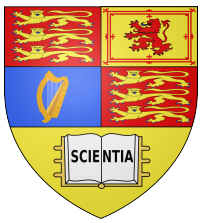


Imperial College London

Independent Study Option

Machine Learning Optimizer for FPGA architecture exploration

Author:
Tianchi Liu

Supervisor:
Prof. Wayne Luk

May 3, 2013

Machine Learning Optimizer for FPGA architecture exploration

Tianchi Liu

Department of Computing, Imperial College London

Abstract. We present a multi-objective machine learning optimization strategy developed based on the previous work on the Machine Learning Optimizer (MLO), which presents a novel technique that uses meta-heuristics and machine learning to automate the optimization of design parameters for reconfigurable designs. The main inspiration for this paper is to extend the application to be able to find trade-offs between different Field programmable gate array (FPGA) performance measures and get well formed Pareto Optimal Set for the parameter space of FPGA architecture. In order to achieve this, we present several versions of Multi-Objective Machine Learning Optimizer (MOMLO) best on Pareto optimal solution in order to find a best one which can present a best approximation of the Pareto Front. Further work is done on the regression test on the MLO to compare the current result to the golden result in order to evaluate how good the new design is. We evaluate our approach using four case studies and compare the results from different version of Multi-Objective Particle Swarm Optimizer (MOMLO). It is noticeable that this is a completely novel way of exploring the FPGA architecture within multi-objective scope and the result gain from artificial continuous functions has proved it will be successful in FPGA design.

Keywords: Optimization, Multi-Objective PSO, GP, SVM, FPGA

As a result the parameter space exploration of FPGA can take a tremendous amount of time, we have shown it to be useful to construct surrogate models of fitness functions representing design quality of reconfigurable hardware designs [1], [2]. As these models are orders of magnitude faster to evaluate than the actual benchmarks and bitstreams, they can substantially accelerate optimization thus allowing for an automated approach. This is the motivation behind our development of the MLO tool which we apply to the problem of reconfigurable designs parameter optimization. Recently however; issues like power efficiency or size have become more prominent, and consequently the design which give trade-off among different FPGA quality measures is extremely important at this perspective. In this paper we expanding the previous FPGA architecture exploration problem into the multi-objective optimization domain. We present a new multi-objective flavor of MLO and evaluate it using a number of examples. The contributions of this paper are:

- A mathematical characterization of multi-objective optimization in reconfigurable applications. We extend the previous description [1] (Section III).
- A implementation of new multi-objective version of MLO. We show how multiple Bayesian regressors and multi-objective meta heuristics can be interlinked (Section IV).
- An evaluation of the extended MOMLO approach using four multi-objective fitness functions(Section V): (a) Two single-objective functions in minimization problem, and (b) Two single-objective functions in maximization problem, and (c) One bi-objective function and (d) One three-objective function.

1 Background

1.1 FPGA

Field-Programmable Gate Arrays (FPGAs) are pre-fabricated devices that can be used to implement logic without going through an expensive fabrication process. The reconfigurability and customization of FPGA designs potentially increases performance and efficiency without increasing the frequency and consume much low power than the traditional processor [1]. However, due to the substantial effort for the FPGA designers to analyze the application and build the corresponding models and benchmarks, it is unlikely for the designer to finish all of the procedures to optimize design within a reasonable time and performance. After taking a scrutiny of the design process, some designers use a exhaustive search method to explore the parameters such as numerical precision, degree of pipelining or number of

cores. Nevertheless, when it comes to realistic problem it is not always applicable because benchmark evaluations and code execution take hours of compute time. Surrogate models approximating fitness functions by substituting lengthy evaluations with estimations based on closeness in a design space have been investigated in reconfigurable computing [3]. Together with machine learning techniques, surrogate modelling then can be used to reduce the original efforts of designers.

1.2 Gaussian Process Regression

Gaussian Process (GP) is a machine learning technology based on strict theoretical fundamentals and Bayesian theory [4, 5]. GP does not require a predefined structure, can approximate arbitrary function landscapes including discontinuities, and includes a theoretical framework for obtaining the optimum hyper-parameters [6]. An advantage of GP is that it provides a predictive distribution, not a point estimate.

A Gaussian process is a collection of random variables, any finite set of which have a joint Gaussian distribution [4]. A Gaussian process is completely specified by its mean function $m(\mathbf{x})$ and the covariance (kernel) function $k(\mathbf{x}, \mathbf{x}')$:

$$\hat{f}(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{1}$$

The $k(\mathbf{x}, \mathbf{x}')$ expresses the covariance between pairs of random variables, and in regression analysis it expresses the relation between input-output pairs. This is based on a training set $\mathcal{D}$ of n observations, $\mathcal{D} = (\mathbf{x}_i, y_i)|i = 1, ...n$, where $\mathbf{x}$ denotes an input vector, y denotes a scalar output. The column vector inputs for all n cases are aggregated in the $D \times n$ design matrix X, and the outputs are collected in the vector $\mathbf{y}$. The goal of Bayesian forecasting is to compute the distribution $p(\hat{f}|\mathbf{x}_*, \mathbf{y}, X)$ of the function $\hat{f}$ at unseen input $\mathbf{x}_*$ given a set of training points $\mathcal{D}$. Using Bayes rule, the predictive posterior for the Gaussian process $\hat{f}$ and the predicted scalar outputs $\hat{f}(\mathbf{x}_*) = y_*$ can be obtained.

