Techniques for Accelerating Multi-Objective Evolutionary Algorithms in PlatEMO

1st Ye Tian
Institutes of Physical Science and Information Technology, Anhui University,
Hefei, China
field910921@gmail.com

2nd Ran Cheng
Department of Computer Science and Engineering, Southern University of Science and Technology,
Shenzhen, China
ranchengcn@gmail.com

3rd Xingyi Zhang
School of Computer Science and Technology, Anhui University,
Hefei, China
xyzhanghust@gmail.com

4th Yaochu Jin
Department of Computer Science, University of Surrey,
Guildford, United Kingdom
yaochu.jin@surrey.ac.uk

Abstract—**It has been widely recognized that evolutionary computation is one of the most effective techniques for solving complex optimization problems. As a group of meta-heuristics inspired by nature, the superiority of evolutionary algorithms is mainly attributed to the evolution of multiple candidate solutions, which can strike a balance between exploration and exploitation. However, the effectiveness of evolutionary algorithms is generally at the expense of efficiency, which reduces the prevalence of evolutionary algorithms in solving real-world optimization problems. In 2017, we proposed the evolutionary multi-objective optimization platform PlatEMO to facilitate the use of multi-objective evolutionary algorithms (MOEAs), where some delicate techniques were developed to improve the computational efficiency of MOEAs. These techniques have not been introduced before, since users need not care about them when using existing MOEAs or developing new MOEAs. To deepen the understanding of the core mechanisms of PlatEMO, this paper gives a comprehensive introduction to these techniques, including new non-dominated sorting approaches, matrix calculation, and parallel computing. Several comparative experiments are conducted for a quantitative understanding of the efficiency improvement brought by these techniques.**

I. Introduction

Evolutionary algorithms broadly refer to a variety of metaheuristics inspired by the biological evolution mechanisms and swarm behaviors in nature, such as genetic algorithms [1], particle swarm optimization [2], and differential evolution [3]. These algorithms have been widely adopted in many scientific and engineering areas [4], [5], since they are versatile for solving different types of optimization problems. The success of evolutionary algorithms is mainly owing to the evolution of multiple candidate solutions (i.e., a population), which enables the algorithm to find optimal or sub-optimal solutions without any a priori knowledge about the problem. On the other hand, the computational complexity is considerably increased since each solution should be updated and reevaluated at each generation; in particular, some evolutionary algorithms even update each solution more than once [6], [7].

Therefore, evolutionary algorithms are often criticized for the low efficiency [8], and it becomes even more serious for multi-objective evolutionary algorithms (MOEAs) [9]. On the one hand, MOEAs have to evaluate each solution by calculating multiple objective functions rather than a single one. On the other hand, the selection operator of single-objective evolutionary algorithms can be achieved by directly comparing the objective values of solutions, whereas complex selection strategies should be designed in MOEAs for striking a balance between convergence and diversity, such as the non-dominated sorting in NSGA-II [10], the truncation method in SPEA2 [11], and the hypervolume based selection in HypE [12].

To address this issue, some work has been dedicated to improving the efficiency as well as the practicality of MOEAs from various aspects. For Pareto dominance based MOEAs whose computational resource is mainly consumed by determining the Pareto dominance relations between solutions, a number of non-dominated sorting approaches have been proposed to improve the efficiency of the sorting procedure. The time complexity of the original non-dominated sorting procedure is $O(mn^3)$ with n denoting the population size and m denoting the number of objectives [13], which is then reduced to $O(mn^2)$ by fast non-dominated sort [10] and $O(n \ln^{m-1} n)$ by Jensen's Sort [9]. Later, the time complexity is further reduced by a series of non-dominated sorting approaches [14]. For decomposition based MOEAs that divide the original problem into a number of simple subproblems to be solved simultaneously, parallelization scheme has been

This work was supported in part by the Key Project of Science and Technology Innovation 2030 supported by the Ministry of Science and Technology of China under Grant 2018AAA0100105, in part by the National Natural Science Foundation of China under Grant 61672033, 61822301, 61876123, 61906001, and U1804262, in part by the Hong Kong Scholars Program under Grant XJ2019035, in part by the Anhui Provincial Natural Science Foundation under Grant 1808085J06 and 1908085QF271, and in part by the State Key Laboratory of Synthetical Automation for Process Industries under Grant PAL-N201805.

adopted to speed up the optimization of subproblems, such as the MOEA deployed on multiple CPUs [15] and the MOEA deployed on GPU [16]. For indicator based MOEAs that must repeatedly calculate the performance indicator, some efforts have been made to accelerate the calculation of hypervolume, which is an effective and popular performance indicator but has a super-polynomial time complexity [17]. For instance, HypE [12] suggests a hypervolume estimation algorithm by using Monte Carlo simulation, FV-MOEA [18] accelerates the calculation of the hypervolume contribution of each solution by ignoring irrelevant solutions, and R2HCA-EMOA [19] approximates the hypervolume contribution of each solution by using an R2 indicator.

In 2017, an evolutionary multi-objective optimization platform was proposed by us, called PlatEMO[1] [20], which contains more than 100 open-source MOEAs and more than 200 open-source benchmark problems. More importantly, it provides a powerful GUI enabling users to implement comparative experiments over multiple MOEAs and multiple benchmark problems by one-click operation, where the statistical results can be directly saved as an Excel table or LaTeX table. PlatEMO also uses some delicate techniques to improve the computational efficiency of MOEAs, including non-dominated sorting, matrix calculation, and parallel computing. These techniques were embedded in public functions, which can be invoked by users to develop new MOEAs without understanding the implementation details of them. In this paper, we would like to give a comprehensive introduction to these techniques, hoping to help users deepen the understanding of the core mechanisms of PlatEMO and to provide the users inspirations for developing new MOEAs. To illustrate the benefits of these techniques, we conduct several experiments to quantitatively show the efficiency improvement brought by these techniques.

