenyu Liang, Hao Li, Naiwei Yu, Kebin Sun, and Ran Cheng, *Senior Member, IEEE*

***Abstract*—Evolutionary multiobjective optimization (EMO) has made significant strides over the past two decades. However, as problem scales and complexities increase, traditional EMO algorithms face substantial performance limitations due to insufficient parallelism and scalability. While most work has focused on algorithm design to address these challenges, little attention has been given to hardware acceleration, thereby leaving a clear gap between EMO algorithms and advanced computing devices, such as GPUs. To bridge the gap, we propose to parallelize EMO algorithms on GPUs via the *tensorization* methodology. By employing tensorization, the data structures and operations of EMO algorithms are transformed into concise tensor representations, which seamlessly enables automatic utilization of GPU computing. We demonstrate the effectiveness of our approach by applying it to three representative EMO algorithms: NSGA-III, MOEA/D, and HypE. To comprehensively assess our methodology, we introduce a multiobjective robot control benchmark using a GPU-accelerated physics engine. Our experiments show that the tensorized EMO algorithms achieve speedups of up to 1113× compared to their CPU-based counterparts, while maintaining solution quality and effectively scaling population sizes to hundreds of thousands. Furthermore, the tensorized EMO algorithms efficiently tackle complex multiobjective robot control tasks, producing high-quality solutions with diverse behaviors. Source codes are available at https://github.com/EMI-Group/evomo.**

***Index Terms*—Evolutionary Multiobjective Optimization, GPU Acceleration, Tensorization, Robot Control.**

I. Introduction

In many real-world optimization problems (e.g., material design [1], [2], energy management [3], [4], network optimization [5], and portfolio optimization [6]), decision-makers must consider multiple (and often conflicting) objectives simultaneously. Without loss of generality, such multiobjective optimization problems (MOPs) can be defined as:

$$\underset{\mathbf{x}}{\text{minimize}} \quad \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})), \tag{1}$$

where $\mathbf{x} = (x_1, x_2, \dots, x_d) \in \mathcal{X} \subseteq \mathbb{R}^d$ is the decision vector and d is the number of decision variables. $\mathbf{f} : \mathcal{X} \rightarrow \mathbb{R}^m$ maps the decision vector to an m-dimensional objective space. Each $f_i : \mathcal{X} \rightarrow \mathbb{R}$ for $i = 1, \dots, m$ represents an objective function that needs to be minimized (or maximized). The key challenge in solving MOPs is to identify a set of trade-off solutions known as the Pareto set (PS), where no single solution can optimize all objectives simultaneously. This set contains all Pareto-optimal or non-dominated solutions, and their corresponding points in the objective space collectively form the Pareto front (PF).

Over the past two decades, the field of evolutionary multiobjective optimization (EMO) [7], [8] has seen rapid advancements, resulting in the development of various effective algorithms for solving MOPs. Broadly, these algorithms can be categorized into three main methods: dominance-based, decomposition-based, and indicator-based. Dominance-based algorithms, such as NSGA-II [9] and NSGA-III [10], select solutions based on dominance relations between individuals. Decomposition-based algorithms, like MOEA/D [11], break down an MOP into multiple simpler subproblems, which are optimized collaboratively. Indicator-based algorithms, such as HypE [12], focus on optimizing specific performance indicators, like hypervolume (HV) [13].

While EMO algorithms have proven effective in solving various MOPs, their performance is significantly constrained by limitations in *computing power*. First, since the majority of existing EMO algorithms still rely on CPUs for execution, their computational efficiency is inherently limited, particularly when addressing large-scale multiobjective optimization problems (LSMOPs) [14]. Second, the inconsistent implementation of EMO algorithms across different methods has led to fragmentation, making it difficult to standardize solutions and apply them across diverse domains. Without a unified framework, efforts to generalize these algorithms and enhance computational efficiency are impeded. Third, much of the current research remains focused on relatively simpler numerical optimization tasks, often neglecting computationally intensive real-world applications. This narrow focus further limits the practical use of EMO algorithms in scenarios where real-time performance and scalability are crucial.

To address these limitations, one promising method is to incorporate modern computational accelerators such as GPUs. With their powerful parallel processing capabilities, GPUs have demonstrated significant performance improvements in fields like deep learning [15]. However, to fully leverage the potential of GPUs for EMO algorithms, a systematic method of parallelization is necessary, yet little effort has been made in this direction so far.

Given the high concurrency enabled by the large number of *tensor cores* [16], GPUs are particularly well-suited for efficient handling and acceleration of large-scale data pro-

This work was supported in part by Guangdong Basic and Applied Basic Research Foundation (No. 2024B1515020019). (*Corresponding author: Ran Cheng*)

Zhenyu Liang, Hao Li, Naiwei Yu, and Kebin Sun are with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China. E-mails: {zhenyuliang97, li7526a, yunaiweiyn, sunkebin.cn}@gmail.com

Ran Cheng is with the Department of Data Science and Artificial Intelligence, and the Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China. He is also with The Hong Kong Polytechnic University Shenzhen Research Institute, Shenzhen, China. (E-mail: ranchengcn@gmail.com)

© 2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

cessing. Correspondingly, one promising method for parallelization on GPUs is *tensorization*, i.e., representing data structures and operations as tensors. Building upon this concept, we introduce a concise and general tensorization methodology for accelerating EMO algorithms on GPUs. By leveraging tensor operations and the inherent parallelism of GPUs, this methodology systematically explains how to transform EMO algorithms into concise tensor representations. Using this methodology, we implement tensorized versions of three representative EMO algorithms from each category: the dominance-based NSGA-III [10], the decomposition-based MOEA/D [11], and the indicator-based HypE [12]. Moreover, to evaluate the performance of tensorized EMO algorithms in GPU computing environments, we develop a multiobjective robot control benchmark using Brax [17], a GPU-accelerated physics engine. The main contributions of this research are as follows:

1) We introduce a general tensorization methodology for EMO algorithms that transforms key data structures (i.e., candidate solutions and objective values) and operations (i.e., crossover, mutation, and selection) into tensor representations. This approach establishes concise yet versatile mathematical models for enabling efficient GPU acceleration of EMO algorithms.
2) We apply the proposed tensorization methodology to three representative EMO algorithms: NSGA-III, MOEA/D, and HypE. The tensorized algorithms achieve up to 1113$\times$ speedup compared to their CPU-based counterparts, while maintaining solution quality and effectively scaling population sizes to hundreds of thousands.
3) We develop a multiobjective robot control benchmark called MoRobtrol. This benchmark represents a computationally intensive scenario with complex black-box properties. It demonstrates the ability of the tensorized EMO algorithms to efficiently generate high-quality solutions with diverse behaviors in such computationally expensive environments.

The structure of this paper is as follows: Section II reviews the background and related work. Section III introduces the tensorization methodology for GPU acceleration. Section IV details the implementations of core operations in three representative EMO algorithms. Section V introduces the multiobjective robot control benchmark. Section VI outlines the experimental setup and results. Section VII summarizes the findings and discusses future work.

II. Background

A. Taxonomy of EMO Algorithms

Traditional evolutionary multiobjective optimization (EMO) algorithms can generally be classified into three main categories based on their selection mechanisms: dominance-based, decomposition-based, and indicator-based [7].

Dominance-based EMO algorithms are pivotal in addressing complex optimization tasks through Pareto dominance. As the pioneering algorithm in this category, NSGA-II [9] introduced a fast non-dominated sorting approach, which has since become a foundation for many subsequent algorithms. SPEA2 [18] introduces a fine-grained fitness assignment strategy, density estimation, and enhanced archive truncation to improve performance. GrEA [19] enhances convergence and diversity balance by using grid dominance. NSGA-III [10] further advances diversity management in higher-dimensional spaces with reference points. Recent algorithms build on these foundations with innovative strategies. BiGE [20] focuses on proximity and diversity through bi-goal optimization, while VaEA [21] balances these using vector-angle-based principles. RSEA [22] improves performance by projecting solutions into a radial space, and NSGA-II/SDR [23] introduces a novel dominance relation with adaptive niching techniques. MSEA [24] divides the optimization process into stages to enhance diversity preservation. PMEA [25] eliminates dominance resistance solutions using an interquartile range method.

Decomposition-based EMO algorithms address MOPs by decomposing them into simpler subproblems [26]. These algorithms can be further categorized into two types: weighted aggregation based and reference set based methods. MOEA/D [11] is the most representative weighted aggregation based method, which aggregates objectives using weight vectors and has inspired variants like MOEA/D-DRA [27] and EAG-MOEA/D [28]. Recent developments include MOEA/D-AAWNs [29], which adapts weight vectors and neighborhoods to enhance diversity. MOEA/D-GLCM [30] employs bidirectional global search and adaptive neighborhood strategies to improve population distribution. Reference set based methods, on the other hand, divide the objective space using reference points or vectors, guiding the search towards underexplored regions while maintaining diversity. Representative algorithms of this type include MOEA/D-M2M [31], RVEA [32], and θ-DEA [33]. A recent work, ECRA-DEA [34], adaptively allocates resources across subspaces using fitness contribution and improvement rates.

Indicator-based EMO algorithms rely on performance indicators for selection [35]. As a notable example, IBEA [36] uses binary ϵ^{+} indicators for decision-making. Following this, SMS-EMOA [37] and HypE [12] are representative hypervolume-based algorithms, with HypE using Monte Carlo sampling to approximate HV. BCE-IBEA [38] integrates bicriterion evolution with the IBEA framework, while SRA [39] combines multiple indicators with stochastic ranking based environmental selection. MOMBI-II [40] uses the R2 metric, and AR-MOEA [41] is guided by the IGD-NS metric. MaOEA/IGD [42] employs the inverted generational distance metric. Recently, R2HCA-EMOA [43], HVCTR [44], and IMOEA-ARP [45] extended the SMS-EMOA framework with innovations in HV approximation, reference point management, and diversity handling, respectively.

B. GPU Acceleration in EMO Algorithms

Most previous efforts in GPU acceleration for EMO have focused on specific algorithms and implementations of certain algorithmic components. An early contribution by Wong *et al.* [46] introduced GPU-accelerated non-dominated sorting in NSGA-II. Building on this, Sharma *et al.* [47] proposed G-

ASREA, a GPU-accelerated variant of NSGA-II that incorporates an external archive to sort non-dominated solutions on the GPU. Arca *et al.* [48] applied GPU acceleration to the evaluation phase of NSGA-II, specifically within the context of fuel treatment optimization. Aguilar-Rivera *et al.* [49] further developed a fully vectorized NSGA-II, employing stochastic non-domination sorting and grid-crowding techniques.

In addition to dominance-based algorithms, Souza *et al.* [50] proposed the GPU-accelerated MOEA/D-ACO algorithm, where solution construction and pheromone matrix updates benefit from data-parallel processing. Lopez *et al.* [51] leveraged GPUs to accelerate hypervolume contribution calculations, improving performance in the SMS-EMOA algorithm. Furthermore, Hussain *et al.* [52] introduced a fast CUDA-based implementation of MOPSO on GPUs.

More recently, frameworks based on Google's JAX [53] have opened new possibilities for GPU acceleration in EMO. Examples include EvoJAX [54], evosax [55], and EvoX [56], all of which provide open-source platforms for GPU-accelerated evolutionary algorithms. While EvoJAX and evosax focus on accelerating evolutionary strategies, EvoX is designed as a distributed GPU-accelerated framework that supports general evolutionary computation. Building upon EvoX, a recent effort has been made to tensorize the RVEA [32] for GPU acceleration [57].

Despite these advances, GPU-accelerated EMO algorithms remain in their infancy. Current research has primarily focused on specific implementations that provide isolated performance improvements. Moreover, many of these implementations rely heavily on CUDA programming [58] and are not open-source, thus making them less accessible, particularly for beginners.

III. Tensorization Methodology

In this section, we present how to adopt the general tensorization methodology in EMO algorithms. Specifically, we begin by defining the notation and preliminary concepts used throughout this paper, including the basic definitions of *tensor* and *tensorization*. Next, we demonstrate how to transform atomic operations such as basic operations and control flow operations into tensors. Furthermore, we discuss why and how tensorization matters for GPU acceleration.

