A Hybrid Development Platform for Evolutionary Multi-Objective Optimization

Ruimin Shen
School of Mathematics & Computational Science
Xiangtan University, Hunan, China
Email: ruimin0shen@gmail.com

Jinhua Zheng
Institute of Information Engineering
Key Laboratory of Intelligent Computing &
Information Processing of Ministry of Education
Xiangtan University, Hunan, China
Email: jhzheng@xtu.edu.cn

Miqing Li
Department of Information Systems & Computing
Brunel University, Uxbridge, Middlesex, U. K.
Email: miqing.li@brunel.ac.uk

***Abstract*—This paper introduces an optimization template library (OTL), a cross-platform C++ template library for multi-objective optimization. OTL has an object-oriented architecture, which allows that different modules can be arbitrarily combined with each other. Moreover, the C++ template technique is used to increase the flexibility of OTL. Meanwhile, generic programming is widely used in OTL, and the generic algorithms can be used to process different data structures. However, compared with C++, the Python script is more suitable for building the experimental platform. To ensure that all attributes of the experimental results can be fully maintained, a database is used to store the experimental data. Moreover, batch experiments can be easily defined in a set of configuration files; thus, the experiments can be executed automatically without human intervention. In addition, serial and various parallel execution modes are supported, and the user can easily switch the running mode to distributed computing to increase the computing speed. Finally, a highly customizable data visualization tool is created to play back the data sample stored in the database. From a series of comparative studies, the accuracy and running performance of OTL are verified by the statistical results.**

***Keywords*—evolutionary multi-objective optimization, object-oriented architecture, generic programming**

I. Introduction

Many real-world problems are multi-objective optimization problems (MOPs), which have multiple conflicting objective functions needing to be optimized simultaneously. Compared with the single-objective optimization, multi-objective optimization algorithms usually provide a set of Pareto optimal [1] solutions for the decision maker. MOPs are usually highly complex, which makes the exact techniques not applicable. But some population-based metaheuristics, such as genetic algorithms, particle swarm optimization algorithms, and ant colony algorithms, usually perform well in these MOPs.

Significant interest in the use of evolutionary algorithms (EAs) to deal with MOPs promotes the development of the software framework with respect to evolutionary multi-objective optimization (EMO). In general, a good EMO software framework has the following features:

- An object-oriented architecture. Polymorphism is one of the most important features in object-oriented design. It creates a consistent interface between different modules so that they can be arbitrarily combined.
- Problem-independent optimizers [2]. MOPs and variation operators (e.g., the crossover and mutation) are typically problem-specific, that is, these modules often rely on the knowledge in the engineering application domain, and their decision variables may be presented by different encodings. However, the optimizers should be problem-independent; thus, can be applied to various real-world applications.
- Ready-to-use modules. The software framework should provide various ready-to-use modules, and the correctness of these modules should be fully verified.
- Automatically performed batch experiments. Experiments can be executed automatically without human intervention. This reduces the possibility of making mistakes, and makes conducting experiments convenient.
- Complete preservation of experimental data. In general the experimental results have various attributes, which are very important in statistical analysis. How to completely save these attributes is one of the most important studies.
- Parallel computing. The software framework not only makes full use of the multiple cores of the modern multi-core CPU but also supports distributed computing.
- Data visualization. A data visualization tool is provided to play back the saved data.

However, most existing source codes provided by researchers lack the above designs, like the Walking-Fish-Group (WFG) toolkit [3], the non-dominated sorting genetic algorithm II (NSGA-II) [4], the decomposition-based multi-objective EA (MOEA/D) [5], and the multiple single objective Pareto sampling (MSOPS) [6]. Recently, some well-designed software frameworks have been proposed including these representative samples:

- Programming language independent interface for search algorithms (PISA) [2]. This is a procedure-oriented software framework. PISA divides the implementation into the problem-specific part and problem-independent part. On the application side, the problem-specific part includes the modules (MOPs

978-1-4799-7492-4/15/$31.00

and variation operators) which are used to process the decision variables. On the algorithmic side, the problem-independent part provides the optimizers that can be applied to solve MOPs with different codings. Moreover, PISA designs a communication protocol based on text files to transfer data and controls between the problem-specific part and problem-independent part; thus, these two parts can be implemented with different programming languages. However, since the problem-specific part and problem-independent part communicate with each other through text files, it will largely reduce the computational efficiency. Moreover, when developing a module, the developer must carefully observe the communication protocol of PISA. This increases the learning cost and the possibility of making mistakes. In addition, batch experiments are difficult to carry out. Once an experiment is finished, the user needs to manually modify the configuration files and proceed to the next experiment. Finally, since the file operations are mutually exclusive, experiment jobs are hard to execute concurrently.

- jMetal [7]. This is a Java library which has an object-oriented architecture. Various MOPs, optimizers, variation operators and indicators are implemented in jMetal, and experiment jobs can be concurrently executed. Since different variable representations are implemented polymorphically, the optimizers can be used to solve MOPs with different codings. However, the abstract class of the optimizers does not provide an abstract method to execute the optimizer step by step, and this makes it difficult to output the experimental results at each step. Moreover, optimizers in jMetal share the same solution class. Consequently, the solution class contains all fitness values of the implemented optimizers (e.g., the crowding distance of the NSGA-II and the k-th nearest distance of the strength Pareto EA 2 (SPEA2) [8]). This design reduces the computational efficiency when copying between solutions as well as wastes the memory space.
- MOEA Framework [9]. This is also a Java library and very similar to jMetal, but some defects in jMetal are fixed in this library. On the one hand, the abstract class of the optimizer provides interfaces to perform initialization and step execution; thus, the user can obtain the experimental results at each step. On the other hand, the optimizers in MOEA Framework also share the same solution class, but this solution class provides a hash table to extend the attributes of the solution object, so that useless information is not included. However, this design requires the developer to append the desired attributes into each solution object at run-time, and the hash operation is required when accessing these attributes. This would reduce the computational efficiency. Meanwhile, like jMetal library, different types of decision variables are also implemented in MOEA Framework using polymorphic technology. When accessing decision variables, a virtual function needs to be called. This increases the computational cost, given that decision variables could be accessed very frequently.