The rest of this paper is organized as follows. Section II introduces three non-dominated sorting approaches used in PlatEMO, and verifies their efficiency by comparing them to other non-dominated sorting approaches. Section III introduces the methods of using matrix calculation in MOEAs, and compares the runtimes of MOEAs with and without matrix calculation. Section IV introduces the way to deploy MOEAs on multiple CPUs in PlatEMO. Section V draws the conclusions.

II. Computationally Efficient Non-Dominated Sorting Techniques

Since the solutions for multi-objective optimization problems (MOPs) have multiple objective values rather than a single one, the Pareto dominance relation is used to compare the quality of two solutions. For minimization MOPs, solution $\mathbf{x}$ is said to Pareto dominate solution $\mathbf{y}$ (denoted by $\mathbf{x} \prec \mathbf{y}$) if and only if

$$\begin{cases} \forall i \in \{1,\dots,m\}: & f_i(\mathbf{x}) \le f_i(\mathbf{y}) \\ \exists j \in \{1,\dots,m\}: & f_j(\mathbf{x}) < f_j(\mathbf{y}) \end{cases}, \tag{1}$$

where $f_i(\mathbf{x})$ denotes the i-th objective value of $\mathbf{x}$ and $m \ge 2$ denotes the total number of objectives. In particular, a solution is called non-dominated if it is not dominated by any other solutions in a population. Based on the Pareto dominance relation, the idea of non-dominated sorting was suggested as the selection strategy [21] and implemented in NSGA [13] for the first time. Specifically, non-dominated sorting performs the following three steps to divide a population P into several subsets:

- *Step 1*: $i = 1$.
- *Step 2*: Find all the non-dominated solutions from P and move them to the non-dominated front F_i.
- *Step 3*: $i = i + 1$; return to *Step 2* until P is empty.

Obviously, the solutions in front F_i are better than those in front F_j for $i < j$, as each solution in F_j must be dominated by at least one solution in F_i. Since the NSGA-II [10] was proposed in 2002, a large number of MOEAs have adopted non-dominated sorting as the basic selection strategy in environmental selection, which are not limited to Pareto dominance based MOEAs [22]–[24] but also include decomposition based MOEAs [25], [26] and indicator based MOEAs [12], [27].

However, non-dominated sorting is relatively time-consuming since the dominance relation between each pair of solutions should be determined by (1). Therefore, a series of non-dominated sorting approaches have been proposed to increase the efficiency of non-dominated sorting [14]. PlatEMO adopts three non-dominated sorting approaches to be used in different cases, including efficient non-dominated sort with sequential search (ENS-SS) [28], tree-based efficient non-dominated sort (T-ENS) [29], and efficient non-dominated level update (ENLU) [30]. In what follows, the three non-dominated sorting approaches are briefly reviewed and verified by comparative experiments.

A. Efficient Non-Dominated Sort (ENS)

In contrast to the conventional non-dominated sorting procedure that sorts the solutions front by front, ENS [28] determines the front number of each solution in sequence, while the sorting results of the two procedures are the same. To be specific, ENS-SS performs the following three steps to divide a population P into several subsets:

- *Step 1*: Sort P in an ascending order of the first objective.
- *Step 2*: $i = 1$; pick up the first solution $\mathbf{x}$ from P.
- *Step 3*: If $\mathbf{x}$ is dominated by any solution in F_i, $i = i+1$ and repeats *Step 3*; otherwise move $\mathbf{x}$ to F_i and return to *Step 2* until P is empty.

[1]PlatEMO 2.4: https://github.com/BIMK/PlatEMO

The core idea of ENS is to presort the population according to the first objective in *Step 1*, so that any solution $\mathbf{x}$ will not be dominated by the solutions after it in the ascending order, and the front number of $\mathbf{x}$ can be determined without checking the solutions after it. Hence, each solution needs to be compared with some solutions before it rather than all the solutions, and a large number of comparisons can be saved by ENS-SS. In short, ENS-SS is much efficient than fast non-dominated sort as evidenced by the empirical comparisons in [28].

B. Tree-Based Efficient Non-Dominated Sort (T-ENS)

As reported in [28], [29], the efficiency of ENS-SS considerably deteriorates as the number of objectives increases. This is because most solutions become mutually non-dominated in high-dimensional objective space [14], and the comparison between each pair of non-dominated solutions cannot be saved by ENS-SS. While most non-dominated sorting approaches including ENS-SS store the solutions in each non-dominated front by an array, a novel tree structure is designed in T-ENS [29] to address the curse of dimensionality, which enables the sorting approach to save many comparisons between non-dominated solutions. That is, ENS-SS can save comparisons by deducing $\mathbf{x} \prec \mathbf{z}$ from $\mathbf{x} \prec \mathbf{y}$ and $\mathbf{y} \prec \mathbf{z}$ owing to the transitivity of Pareto dominance, while T-ENS can also deduce $\mathbf{x} \nprec \mathbf{z}$ from $\mathbf{x} \nprec \mathbf{y}$ and $\mathbf{y} \nprec \mathbf{z}$ in some cases. As a result, T-ENS is more efficient than ENS-SS when the number of objectives is high.