A. Preliminaries

A *tensor* is a multidimensional array that generalizes scalars, vectors, and matrices to higher dimensions [59]. Formally, a k-th order tensor is an element of the tensor product of k vector spaces: $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_k}$, where d_i represents the dimension along the i-th mode (axis) of the tensor. Scalars are zero-order tensors, vectors are first-order tensors, and matrices are second-order tensors. Correspondingly, *tensorization* refers to the process of transforming algorithmic data structures and operations into tensor representations. This transformation facilitates efficient parallel computation, particularly on massively parallel hardware such as GPUs, by leveraging the inherent parallelism in tensor operations to improve computational performance and scalability.

TABLE I
Tensor Variables in EMO Algorithms

Notation	Description
n	Population size
m	Number of objectives
d	Problem dimension
$\boldsymbol{X}$	Solution tensor
$\boldsymbol{F}$	Objective tensor
$\boldsymbol{R}, \boldsymbol{W}$	Reference and weight tensors
$\boldsymbol{U}, \boldsymbol{L}$	Upper and lower bound tensors

In this paper, scalars (0-order tensors) are denoted by lowercase letters (e.g., a), vectors (1st-order tensors) are denoted by italicized bold lowercase letters (e.g., $\boldsymbol{a}$), matrices (2nd-order tensors) are denoted by italicized bold uppercase letters (e.g., $\boldsymbol{A}$), and higher-order tensors are denoted by calligraphic letters (e.g., $\mathcal{T}$). The tensor variables related to EMO algorithms are summarized in Table I. Correspondingly, a tensorized MOP can be formulated as:

$$\underset{\boldsymbol{X}}{\text{minimize}} \quad \boldsymbol{F}(\boldsymbol{X}) = (\boldsymbol{f}_1(\boldsymbol{X}), \boldsymbol{f}_2(\boldsymbol{X}), \ldots, \boldsymbol{f}_m(\boldsymbol{X})), \qquad (2)$$

where $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ is the solution tensor for n individuals and d dimensions, and $\boldsymbol{F} \in \mathbb{R}^{n \times m}$ is the corresponding objective tensor.

TABLE II
Basic Tensor Operations

Operation	Description
$\boldsymbol{A} \cdot \boldsymbol{B}$	Tensor multiplication: Product of two tensors $\boldsymbol{A}$ and $\boldsymbol{B}$.
$\boldsymbol{A} \odot \boldsymbol{B}$	Hadamard product: Element-wise multiplication of $\boldsymbol{A}$ and $\boldsymbol{B}$.
$H(\boldsymbol{A})$	Heaviside step function: Returns 1 if $\boldsymbol{A}_{ij} \geq 0$, 0 otherwise.
$\mathbb{1}_{\boldsymbol{A}}$	Indicator function: Returns 1 if $\boldsymbol{A}_{ij}$ is true, 0 otherwise.
sort	Arranges elements in ascending order.
argsort	Returns indices of sorted elements.
min, max	Returns the smallest or largest element along the specified axis: column-wise if axis $= 0$, row-wise if axis $= 1$.
argmin	Returns the indices of the smallest elements.
`vmap`	Vectorization map: Applies a function across an array axis.

B. Tensorization of Data Structures in EMO Algorithms

In EMO algorithms, the candidate solutions and their corresponding objective values are two critical data structures, as expressed in Eq. 2. They can be encoded as tensors $\boldsymbol{X}$ and $\boldsymbol{F}$, respectively. Moreover, decomposition-based algorithms use additional structures such as reference vectors or weights, which are similarly encoded as tensors $\boldsymbol{R}$ and $\boldsymbol{W}$, respectively. Here, $\boldsymbol{R} \in \mathbb{R}^{r \times m}$ typically represents r reference vectors, each corresponding to a different point in the objective space, and $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ denotes a weight tensor with n different weights. These tensor representations allow for the parallel and batch processing of operations along the population dimension, capitalizing on the independence of individuals within the population. Consequently, the EMO algorithm can efficiently process one entire population at a time.

C. Tensorization of Operations in EMO Algorithms

After establishing an effective tensorized data structure, the next crucial step is to consider tensorization for core operations in EMO algorithms, such as crossover, mutation, and selection mechanisms (e.g., environmental selection). These operations consist of numerous atomic operations, including both basic tensor operations and control flow operations.

1) Basic Tensor Operations: Basic tensor operations serve as the foundation for transforming more complex operations in EMO algorithms. Table II summarizes these operations, including tensor multiplication, Hadamard product, Heaviside step function, and indicator function, as well as advanced functions like sort, argsort, min, max, and argmin. These functions are essential for implementing selection and ranking strategies, enabling GPU-accelerated non-dominated sorting, and diversity maintenance in EMO algorithms.

2) Control Flow Operations: The tensorization of control flow operations, including *loops* and *branches*, is a key challenge in transforming traditional EMO algorithms into their tensor-based counterparts. Control flow operations are commonly used to implement iterative procedures and define selection rules based on specific conditions. However, traditional control flow operations such as loops (`for` and `while`) and `if-else` branches introduce sequential dependencies that hinder parallel execution and reduce the efficiency of GPU computations. Tensorizing these operations requires replacing them with tensor-based operations that can execute in parallel.

Loop operations are often used in EMO algorithms for tasks like calculating distances or updating solutions. However, with a large population size, sequential processing becomes inefficient. These loops can be replaced by either vectorized mapping functions (e.g., `vmap`), or by using broadcasting combined with basic operations. The `vmap` function automatically applies a specified function across all elements in a given tensor dimension, eliminating explicit loops. Mathematically, `vmap` can be expressed as:

$$\texttt{vmap}(f)(\boldsymbol{A}) = [f(\boldsymbol{A}_1), f(\boldsymbol{A}_2), \ldots, f(\boldsymbol{A}_n)]^\top, \tag{3}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ is the input tensor and f is the function applied to each individual $\boldsymbol{A}_i$. The function f operates independently on each element, with `vmap` managing parallel computation and concatenation of results. Alternatively, broadcasting can be used to perform the same operations without any loops. Broadcasting works by expanding the dimensions of tensors to align them, enabling element-wise operations to be performed simultaneously across the entire population.

Branch, such as `if-else` operations, are another type of control flow that poses challenges for tensorization. In traditional implementations, `if-else` operations introduce branching and disrupt parallel execution. To address this issue, tensorization replaces branch with element-wise masking operations such as `where`. For example, a traditional branch assigning values to a population matrix based on a threshold can be written as:

$$\boldsymbol{Y}_{ij} = \begin{cases} \boldsymbol{A}_{ij}, & \text{if } \boldsymbol{M}_{ij} > \tau \\ \boldsymbol{B}_{ij}, & \text{otherwise} \end{cases}, \tag{4}$$

```python
import torch

# Conventional implementation
def dominance_detection_conventional(P):
    n = P.size(0)
    dom = torch.zeros(n, n, dtype=torch.bool)
    for i in range(n):
        for j in range(n):
            if i != j:
                if (P[i] <= P[j]).all() and
                    (P[i] < P[j]).any():
                    dom[i, j] = True
    return dom

# Tensor-based implementation
@torch.compile
def dominance_detection_tensor(P):
    P1 = P.unsqueeze(1)
    P2 = P.unsqueeze(0)
    return (P1 <= P2).all(dim=2) & \
           (P1 < P2).any(dim=2)
```

Listing 1: Comparison between conventional and tensor-based implementations of Pareto dominance detection.

where $\boldsymbol{A}$, $\boldsymbol{B}$ are input tensors, τ is a threshold, and $\boldsymbol{M}$ is the mask tensor. It can be replaced by a tensorized masking operation:

$$\boldsymbol{Y} = \mathbb{1}_{\boldsymbol{M}>\tau} \odot \boldsymbol{A} + (1 - \mathbb{1}_{\boldsymbol{M}>\tau}) \odot \boldsymbol{B}, \tag{5}$$

which can be implemented as $\texttt{where}(\boldsymbol{M} > \tau, \boldsymbol{A}, \boldsymbol{B})$.

3) Advantages over Conventional Operations: Tensorization offers several key advantages over conventional EMO implementations. First, it provides greater flexibility by handling multi-dimensional data, whereas conventional matrix operations are often limited to two dimensions. Second, tensorization enhances computational efficiency by enabling parallel processing and removing the need for explicit loops and conditional branches. Finally, tensorization simplifies the code, making it more concise and easier to maintain. For instance, as shown in Listing 1, conventional Pareto dominance detection relies on nested `for` loops and `if-else` statements to compare individuals. In contrast, the tensorized version uses element-wise operations, broadcasting, and masking to perform these comparisons in parallel, significantly reducing code complexity and boosting performance.

D. Discussion

1) Why is tensorization crucial for GPU acceleration: With thousands of cores designed to manage multiple tasks simultaneously, GPUs are intrinsically tailored for parallel computing. The architectures of GPUs, such as the SIMT (Single Instruction, Multiple Threads) model, enable efficient execution of tensor operations. Moreover, the specialized features like NVIDIA's Tensor Cores further enhance performance by accelerating matrix multiplication and accumulation tasks. This makes the tensorization methodology ideally suited for GPUs, which inherently involve large-scale parallel computations.

2) What algorithms are suitable for tensorization: Algorithms with independent computations and minimal branching are ideal for tensorization, as they can be easily parallelized.

In contrast, algorithms that depend on sequential processes, frequent branching, or recursion pose challenges for parallelization. For example, in traditional MOEA/D, the aggregate function computation relies on results from previous iterations, thus making direct tensorization challenging. Nonetheless, by restructuring and decoupling such algorithms, the tensorization methodology can still be effectively applied.

IV. Tensorization Implementation in Representative EMO Algorithms

In this section, we present the application of tensorization methodology in three representative EMO algorithms: NSGA-III, MOEA/D, and HypE. The genetic operators, including mating selection, crossover, and mutation, are common across most EMO algorithms and follow similar tensorization procedures, which are elaborated in Section S.I of the Supplementary Document. Here, we focus on the tensorized implementation of the environmental selection operators specific to each algorithm.

It is important to note that both the environmental selection in NSGA-III and the Monte Carlo-based selection in HypE are inherently highly parallelizable, which allows for straightforward tensorization. In contrast, the MOEA/D algorithm presents a unique challenge due to its fundamentally sequential nature. As shown in Algorithm 1, each subproblem in MOEA/D involves four interdependent steps that must be executed in sequence. This sequential dependency prevents direct tensorization and requires a reconfiguration of the entire process to enable parallel computation, which will be elaborated in Section IV-B.

A. Tensorized Environmental Selection in NSGA-III

The key components of environmental selection in NSGA-III include *non-dominated sorting*, *normalization*, *association*, *niche count calculation*, and *niche selection*. The tensorization process of each component is elaborated as follows. The pseudocode of both the original and tensorized algorithms is provided in Section S.II of the Supplementary Document.

1) Non-dominated Sorting: Given the combined objective tensor $\boldsymbol{F} \in \mathbb{R}^{2n\times m}$, representing the objective tensor of both parent and offspring populations, the non-dominated rank is computed iteratively. The primary goal is to assign a non-dominated rank to each individual, where lower ranks correspond to better solutions.

First, the dominance relation tensor $\boldsymbol{D} \in \{0,1\}^{2n\times 2n}$ is computed using `vmap` or broadcasting for parallel processing. Each element $\boldsymbol{D}_{ij}$ indicates whether the solution $\boldsymbol{F}_i$ dominates $\boldsymbol{F}_j$:

$$\boldsymbol{D}_{ij} = \boldsymbol{F}_i \prec \boldsymbol{F}_j, \quad i,j = 1,2,\ldots,2n. \tag{6}$$

Next, we calculate the dominance count tensor $\boldsymbol{c} \in \mathbb{Z}^{2n}$, which indicates how many individuals each individual dominates:

$$\boldsymbol{c} = \sum_{j=1}^{2n} \boldsymbol{D}_{ij}. \tag{7}$$

The rank tensor $\boldsymbol{r} \in \mathbb{Z}^{2n}$ is initialized to zeros, and the rank counter k is set to zero. The boolean tensor $\boldsymbol{p} \in \{0,1\}^{2n}$, the set of all non-dominated solutions at the current rank, is obtained by $\boldsymbol{p} = \mathbb{1}_{\boldsymbol{c}=0}$.