In this paper, we propose a C++ template library for multi-objective optimization, called optimization template library (OTL). OTL has an object-oriented design, so that different encapsulated classes can be freely combined through a unified interface. Moreover, a C++ template and generic programming are introduced in OTL. The former is used to arbitrarily define the various data structures (e.g., the type of decision variables and the type of real numbers), and the latter is applied for some algorithms to process a set of data structures. These approaches take effect at compile-time and largely reduce the efficiency loss. In addition, OTL contains a number of ready-to-use classes and functions, providing convenience for scientific research and engineering applications. Finally, OTL is a cross-platform and cross-complier library. Various operating systems (e.g., Linux, Macintosh and Windows) and compliers (like Clang, GCC and Visual C++) are supported. With the help of CMake, projects of different integrated development environments (IDEs), for example, Eclipse, Code::Blocks and Visual Studio, can be generated without any change of the source codes.

However, compared with C++, Python is more suitable for building the experimental platform. Python is a dynamic scripting language, with a garbage collection mechanism, a dynamic type system, and negligible compilation time. The development efficiency of Python is much higher than that of C++. Despite having slower running speed than C++, Python allows the developer to convert the C++ codes (or C codes) into Python modules; thus, Python can be used as a "glue language" to invoke these converted modules to reduce the running efficiency loss. Thus, we use Python to implement the experimental platform, which consists of the following 2 projects:

- PyOTL. It converts the C++ programs of OTL into Python modules using the Boost library (supports both versions 2 and 3 of Python). The unit-test technique is adopted to ensure the correctness of all converted Python modules. For the randomized algorithms (e.g., variation operators and optimizers), the independent-sample T test is used to verify whether the performance metrics of the final solution set are significantly identical to the correct data obtained by the original source codes.
- PyOptimization. This project uses the Python modules generated by PyOTL project to carry out experiments. A database is adopted to store experimental data, ensuring that all attributes associated with the experimental results can be saved. Moreover, batch experiments can be easily defined using a set of configuration files, which allow non-expert users to carry out experiments. In addition, the execution mode can be easily switched between different running modes (i.e., serial computing, multi-process parallel computing, and message passing interface (MPI) based distributed computing). Finally, a highly customizable data visualization tool is provided to replay the data stored in the database, being able to adaptively display experimental data according to their dimensionality.

The three projects mentioned above are released as the open source software with the GNU lesser general public

license (LGPL) version 3, hosted in the GitHub page[1]. The rest of this paper is organized as follows. Section II first introduces the overall architecture of OTL, and then describes the C++ template technique and the generic programming which are used in OTL. Section III describes the proposed experimental platform, and two Python-based projects (PyOTL and PyOptimization) are separately introduced. Experimental results are presented and discussed in Section IV. Finally, Section V concludes the paper and presents some lines of future work.

II. Optimization Template Library (OTL)

There are a large number of minimization MOPs (e.g., the Deb–Thiele–Laumanns–Zitzler (DTLZ) test problems [10] and traveling salesman problem (TSP) [11]) and maximization MOPs (like the 0/1 knapsack problem [12]). In the engineering application field, some real-world optimization problems may simultaneously contain minimization objectives and maximization objectives (e.g., the radar waveform optimization problem [13] and the water resource optimization problem [14]). To support these MOPs, we stipulate that all MOPs in OTL are minimization optimization problems. The maximization objectives can be converted into their opposite values, and MOPs provide a restore function to get the original objective values. This design simplifies the architecture of OTL, and increases the computing efficiency. OTL provides a number of ready-to-use modules as follows:

- MOPs. Including the real-coded problems (e.g., the Zitzler–Deb–Thiele (ZDT) test problems [15], DTLZ test problems, WFG test problems, CEC 2009 test problems [16], Kursawe [17], Rectangle [18], etc.), binary-coded problems (like the 0/1 knapsack problem), and permutation optimization problems (e.g., the TSP and its multi-objective version [19]).
- Variation operators. Including the real-coded operators (i.e., simulated binary crossover (SBX) [20], polynomial mutation (PM) [21] and differential evolution (DE) operator [22]), binary-coded operators (like the single-point crossover and bitwise mutation), and operators designed for TSP (e.g., the partially mapped crossover (PMX) [23], order crossover (OX) [24], insertion mutation [25], etc.).
- Optimizers. Such as the NSGA-II, NSGA-III [26], SPEA2, MOEA/D, ϵ-dominance based multi-objective EA (ϵ-MOEA) [27], territory defining multi-objective EA (TDEA) [28], generalized differential evolution 3 (GDE3) [29], grid-based EA (GrEA) [30], SPEA2 incorporated with shift-based density estimation (SDE) strategy [31] (SPEA2+SDE), $\mathscr{S}$ metric selection EMO algorithm (SMS-EMOA) [32], hypervolume estimation algorithm (HypE) [33], indicator-based evolutionary algorithm (IBEA) [34], MSOPS, controlling dominance area of solutions (CDAS) [35], NSGA-II incorporated with g-dominance [36] (G-NSGA-II), average ranking (AR) [37], and AR incorporated with both diversity management operator (DMO) [38], [39] and improved crowding distance [40], [39].
- Indicators. Such as the generational distance (GD) [41], inverted generational distance (IGD) [42], spacing [43], diversity measure (DM) [44], hypervolume (also called $\mathscr{S}$ metric) [12], maximum spread (MS) [15] and its improved version [45].

[1]OTL, PyOTL and PyOptimization: github.com/O-T-L

A. Architecture of OTL

OTL has an object-oriented design. The unified modeling language (UML) diagram shown in Figure 1 illustrates the basic architecture of OTL. Different modules are divided into different namespaces. Each namespace has at least one abstract class, which provides a set of abstract methods (pure virtual functions in C++) to create a unified interface between different namespaces. Thus, the derived classes in different namespaces can be arbitrarily combined through the unified interface rather than directly invoking each other.

The class `Solution` is the most basic module in OTL. The template parameter `TReal` represents the real number type (can be set to `float`, `double`, `long double`, etc.), which determines the precision and running speed of the program. The template parameter `TDecision` represents the type of the decision variable and can be specified to any data structure. The class `Solution` provides 4 member variables: `decision_` and `objective_` denote the decision variable and objective vector, respectively. The member variables `inequality_` and `equality_` provided in the parent class `Constraint` represent the inequality constraints and equality constraints, respectively.