C. Efficient Non-Dominated Level Update (ENLU)

There exist some MOEAs based on the steady-state evolution model, which generate a single offspring and update the population each time [31]. Assuming that the population size is n and the number of generations is g, the non-dominated sorting should be performed g times in general MOEAs, while it should be performed $n \times g$ times in steady-state MOEAs. As a result, steady-state MOEAs are generally less efficient than general MOEAs with the same number of function evaluations. To address this issue, ENLU [30] is tailored for steady-state MOEAs, which performs non-dominated sorting by taking advantage of the historical sorting result. More specifically, when an offspring is added to or deleted from the population, ENLU updates the non-dominated fronts rather than sorts the new population from scratch.

D. Empirical Studies

To illustrate the superiority of the above three non-dominated sorting approaches, several experiments are conducted to compare them to other existing non-dominated sorting approaches. Firstly, Fig. 1 depicts the runtime of best order sort (BOS) [32], corner sort [33], deductive sort [34], ENS-SS, fast non-dominated sort [10], and T-ENS on random populations with 2–4 objectives and 100–500 solutions, averaged over 20 runs. It is obvious that ENS-SS outperforms the other approaches on all the populations. By contrast, as shown in Fig. 2, T-ENS becomes more efficient than ENS-SS when the number of solutions and the number of objectives increase. One the other hand, Fig. 3 plots the runtime of ENS-SS, fast non-dominated sort, T-ENS, and ENLU on random populations with 2–10 objectives. To mimic the steady-state evolutionary process, an initial population with 100 solutions is randomly generated, then it is iteratively extended with a single solution until its size reaches 200. According to the results shown in Fig. 3, it can be found that ENLU is much more efficient than ENS-SS and T-ENS.

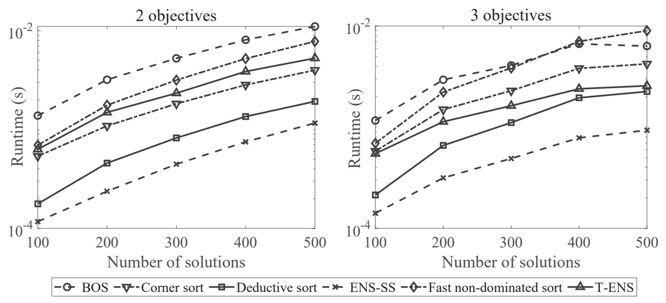


Fig. 1. Runtime (in second) of BOS, corner sort, deductive sort, ENS-SS, fast non-dominated sort, and T-ENS on random populations with 2–3 objectives and 100–500 solutions.

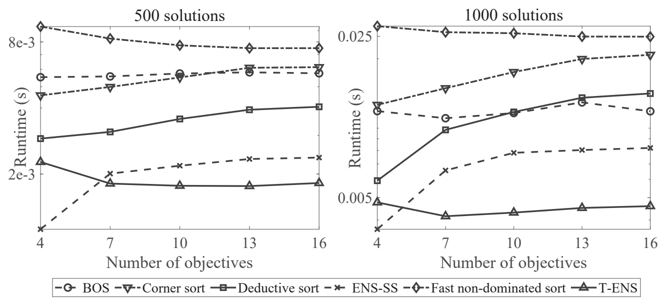


Fig. 2. Runtime (in second) of BOS, corner sort, deductive sort, ENS-SS, fast non-dominated sort, and T-ENS on random populations with 500–1000 solutions and 4–16 objectives.

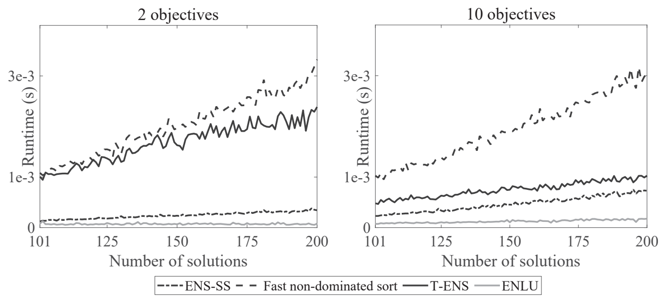


Fig. 3. Runtime (in second) of ENS-SS, fast non-dominated sort, T-ENS, and ENLU on random populations with 2–10 objectives and 100–200 solutions. Note that the population is increased by new solutions steadily.

TABLE I
RUNTIME (IN SECOND) OF NSGA-III EMBEDDED WITH BOS, CORNER SORT, DEDUCTIVE SORT, ENS-SS, FAST NON-DOMINATED SORT, AND T-ENS ON DTLZ2 WITH DIFFERENT NUMBER OF OBJECTIVES m AND POPULATION SIZE n. BEST RESULT IN EACH ROW IS HIGHLIGHTED.

Number of objectives m, population size n	NSGA-III embedded with					
	BOS	Corner sort	Deductive sort	ENS-SS	Fast non-dominated sort	T-ENS
$m = 2, n = 100$	1.3342	1.2383	1.1442	1.0901	1.2392	1.2143
$m = 2, n = 500$	1.5586	1.4419	1.3019	1.1114	1.8105	1.3777
$m = 3, n = 100$	1.5822	1.5111	1.3977	1.3537	1.5071	1.3874
$m = 3, n = 500$	2.4367	2.5265	2.3762	2.1926	2.8630	2.1576
$m = 5, n = 100$	1.9484	1.7922	1.6631	1.5907	1.8241	1.6057
$m = 5, n = 500$	3.1479	3.3340	3.1284	2.9138	3.6098	2.7695
$m = 10, n = 100$	1.9484	1.7922	1.6631	1.5907	1.8241	1.6057
$m = 10, n = 500$	5.0384	5.3235	5.0838	4.8232	5.6854	4.6039

TABLE II
RUNTIME (IN SECOND) OF MSEA EMBEDDED WITH ENS-SS, FAST NON-DOMINATED SORT, T-ENS, AND ENLU ON DTLZ7 WITH DIFFERENT NUMBER OF OBJECTIVES m AND POPULATION SIZE n. BEST RESULT IN EACH ROW IS HIGHLIGHTED.