In each iteration, individuals sharing the same dominance rank are identified and processed collectively. This method ensures that even with a large population size, the number of iterations remains relatively low. Additionally, the `while` function is optimized for accelerated computation. The rank tensor $\boldsymbol{r}$ is updated as follows:

$$\boldsymbol{r} = H(\boldsymbol{p}) \cdot k + H(1-\boldsymbol{p}) \odot \boldsymbol{r}. \tag{8}$$

After rank assignment, the dominance count $\boldsymbol{c}$ is updated by:

$$\boldsymbol{c} = \boldsymbol{c} - \sum_{i=1}^{2n} \boldsymbol{p}_i \cdot \boldsymbol{D}_{ij} - \boldsymbol{p}. \tag{9}$$

The process repeats until all individuals are ranked, which means that all elements in $\boldsymbol{p}$ are zero. Once all ranks are assigned, the tensor $\boldsymbol{r}$ is sorted to determine the rank l of the n-th individual.

2) Normalization: After performing non-dominated sorting, the objective tensor $\boldsymbol{F}$ undergoes a normalization process, similar to that in the original NSGA-III algorithm. This normalization ensures that the objectives are comparable by mapping them onto a hyperplane, enabling the algorithm to maintain diversity across generations.

3) Association: In this step, each individual in the population is associated with the closest reference point. The distance between the normalized objective tensor $\boldsymbol{F}'$ and the reference tensor $\boldsymbol{R}$ is computed using the perpendicular distance:

$$\boldsymbol{D} = \|\boldsymbol{F}'\| \cdot \sqrt{1 - \left(\boldsymbol{F}' \cdot \boldsymbol{R}^\top / (\|\boldsymbol{F}'\| \cdot \|\boldsymbol{R}\|)\right)^2}. \tag{10}$$

Based on the distance tensor $\boldsymbol{D}$, the index of the closest reference point $\boldsymbol{\pi}$ is the index of the minimum value in each row of $\boldsymbol{D}$, and the corresponding distance $\boldsymbol{d}$ represents the minimum value in each row.

4) Niche Count Calculation: For each reference point, the niche count is computed, which indicates how many individuals are associated with that reference point. The niche count tensor $\boldsymbol{\rho}$ is calculated as:

$$\boldsymbol{\rho}_j = \sum_{i=1}^{2n} H(l - \boldsymbol{r}_i) \cdot \mathbb{1}_{\boldsymbol{\pi}_i = j}, \quad j = 1,\ldots,n_r, \tag{11}$$

where n_r is the number of reference points. The tensor $\boldsymbol{\rho}_l$ represents the niche count for the last front (i.e., the niche count corresponding to the front when $\boldsymbol{r}_i = l$):

$$\boldsymbol{\rho}_{l,j} = \sum_{i=1}^{2n} \mathbb{1}_{\boldsymbol{r}_i = l} \cdot \mathbb{1}_{\boldsymbol{\pi}_i = j}, \quad j = 1,\ldots,n_r. \tag{12}$$

The total number of selected individuals n_s is then updated as $n_s = \sum \boldsymbol{\rho}$.

5) Niche Selection: In the niche selection process of NSGA-III, the distance tensor $\boldsymbol{D}'$ represents $\boldsymbol{D}$ adjusted by niche counts for each reference tensor. This tensor is used to identify individuals closest to underpopulated niches. The index of the selected individual $\boldsymbol{q}$ is determined by minimizing the distance in each row of $\boldsymbol{D}'$, i.e., $\boldsymbol{q} = \arg\min_j(\boldsymbol{D}')$. After selecting the individual, the rank tensor $\boldsymbol{r}$ is updated by setting

Algorithm Environmental Selection of TensorMOEA/D

Input: Population tensor $\boldsymbol{X}$, Objective tensor $\boldsymbol{F}_1$, Offspring tensor $\boldsymbol{O}$, the objective of offspring $\boldsymbol{F}_2$, the population size n, the ideal points $\boldsymbol{z}$, the weights $\boldsymbol{W}$, the neighbors indices $\boldsymbol{I}_{\text{nb}}$, and the PBI function f_{PBI};

Output: Next population tensor $\boldsymbol{X}_{\text{next}}$ and next objective tensor $\boldsymbol{F}_{\text{next}}$;

1: $\boldsymbol{z}_{\min} \leftarrow \min_i(\boldsymbol{z} \cup \boldsymbol{F}_2)$;
2: $\boldsymbol{I}_{\text{sub}} \leftarrow [i \mid i \in \mathbb{N}, 0 \leq i < n]$;
3: $\boldsymbol{M} \leftarrow \boldsymbol{0}_n$;
4: **Function** $f_{\text{op1}}(\boldsymbol{i}_{\text{nb}}, \boldsymbol{f}_2)$
5: $\quad \boldsymbol{g}_{\text{old}} \leftarrow f_{\text{PBI}}(\boldsymbol{F}_1[\boldsymbol{i}_{\text{nb}}], \boldsymbol{w}[\boldsymbol{i}_{\text{nb}}], \boldsymbol{z}_{\min})$;
6: $\quad \boldsymbol{g}_{\text{new}} \leftarrow f_{\text{PBI}}(\boldsymbol{f}_2, \boldsymbol{w}[\boldsymbol{i}_{\text{nb}}], \boldsymbol{z}_{\min})$; (1)
7: $\quad \boldsymbol{M}[\boldsymbol{i}_{\text{nb}}] \leftarrow H(\boldsymbol{g}_{\text{old}} - \boldsymbol{g}_{\text{new}})$;
8: $\quad \boldsymbol{I}_{\text{sub}} \leftarrow \boldsymbol{M} \odot (-1) + (1 - \boldsymbol{M}) \odot \boldsymbol{I}_{\text{sub}}$;
9: $\quad$ **return** $\boldsymbol{I}_{\text{sub}}$
10: $\boldsymbol{I}_{\text{new}} \leftarrow \texttt{vmap}(f_{\text{op1}})(\boldsymbol{I}_{\text{nb}}, \boldsymbol{F}_2)$;
11: **Function** $f_{\text{op2}}(\boldsymbol{i}_{\text{new}}, \boldsymbol{x}, \boldsymbol{f}_1, \boldsymbol{w})$
12: $\quad \boldsymbol{f} \leftarrow \mathbb{1}_{\boldsymbol{i}_{\text{new}}=-1} \odot \boldsymbol{F}_2 + (1 - \mathbb{1}_{\boldsymbol{i}_{\text{new}}=-1}) \odot \boldsymbol{f}_1$;
13: $\quad \boldsymbol{x} \leftarrow \mathbb{1}_{\boldsymbol{i}_{\text{new}}=-1} \odot \boldsymbol{O} + (1 - \mathbb{1}_{\boldsymbol{i}_{\text{new}}=-1}) \odot \boldsymbol{x}$; (2)
14: $\quad i \leftarrow \text{argmin}(f_{\text{PBI}}(\boldsymbol{f}, \boldsymbol{w}, \boldsymbol{z}_{\min}))$;
15: $\quad$ **return** $\boldsymbol{x}[i], \boldsymbol{f}[i]$
16: $\boldsymbol{X}_{\text{next}}, \boldsymbol{F}_{\text{next}} \leftarrow \texttt{vmap}(f_{\text{op2}})(\boldsymbol{I}_{\text{new}}^{\top}, \boldsymbol{X}, \boldsymbol{F}_1, \boldsymbol{w})$;
17: $\boldsymbol{z} \leftarrow \boldsymbol{z}_{\min}$;

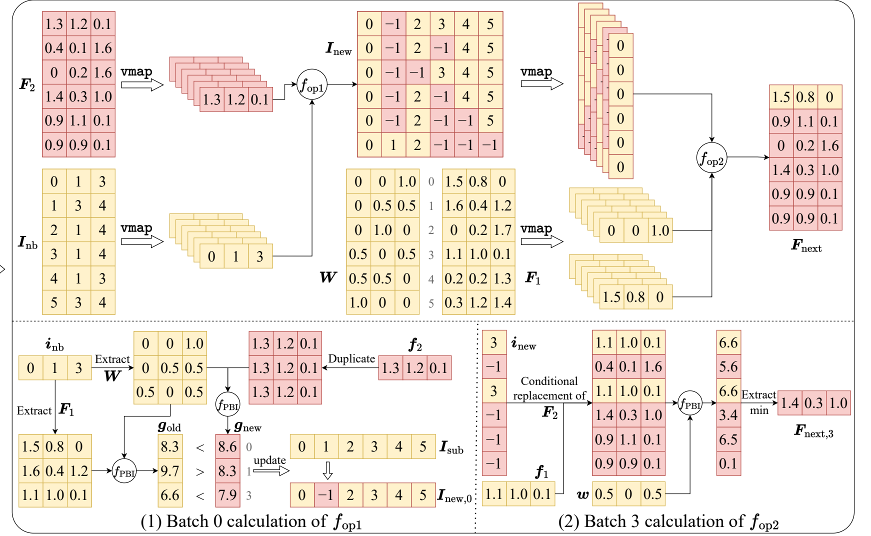


Fig. 1: Overview of the environmental selection in the TensorMOEA/D algorithm. **Left**: Pseudocode of the algorithm. **Right**: Tensor dataflow of module (1) and module (2). The upper part of the right figure shows the overall tensor dataflow for modules (1) and (2), while the lower part presents the batch calculation tensor dataflow, with module (1) on the left and module (2) on the right.

$\boldsymbol{r}[\boldsymbol{q}] = l - 1$, with further updates in subsequent iterations reflecting the inclusion of individuals in the current front.

Once the selection is complete, the indices of the top n individuals for the next generation are determined by:

$$\boldsymbol{i}_{\text{next}} = \text{sort}(H(l - \boldsymbol{r}) \odot \boldsymbol{a} + \infty \cdot (1 - H(l - \boldsymbol{r})))[: n], \quad (13)$$

where $\boldsymbol{a} = [0, 1, \ldots, n-1]$. The final solution tensors $\boldsymbol{X}_{\text{next}}$ and $\boldsymbol{F}_{\text{next}}$ are formed by selecting individuals using the sorted indices: $\boldsymbol{X}_{\text{next}} = \boldsymbol{X}[\boldsymbol{i}_{\text{next}}]$ and $\boldsymbol{F}_{\text{next}} = \boldsymbol{F}[\boldsymbol{i}_{\text{next}}]$.

Algorithm 1 Main Framework of Original MOEA/D

Input: The maximal number of generations $t_{\max}$; n weight vectors; the neighborhood size T of each weight vector;

Output: Final population;

1: **Initialization**;
2: **for** $t = 1$ to $t_{\max}$ **do**
3: $\quad$ **for** $i = 1$ to n **do**
4: $\quad\quad$ Reproduction;
5: $\quad\quad$ Fitness Evaluation;
6: $\quad\quad$ Ideal Point Update;
7: $\quad\quad$ Neighborhood Update;
8: $\quad$ **end for**
9: **end for**

B. Tensorized Environmental Selection in MOEA/D

In the original MOEA/D algorithm, as shown in Algorithm 1, the reproduction, fitness evaluation, ideal point update, and neighborhood update are executed *sequentially* within a single loop. This method requires processing individuals one by one in a specific order, which can significantly impede the execution speed of the algorithm. To address this limitation, we apply tensorization methodology to decouple these four steps in the inner loop (i.e., environmental selection), treating them as independent operations. This adjustment enables parallel processing of all individuals in the tensorized version, referred to as TensorMOEA/D.

In TensorMOEA/D, reproduction generates n individuals simultaneously based on the neighborhood, contrasting with the original MOEA/D, which produces one individual at a time. The environmental selection process is further divided into two main steps: *comparison and population update*, and *elite selection*. These two steps are primarily implemented using two `vmap` operations, with the tensorization process detailed as follows. The pseudocode of both the original and tensorized algorithms is provided in Section S.III of the Supplementary Document.

1) Comparison and Population Update: The primary objective of this step is to determine the indices for the updated population by comparing the aggregated function values of the old and new populations. This process ultimately generates an updated index tensor $\boldsymbol{I}_{\text{new}}$, where each row corresponds to a subpopulation that mirrors the structure of the original population, containing indices of n individuals. Positions that require updates are indicated by -1.

Given the solution tensor $\boldsymbol{X}$, objective tensor $\boldsymbol{F}_1$, offspring tensor $\boldsymbol{O}$, the objective of offspring $\boldsymbol{F}_2$, the ideal points $\boldsymbol{z}$, the weights $\boldsymbol{W}$, and the neighbors indices $\boldsymbol{I}_{\text{nb}}$, the process begins by calculating the minimal objective values to update the reference points, $\boldsymbol{z}_{\min}$, by finding the minimum objective values between the current population and the offspring.