The namespace `problem` contains the MOPs which inherit from the abstract class `Problem`. In the constructor of the abstract class, the number of objectives is saved into the member variable `nObjectives_`. In the derived classes, the protected abstract method `_DoEvaluate` should be realized to evaluate a solution, and finally return a value, which represents the number of evaluations (usually 1), and this number will be added in the member variable `nEvaluations_` (initialized with a zero value). Since OTL stipulates that all MOPs are minimization optimization problems, the abstract method `_DoFix` is used to restore the maximization objective values. For real-coded MOPs (e.g., the DTLZ1 and DTLZ2 test problems), the type of the decision variable (template parameter `TDecision`) is set to a real-valued vector (the C++ type `std::vector<TReal>`). For permutation optimization problems (like TSP), the type of the decision variable is set to an integer-valued vector (the C++ type `std::vector<size_t>`).

The crossover operators are included in the namespace `crossover`, and these operators inherit from different abstract classes. For example, the SBX operator (the class `SimulatedBinaryCrossover`) realizes the abstract method `_DoCrossover` of the abstract class `CoupleCoupleCrossover`, and 2 parent solutions (`p1` and `p2`) are recombined into 2 offspring solutions (`c1` and `c2`). Thus, the abstract class `CoupleCoupleCrossover` can be directly used in the optimizers which adopt such crossover operation (like the ϵ-MOEA). However, some optimizers (e.g., the NSGA-II) require a recombination of the parent solution set (`ancestor`) into the offspring solution set (`offspring`); thus, the abstract class `Crossover` is needed. The class

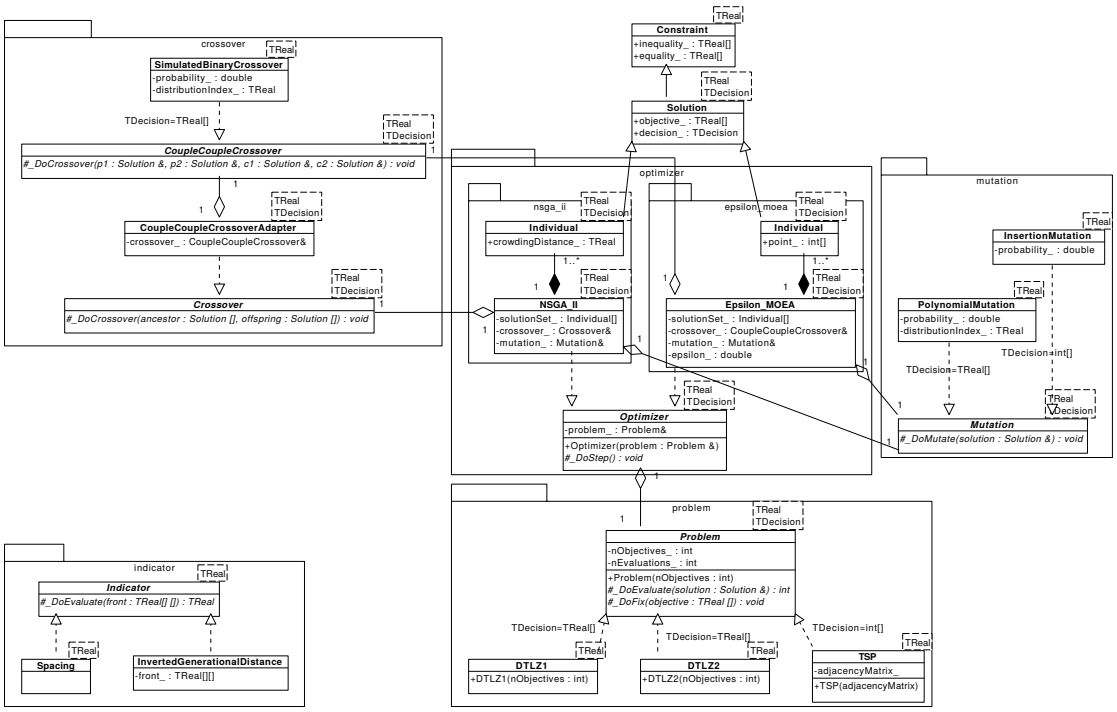


Figure 1. Architecture of OTL.

`CoupleCoupleCrossoverAdapter` is created to convert a `CoupleCoupleCrossover` object into a `Crossover` object; thus, the SBX operator can also be used in these optimizers.

The architecture of mutation operators in the namespace `mutation` is simpler than the crossover operators. These mutation operators inherit from the abstract class `Mutation`, and the abstract method `_DoMutate` must be realized. Like the MOPs and crossover operators, the mutation operators are also problem-specific; thus, these variation operators can only be applied to the MOPs with the same coding. For example, the PM operator (class `PolynomialMutation`) and the insertion mutation (class `InsertionMutation`) can be used to solve real-coded MOPs (like DTLZ1 and DTLZ2) and TSP, respectively.

In the namespace `optimizer`, all optimizers inherit from the abstract class `Optimizer`. In the constructor of the abstract class, a MOP is associated with the optimizer by saving the `Problem` object into the member variable `problem_`. Meanwhile, an abstract method `_DoStep` is provided to execute the optimizer a step. Moreover, different optimizers are included in different sub-namespaces, and their individual classes inherit from the class `Solution`. However, new attributes are extended to these individual classes since different optimizers have different fitness assignment strategies (For example, member variable `crowdingDistance_` and `point_` are extended to the individual classes of NSGA-II and ϵ-MOEA, respectively). Finally, the optimizers are problem-independent and can be used to solve MOPs with different codings since the type of decision variable can be arbitrarily defined by the template parameter `TDecision`.