1.3 Support Vector Machines Classification

Support Vector Machine (SVM) is a maximum margin classifier, which constructs a hyperplane used for classification (or regression) [7]. SVMs use kernel functions $k(\mathbf{x}, \mathbf{x}')$ to transform the original feature space to a different space where a linear model is used for classification. SVMs are a class of decision machines and so do not provide posterior probabilities. There is a training set $\mathcal{D}$ of n observations, $\mathcal{D} = (\mathbf{x}_i, t_i)|i = 1, ...n$, where $\mathbf{x}$ denotes an input vector, t denotes a target value. The column vector inputs for all n cases are aggregated in the $D \times n$ design matrix X, and the targets in the vector $\mathbf{t}$. The goal is to classify an unseen input $\mathbf{x}_*$ based on X and $\mathbf{t}$ by computing a decision boundary.

1.4 Particle Swarm Algorithm (PSO)

Particle Swarm Optimization (PSO) is a heuristic search technique (which is considered as an evolutionary algorithm by its authors [8]) that simulates the movements of a flock of birds which aim to find food. The relative simplicity of PSO and the fact that is a population-based technique have made it a natural candidate to be extended for multi-objective optimization. In the following we provide some definitions of several technical terms commonly used:

- *Swarm*: Population of the algorithm.
- *Particle*: Member (individual) of the swarm. Each particle represents a potential solution to the problem being solved. The position of a particle is determined by the solution it currently represents.
- *pbest* (personal best): Personal best position of a given particle, so far. That is, the position of the particle that has provided the greatest success (measured in terms of a scalar value analogous to the fitness adopted in evolutionary algorithms).
- *lbest* (local best): Position of the best particle member of the neighborhood of a given particle.
- *gbest* (global best): Position of the best particle of the entire swarm.

- *Leader*: Particle that is used to guide another particle towards better regions of the search space.
- *Velocity* (vector): This vector drives the optimisation process, that is, it determines the direction in which a particle needs to "fly" (move), in order to improve its current position.
- *Inertiaweight*: Denoted by W , the inertia weight is employed to control the impact of the previous history of velocities on the current velocity
- *Learning factor*: Represents the attraction that a particle has toward either its own success or that of its neighbors. Two are the learning factors used: $C1$ and $C2$. $C1$ is the cognitive learning factor and represents the attraction that a particle has toward its own success. $C2$ is the social learning factor and represents the attraction that a particle has toward the success of its neighbors. Both, $C1$ and $C2$, are usually defined as constants.
- *Neighborhood topology*: Determines the set of particles that contribute to the calculation of the lbest value of a given particle. In PSO, particles are "flown" through hyper dimensional search space. Changes to the position of the particles within the search space are based on the social-psychological tendency of individuals to emulate the success of other individuals. The position of each particle is changed according to its own experience and that of its neighbors.

Particles tend to be influenced by the success of anyone they are connected to. These neighbors are not necessarily particles which are close to each other in parameter (decision variable) space, but instead are particles that are close to each other based on a neighborhood topology that defines the social structure of the swarm [9]. The algorithm starts by randomly initializing N particles where each individual is a point in the $\mathcal{X} = \mathbb{R} \times ... \times \mathbb{R}$ search space. The population is updated in an iterative manner where each particle is displaced based on its velocity v_{id}. The x_{id} represents the dth coordinate of particle i from the set X_* of N particles, where particle is a point within $\mathcal{X}$. In the most basic form of Particle Swarm Optimization (PSO) Eq. 2-3 govern movement of particles. $r_1 \sim U(0,1)$ and $r_2 \sim U(0,1)$ are two independent uniformly distributed random numbers, c_1 and c_2 are acceleration coefficients and p_{gd} and p_{id} are dth coordinates of the global best and personal best positions.

$$v_{id} = v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \tag{2}$$

$$x_{id} = x_{id} + v_{id} \tag{3}$$

1.5 MLO

Algorithm Review In the previous approach of MLO the user supplies a benchmark along with constraints and goals, and the MLO automatically carries out the optimization (Algorithm 2) [1]. The approach consists of the following steps:

1. Build application and benchmark returning design quality metrics.
2. Specify search space boundaries and optimization goal.
3. Automatically optimize design with MLO.
4. If result is not satisfactory, redesign or revised time budget and search space.