Next, the subpopulation indices $\boldsymbol{I}_{\text{sub}}$ are created to track population updates. The `vmap` function is utilized to calculate the update indices $\boldsymbol{I}_{\text{new}}$:

$$\boldsymbol{I}_{\text{new}} = \texttt{vmap}(f_{\text{op1}})(\boldsymbol{I}_{\text{nb}}, \boldsymbol{F}_2), \quad (14)$$

where $f_{\text{op1}}(\boldsymbol{i}_{\text{nb}}, \boldsymbol{f}_2) = \boldsymbol{I}_{\text{sub}}$. This process is conducted in parallel by batching the rows of $\boldsymbol{I}_{\text{nb}}$ and $\boldsymbol{F}_2$, with $\boldsymbol{i}_{\text{nb}}$ and

$\boldsymbol{f}_2$ representing a single batch of these tensors. Each batch undergoes the f_{op1} operation, and all batches are processed simultaneously to yield results for all updates. A visual representation of this operation can be found in Fig. 1, with an example for the first batch illustrated in the lower left corner of the tensor dataflow on the right.

In the f_{op1} function, $\boldsymbol{i}_{\text{nb}}$ is used as an index to extract the corresponding rows from $\boldsymbol{F}_1$ and $\boldsymbol{W}$, and the entries of $\boldsymbol{f}_2$ are replicated to obtain the inputs needed for the aggregation function. The aggregation function employs the penalty-based boundary intersection (PBI) function:

$$f_{\text{PBI}}(\boldsymbol{f}, \boldsymbol{w}, \boldsymbol{z}) = d_1 + \theta \cdot d_2, \tag{15}$$

where $d_1 = \frac{\|(\boldsymbol{f}-\boldsymbol{z})^\top \cdot \boldsymbol{w}\|}{\|\boldsymbol{w}\|}$, $d_2 = \|(\boldsymbol{f} - \boldsymbol{z}) - (d_1 \cdot \boldsymbol{w})\|$, and θ is a preset penalty parameter. The old and new aggregated values $\boldsymbol{g}_{\text{old}}$ and $\boldsymbol{g}_{\text{new}}$ are computed by applying the PBI function. The mask $\boldsymbol{M}$ stores the comparison results and is initially a zero tensor, updated as follows:

$$\boldsymbol{M}[\boldsymbol{i}_{\text{nb}}] = H(\boldsymbol{g}_{\text{old}} - \boldsymbol{g}_{\text{new}}). \tag{16}$$

Finally, the subpopulation indices $\boldsymbol{I}_{\text{sub}}$ are updated based on $\boldsymbol{M}$:

$$\boldsymbol{I}_{\text{sub}} = \boldsymbol{M} \odot (-1) + (1 - \boldsymbol{M}) \odot \boldsymbol{I}_{\text{sub}}, \tag{17}$$

where positions that need updates are assigned a value of -1.

2) Elite Selection: The purpose of this step is to select the best individuals along n distinct weighted directions based on the aggregated function values. Each direction yields a single elite individual, resulting in a total of n individuals that form the population of next generation.

To efficiently update the solution and objective tensor for the entire population, the function `vmap` is applied to parallelize the computation over all rows of the input:

$$\boldsymbol{X}_{\text{next}}, \boldsymbol{F}_{\text{next}} = \texttt{vmap}(f_{\text{op2}})(\boldsymbol{I}_{\text{new}}^\top, \boldsymbol{X}, \boldsymbol{F}_1, \boldsymbol{W}), \tag{18}$$

where `vmap` maps the function $f_{\text{op2}}(\boldsymbol{i}_{\text{new}}, \boldsymbol{x}, \boldsymbol{f}_1, \boldsymbol{w})$ to each row of the provided inputs in parallel. Fig. 1 shows this process in the bottom-right corner, highlighting the computation for batch 3 (i.e., the 4th row).

The function f_{op2} is defined to update the population and objective tensor based on the new indices $\boldsymbol{i}_{\text{new}}, \boldsymbol{f}_1, \boldsymbol{x},$ and $\boldsymbol{w}$. Specifically, the objective values $\boldsymbol{f}$ are updated as $\boldsymbol{f} = \mathbb{1}_{\boldsymbol{i}_{\text{new}}=-1} \odot \boldsymbol{F}_2 + (1 - \mathbb{1}_{\boldsymbol{i}_{\text{new}}=-1}) \odot \boldsymbol{f}_1$. And the individual is updated as $\boldsymbol{x} = \mathbb{1}_{\boldsymbol{i}_{\text{new}}=-1} \odot \boldsymbol{O} + (1 - \mathbb{1}_{\boldsymbol{i}_{\text{new}}=-1}) \odot \boldsymbol{x}$. The index $i = \text{argmin}(f_{\text{PBI}}(\boldsymbol{f}, \boldsymbol{w}, \boldsymbol{z}_{\text{min}}))$ is used to select the best solution in $\boldsymbol{x}$. Finally, the ideal points $\boldsymbol{z}$ are updated to the minimum objective values $\boldsymbol{z}_{\text{min}}$ for the next iteration.

C. Tensorized Environmental Selection in HypE

The environmental selection in HypE primarily involves non-dominated sorting and HV calculation. In this paper, we utilize the Monte Carlo estimation method for HV calculation that is well-suited for tensorization. This Monte Carlo-based HV calculation consists of four key steps: *sampling bound determination*, *sampling weights calculation*, *dominance score and distance update*, and *hypervolum calculation*. The implementation details for these steps are as follows. The pseudocode of both the original and tensorized algorithms are provided in Section S.IV of the Supplementary Document.

1) Sampling Bound Determination: The lower and upper bounds for sampling are determined by calculating the minimum objective values, $\boldsymbol{f}_l = \min_i(\boldsymbol{F})$, and using the reference point, $\boldsymbol{f}_u = \boldsymbol{v}_{\text{ref}}$. These bounds define the hyperrectangle for sampling. Next, uniformly sample points from the hyperrectangle defined by $\boldsymbol{f}_l$ and $\boldsymbol{f}_u$ to generate a sample tensor $\boldsymbol{S}$ of dimension $s \times m$. The initial distance score $\boldsymbol{v}_{\text{ds}}$ is then initialized to zeros with dimensions $1 \times s$.

2) Sampling Weights Calculation: The sampling weights $\boldsymbol{\alpha}$ are calculated as follows:

$$\boldsymbol{\alpha}_j = \prod_{i=1}^{j} \boldsymbol{\lambda}_i / j, \quad j = 1, 2, \ldots, k, \tag{19}$$

where $\boldsymbol{\lambda}$ is defined as $\boldsymbol{\lambda} = [1, (k - \boldsymbol{l})/(n_1 - \boldsymbol{l})]$. Here, $\boldsymbol{l} = [i \mid i \in \mathbb{N}, 1 \leq i < n_1]$, and n_1 is the number of rows in $\boldsymbol{F}$.

3) Dominance Score and Distance Updating: Calculate the dominance scores for each sample i using the function f_{pds}:

$$f_{\text{pds}}(\boldsymbol{f}) = \mathbb{1}_{\sum_{j=1}^{m} H(\boldsymbol{S}_{ij} - \boldsymbol{f}) = m}, \tag{20}$$

where $i = 1, 2, \ldots, s$. The dominance scores $\boldsymbol{T}_{\text{pds}}$ are then computed by applying `vmap` in parallel to the function f_{pds} across all rows of the objective tensor $\boldsymbol{F}$.

The distance score $\boldsymbol{v}_{\text{ds}}$ is then updated based on a temporary matrix $\boldsymbol{T}_{\text{temp}}$, which is computed as $\boldsymbol{T}_{\text{pds}}$ combined with $\boldsymbol{v}_{\text{ds}}$ and a tensor of ones $\mathbb{1}_{n \times 1}$:

$$\boldsymbol{v}_{\text{ds}} = \text{maximum}\left(\sum_{i=1}^{n_1} (\boldsymbol{T}_{\text{temp}})_i - 1, 0\right), \tag{21}$$

where $\boldsymbol{T}_{\text{temp}}$ is calculated as $\boldsymbol{T}_{\text{pds}} \odot (\mathbf{1}_{n \times 1} \cdot \boldsymbol{v}_{\text{ds}} + 1) + (1 - \boldsymbol{T}_{\text{pds}}) \odot (\mathbf{1}_{n \times 1} \cdot \boldsymbol{v}_{\text{ds}})$, and $\text{maximum}(\cdot, 0)$ is an element-wise operation that compares each element with 0, returning the element itself if it is greater than or equal to 0, and 0 otherwise.

4) Hypervolume Calculation: The HV contributions are calculated using the function f_{hv}, which sums the contributions from each sample:

$$f_{\text{hv}}(\boldsymbol{t}_{\text{pds}}) = \sum_{i=1}^{s} \left(\boldsymbol{\alpha}[\delta] \odot \mathbb{1}_{\boldsymbol{t}_{\text{pds}} \neq -1}\right)_i, \tag{22}$$

where $\boldsymbol{\delta} = \boldsymbol{t}_{\text{pds}} \odot \boldsymbol{v}_{\text{ds}} - (1 - \boldsymbol{t}_{\text{pds}})$. This function is then applied in parallel to each row of the point dominance score tensor $\boldsymbol{T}_{\text{pds}}$ using `vmap`, which efficiently computes the HV contributions across all samples.

Finally, the total HV $\boldsymbol{v}_{\text{hv}}$ is obtained by aggregating these contributions and normalizing:

$$\boldsymbol{v}_{\text{hv}} = \boldsymbol{v}_{\text{hv}} \cdot \prod_{i=1}^{m} \left(\boldsymbol{v}_{\text{ref},i} - \boldsymbol{f}_{l,i}\right) / s. \tag{23}$$

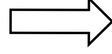
For other indicator-based EMO algorithms, the indicator is typically calculated using mathematical expressions, which facilitates straightforward tensorization. In the case of more complex indices, such as HV, Monte Carlo sampling can be employed to enhance algorithm efficiency by approximating these values. Additionally, to improve efficiency and scalability, the niche selection phase can use a one-shot method [12] for parallel selection.

Algorithm Tensorized Non-dominated Sorting

Input: Objective tensor $\boldsymbol{F} \in \mathbb{R}^{2n \times m}$ and population size n.

Output: Nondomination rank tensor $\boldsymbol{r}$ and the last rank $l \in \mathbb{N}$.

1: $\boldsymbol{D} \leftarrow [\boldsymbol{F}_i \prec \boldsymbol{F}_j]_{i,j}, \quad i,j = 1,2,\ldots,2n$;
2: $\boldsymbol{c} \leftarrow \sum_{j=1}^{2n} \boldsymbol{D}_{ij}, \quad i = 1,2,\ldots,2n$;
3: $\boldsymbol{r} \leftarrow \boldsymbol{0}_{2n\times 1}$;
4: $\boldsymbol{p} \leftarrow \mathbb{1}_{\boldsymbol{c}=0}$;
5: $k \leftarrow 0$;
6: **while** any($\boldsymbol{p}$) **do**
7: $\quad \boldsymbol{r} \leftarrow H(\boldsymbol{p}) \cdot k + H(1-\boldsymbol{p}) \odot \boldsymbol{r}$;
8: $\quad \boldsymbol{d}_j \leftarrow \sum_{i=1}^{2n} (\boldsymbol{p}_i \cdot \boldsymbol{D}_{ij}), \quad j = 1,2,\ldots,2n$;
9: $\quad \boldsymbol{c} \leftarrow \boldsymbol{c} - \boldsymbol{d} - \boldsymbol{p}$;
10: $\quad k \leftarrow k+1$;
11: $\quad \boldsymbol{p} \leftarrow \mathbb{1}_{\boldsymbol{c}=0}$;
12: **end while**
13: $l \leftarrow \text{sort}(\boldsymbol{r})[n]$;



```python
import torch

def dominate_relation(X, Y):
    X = X.unsqueeze(1)
    Y = Y.unsqueeze(0)
    return (X <= Y).all(dim=2) & (X < Y).any(dim=2)


def non_dominated_sort(F, n):
    D = dominate_relation(F, F)
    c = torch.sum(D, dim=0)
    r = torch.zeros(F.size(0), dtype=torch.int32)
    p = c == 0
    k = 0

    while p.any():
        r = torch.where(p, c, r)
        d = torch.sum(p.unsqueeze(1) * D, dim=0)
        c = c - d - p
        k += 1
        p = c == 0

    l = torch.sort(r)[n]
    return r, l
```

Fig. 2: The seamless transformation of the tensorized non-dominated sorting from pseudocode (Left) to Python code (Right).