Finally, we introduce the performance indicators which are contained in the namespace `indicator`. Since the indicators are used to assess the performance of the obtained objective vectors (see the abstract method `_DoEvaluate` in the abstract class `Indicator`) when the optimization process is finished, these indicators are isolated from the modules in other namespaces.

```
typedef std::mt19937 _TRandom;
typedef double _TReal;
typedef otl::problem::dtlz::DTLZ2<_TReal> _TProblem;
typedef otl::crossover::SimulatedBinaryCrossover<_TReal, _TRandom &> _TCrossover;
typedef otl::mutation::PolynomialMutation<_TReal, _TRandom &> _TMutation;
typedef _TProblem::TDecision _TDecision; // real-coded MOP: std::vector<_TReal>
typedef otl::optimizer::nsga_ii::NSGA_II<_TReal, _TDecision, _TRandom &>
        _TOptimizer;
_TRandom random(std::time(0));
_TProblem problem(3); // 3-objective optimization
_TCrossover _crossover(random, 1, problem.GetBoundary(), 20);
otl::crossover::CoupleCoupleCrossoverAdapter<_TReal, _TDecision, _TRandom &>
        crossover(_crossover, random);
_TMutation mutation(random, 1 / (_TReal)problem.GetBoundary().size(), problem.
        GetBoundary(), 20);
const std::vector<_TDecision> initial = otl::initial::PopulationUniformReal(random
        , problem.GetBoundary(), 100); // 100 individuals
_TOptimizer optimizer(random, problem, initial, crossover, mutation);
for (size_t generation = 1; problem.GetNumberOfEvaluations() < 30000; ++generation
        ) // 300 generations
{
    std::cout << "generation=" << generation << std::endl;
    optimizer();
}
```

Listing 1. The C++ source code that uses the NSGA-II to solve the real-coded MOP (DTLZ2 test problem).

B. C++ Template and Generic Programming

Since the C++ template is adopted in OTL, various data structures can be arbitrarily defined by the template parameter. Moreover, the C++ template can realize generic programming, which allows the generic algorithms to process different data structures. In C++, these technologies take effect at compile-time; thus, largely reduce efficiency losses. Here, we use the following 2 C++ source codes to illustrate how the C++ template and generic programming take effect in OTL.

```
1  typedef std::mt19937 _TRandom;
2  typedef double _TReal;
3  typedef otl::problem::tsp::TSP<_TReal> _TProblem;
4  typedef otl::crossover::OrderBasedCrossover<_TReal, _TRandom &> _TCrossover;
5  typedef otl::mutation::InversionMutation<_TReal, _TRandom &> _TMutation;
6  typedef _TProblem::TDecision _TDecision; // permutation optimization problem: std
       ::vector<size_t>
7  typedef otl::optimizer::nsga_ii::NSGA_II<_TReal, _TDecision, _TRandom &>
       _TOptimizer;
8  _TRandom random(std::time(0));
9  auto adjacencyMatrix = otl::problem::tsp::CalculateAdjacencyMatrix<_TReal>(cities.
       begin(), cities.end());
10 _TProblem problem(adjacencyMatrix); // single objective optimization
11 _TCrossover _crossover(random, 1);
12 otl::crossover::CoupleCoupleCrossoverAdapter<_TReal, _TDecision, _TRandom &>
       crossover(_crossover, random);
13 _TMutation mutation(random, 1 / (_TReal)problem.GetNumberOfCities());
14 const std::vector<_TDecision> initial = otl::initial::PopulationShuffleTSP(random,
       problem.GetNumberOfCities(), 100); // 100 individuals
15 _TOptimizer optimizer(random, problem, initial, crossover, mutation);
16 for (size_t generation = 1; problem.GetNumberOfEvaluations() < 30000; ++generation
       ) // 300 generations
17 {
18     std::cout << "generation=" << generation << std::endl;
19     optimizer();
20 }
```

Listing 2. The C++ source code that uses the NSGA-II to solve the TSP.

Listing 1 presents the source code that uses the NSGA-II to solve the real-coded MOP (DTLZ2 test problem). First, the real number type is defined (line 2). Thereafter, the problem-specific modules (MOP, crossover and mutation operators) are defined (line 3, 4, and 5). Notably, since these modules are real-coded, their member types `TDecision` must be the same. To define the optimizer (line 7), its decision variable type should be set to the member type `TDecision` of the MOP (line 6). Since the decision variable type (`TDecision`) of the template class `NSGA_II` can be arbitrarily defined by the developer, the NSGA-II can be used to solve MOPs with different codings. As can be seen in Listing 2, the template class `NSGA_II` (line 7) is used to solve the TSP (line 3) with the variation operators which are designed for TSP (line 4 and 5). Hence the flexibility and code reusability of OTL is largely improved.

III. Experimental Platform Based in Python

OTL is implemented with C++, which has a high running efficiency, and does not rely on an interpreter; thus, it is suitable for real-world engineering applications. However, due to the low compilation speed and development efficiency of C++, it is not suitable for building the experimental platform, hence we adopt Python. Python is a dynamic scripting language, which presents features such as cross-platform, negligible compilation time, and high development efficiency. The running speed of the Python script is much slower than the C++ program, but it allows the developer to convert the C++ codes into Python modules; thus, Python can be used as a "glue language" to invoke these converted modules to increase the running speed. Here, first we describe how PyOTL project converts the C++ programs in OTL into Python modules. Thereafter, we introduce PyOptimization project which is developed with Python.

A. PyOTL

PyOTL project converts the C++ programs in OTL into Python modules using the Boost library, and supports both version 2 and version 3 of Python. Python script is much simpler than C++ codes. Here, we use an example to show how to implement the same functionality with the C++ program presented in Listing 1 using the Python script (shown in Listing 3). The NSGA-II (line 7) is used to solve the 3-objective DTLZ2 test problem (line 2) with the SBX and PM

```
1 random = pyotl.utility.Random()
2 problem = pyotl.problem.real.DTLZ2(3) # 3 objectives
3 _crossover = pyotl.crossover.real.SimulatedBinaryCrossover(random, 1, problem.
      GetBoundary(), 20)
4 crossover = pyotl.crossover.real.CoupleCoupleCrossoverAdapter(_crossover, random)
5 mutation = pyotl.mutation.real.PolynomialMutation(random, 1 / float(len(problem.
      GetBoundary())), problem.GetBoundary(), 20)
6 initial = pyotl.initial.real.PopulationUniform(random, problem.GetBoundary(), 100)
       # 100 individuals
7 optimizer = pyotl.optimizer.real.NSGA_II(random, problem, initial, crossover,
      mutation)
8 while optimizer.GetProblem().GetNumberOfEvaluations() < 30000: # 300 generations
9     optimizer()
```

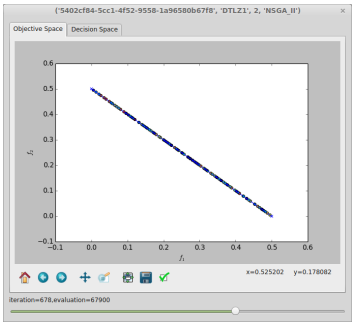
Listing 3. The Python source code that uses the NSGA-II to solve the real-coded MOP (DTLZ2 test problem).

operators (line 3 and 5). Clearly, using Python requires fewer source codes, and reduces the possibility of making mistakes. Thus, Python has a higher development efficiency than C++, and the developer can implement the experimental platform very easily.