Number of objectives m, population size n	MSEA embedded with			
	ENS-SS	Fast non-dominated sort	T-ENS	ENLU
$m = 2, n = 100$	12.0792	19.1943	17.0460	11.7577
$m = 2, n = 200$	15.4912	22.4370	18.1422	13.1762
$m = 5, n = 100$	16.3255	22.2902	19.1006	14.0555
$m = 5, n = 200$	20.7603	48.0474	35.7418	19.5126
$m = 10, n = 100$	29.1788	59.0508	33.4465	21.4377
$m = 10, n = 200$	32.4758	56.3583	36.3406	23.9769

Secondly, the efficiency of the three non-dominated sorting approaches is investigated by embedding them into representative MOEAs. Table I lists the runtime of NSGA-III [35] embedded with six non-dominated sorting approaches on DTLZ2 [36], where the number of function evaluations is set to 20000, the population size is set to 100 and 500, and the other parameter settings are the same to those in [35]. The experiment uses NSGA-III rather than NSGA-II since the former can better solve the MOPs with many objectives. To make the number of reference points consistent with the population size, the mixture uniform design [37] is adopted to generate reference points instead of the original sampling method of NSGA-III. As can be observed from Table I, ENS-SS is the most efficient approach in most cases, while T-ENS becomes more efficient than ENS-SS when the number of objectives is more than 2 and the population size is 500. Moreover, Table II presents the runtime of MSEA [38] embedded with four non-dominated sorting approaches on DTLZ7 [36], where MSEA is a recently proposed steady-state MOEA having excellent diversity performance. The number of function evaluations is set to 20000 and the population size is set to 100 and 200. It can be found from Table II that the MSEA with ENLU consumes less runtime than MSEA with the other non-dominated sorting approaches.

As a consequence, the above experiments indicate that ENS-SS has the best efficiency when the number of objectives or the population size is small, T-ENS has the best efficiency when the number of objectives and the population size are large, and ENLU has the best efficiency for the steady-state evolutionary process. In PlatEMO, a function `NDSort.m` is provided for the non-dominated sorting of 77 MOEAs, where the sorting approach to be used is automatically determined based on the above experimental results. More specifically, ENS-SS is used when the number of objectives is 2 or the population size is smaller than 500, and T-ENS is used otherwise. Besides, a function `UpdateFront.m` is provided for the non-dominated sorting of four steady-state MOEAs, where ENLU is used.

III. MATRIX CALCULATION FOR EFFICIENT GENETIC OPERATORS AND FUNCTION EVALUATIONS

Since matrix calculation can be accelerated by some programming languages (e.g., MATLAB) and hardware devices (e.g., GPU), it is desirable to improve the efficiency of MOEAs by matrix calculation. For example, the decision variables of all the solutions in a population can be represented by a matrix, where each row denotes a solution and each column denotes a decision variable. Thus, the generation of offsprings can be accelerated by performing genetic operators on the matrix, and the function evaluation can be accelerated by calculating the objective values of the matrix.

PlatEMO fits well with matrix calculation since it is fully developed in MATLAB, which provides various methods to accelerate matrix calculation. In the rest of this section, we introduce the three methods for accelerating matrix calculation in PlatEMO, including matrix operators, matrix functions, and GPU acceleration.

A. Matrix Operators

All the operators in PlatEMO are in fact conducted based on matrix calculation, including genetic operators (for real encoding [39], [40], binary coding [41], and permutation based encoding [42], [43]), particle swarm optimization [44] and its variants [45], [46], differential evolution [47], covariance matrix adaptation evolution strategy [48], and so on. In particular, the simulated binary crossover (SBX) [39] includes complicated formulas and procedures, which is taken as an example to illustrate the way to accelerate all its steps by matrix operators.

Given two parents $\mathbf{x}_1 = (x_1^1, \ldots, x_1^d)$, $\mathbf{x}_2 = (x_2^1, \ldots, x_2^d)$ and the crossover probability p, SBX performs the following steps to generate two offsprings $\mathbf{o}_1 = (o_1^1, \ldots, o_1^d)$,

$\mathbf{o}_2 = (o_2^1, \ldots, o_2^d)$, where $rand$ denotes a random number within $[0, 1]$:

- *Step 1*: If $rand > p$, let $\mathbf{o}_1 = \mathbf{x}_1$ and $\mathbf{o}_2 = \mathbf{x}_2$; otherwise go to *Step 2*.
- *Step 2*: For each dimension i, if $rand > 0.5$, let $o_1^i = x_1^i$ and $o_2^i = x_2^i$; otherwise go to *Step 3*.
- *Step 3*: Calculate o_1^i and o_2^i by

$$\begin{cases} o_1^i = [(1+\beta)x_1^i + (1-\beta)x_2^i]/2 \\ o_2^i = [(1-\beta)x_1^i + (1+\beta)x_2^i]/2 \end{cases}, \tag{2}$$

where

$$\beta = \begin{cases} (2\mu)^{\frac{1}{\eta+1}} & , \ \mu \leq 0.5 \\ (2-2\mu)^{-\frac{1}{\eta+1}} & , \ \mu > 0.5 \end{cases}, \tag{3}$$

μ is a random number within $[0, 1]$, and η is a parameter controlling the distribution of the offsprings.
- *Step 4*: If $rand > 0.5$, exchange o_1^i and o_2^i; return to *Step 2* until all the variables of the offsprings are generated.