D. Discussion

The tensorization methodology offers unique transformative benefits in EMO algorithm design and implementation. One of the key advantages is the seamless transformation of EMO algorithms from mathematical formulations into efficient code implementations. This bridge between algorithm design and programming is particularly valuable in GPU computing, where tensor operations can be leveraged for significant performance gains. For instance, Fig. 2 illustrates how pseudocode for tensorized non-dominated sorting can be directly translated into Python code. This straightforward translation reduces the gap between high-level algorithmic design and practical implementation, thereby enabling practitioners to focus more on the theoretical aspects without the burden of intricate code optimization.

Tensorization also provides a high level of conciseness, significantly reducing code complexity compared to traditional iterative pseudocode. By representing population-wide operations as single tensor expressions, tensorization minimizes the need for loops and conditional statements, making the codebase more compact and readable. This conciseness not only eases code maintenance but also reduces the risk of programming errors by limiting procedural complexity.

Moreover, tensorization facilitates reproducibility in the field of EMO. As tensorized code relies on structured mathematical expressions, it becomes easier for researchers and developers to replicate results and benchmark different methods. This standardization paves the way for creating robust and high-performance libraries that can be shared and reused across various applications, thereby ultimately advancing research and industrial applications in EMO.

V. Multiobjective Robot Control Benchmark

Traditional EMO benchmarks, such as ZDT [60], DTLZ [61], WFG [62], LSMOP [63], and MaF [64], primarily focus on numerical optimization problems. While these benchmarks are effective for evaluating the basic mechanisms of EMO algorithms, they are limited in their ability to leverage hardware acceleration, thereby reducing their relevance in GPU computing environments.

In contrast, multiobjective robot control tasks present more realistic and computationally intensive challenges that better reflect real-world applications, which is particularly significant in the emerging area of *embodied artificial intelligence* (Embodied AI) [65]. These tasks provide complex and dynamic environments where multiple objectives must be balanced, such as energy efficiency and stability, making them well-suited for testing the adaptability and robustness of EMO algorithms. However, these tasks have been underexplored in the EMO community due to a lack of suitable benchmarks and the significant computational cost of running these environments.

To address this gap, we introduce the multiobjective robot control benchmark test suite, dubbed *MoRobtrol*, which reformulates nine tasks from the Brax environment [17] into MOPs. As a GPU-accelerated physics simulation engine, Brax provides a substantial performance improvement over CPU-based platforms such as OpenAI Gym [66] and Mo-Gymnasium [67]. By leveraging Brax's GPU computing capabilities, MoRobtrol enables scalable and rapid evaluations, making it an ideal benchmark for testing EMO algorithms in computationally demanding settings in practice.

As illustrated in Fig. S.1, the MoRobtrol benchmark includes nine robot control tasks: MoHalfcheetah, MoHopper, MoSwimmer, MoInvertedDoublePendulum (MoIDP), MoWalker2d, MoPusher, MoReacher, MoHumanoid, and MoHumanoidStandup (MoHumanoid-s). These tasks involve optimizing multiple conflicting objectives, such as speed, energy consumption, and distance to target, reflecting tradeoffs commonly encountered in robotics applications. Specifically, in MoRobtrol, the parameters being optimized are the weights of a multilayer perceptron (MLP), a common design in control policy modeling for evolutionary reinforcement learning (EvoRL) [68]. These parameters are optimized by EMO algorithms to enable agents to maximize performance

TABLE III
OVERVIEW OF MULTIOBJECTIVE ROBOT CONTROL PROBLEMS IN THE PROPOSED MOROBTROL BENCHMARK TEST SUITE

Problem	MLP architecture†	d	m
MoHalfcheetah	$17 \times 16 \times 6$	390	f_v, f_c
MoHopper	$11 \times 16 \times 3$	243	f_v, f_h, f_c
MoSwimmer	$8 \times 16 \times 2$	178	f_v, f_c
MoIDP	$8 \times 16 \times 1$	161	f_{dp}, f_{sp}
MoWalker2d	$17 \times 16 \times 6$	390	f_v, f_c
MoPusher	$23 \times 16 \times 7$	503	f_n, f_d, f_c
MoReacher	$11 \times 16 \times 2$	226	f_d, f_c
MoHumanoid	$244 \times 16 \times 17$	4209	f_v, f_c
MoHumanoid-s	$244 \times 16 \times 17$	4209	f_v, f_c

† All MLP networks use the `tanh` activation function.

across conflicting objectives.

Table III provides an overview of the nine tasks, including the MLP structure, number of parameters (d), and specific objectives for each task. The key objectives across tasks include forward reward (f_v), control cost (f_c), height (f_h), distance penalty (f_{dp}), speed penalty (f_{sp}), distance reward (f_d), and near reward (f_n). The number of objectives m varies depending on the task, allowing for detailed evaluations of algorithmic performance in diverse, real-world-inspired control scenarios. Detailed mathematical definitions are provided in Section S.VI of the Supplementary Document.

VI. EXPERIMENTAL STUDY

In this section, we conduct experiments to evaluate the performance of the tensorized EMO algorithms, including TensorNSGA-III, TensorMOEA/D, and TensorHypE. The experiments are categorized into three main aspects: acceleration performance, benchmarking on standard EMO test problems, and evaluation in multiobjective robot control tasks. All experiments are conducted on an RTX 4090 GPU server with AMD EPYC 7543 CPUs using the EvoX [56] framework.

A. Acceleration Performance

To verify the acceleration performance of the three proposed algorithms, we have conducted two sub-experiments. The first sub-experiment doubles the population size and observes the average runtime per generation. The second sub-experiment doubles the problem dimension and observes the average runtime per generation. We compare the performance of NSGA-III, MOEA/D, and HypE before and after tensorization on both CPU and GPU platforms using the DTLZ1 [61] problem.

Additionally, further experiments have been conducted to compare the performance of the tensorized algorithms in comparison with CUDA-accelerated algorithms in EvoTorch [69], as well as to investigate the impact of different types of GPUs on performance. Detailed results of these experiments can be found in Section S.VII-B and Section S.VII-C of Supplementary Document, respectively.

1) Experimental Settings: In the two sub-experiments, the tensorized and non-tensorized[1] algorithms are independently repeated 10 times on both CPU and GPU devices, with each algorithm evolving for 100 generations. The average runtime per generation is then calculated. In the first sub-experiment, the DTLZ1 problem has a dimension $d = 500$, with $m = 3$, and the population size n is doubled incrementally from 128 to 32 768. In the second sub-experiment, the DTLZ1 problem is configured with $m = 3$, $n = 100$, and d is incrementally doubled from 1024 to 1 048 576.

2) Comparison Results: Fig. 3 shows that TensorNSGA-III, TensorMOEA/D, and TensorHypE consistently achieve faster runtimes on GPU compared to their non-tensorized versions on CPUs. When the population size n reaches 32 768, TensorNSGA-III, TensorMOEA/D, and TensorHypE attain speedups of approximately 191×, 1113×, and 186×, respectively, compared to their CPU-based counterparts. As the problem dimension increases to 1 048 576, these speedups rise to 304×, 228×, and 263×. Although the runtime of tensorized algorithms may increase with n, they consistently outperform the non-tensorized versions. Additionally, as problem dimensions scale up, the runtime of tensorized algorithms remains relatively stable, maintaining a significant performance advantage over the original algorithms.

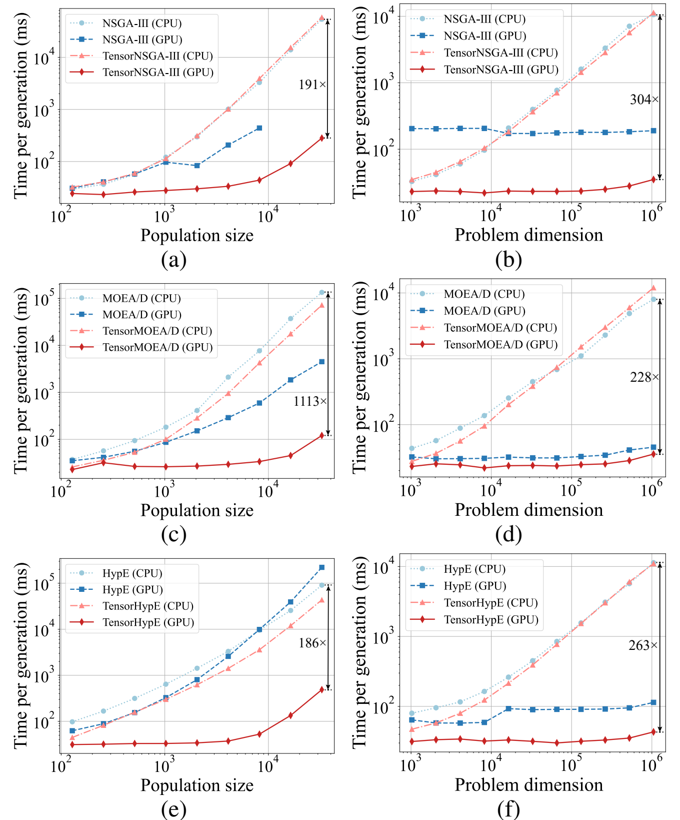


Fig. 3: Comparative acceleration performance of NSGA-III, MOEA/D, and HypE with their tensorized counterparts on CPU and GPU platforms across varying population sizes and problem dimension.

Notably, when n exceeds 16 384, the runtime for NSGA-III on GPU surpasses the preset threshold of 5 hours, resulting in missing data points for larger populations. For the HypE algorithm, once n exceeds 1024, its average runtime per generation on GPU begins to exceed that of TensorHypE on

[1]The non-tensorized algorithms are partially tensorized for efficiency, as the original versions are time-consuming for large populations.

TABLE IV
STATISTICAL RESULTS (MEAN AND STANDARD DEVIATION) OF THE IGD AND RUNTIME (S) FOR NON-TENSORIZED AND TENSORIZED EMO ALGORITHMS IN LSMOP1–LSMOP9. ALL EXPERIMENTS ARE ON AN RTX 4090 GPU AND THE BEST RESULTS ARE HIGHLIGHTED.