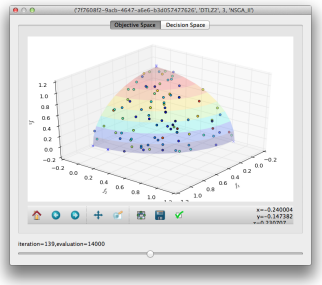
B. PyOptimization

The latest version of Python (Python 3) is used to implement PyOptimization project, which uses the Python modules generated by PyOTL project to do experiments. It has the following design:

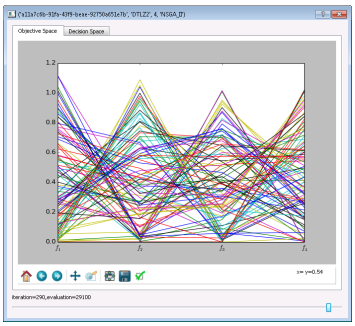
- Using a database to store the experimental results. Most existing software frameworks (e.g., jMetal and MOEA Framework) directly use the file system to save the final solution set (e.g., the objective vectors and decision variables), and some common attributes (e.g., the problem name, number of objectives, and optimizer name) are separately recorded into the folder names on different levels. These attributes are saved in the file path with a fixed order; thus, the program can identify the attribute type of the folder name. However, different experiments may have different attributes. For example, compared with the NSGA-II, the ϵ-MOEA has an extra parameter ϵ, which is important in parameter analysis experiments (i.e., analyze the relationship between the performance metrics (like GD and IGD) of ϵ-MOEA and the parameter ϵ). Since different attributes saved in the file path should have a fixed order, these attributes of the experimental data are very difficult to fully saved in the file system. In PyOptimization, these attributes are saved in a table of the Sqlite database. Each row of the table represents an independent experimental result, and different attributes of the experimental result are stored in the corresponding columns. Using the structured query language (SQL) can easily get the desired data, which can be directly exported into the statistical software (like R), making statistical analysis convenient.
- Experiments can be automatically executed in a batch, which accordingly increases the productivity of the experimental platform. PyOptimization uses the INI file to configure the batch experiments, even non-expert users can do highly complicated experiments.
- A general computing model is designed. The user can freely switch the running mode among serial computing, multi-process parallel computing, and distributed parallel computing based in MPI. The distributed parallel computing mode is implemented with the



(a) Visualizing the solution set of NSGA-II on a 2-objective DTLZ1 test problem. Different colors and the "x" marker represent the crowding distance values and their extreme values, respectively.



(b) Visualizing the solution set of SPEA2 on a 3-objective DTLZ2 test problem. Different colors represent the different fitness values.



(c) Visualizing the solution set of SPEA2 on a 4-objective DTLZ2 test problem.

Figure 2. The visualization tool can play back the data stored in the database under various operating systems (e.g., Linux, Macintosh and Windows).

MPI4Py library; thus, the experimental platform can take full advantage of the parallel computing ability of the computing cluster.

- A cross-platform data visualization tool is implemented with the Matplotlib library. It not only can visualize the low-dimensional data (2- or 3- dimensional), but also uses the parallel coordinate [46], [47] to display the high-dimensional data (more than three dimensions). Moreover, since the visualization tool is designed to play back the data stored in the database, the visualization is independent with the optimization process. Thus, the user can observe the saved data after the optimization process has finished, and the play back progress can be freely controlled. Finally, the visualization tool is highly customizable. As can be seen in Figure 2, it not only can display the true Pareto front of different MOPs, but can also visualize the fitness values of the optimizers with different colors and markers.

IV. Comparative Studies

In order to verify the accuracy and computing performance of the proposed software framework, we compare it with the above software frameworks (the latest C codes of both DTLZ and NSGA-II packages in PISA, jMetal 4.5, and MOEA Framework 2.1). The hardware used in the comparative studies is the Inspur rack server NF5240M3 (with Intel Xeon E5-2420 CPU, 8 GiB RAM, and 2×250 GB hard disk with RAID controller for RAID1 mirroring). The operating system is Linux Mint 17 x86_64. For C/C++ compiler, Java programs, and Python interpreter, Clang 3.4, OpenJDK 7u55-2.4.7, and Python 3.4.0 are used, respectively.

A. Statistics

In the comparative experiments, we execute each test instance independently 30 times, and give the mean and the standard deviation results. However, the limited sample may cause sampling error; thus, we adopt multiple comparisons [48] in the analysis of variance (ANOVA). Here, the Tamhane's T2 [49] method is used since it does not require the samples to satisfy the homogeneity of variance, and the obtained p-value shows if the mean values of the multiple samples are significantly identical. A p-value greater than the significance level α means the null hypothesis is accepted and vice versa.

Table I. Terminate Condition

Problem	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ7
Evaluation	100000	30000	100000	30000	30000

B. Experimental Settings

For the MOPs, we use the DTLZ1-4 and DTLZ7 test problems (DTLZ5 and DTLZ6 are not implemented in MOEA Framework 2.1). Meanwhile, the SBX, PM, and NSGA-II are used. To verify the accuracy of the obtained solution set, the convergence indicator GD and the comprehensive indicator IGD are used. GD is defined in Equation (1), where $d(x, P^*)$ denotes the Euclidean distance from solution x to its nearest point in P^*. IGD is defined in Equation (2), where $d'(x^*, P)$ denotes the Euclidean distance between $f(\vec{x^*})$ and its nearest objective vector in P. Low GD and IGD values are preferable. Moreover, we compare the running time of the above software frameworks.

$$\begin{aligned} GD(P) &= \frac{\sqrt{\sum_{x \in P} d(x, P^*)^2}}{|P|} \\ where\, d(x, P^*) &= \min_{x^* \in P^*} \left\| f(\vec{x}) - f(\vec{x^*}) \right\| \end{aligned} \quad (1)$$

$$\begin{aligned} IGD(P) &= \frac{\sum_{x^* \in P^*} d'(x^*, P)}{|P^*|} \\ where\, d'(x^*, P) &= \min_{x \in P} \left\| f(\vec{x^*}) - f(\vec{x}) \right\| \end{aligned} \quad (2)$$

The number of objectives used in the DTLZ test problems is 2 and 3. A crossover probability $p_c = 1$ and a mutation probability $p_m = \frac{1}{n}$ are used for the SBX and PM operators,

Table II. GD COMPARISON OF THE NSGA-II BETWEEN THE 4 SOFTWARE FRAMEWORKS REGARDING THE MEAN AND STANDARD DEVIATION (SD) VALUES. THE DARK AND LIGHT GRAY REPRESENT THE BEST AND SECOND-RANK MEAN VALUES, RESPECTIVELY. "†" INDICATES THAT THE DIFFERENCE OF MEAN VALUES BETWEEN PYOTL AND THE CORRESPONDING SOFTWARE FRAMEWORK IS STATISTICALLY SIGNIFICANT.