Obviously, the procedure of SBX is quite complex that it contains two `for-end` blocks and three `if-else` blocks for generating $2n$ offsprings.

In order to improve the efficiency of SBX, PlatEMO uses four matrices X_1, X_2, O_1, and O_2 to represent the parents and offsprings, where each matrix has a size of $n \times d$ with each row denoting a solution and each column denoting a decision variable. Generally, it is not difficult to calculate O_1, O_2 based on X_1, X_2 by (2) and (3), which can eliminate all the `for-end` blocks. But, it is not intuitive to eliminate the `if-else` blocks by matrix operators. Taking a closer look at (2), it can be found that the decision variables of the offsprings are mainly controlled by the value of β. More specifically, the following assignments of β can lead to some special results:

- $\beta = 1$ will make $o_1^i = x_1^i$ and $o_2^i = x_2^i$;
- $\beta = -\beta$ will make $o_1^i = o_2^i$ and $o_2^i = o_1^i$.

Therefore, we can set $\beta = 1$ to prevent the crossover and set $\beta = -\beta$ to exchange the decision variables of the two offsprings. As a consequence, PlatEMO performs the following matrix calculations to achieve the same function to SBX, where all the involved matrix operators can be directly used in MATLAB:

- *Step 1*: Generate four random matrices $M \in [0, 1]^{n\times d}$, $R_1 \in [0, 1]^{n\times d}$, $R_2 \in [0, 1]^{n\times d}$, and $R_3 \in [0, 1]^{n\times 1}$.
- *Step 2*: Perform the following matrix calculations:

$$T_1 = M \leq 0.5, \quad ^2 \tag{4}$$

$$T_2 = \text{sign}(R_1 - 0.5), \quad ^3 \tag{5}$$

$$T_3 = R_2 > 0.5, \tag{6}$$

$$T_4 = \text{repmat}(R_3 > p, d), \quad ^4 \tag{7}$$

$$B_1 = (2 \times M)^{\frac{1}{\eta+1}}, \quad ^5 \tag{8}$$

$$B_2 = (2 - 2 \times M)^{-\frac{1}{\eta+1}}, \tag{9}$$

$$B = T_1 \cdot B_1 + (1 - T_1) \cdot B_2, \quad ^6 \tag{10}$$

$$B = T_2 \cdot B, \tag{11}$$

$$B = (1 - T_3) \cdot B + T_3, \tag{12}$$

$$B = (1 - T_4) \cdot B + T_4, \tag{13}$$

$$O_1 = [(1 + B) \cdot X_1 + (1 - B) \cdot X_2]/2, \tag{14}$$

$$O_2 = [(1 - B) \cdot X_1 + (1 + B) \cdot X_2]/2. \tag{15}$$

By doing so, the SBX only needs to perform several matrix calculations without any `for-end` block or `if-else` block, where (10) corresponds to (3), (11) corresponds to *Step 4*, (12) corresponds to *Step 2*, (13) corresponds to *Step 1*, and (14)(15) corresponds to (2). In PlatEMO, the above steps are implemented in the function `GA.m`.

B. Matrix Functions

The above formulas adopt not only matrix operators but also some matrix functions provided by MATLAB (i.e., `sign()` and `repmat()`). When calculating the objective values of a population containing n solutions, PlatEMO also adopt several matrix functions to improve the efficiency. Taking the DTLZ2 [36] problem as an example, the objective values of a solution $\mathbf{x}_1 = (x^1, \ldots, x^d)$ are calculated by

$$\begin{cases} f_1(\mathbf{x}) = (1 + g(\mathbf{x}))\cos\left(\frac{\pi}{2}x^1\right)\ldots\cos\left(\frac{\pi}{2}x^{m-2}\right)\cos\left(\frac{\pi}{2}x^{m-1}\right) \\ f_2(\mathbf{x}) = (1 + g(\mathbf{x}))\cos\left(\frac{\pi}{2}x^1\right)\ldots\cos\left(\frac{\pi}{2}x^{m-2}\right)\sin\left(\frac{\pi}{2}x^{m-1}\right) \\ f_3(\mathbf{x}) = (1 + g(\mathbf{x}))\cos\left(\frac{\pi}{2}x^1\right)\ldots\sin\left(\frac{\pi}{2}x^{m-2}\right) \\ \quad \ldots \\ f_m(\mathbf{x}) = (1 + g(\mathbf{x}))\sin\left(\frac{\pi}{2}x^1\right) \end{cases}, \tag{16}$$

where

$$g(\mathbf{x}) = \sum_{i=m}^{d}(x^i - 0.5)^2. \tag{17}$$

Obviously, it is also not intuitive to replace the variables in the above formulas by matrices, since each objective function is related to different decision variables. Fortunately, the goal can be achieved by taking advantages of the matrix functions provided by MATLAB. More

[2] $A < 0.5$ returns a Boolean matrix, in which the value 1 denotes the corresponding element in A is smaller than 0.5 and 0 otherwise.

[3] sign(A) returns a symbol matrix, in which the value 1 denote the corresponding element in A is larger than 0 and -1 otherwise.

[4] repmat(A, d) returns a replicate matrix, in which A is repeated for d times along the column.

[5] The power operator works on each element rather than the whole matrix.