Algorithm	Problem	IGD (Non-Tensorized)	IGD (Tensorized)	Time (Non-Tensorized)	Time (Tensorized)
NSGA-III	LSMOP1	**8.2378e−01 (5.2173e−03)**	1.0399e+00 (1.2327e−01)	3.5048e+03 (2.7933e+00)	**7.8573e+00 (2.0971e−01)**
	LSMOP2	**3.5883e−01 (5.9506e−06)**	**3.5883e−01 (2.0165e−05)**	6.0457e+01 (1.2916e+00)	**1.6127e+00 (1.8569e−02)**
	LSMOP3	**1.4106e+00 (7.5580e−02)**	7.3244e+00 (5.3558e−01)	6.1996e+01 (1.6457e+00)	**2.4987e+00 (3.4992e−02)**
	LSMOP4	**5.9798e−01 (1.5968e−04)**	5.9815e−01 (2.0699e−04)	6.5350e+01 (1.8958e+00)	**1.7237e+00 (1.5717e−02)**
	LSMOP5	**5.6862e−01 (2.3841e−03)**	6.4269e−01 (6.6591e−03)	6.0026e+01 (1.0193e+00)	**7.0027e+00 (8.2602e−02)**
	LSMOP6	**2.8277e+00 (5.9075e−01)**	2.3737e+01 (2.9479e+00)	6.3764e+01 (1.8220e+00)	**3.5406e+00 (5.6013e−02)**
	LSMOP7	**1.8425e+00 (7.5222e−03)**	1.8474e+00 (3.7155e−03)	6.2175e+01 (1.2735e+00)	**5.0992e+00 (1.3959e−01)**
	LSMOP8	**3.0317e−01 (2.9719e−02)**	3.5186e−01 (1.3261e−02)	6.6262e+01 (1.4221e+00)	**5.6878e+00 (4.9742e−02)**
	LSMOP9	**7.4488e−01 (9.4755e−03)**	7.6837e−01 (2.9024e−03)	6.1194e+01 (1.6353e+00)	**3.4327e+00 (2.2911e−02)**
MOEA/D	LSMOP1	**7.3475e−01 (3.8276e−04)**	7.4365e−01 (2.1169e−03)	9.4239e+01 (3.0883e+00)	**5.4412e+00 (2.8179e−01)**
	LSMOP2	**3.5885e−01 (3.4131e−05)**	3.5888e−01 (5.9703e−06)	9.9946e+01 (2.7744e+00)	**5.7516e+00 (6.0279e−01)**
	LSMOP3	1.0815e+00 (1.0631e−01)	**8.2722e−01 (5.3689e−02)**	1.0148e+02 (2.6072e+00)	**7.0391e+00 (4.7843e−01)**
	LSMOP4	5.9570e−01 (1.5407e−03)	**5.9420e−01 (1.1813e−03)**	1.0919e+02 (2.2606e+00)	**7.1559e+00 (3.8251e−01)**
	LSMOP5	4.0554e−01 (8.8890e−03)	**3.7065e−01 (8.5792e−03)**	9.7131e+01 (2.9509e+00)	**7.1893e+00 (4.6098e−01)**
	LSMOP6	**1.6366e+00 (7.7256e−02)**	1.8622e+00 (1.2612e−01)	9.6220e+01 (2.9977e+00)	**6.8919e+00 (2.4831e−01)**
	LSMOP7	**1.5534e+00 (2.6469e−01)**	1.7345e+00 (3.6746e−03)	1.0456e+02 (2.4456e+00)	**7.0900e+00 (4.0993e−01)**
	LSMOP8	2.1207e−01 (6.4136e−03)	**2.0107e−01 (1.8164e−02)**	1.0154e+02 (2.0132e+00)	**7.1273e+00 (4.2536e−01)**
	LSMOP9	5.1964e−01 (1.7163e−02)	**5.0269e−01 (1.0548e−02)**	1.0443e+02 (2.9910e+00)	**7.2184e+00 (7.7100e−01)**
HypE	LSMOP1	**8.2757e−01 (1.2610e−03)**	**8.2747e−01 (1.0132e−03)**	6.3918e+01 (6.1344e−01)	**1.8814e+00 (9.9671e−02)**
	LSMOP2	**3.5912e−01 (5.5267e+00)**	**3.5912e−01 (4.9532e−05)**	6.4140e+01 (4.4810e−01)	**1.5211e+00 (2.8512e−02)**
	LSMOP3	**4.6906e+00 (3.2754e−01)**	**4.6906e+00 (3.2754e−01)**	6.4090e+01 (4.1599e−01)	**1.7588e+00 (4.3342e−02)**
	LSMOP4	**5.9841e−01 (2.3223e−04)**	**5.9847e−01 (2.5082e−04)**	6.3900e+01 (2.4586e−01)	**1.6785e+00 (4.7110e−02)**
	LSMOP5	**5.2160e−01 (1.3777e−03)**	**5.2153e−01 (1.6406e−03)**	6.4216e+01 (2.9358e−01)	**2.0728e+00 (1.6839e−02)**
	LSMOP6	**3.0407e+00 (1.2805e−01)**	**3.0267e+00 (1.1215e−01)**	6.5536e+01 (3.2336e−01)	**3.4693e+00 (7.3326e−02)**
	LSMOP7	**1.8223e+00 (2.5992e−03)**	**1.8220e+00 (2.5472e−03)**	6.4099e+01 (2.5939e−01)	**1.7594e+00 (2.9685e−02)**
	LSMOP8	**3.6062e−01 (9.5863e−03)**	**3.6060e−01 (9.5862e−03)**	6.4537e+01 (3.3889e−01)	**2.3856e+00 (2.8679e−02)**
	LSMOP9	**6.5873e−01 (3.0162e−03)**	**6.5797e−01 (3.4461e−03)**	6.3643e+01 (2.6093e−01)	**1.5741e+00 (1.2765e−02)**

CPU, and for $n > 8192$, the GPU version of HypE becomes even slower than the original CPU-based implementation.

These performance drops can be attributed to two main factors. First, the original implementations of NSGA-III and HypE do not fully leverage the multicore parallel processing capabilities of GPUs, leading to underutilization of GPU cores. Second, the data transfer overhead between CPU and GPU further reduces efficiency, particularly for large population sizes where the NSGA-III and HypE algorithms incur higher computational costs on GPU than on CPU.

Additionally, the acceleration performance varies across tensorized algorithms. MOEA/D and HypE, due to their simpler operations and fewer conditional branches, achieve greater acceleration after tensorization compared to their original versions. Conversely, NSGA-III shows more limited acceleration, as its more complex operations involve intricate loops and branches, which are less amenable to GPU parallelization.

B. Performance in Numerical Optimization

To verify the precision before and after tensorization, the three proposed algorithms and their original versions are comprehensively tested on the LSMOP [63] and DTLZ [61] test suites. The detailed results for the DTLZ are provided in Section S.VII-D of Supplementary Document.

1) Experimental Settings: In this experiment, all algorithms are independently repeated 31 times on 9 LSMOP problems and 7 DTLZ problems. Each algorithm is run for 100 generations with a population size of $10\,000$. Each problem has a dimension of 5000. Performance is measured using the average inverted generational distance (IGD) [70] and average runtime over 31 runs. A Wilcoxon rank-sum test is used to compare tensorized and non-tensorized algorithms. If the test shows no significant difference (i.e., $p > 0.05$), both performance indicators are highlighted in bold.

2) Comparison Results: Table IV demonstrates that the tensorized algorithms maintain comparable precision to their non-tensorized counterparts while achieving significantly faster runtimes. The average IGD between the tensorized and non-tensorized algorithms consistently remain within the same order of magnitude, indicating similar levels of solution quality. In some instances, the tensorized algorithms even exhibit superior indicator performance. Additionally, the average runtime for tensorized algorithms is consistently lower than that for non-tensorized algorithms, underscoring the efficiency gains enabled by tensorization. Notably, when comparing performance within equivalent time frames, tensorized algorithms consistently achieve better indicator performance, highlighting their effectiveness in optimizing both time and solution quality metrics.

However, some performance degradation is observed in certain cases. In TensorNSGA-III, batch random operations simulate the niche selection process of the original algorithm, which differs from the original method's precise operation on individual solutions. This discrepancy can lead to performance degradation, as the batch approach may not capture niche selection as effectively. Similarly, TensorMOEA/D adopts batch offspring generation and updates, which accelerate computation per generation but may result in lower performance for the same number of generations. Despite this, TensorMOEA/D consistently achieves or sometimes surpasses the performance of the original algorithm when comparing performance over

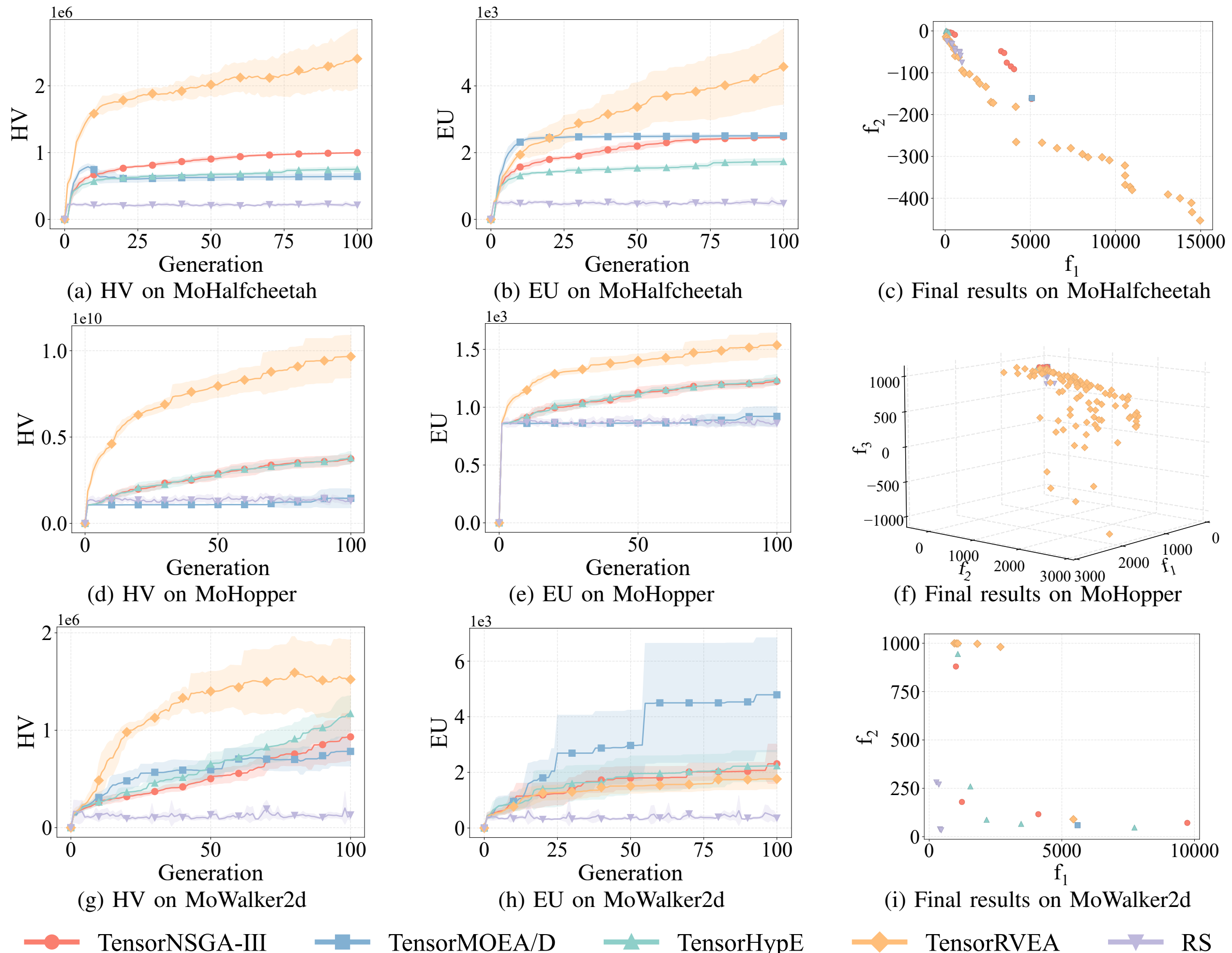


Fig. 4: Comparative performance (HV, EU, and visualization of final results) of TensorNSGA-III, TensorMOEA/D, TensorHypE, TensorRVEA, and random search (RS) across varying problems: MoHalfcheetah (390D), MoHopper (243D), and MoWalker2d (390D). *Note:* Higher values for all metrics indicate better performance.

equivalent time periods. As for HypE, since its method of calculating the HV is largely similar to that of the original algorithm, the primary difference lies in computation speed.

C. Performance in Multiobjective Robot Control Benchmark

In this experiment, we evaluate the advantages of tensorization and the effectiveness of the three proposed tensorized algorithms in solving the multiobjective rotbot control tasks using the proposed MoRobtrol benchmark test suite. Following the paradigm of EvoRL, the EMO algorithms evolve a population of MLP neural networks, with each MLP serving as a policy model within the simulation environment of each task.

1) Experimental Settings: In this experiment, we apply TensorNSGA-III, TensorMOEA/D, TensorHypE, TensorRVEA [57], and random search (RS) algorithms to solve 9 multiobjective robot control problems in MoRobtrol. Each algorithm is repeated 10 times with a population size of 10 000. Performance is evaluated using hypervolume (HV) [13], expected utility (EU) [71], and visualization of the final non-dominated solutions. These indicators are calculated based on the non-dominated solutions of each generation. Details on the TensorRVEA algorithm and the reference points for HV calculation are provided in Sections S.V and S.VII-E of the Supplementary Document, respectively.

2) Comparison Results: As shown in Fig. 4, TensorRVEA achieves the highest HV on the MoHalfcheetah, MoHopper, and MoWalker2d problems, followed closely by TensorHypE and TensorNSGA-III, which show similar performance. TensorHypE outperforms TensorNSGA-III on MoWalker2d, while TensorNSGA-III performs better on MoHalfcheetah. For EU and final results, TensorRVEA demonstrates high EU scores and strong diversity in MoHalfcheetah and MoHopper. On MoWalker2d, TensorRVEA achieves better HV, while TensorMOEA/D scores higher in EU, indicating a better preference under uniform weights. TensorNSGA-III performs best on MoWalker2d for the first objective, achieving very high speeds. Although TensorMOEA/D does not perform as well as the other algorithms, it still significantly surpasses random search.