Problem	Obj.	PISA	jMetal	MOEA Framework	*PyOTL*
DTLZ1	2	8.21E-03 (2.10E-02)	1.14E-05 (8.84E-06)	8.45E-06 (6.38E-06)	7.91E-06 (8.69E-06)
	3	3.99E-01 (4.18E-01)†	1.01E-02 (5.45E-02)	3.43E-03 (9.77E-03)	8.74E-04 (3.82E-03)
DTLZ2	2	2.44E-04 (5.96E-05)†	1.04E-04 (2.48E-05)	1.11E-04 (2.54E-05)	1.09E-04 (2.21E-05)
	3	6.16E-03 (1.94E-03)†	1.18E-03 (3.30E-04)	1.20E-03 (1.93E-04)	1.23E-03 (3.20E-04)
DTLZ3	2	2.71E-01 (3.83E-01)†	1.07E-04 (5.99E-05)	1.12E-04 (7.89E-05)	1.05E-04 (6.98E-05)
	3	2.24E+01 (5.90E+00)†	7.02E-04 (7.06E-04)	1.10E-01 (5.67E-01)	4.88E-02 (1.74E-01)
DTLZ4	2	1.22E-04 (1.06E-04)	7.95E-05 (4.49E-05)	7.31E-05 (5.85E-05)	9.41E-05 (4.39E-05)
	3	1.97E-03 (1.32E-03)†	1.19E-03 (2.64E-04)	1.09E-03 (1.45E-04)	1.13E-03 (2.71E-04)
DTLZ7	2	5.99E-04 (2.27E-04)†	2.22E-04 (2.82E-05)	2.37E-04 (3.82E-05)	2.19E-04 (1.91E-05)
	3	5.96E-02 (2.42E-02)†	3.42E-03 (6.31E-04)	3.57E-03 (1.08E-03)	3.86E-03 (1.17E-03)

Table III. IGD COMPARISON OF THE NSGA-II BETWEEN THE 4 SOFTWARE FRAMEWORKS REGARDING THE MEAN AND STANDARD DEVIATION (SD) VALUES. THE DARK AND LIGHT GRAY REPRESENT THE BEST AND SECOND-RANK MEAN VALUES, RESPECTIVELY. "†" INDICATES THAT THE DIFFERENCE OF MEAN VALUES BETWEEN PYOTL AND THE CORRESPONDING SOFTWARE FRAMEWORK IS STATISTICALLY SIGNIFICANT.

Problem	Obj.	PISA	jMetal	MOEA Framework	*PyOTL*
DTLZ1	2	2.40E-03 (1.24E-04)†	2.22E-03 (8.41E-05)	2.16E-03 (6.10E-05)	2.18E-03 (6.50E-05)
	3	2.63E-01 (3.04E-01)†	2.56E-02 (2.25E-03)	2.56E-02 (2.03E-03)	2.51E-02 (1.59E-03)
DTLZ2	2	5.34E-03 (1.73E-04)†	4.96E-03 (1.69E-04)	4.95E-03 (1.67E-04)	4.95E-03 (1.25E-04)
	3	7.69E-02 (4.09E-03)†	6.85E-02 (2.50E-03)	6.89E-02 (3.51E-03)	6.80E-02 (3.20E-03)
DTLZ3	2	7.79E-02 (2.22E-01)	5.09E-03 (2.58E-04)	5.15E-03 (3.21E-04)	5.11E-03 (3.71E-04)
	3	2.29E+01 (7.16E+00)†	6.74E-02 (3.09E-03)	6.67E-02 (2.86E-03)	6.65E-02 (3.98E-03)
DTLZ4	2	2.77E-01 (3.63E-01)	1.53E-01 (3.01E-01)	2.52E-01 (3.55E-01)	1.04E-01 (2.56E-01)
	3	2.39E-01 (3.12E-01)†	6.62E-02 (2.79E-03)	6.53E-02 (2.47E-03)	6.58E-02 (3.19E-03)
DTLZ7	2	4.39E-01 (3.25E-04)†	4.38E-01 (1.78E-04)	4.38E-01 (1.62E-04)	4.38E-01 (1.66E-04)
	3	1.07E-01 (1.31E-02)†	7.70E-02 (4.39E-03)	8.60E-02 (4.94E-02)	7.66E-02 (3.55E-03)

respectively. For both the SBX and PM operators, a distribution index $\eta = 20$ is used. For the NSGA-II, the population size is set to 100, and the number of evaluations of different test problems is shown in Table I. For multiple comparisons, a significance level $\alpha = 0.05$ is adopted. In PISA, the polling interval is set to its minimal allowable value (0.01) to maximize the use of the CPU.

C. Experimental Results

In this section, we give the experimental results of the above software frameworks. In the following tables, the results regarding the mean and the standard deviation (SD) values are given. The dark and light gray represent the best and the second-rank results, respectively. "†" indicates that the difference of mean values between PyOTL and the corresponding software framework is statistically significant.

As can be seen in Table II and Table III, both GD and IGD results of NSGA-II in jMetal, MOEA Framework and PyOTL have no significant difference. However, these results obtained by PISA are significantly worse than the other software frameworks.