[6] The operator $\cdot$ denotes the element-wise multiplication.

specifically, PlatEMO uses a matrix X with size $n \times d$ to represent the decision variables of all the solutions in the population, and calculates the objective values on DTLZ2 by

$$G = \mathrm{sum}((X^{m\ldots d} - 0.5)^2), \quad ^{7} \tag{18}$$

$$\begin{aligned} F = {} & \mathrm{repmat}(1 + G, m) \cdot \\ & \mathrm{flip}(\mathrm{cumprod}(\mathrm{cat}(I^n, \cos(\frac{\pi}{2} \cdot X^{1\ldots m-1})))) \cdot \\ & \mathrm{cat}(I^n, \sin(\frac{\pi}{2} \cdot X^{m-1\ldots 1})). \quad ^{8} \end{aligned} \tag{19}$$

The obtained matrix F stores all the objective values of X, where each row denotes a solution and each column denotes an objective value. It can be seen that (18) corresponds to (17) and (19) corresponds to (16). In PlatEMO, the above steps are implemented in the function `DTLZ2.m`.

C. GPU Acceleration

The above matrix operators and matrix functions can also be accelerated by GPU, especially when the population size or the number of decision variables is large, e.g., when solving large-scale MOPs [49]. In MATLAB, all the matrices can be sent to GPU by the function `gpuArray()` and got back from GPU by the function `gather()`, while the matrix calculations on GPU uses the same matrix operators and matrix functions, i.e., the MATLAB codes do not need to be modified at all. However, it should be noted that the communication with GPU is also time-consuming, which can even decrease the efficiency if we send the matrices to GPU, perform matrix calculations on GPU, and get the matrices back from GPU. Instead, we should directly generate all the initial matrices (e.g., the decision variables of the initial population and the random matrices used in SBX) on GPU by the function `gpuArray.rand()` and make all the matrices stored in GPU all the time, thus saving the time consumed by the communication with GPU.

D. Empirical Studies

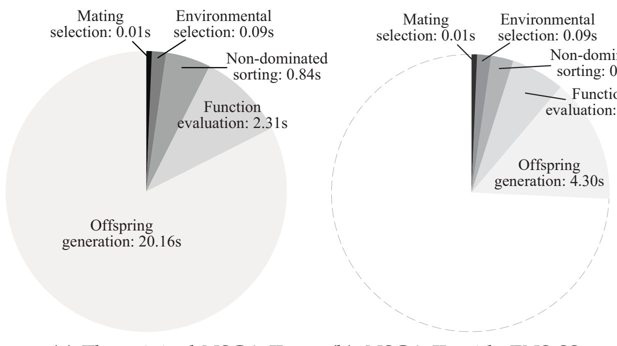
To verify the efficiency of matrix calculation based MOEA, Table III gives the runtime of NSGA-II on DTLZ2, where the number of function evaluations is set to 20000, the number of objectives is set to 3, the population size is set to 100 and 500, and the number of decision variables is set to 100, 500, 1000, and 5000. It can be seen from the table that the NSGA-II with matrix operators is more efficient than the original NSGA-II, while the efficiency can be further improved by matrix functions.

[7] $A^{m\ldots d}$ returns a matrix containing the m-th to the d-th columns of A, and $\mathrm{sum}(A)$ returns a column vector containing the sum of each row of A.

[8] I^n denotes a $n \times 1$ column vector with all elements being 1, $\mathrm{cat}(A, B)$ returns a matrix with B being concatenated to the last column of A, $\mathrm{cumprod}(A)$ returns a matrix containing the cumulative product of each row of A, and $\mathrm{flip}(A)$ returns A with its columns flipped in the left-right direction.

TABLE III
RUNTIME (IN SECOND) OF NSGA-II USING DIFFERENT METHODS FOR ACCELERATING MATRIX CALCULATION ON DTLZ2 WITH DIFFERENT NUMBER OF DECISION VARIABLES d AND POPULATION SIZE n. BEST RESULT IN EACH ROW IS HIGHLIGHTED.

Number of variables d, population size n	NSGA-II with			
	Original version	Matrix operators	Matrix operators + Matrix functions	Matrix operators + Matrix functions + GPU
$d = 100, n = 100$	1.1422	0.9440	0.8952	3.8038
$d = 100, n = 500$	0.8527	0.6797	0.6396	3.1963
$d = 500, n = 100$	2.8561	1.8656	1.8094	4.0788
$d = 500, n = 500$	2.3751	1.6227	1.6101	3.2507
$d = 1000, n = 100$	4.7466	2.8531	2.8035	4.4720
$d = 1000, n = 500$	4.4219	3.0301	3.0040	3.5709
$d = 5000, n = 100$	23.1272	13.8115	13.9500	6.6578
$d = 5000, n = 500$	22.7842	15.1807	14.9286	5.9857



(a) The original NSGA-II

(b) NSGA-II with ENS-SS, matrix operators, matrix functions, and GPU acceleration

Fig. 4. Runtime (in second) of each component of the original NSGA-II and the accelerated NSGA-II on DTLZ2, where the number of decision variables is 5000 and the population size is 500.

As for the NSGA-II based on GPU acceleration, it has the best efficiency when the number of decision variables is large. To summarize, the experiment indicates that the matrix operators and matrix functions can considerably improve the efficiency of MOEAs, and the efficiency can be further improved by GPU when solving large-scale MOPs.