As shown in Fig. 5 and Fig. 6, TensorRVEA achieves higher HV on MoPusher and MoReacher, although its EU is lower

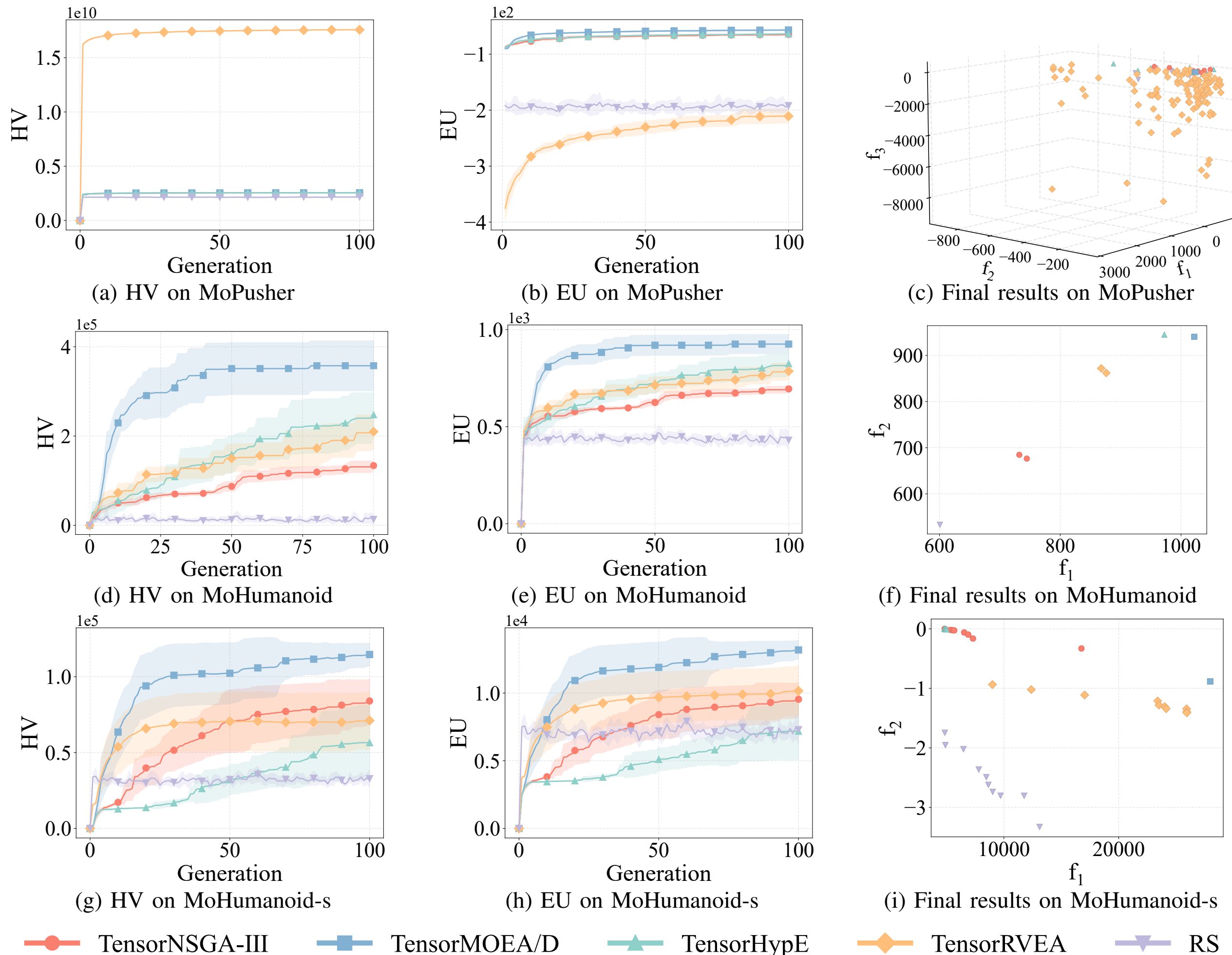


Fig. 5: Comparative performance (HV, EU, and visualization of final results) of TensorNSGA-III, TensorMOEA/D, TensorHypE, TensorRVEA, and random search (RS) across varying problems: MoPusher (503D), MoHumanoid (4209D), and MoHumanoid-s (4209D). *Note:* Higher values for all metrics indicate better performance.

than that of the other algorithms. On MoReacher, solutions from TensorNSGA-III, TensorMOEA/D, and TensorHypE dominate parts of TensorRVEA's solutions, though TensorRVEA still maintains superior HV. TensorMOEA/D performs significantly better on the MoHumanoid and MoHumanoid-s problems, highlighting its effectiveness in handling large-scale problems.

Another important observation is that TensorRVEA and TensorMOEA/D can exhibit an initial rise in HV, followed by a decline before stabilizing, as seen in problems like MoSwimmer in Fig. 6. This phenomenon can be attributed to the weight or reference tensors guiding the population to optimize in specific directions early in the process, leading to premature convergence and a subsequent decline in HV as diversity decreases. By contrast, on the MoIDP problem, TensorRVEA achieves the highest HV, although its EU and visualization results remain comparable to those of the other algorithms.

Overall, decomposition-based algorithms such as TensorRVEA and TensorMOEA/D exhibit superior performance in large-scale multiobjective robot control tasks, particularly in handling high-dimensional decision spaces and maintaining solution diversity.

VII. Conclusion

This paper introduces a tensorization approach to address the computational limitations of traditional CPU-based evolutionary multiobjective optimization (EMO) algorithms, enhancing both speed and scalability. We applied this approach across three representative EMO algorithm classes: dominance-based (NSGA-III), decomposition-based (MOEA/D), and indicator-based (HypE), which demonstrated substantial performance improvements on GPU platforms. Our results confirm that tensorized algorithms can significantly accelerate computations while maintaining solution quality comparable to their original CPU-based counterparts.

To demonstrate the applicability of tensorized EMO algorithms in GPU computing environments, we also developed MoRobtrol, a comprehensive benchmark test suite that reformulates complex multiobjective robot control tasks from the physics simulation environments into multiobjective optimization problems. MoRobtrol underscores the potential of tensorized EMO algorithms to efficiently address the high

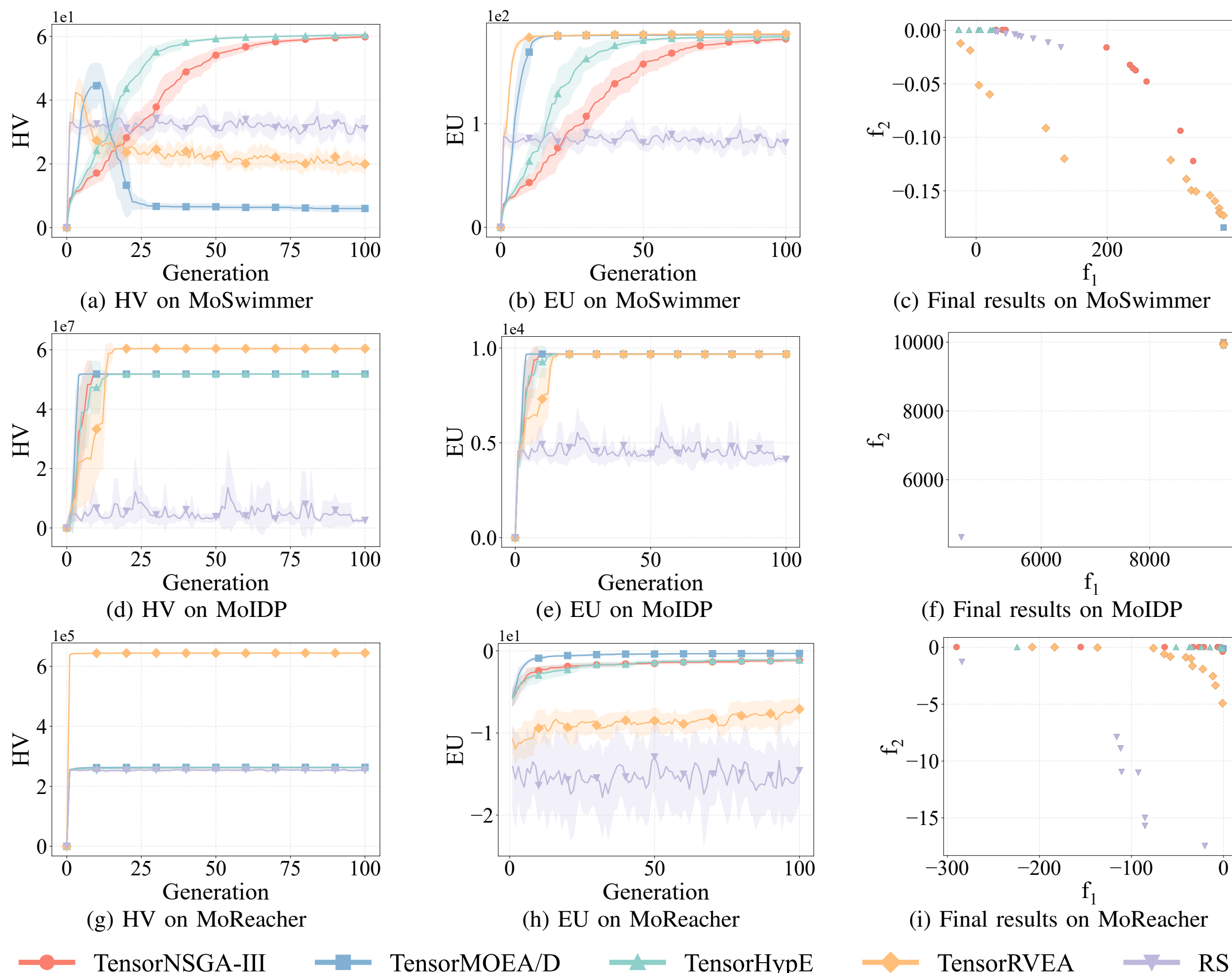


Fig. 6: Comparative performance (HV, EU, and visualization of final results) of TensorNSGA-III, TensorMOEA/D, TensorHypE, TensorRVEA, and random search (RS) across varying problems: MoSwimmer (178D), MoIDP (161D), and MoReacher (226D). *Note:* Higher values for all metrics indicate better performance.

computational demands of Embodied AI, illustrating their relevance to dynamic real-world applications.

While tensorization has substantially improved algorithmic efficiency, opportunities remain to further optimize speed and memory use. Future work will focus on refining key operators such as non-dominated sorting and exploring new tensorized operators optimized for multi-GPU environments to maximize performance. Additionally, leveraging large population data to strengthen search strategies and integrating deep learning techniques may further extend the capabilities of EMO algorithms in tackling large-scale challenges.

References

[1] P. R. Wiecha, A. Arbouet, C. Girard, A. Lecestre, G. Larrieu, and V. Paillard, "Evolutionary multi-objective optimization of colour pixels based on dielectric nanoantennas," *Nature Nanotechnology*, vol. 12, no. 2, pp. 163–169, 2017.

[2] B. Peng, Y. Wei, Y. Qin, J. Dai, Y. Li, A. Liu, Y. Tian, L. Han, Y. Zheng, and P. Wen, "Machine learning-enabled constrained multi-objective design of architected materials," *Nature Communications*, vol. 14, no. 1, p. 6630, 2023.

[3] Y. Cui, Z. Geng, Q. Zhu, and Y. Han, "Multi-objective optimization methods and application in energy saving," *Energy*, vol. 125, pp. 681–704, 2017.

[4] X. Wang, X. Mao, and H. Khodaei, "A multi-objective home energy management system based on internet of things and optimization algorithms," *Journal of Building Engineering*, vol. 33, p. 101603, 2021.

[5] Z. Fei, B. Li, S. Yang, C. Xing, H. Chen, and L. Hanzo, "A survey of multi-objective optimization in wireless sensor networks: Metrics, algorithms, and open problems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 550–586, 2016.

[6] A. Ponsich, A. L. Jaimes, and C. A. C. Coello, "A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 3, pp. 321–344, 2012.

[7] K. Deb, K. Sindhya, and J. Hakanen, "Multi-objective optimization," in *Decision Sciences*. CRC Press, 2016, pp. 161–200.