Table IV gives the running time comparison results (in seconds) of the above software frameworks. Meanwhile, we also compare the running time between OTL and PyOTL. As can be seen in the table, firstly, the running speed of PyOTL is much faster than PISA, jMetal, or MOEA Framework. Secondly, for the 2 Java libraries, jMetal is faster than MOEA Framework on all test instances. Thirdly, although the problem-specific part and problem-independent part in

Table IV. RUNNING TIME (IN SECONDS) COMPARISON OF THE NSGA-II BETWEEN THE 5 SOFTWARE FRAMEWORKS REGARDING THE MEAN AND STANDARD DEVIATION (SD) VALUES. THE DARK AND LIGHT GRAY REPRESENT THE BEST AND SECOND-RANK MEAN VALUES, RESPECTIVELY. "†" INDICATES THAT THE DIFFERENCE OF MEAN VALUES BETWEEN PYOTL AND THE CORRESPONDING SOFTWARE FRAMEWORK IS STATISTICALLY SIGNIFICANT.

Problem	Obj.	PISA	jMetal	MOEA Framework	OTL	*PyOTL*
DTLZ1	2	1.38E+01 (4.41E-01)†	1.12E+00 (1.09E-01)†	1.32E+00 (7.76E-02)†	5.00E-01 (7.65E-03)†	5.09E-01 (3.44E-03)
	3	1.33E+01 (1.68E+00)†	1.29E+00 (8.01E-03)†	1.76E+00 (9.82E-03)†	5.81E-01 (4.81E-03)†	5.93E-01 (3.05E-03)
DTLZ2	2	4.16E+00 (5.82E-02)†	3.34E-01 (7.55E-03)†	5.45E-01 (3.51E-03)†	1.89E-01 (3.65E-03)†	1.93E-01 (2.59E-03)
	3	4.45E+00 (6.84E-01)†	4.10E-01 (2.60E-03)†	7.66E-01 (4.16E-03)†	2.45E-01 (5.72E-03)†	2.49E-01 (1.48E-03)
DTLZ3	2	1.46E+01 (1.43E-01)†	1.32E+00 (8.14E-03)†	1.47E+00 (8.08E-03)†	7.75E-01 (5.71E-03)†	7.82E-01 (5.15E-03)
	3	1.51E+01 (5.69E-01)†	1.51E+00 (7.06E-03)†	1.89E+00 (1.11E-02)†	8.97E-01 (5.96E-03)†	9.11E-01 (6.04E-03)
DTLZ4	2	4.43E+00 (3.86E-01)†	3.59E-01 (2.65E-02)†	5.14E-01 (5.91E-02)†	1.92E-01 (7.28E-03)	1.97E-01 (6.36E-03)
	3	4.44E+00 (1.14E-01)†	4.54E-01 (3.98E-03)†	8.02E-01 (5.62E-03)†	2.58E-01 (3.79E-03)†	2.61E-01 (2.78E-03)
DTLZ7	2	4.64E+00 (4.63E-01)†	4.44E-01 (2.59E-03)†	6.24E-01 (5.97E-03)†	2.63E-01 (4.79E-03)	2.66E-01 (1.72E-03)
	3	4.71E+00 (6.02E-02)†	5.18E-01 (4.00E-03)†	8.40E-01 (5.85E-03)†	3.09E-01 (3.46E-03)†	3.12E-01 (2.28E-03)

PISA are both implemented with C, the time cost is much higher than the other 4 software frameworks. This phenomenon may be attributed to the inefficiency of the communication protocol based on text files. Finally, converting C++ codes into Python modules successfully reduced the computational efficiency loss. Although PyOTL is slightly slower than OTL, their running speed is very close. The development of Python script is much easier than C++ code; thus, the negligible computational efficiency loss is acceptable.

V. CONCLUSIONS

This paper has presented an optimization template library (OTL) for multi-objective optimization. OTL has an object-oriented design, and adopts C++ template and generic programming to improve its flexibility and code reusability. To achieve high development efficiency, we convert the C++ codes into Python modules (PyOTL) to build the experimental platform (PyOptimization). Various features are implemented in PyOptimization, including full preservation of experimental results, batch experiments support, parallel and distributed computing, and data visualization. Finally, the comparative experiments have verified the accuracy and running performance of the proposed software framework.

It is worth mentioning that OTL depends on the latest C++ standard (C++11) and the Boost library. PyOptimization also relies on some Python libraries (e.g., NumPy, SciPy, MPI4Py, PyQT and Matplotlib). This may increase the difficulty for beginners. Moreover, more MOPs, variation operators and optimizers should be implemented. Directly transplanting the existing codes from other software frameworks into OTL may be a feasible way. Currently, the C or C++ codes can be directly used, but how to use the codes of other languages (e.g., Java and MATLAB) still needs further study. In addition, applying OTL to real-world engineering optimization is another important issue. Since LGPL is adopted, OTL can be used in commercial projects, which enlarges its application area.

ACKNOWLEDGMENT

The authors wish to thank the support of the Hunan Provincial Innovation Foundation For Postgraduate (Grant No. CX2013A011), the National Natural Science Foundation of China (Grant Nos. 61379062, 61372049), the Science Research Project of the Education Office of Hunan Province (Grant Nos. 12A135, 12C0378), the Hunan Province Natural Science Foundation (Grant Nos. 14JJ2072, 13JJ8006), the Science and Technology Project of Hunan Province (Grant

No. 2014GK3027), and the Construct Program of the Key Discipline in Hunan Province.

REFERENCES

[1] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, 1st ed., ser. Wiley-Interscience series in systems and optimization. Chichester, New York: John Wiley & Sons, 2001.

[2] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA—a platform and programming language independent interface for search algorithms," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science. Berlin: Springer, 2003, pp. 494–508.

[3] 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.

[4] 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.

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

[6] E. J. Hughes, "Multiple single objective Pareto sampling," in *IEEE Congress on Evolutionary Computation*, vol. 4. Canberra, Australia: IEEE, 8-12 December 2003, pp. 2678–2684.

[7] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, pp. 760–771, 2011.

[8] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization," in *Evolutionary Methods for Design, Optimisation, and Control*. CIMNE, Barcelona, Spain, 2002, pp. 95–100.

[9] D. Hadka, *MOEA Framework: A Free and Open Source Java Framework for Multiobjective Optimization*, 2011. [Online]. Available: www.moeaframework.org

[10] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable test problems for evolutionary multi-objective optimization," in *Evolutionary Multiobjective Optimization*, ser. Advanced Information and Knowledge Processing. Springer, 2005, pp. 105–145.