For a comprehensive study, Fig. 4 compares the runtimes of the original NSGA-II and the NSGA-II accelerated by ENS-SS (i.e., non-dominated sorting), matrix operators (i.e., offspring generation), matrix operators (i.e., function evaluation), and GPU, where the number of function evaluations, the number of objectives, the population size, and the number of decision variables are set to 20000, 3, 500, and 5000, respectively. It is obvious that the runtime of the accelerated NSGA-II is approximately a quarter of the original version. As a consequence, the effectiveness of these techniques used

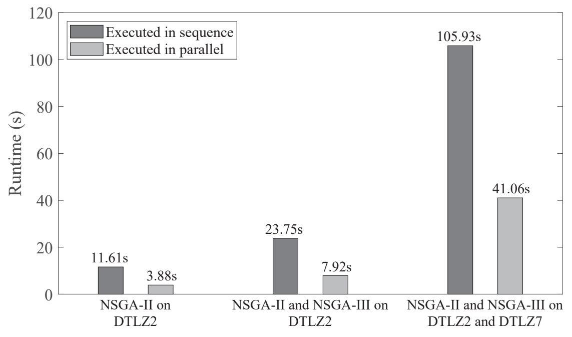


Fig. 5. Runtime (in second) of executing NSGA-II and NSGA-III on DTLZ2 and DTLZ7 for 20 times in sequence and in parallel.

in PlatEMO can be confirmed.

IV. Parallelization of the Execution of MOEAs

When comparing the performance of several MOEAs, it needs to execute each MOEA for multiple independent runs to eliminate the effect of random initialization and perform statistical analysis. To improve the efficiency of the experimental comparison between MOEAs, PlatEMO uses parallel computing to accelerate such a repetitive execution. More specifically, the `parfor-end` block is used to execute an MOEA for multiple runs instead of the `for-end` block, which can execute the loop in parallel by using all the CPUs in the computer.

To illustrate the efficiency improved by parallel computing, Fig. 5 depicts the runtime of executing NSGA-II and NSGA-III on DTLZ2 and DTLZ7 for 20 times. It can be found that the experiments performed in parallel are much more efficient than the experiments performed in sequence, where the parallel computing can reduce the runtime to a third. It is worth to note that the parallel computing is supported by four CPUs here, while the runtime cannot be reduced to a quarter since some runtime is consumed by the communication between CPUs.

In PlatEMO, the experimental module provided by the GUI can automatically perform the experiments in parallel. In other words, users can achieve parallel computing by one-click operation without writing any code.

V. Conclusions

Evolutionary algorithms have demonstrated high effectiveness in solving complex optimization problems, but they are often criticized for the low efficiency. As an evolutionary multi-objective optimization platform, PlatEMO provides more than 100 MOEAs and improves their efficiency by various techniques. This paper has given a detailed introduction to these techniques, including three non-dominated sorting approaches, matrix operators, matrix functions, GPU acceleration, and parallel computing. Moreover, several experiments have been conducted to quantitatively show the efficiency improved by these techniques.

The techniques provided by programming languages and hardware devices do improve the computational efficiency of MOEAs to some extent, while novel search strategies should be developed to fundamentally improve the efficiency of MOEAs in solving real-world MOPs, especially those with a large number of decision variables [50]. Although some variable decomposition based MOEAs have been tailored for large-scale MOPs [49], [51], they require a large number of function evaluations that highly increase the computational complexity. Therefore, some other strategies (e.g., Pareto optimal space learning [52]) should be employed to solve large-scale MOPs with both high effectiveness and high efficiency.

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.

[2] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43.

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

[4] T. Back, *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

[5] C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.

[6] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied mathematics and computation*, vol. 214, no. 1, pp. 108–132, 2009.

[7] Y. Tan and Y. Zhu, "Fireworks algorithm for optimization," in *International conference in swarm intelligence*, 2010, pp. 355–364.

[8] P. Pospichal, J. Jaros, and J. Schwarz, "Parallel genetic algorithm on the CUDA architecture," in *European conference on the applications of evolutionary computation*. Springer, 2010, pp. 442–451.

[9] M. T. Jensen, "Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 503–515, 2003.

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

[11] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization," in *Proceedings of the Fifth Conference on Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, 2001, pp. 95–100.

[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, 2011.

[13] N. Srinivas and K. Deb, "Multiobjective optimization using non-dominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1995.

[14] Y. Tian, H. Wang, X. Zhang, and Y. Jin, "Effectiveness and efficiency of non-dominated sorting for evolutionary multi- and many-objective optimization," *Complex & Intelligent Systems*, vol. 3, no. 4, pp. 247–263, 2017.

[15] A. Mambrini and D. Izzo, "PaDe: A parallel algorithm based on the MOEA/D framework and the island model," in *Proceedings of the International Conference on Parallel Problem Solving from Nature*, 2014, pp. 711–720.

[16] M. Z. de Souza and A. T. R. Pozo, "A GPU implementation of MOEA/D-ACO for the multiobjective traveling salesman problem," in *Proceedings of the 2014 Brazilian Conference on Intelligent Systems*, 2014, pp. 324–329.

[17] L. Russo and A. P. Francisco, "Quick hypervolume," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 481–502, 2014.

[18] S. Jiang, J. Zhang, Y.-S. Ong, A. N. Zhang, and P. S. Tan, "A simple and fast hypervolume indicator-based multiobjective evolutionary algorithm," *IEEE Transactions on Cybernetics*, vol. 45, no. 10, pp. 2202–2213, 2015.

[19] K. Shang and H. Ishibuchi, "A new hypervolume-based evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, 2020, in press.

[20] Y. Tian, R. Cheng, X. Zhang, and Y. Jin, "PlatEMO: A MATLAB platform for evolutionary multi-objective optimization," *IEEE Computational Intelligence Magazine*, vol. 12, no. 4, pp. 73–87, 2017.