[8] B. Li, J. Li, K. Tang, and X. Yao, "Many-objective evolutionary algorithms: A survey," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, sep 2015.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[10] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014.

[11] Qingfu Zhang and Hui Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, Dec. 2007.

[12] J. Bader and E. Zitzler, "HypE: An algorithm for fast hypervolume-based

many-objective optimization," *Evolutionary Computation*, vol. 19, no. 1, pp. 45–76, 03 2011.
[13] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.
[14] Y. Tian, L. Si, X. Zhang, R. Cheng, C. He, K. C. Tan, and Y. Jin, "Evolutionary large-scale multi-objective optimization: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, oct 2021.
[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[16] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter, "NVIDIA Tensor Core programmability, performance & precision," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 522–531.
[17] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax - a differentiable physics engine for large scale rigid body simulation," 2021. [Online]. Available: http://github.com/google/brax
[18] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," Comput. Eng. Netw. Lab., Swiss Fed. Inst. Technol., Zürich, Switzerland, Tech. Rep. TIK-103, 2001.
[19] S. Yang, M. Li, X. Liu, and J. Zheng, "A grid-based evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 721–736, 2013.
[20] M. Li, S. Yang, and X. Liu, "Bi-goal evolution for many-objective optimization problems," *Artificial Intelligence*, vol. 228, pp. 45–65, 2015.
[21] Y. Xiang, Y. Zhou, M. Li, and Z. Chen, "A vector angle-based evolutionary algorithm for unconstrained many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 1, pp. 131–152, 2016.
[22] C. He, Y. Tian, Y. Jin, X. Zhang, and L. Pan, "A radial space division based evolutionary algorithm for many-objective optimization," *Applied Soft Computing*, vol. 61, pp. 603–621, 2017.
[23] Y. Tian, R. Cheng, X. Zhang, Y. Su, and Y. Jin, "A strengthened dominance relation considering convergence and diversity for evolutionary many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 2, pp. 331–345, 2018.
[24] Y. Tian, C. He, R. Cheng, and X. Zhang, "A multistage evolutionary algorithm for better diversity preservation in multiobjective optimization," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 9, pp. 5880–5894, 2019.
[25] Y. Liu, N. Zhu, and M. Li, "Solving many-objective optimization problems by a Pareto-based evolutionary algorithm with preprocessing and a penalty mechanism," *IEEE Transactions on Cybernetics*, vol. 51, no. 11, pp. 5585–5594, 2021.
[26] K. Li, "A survey of multi-objective evolutionary algorithm based on decomposition: Past and future," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2024.
[27] Q. Zhang, W. Liu, and H. Li, "The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances," in *2009 IEEE Congress on Evolutionary Computation*, 2009, pp. 203–208.
[28] X. Cai, Y. Li, Z. Fan, and Q. Zhang, "An external archive guided multiobjective evolutionary algorithm based on decomposition for combinatorial optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 4, pp. 508–523, 2015.
[29] Q. Zhao, Y. Guo, X. Yao, and D. Gong, "Decomposition-based multiobjective optimization algorithms with adaptively adjusting weight vectors and neighborhoods," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 5, pp. 1485–1497, 2023.
[30] Q. Wang, Q. Gu, L. Chen, Y. Guo, and N. Xiong, "A MOEA/D with global and local cooperative optimization for complicated bi-objective optimization problems," *Applied Soft Computing*, vol. 137, p. 110162, 2023.
[31] H.-L. Liu, F. Gu, and Q. Zhang, "Decomposition of a multiobjective optimization problem into a number of simple multiobjective subproblems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 450–455, 2014.
[32] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff, "A reference vector guided evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 5, pp. 773–791, 2016.
[33] Y. Yuan, H. Xu, B. Wang, and X. Yao, "A new dominance relation-based evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 16–37, 2016.
[34] J. Zhou, L. Gao, and X. Li, "Ensemble of dynamic resource allocation strategies for decomposition-based multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 710–723, 2021.
[35] J. G. Falcón-Cardona and C. A. C. Coello, "Indicator-based multi-objective evolutionary algorithms: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, Mar. 2020.
[36] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Parallel Problem Solving from Nature - PPSN VIII*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 832–842.
[37] M. Emmerich, N. Beume, and B. Naujoks, "An EMO algorithm using the hypervolume measure as selection criterion," in *Evolutionary Multi-Criterion Optimization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 62–76.
[38] M. Li, S. Yang, and X. Liu, "Pareto or non-Pareto: Bi-criterion evolution in multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 5, pp. 645–665, 2016.
[39] B. Li, K. Tang, J. Li, and X. Yao, "Stochastic ranking algorithm for many-objective optimization based on multiple indicators," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 6, pp. 924–938, 2016.
[40] R. Hernández Gómez and C. A. Coello Coello, "Improved metaheuristic based on the R2 indicator for many-objective optimization," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '15. ACM, 2015, p. 679–686.
[41] Y. Tian, R. Cheng, X. Zhang, F. Cheng, and Y. Jin, "An indicator-based multiobjective evolutionary algorithm with reference point adaptation for better versatility," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 4, pp. 609–622, 2018.
[42] Y. Sun, G. G. Yen, and Z. Yi, "IGD indicator-based evolutionary algorithm for many-objective optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 2, pp. 173–187, 2019.
[43] K. Shang and H. Ishibuchi, "A new hypervolume-based evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 5, pp. 839–852, 2020.
[44] L. M. Pang, H. Ishibuchi, L. He, K. Shang, and L. Chen, "Hypervolume-based cooperative coevolution with two reference points for multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 28, no. 4, pp. 1054–1068, 2024.
[45] Z. Wang, K. Lin, G. Li, and W. Gao, "Multi-objective optimization problem with hardly dominated boundaries: Benchmark, analysis, and indicator-based algorithm," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2024.
[46] M. L. Wong, "Parallel multi-objective evolutionary algorithms on graphics processing units," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, ser. GECCO '09. ACM, 2009, pp. 2515–2522.
[47] D. Sharma and P. Collet, "GPGPU-compatible archive based stochastic ranking evolutionary algorithm (G-ASREA) for multi-objective optimization," in *Parallel Problem Solving from Nature, PPSN XI*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 111–120.
[48] B. Arca, T. Ghisu, and G. A. Trunfio, "GPU-accelerated multi-objective optimization of fuel treatments for mitigating wildfire hazard," *Journal of Computational Science*, vol. 11, pp. 258–268, 2015.
[49] A. Aguilar-Rivera, "A GPU fully vectorized approach to accelerate performance of NSGA-2 based on stochastic non-domination sorting and grid-crowding," *Applied Soft Computing*, vol. 88, p. 106047, 2020.
[50] M. Z. De Souza and A. T. R. Pozo, "A GPU implementation of MOEA/D-ACO for the multiobjective traveling salesman problem," in *2014 Brazilian conference on intelligent systems*. IEEE, 2014, pp. 324–329.
[51] E. M. Lopez, L. M. Antonio, and C. A. Coello Coello, "A GPU-based algorithm for a faster hypervolume contribution computation," in *Evolutionary Multi-Criterion Optimization*. Cham: Springer International Publishing, 2015, pp. 80–94.
[52] M. M. Hussain and N. Fujimoto, "GPU-based parallel multi-objective particle swarm optimization for large swarms and high dimensional problems," *Parallel Computing*, vol. 92, p. 102589, 2020.
[53] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax
[54] Y. Tang, Y. Tian, and D. Ha, "EvoJAX: Hardware-accelerated neuroevolution," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '22. ACM, 2022, p. 308–311.

[55] R. T. Lange, "evosax: JAX-based evolution strategies," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, ser. GECCO '23 Companion. ACM, 2023, p. 659–662.
[56] B. Huang, R. Cheng, Z. Li, Y. Jin, and K. C. Tan, "EvoX: A distributed GPU-accelerated framework for scalable evolutionary computation," *IEEE Transactions on Evolutionary Computation*, 2024.
[57] Z. Liang, T. Jiang, K. Sun, and R. Cheng, "GPU-accelerated evolutionary multiobjective optimization using tensorized RVEA," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '24, 2024, pp. 566–575.
[58] D. Kirk, "NVIDIA CUDA software and GPU parallel computing architecture," ser. ISMM '07. ACM, 2007, p. 103–104.
[59] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.
[60] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2000.
[61] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, *Scalable test problems for evolutionary multiobjective optimization*. London: Springer London, 2005, pp. 105–145.
[62] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 477–506, 2006.
[63] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff, "Test problems for large-scale multiobjective and many-objective optimization," *IEEE Transactions on Cybernetics*, vol. 47, no. 12, pp. 4108–4121, 2017.
[64] R. Cheng, M. Li, Y. Tian, X. Zhang, S. Yang, Y. Jin, and X. Yao, "A benchmark test suite for evolutionary many-objective optimization," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 67–81, Mar. 2017.
[65] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A platform for embodied AI research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.
[66] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.
[67] F. Felten, L. N. Alegre, A. Nowé, A. L. C. Bazzan, E. G. Talbi, G. Danoy, and B. C. da. Silva, "A toolkit for reliable benchmarking and research in multi-objective reinforcement learning," in *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023.
[68] H. Bai, R. Cheng, and Y. Jin, "Evolutionary reinforcement learning: A survey," *Intelligent Computing*, vol. 2, p. 0025, 2023.
[69] N. E. Toklu, T. Atkinson, V. Micka, P. Liskowski, and R. K. Srivastava, "EvoTorch: Scalable evolutionary computation in Python," *arXiv preprint*, 2023, https://arxiv.org/abs/2302.12600.
[70] C. A. C. Coello and N. C. Cortés, "Solving multiobjective optimization problems using an artificial immune system," *Genetic Programming and Evolvable Machines*, vol. 6, pp. 163–190, 2005.
[71] L. M. Zintgraf, T. V. Kanters, D. M. Roijers, F. Oliehoek, and P. Beau, "Quality assessment of MORL algorithms: A utility-based approach," in *Benelearn 2015: Proceedings of the 24th Annual Machine Learning Conference of Belgium and the Netherlands*, 2015.

Zhenyu Liang received the B.S. degree in Computer Science and Technology from Southern University of Science and Technology, Shenzhen, China, in 2020, where he is currently pursuing an M.S. degree in Electronic Science and Engineering. His main research interests are evolutionary computation and multi-objective optimization. He serves as the Social Media Chair of IEEE Computational Intelligence Society Shenzhen Chapter.

Hao Li received the B.S. degree in Computer Science and Technology from the Southern University of Science and Technology, Shenzhen, China, in 2021, where he is currently pursuing the M.S. degree in Electronic Science and Engineering. His primary research focus lies in developing GPU-accelerated multi-objective evolutionary algorithms.

Naiwei Yu received the B.S. degree in Computer Science and Technology from the Southern University of Science and Technology, Shenzhen, China, in 2024, where he is currently pursuing the M.S. degree in Electronic Information. His research interests include artificial intelligence and its applications in robotic control.

Kebin Sun received his B.S. degree in Computer Science and Technology from Southern University of Science and Technology, Shenzhen, China, in 2019, where he is currently pursuing an M.S. degree in Electronic Science and Engineering. From 2019 to 2021, he worked as a research assistant with the Institute of Quantum Science and Engineering, Shenzhen, China. His research interests include black-box optimization and machine learning, particularly in generative learning, and evolutionary computation.

Ran Cheng (Senior Member, IEEE) received the B.Sc. degree from the Northeastern University, Shenyang, China, in 2010, and the Ph.D. degree from the University of Surrey, Guildford, U.K., in 2016. He is currently an Associate Professor with the Department of Data Science and Artificial Intelligence, and the Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China. He is a recipient of the IEEE Transactions on Evolutionary Computation Outstanding Paper Award (2018 and 2021), the IEEE Computational Intelligence Society Outstanding Ph.D. Dissertation Award (2019), the IEEE Computational Intelligence Magazine Outstanding Paper Award (2020), and the IEEE Computational Intelligence Society Early Career Award (2025). He is the Founding Chair of the IEEE Computational Intelligence Society Shenzhen Chapter. He is an Associate Editor of IEEE Transactions on Evolutionary Computation, IEEE Transactions on Artificial Intelligence, IEEE Transactions on Emerging Topics in Computational Intelligence, and IEEE Transactions on Cognitive and Developmental Systems.