[11] Z. Michalewicz and D. B. Fogel, *How to Solve it: Modern Heuristics*. Springer, 2000.

[12] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms—a comparative case study," in *Parallel Problem Solving from Nature*. Springer, 1998, pp. 292–301.

[13] E. J. Hughes, "Radar waveform optimisation as a many-objective application benchmark," in *Evolutionary Multi-Criterion Optimization*. Springer, 2007, pp. 700–714.

[14] P. M. Reed and J. B. Kollat, "Save now, pay later? multi-period many-objective groundwater monitoring design given systematic model errors and uncertainty," *Advances in Water Resources*, vol. 35, pp. 55–68, January 2012.

[15] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, June 2000.

[16] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, and S. Tiwari, "Multiobjective optimization test instances for the CEC 2009 special session and competition," School of Computer Science and Electrical Engineering, University of Essex, Tech. Rep. CES-887, 2008.

[17] F. Kursawe, "A variant of evolution strategies for vector optimization," in *Parallel Problem Solving from Nature*, vol. 496. Dortmund: Springer, 1-3 October 1991, pp. 193–197.

[18] M. Li, S. Yang, and X. Liu, "A test problem for visual investigation of high-dimensional multi-objective search," in *IEEE Congress on Evolutionary Computation*. Beijing, China: IEEE, 6-11 July 2014, pp. 2140–2147.

[19] D. W. Corne and J. D. Knowles, "Techniques for highly multiobjective optimisation: Some nondominated points are better than others," in *Genetic and Evolutionary Computation Conference*, London, England, UK, July 7-11 2007, pp. 773–780.

[20] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, no. 2, pp. 115–148, 1994.

[21] 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.

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

[23] D. E. Goldberg and R. Lingle, "Alleles, loci, and the traveling salesman problem," in *International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, 1985, pp. 154–159.

[24] L. Davis, "Applying adaptive algorithms to epistatic domains," in *International Joint Conference on Artificial Intelligence*, vol. 1. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1985, pp. 162–164.

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

[26] 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, 16 September 2014.

[27] K. Deb, M. Mohan, and S. Mishras, "Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of Pareto-optimal solutions," *Evolutionary Computation*, vol. 13, no. 4, pp. 501–525, 2005.

[28] İbrahim Karahan and M. Köksalan, "A territory defining multiobjective evolutionary algorithms and preference incorporation," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 4, pp. 636–664, 2010.

[29] S. Kukkonen and J. Lampinen, "GDE3: The third evolution step of generalized differential evolution," in *IEEE Congress on Evolutionary Computation*, vol. 1. Edinburgh, Scotland: IEEE Service Center, September 2005, pp. 443–450.

[30] 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, January 2013.

[31] M. Li, S. Yang, and X. Liu, "Shift-based density estimation for Pareto-based algorithms in many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 348–365, June 2014.

[32] N. Beume, B. Naujoks, and M. Emmerich, "SMS-EMOA: Multiobjective selection based on dominated hypervolume," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, 2007.

[33] 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.

[34] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Parallel Problem Solving from Nature*. Springer, 2004, pp. 832–842.

[35] H. Sato, H. E. Aguirre, and K. Tanaka, "Controlling dominance area of solutions and its impact on the performance of MOEAs," in *Evolutionary Multi-Criterion Optimization*. Springer, March 5-8 2007, pp. 5–20.

[36] J. M. Luque, L. V. Santana-Quintero, A. G. Hernández-Díaz, C. A. Coello Coello, and R. Caballero, "g-dominance: Reference point based dominance for multiobjective metaheuristics," *European Journal of Operational Research*, vol. 197, no. 2, pp. 685–692, 2009.

[37] P. J. Bentley and J. P. Wakefield, "Finding acceptable Pareto-optimal solutions using multiobjective genetic algorithms," *Soft Computing in Engineering Design and Manufacturing*, vol. 5, pp. 231–240, 1997.

[38] S. F. Adra and P. J. Fleming, "A diversity management operator for evolutionary many-objective optimisation," in *Evolutionary Multi-Criterion Optimization*. Nantes, France: Springer, 7-10 April 2009, pp. 81–94.

[39] M. Li, J. Zheng, K. Li, Q. Yuan, and R. Shen, "Enhancing diversity for average ranking method in evolutionary many-objective optimization," in *Parallel Problem Solving from Nature*. Springer, September 11-15 2010, pp. 647–656.

[40] T. Wagner, N. Beume, and B. Naujoks, "Pareto-, aggregation-, and indicator-based methods in many-objective optimization," in *Evolutionary Multi-Criterion Optimization*. Springer, March 5-8 2007, pp. 742–756.

[41] D. A. V. Veldhuizen and G. B. Lamont, "Evolutionary computation and convergence to a Pareto front," in *Late Breaking Papers at the Genetic Programming 1998 Conference*. University of Wisconsin, Madison, Wisconsin, USA: Stanford University Bookstore, October 1998, pp. 221–228.

[42] P. A. Bosman and D. Thierens, "The balance between proximity and diversity in multi-objective evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 174–188, April 2003.

[43] J. R. Schott, "Fault tolerant design using single and multicriteria genetic algorithm optimization," Ph.D. dissertation, Massachusetts Institute of Technology. Department of Aeronautics and Astronautics, 1995.

[44] K. Deb and S. Jain, "Running performance metrics for evolutionary multi-objective optimization," Indian Institute of Technology, Tech. Rep. Kangal Report No. 2002004, May 2002.

[45] C. K. Goh and K. C. Tan, "An investigation on noisy environments in evolutionary multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 3, pp. 354–381, June 2007.

[46] A. Inselberg and B. Dimsdale, "Parallel coordinates: A tool for visualizing multi-dimensional geometry," in *IEEE Conference on Visualization*. IEEE Computer Society Press, 23-26 October 1990, pp. 361–378.

[47] A. Inselberg, "The plane with parallel coordinates," *The Visual Computer*, vol. 1, no. 4, pp. 69–91, 1985.

[48] R. G. J. Miller, *Simultaneous Statistical Inference*, 2nd ed., G. Enderlein, Ed. New York: Springer Verlag, 1981.

[49] A. C. Tamhane, "Multiple comparisons in model I one-way ANOVA with unequal variances," *Communications in Statistics*, vol. 6, no. 1, pp. 15–32, 1977.