[21] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Boston, USA: Addison-Wesley, 1989.

[22] X. Zhang, Y. Tian, and Y. Jin, "A knee point driven evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 6, pp. 761–776, 2015.

[23] 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.

[24] Y. Liu, D. Gong, J. Sun, and Y. Jin, "A many-objective evolutionary algorithm using a one-by-one selection strategy," *IEEE Transactions on Cybernetics*, vol. 47, no. 9, pp. 2689–2702, 2017.

[25] K. Li, K. Deb, Q. Zhang, and S. Kwong, "Combining dominance and decomposition in evolutionary many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 694–716, 2015.

[26] M. Li, S. Yang, and X. Liu, "Pareto or non-Pareto: Bi-criterion evolution in multi-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 5, pp. 645–665, 2015.

[27] Y. Tian, R. Cheng, X. Zhang, F. Cheng, and Y. Jin, "An indicator based multi-objective evolutionary algorithm with reference point adaptation for better versatility," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 4, pp. 609–622, 2018.

[28] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "An efficient approach to non-dominated sorting for evolutionary multi-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 201–213, 2015.

[29] ——, "Empirical analysis of a tree-based efficient non-dominated sorting approach for many-objective optimization," in *Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence*, 2016.

[30] K. Li, K. Deb, Q. Zhang, and Q. Zhang, "Efficient nondomination level update method for steady-state evolutionary multiobjective optimization," *IEEE Transactions on Cybernetics*, vol. 47, no. 9, pp. 2838–2849, 2016.

[31] J. J. Durillo, A. J. Nebro, F. Luna, and E. Alba, "On the effect of the steady-state selection scheme in multi-objective genetic algorithms," in *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*, 2009, pp. 183–197.

[32] P. C. Roy, M. M. Islam, and K. Deb, "Best order sort: A new algorithm to non-dominated sorting for evolutionary multi-objective optimization," in *Proceedings of the 2016 Genetic and Evolutionary Computation Conference Companion*, 2016, pp. 1113–1120.

[33] H. Wang and X. Yao, "Corner sort for Pareto-based many-objective optimization," *IEEE Transactions on Cybernetics*, vol. 44, no. 1, pp. 92–102, 2014.

[34] K. M. Clymont and E. Keedwell, "Deductive sort and climbing sort: New methods for non-dominated sorting," *Evolutionary Computation*, vol. 20, no. 1, pp. 1–26, 2012.

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

[36] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable test problems for evolutionary multiobjective optimization," in *Evolutionary Multiobjective Optimization. Theoretical Advances and Applications*, 2005, pp. 105–145.

[37] Y. Tian, X. Xiang, X. Zhang, R. Cheng, and Y. Jin, "Sampling reference points on the Pareto fronts of benchmark multi-objective optimization problems," in *Proceedings of the 2018 IEEE Congress on Evolutionary Computation*, 2018.

[38] 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*, 2019.

[39] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, no. 4, pp. 115–148, 1995.

[40] K. Deb and M. Goyal, "A combined genetic adaptive search (GeneAS) for engineering design," *Computer Science and Informatics*, vol. 26, no. 4, pp. 30–45, 1996.

[41] Y. Tian, X. Zhang, C. Wang, and Y. Jin, "An evolutionary algorithm for large-scale sparse multi-objective optimization problems," *IEEE Transactions on Evolutionary Computation*, 2019, in press.

[42] L. Davis, "Applying adaptive algorithms to epistatic domains," in *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 1, 1985, pp. 162–164.

[43] D. B. Fogel, "An evolutionary approach to the traveling salesman problem," *Biological Cybernetics*, vol. 60, no. 2, pp. 139–144, 1988.

[44] J. Kennedy, J. F. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*. Morgan Kaufmann, 2001.

[45] A. J. Nebro, J. J. Durillo, J. Garcia-Nieto, C. A. C. Coello, F. Luna, and E. Alba, "SMPSO: A new PSO-based metaheuristic for multi-objective optimization," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making*, 2009, pp. 66–73.

[46] Y. Tian, X. Zheng, X. Zhang, and Y. Jin, "Efficient large-scale multi-objective optimization based on a competitive swarm optimizer," *IEEE Transactions on Cybernetics*, vol. 2019, in press.

[47] K. Price, R. M. Storn, and J. A. Lampinen, *Differential evolution: A practical approach to global optimization*. Springer Science & Business Media, 2006.

[48] H. Li, Q. Zhang, and J. Deng, "Biased multiobjective optimization and decomposition algorithm," *IEEE Transactions on Cybernetics*, vol. 47, no. 1, pp. 52–66, 2017.

[49] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "A decision variable clustering-based evolutionary algorithm for large-scale many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 1, pp. 97–112, 2018.

[50] C. He, R. Cheng, C. Zhang, Y. Tian, Q. Chen, and X. Yao, "Evolutionary large-scale multiobjective optimization for ratio error estimation of voltage transformers," *IEEE Transactions on Evolutionary Computation*, 2020, in press.

[51] X. Ma, F. Liu, Y. Qi, X. Wang, L. Li, L. Jiao, M. Yin, and M. Gong, "A multiobjective evolutionary algorithm based on decision variable analyses for multiobjective optimization problems with large-scale variables," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 2, pp. 275–298, 2016.

[52] Y. Tian, C. Lu, X. Zhang, K. C. Tan, and Y. Jin, "Solving large-scale multi-objective optimization problems with sparse optimal solutions via unsupervised neural networks," *IEEE Transactions on Cybernetics*, vol. 2020, in press.