The main concept used in the MLO is to explore the parameter space and get the outcome of different benchmark configurations. The evaluations then used to generated surrogate model by a regressor and the invalid region will also been marked using a classifier. Last, the PSO is used to explore the parameter space using the surrogate model to find a global minimum. The idea of surrogate modeling is illustrated in Fig. 1 [1]. The MLO algorithm explores the parameter space by evaluating different benchmark configurations as presented in the left figure. The results obtained during evaluations are used to build a surrogate model by using a regression of the fitness function and identifies invalid regions of the parameter space. A meta-heuristic (currently PSO) is used to guide the exploration of the parameter space which is presented in the surrogate model.

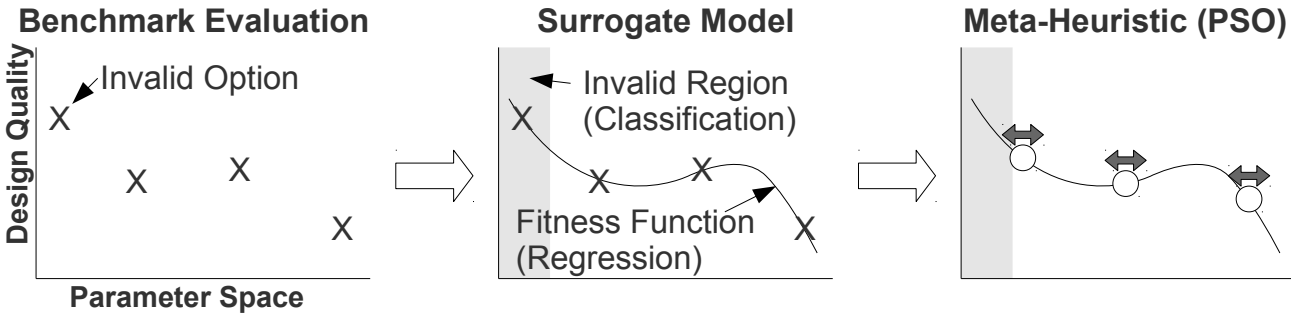


Fig. 1: Benchmark evaluations, surrogate model and model guided search.

MLO Surrogate Model There are a few steps to build the surrogate model and use it to predict:

1. Make use of Bayesian regressors to access the probability of prediction of $\hat{f}(\mathbf{x}_*)$ of non-examined parameter configurations $\mathbf{x}_*$
2. Use classifiers to predict exit codes (valid or not) of X_* across $\mathcal{X}$.
3. Predict the exit code and fitness value for a unknown parameter using both models.

Admittedly, building surrogate model will also need a certain amount of effort by using a Bayesian regressor together with a classifier to create a novel surrogate model based on the given fitness function. However, it is worth doing since it will reduce substantial effort afterwards.

Parameter Space The parameter space $\mathcal{X}$ of a reconfigurable design is spanned by discrete and continuous parameters determining both the architecture and physical settings of FPGA designs [1]. A vector $\mathbf{x}$ represents a parameter configuration within the parameter space $\mathcal{X} = \mathcal{X}_1 \times ... \times \mathcal{X}_D$ such that any $\mathcal{X}_d \subseteq \mathbb{R}$. We should notice that in the real problem of FPGA design, the parameter is either discrete or continuous, which make the design space more sophistic and as a consequence, several effort should be done to handle this well.

However, when there are more than one objective functions for optimizing, we should take all of the fitness functions into consideration when we do the meta-iteraration. Also, since there are multiple objectives, it's hard to find a 'best' particle which is both the best for all objectives. Instead, we wish to find trade-off between these objectives, thus the concept of MOMLO arises.

2 MOMLO

2.1 Objective Functions

Multi-objective optimization problems occur in many different real-world domains and particularly in the FPGA design problem. Optimization problems that have more that one objective function are rather common in every field or area of knowledge. In such problems, the objectives to be optimized are normally in conflict with respect to each other, which means it is hard to find a single solution for all of these problems. Consider the following 4 scenarios for the FPGA design:

- We evaluate our first configuration with the parameters $\mathbf{x_1}$ in the FPGA architecture, it shows poor performance on both power consumption and accuracy, with a score of 54 and 70 on the evaluation measure respectively.
- Then evaluate one another configuration with the parameters $\mathbf{x_2}$, it shows really good performance on the power consumption, say 20. Unfortunately it has a poor accuracy of only 68.
- After we evaluate one more configuration $\mathbf{x_3}$ which shows good performance on the accuracy, say 39. Unfortunately it has a poor performance on the power consumption with 60 on the evaluation measure.
- At last we evaluate a slightly better configuration $\mathbf{x_4}$ which has a power consumption of 42 and accuracy of 48.

From the above 4 scenarios, we would first declaim that the $\mathbf{x_1}$ is poor on both power consumption and accuracy and therefore it is thrown away. When considering the next 3 designs, however, we cannot say which one is better than the rest because each of them has its own strength on one aspect. Therefore, the algorithm should return all of them and let the designers to decide which one to use.

To conclude, in multi-objective MLO we want to achieve a balanced solution for the FPGA design with respect to several objectives such as power, execution time or accuracy, etc. Solving multi-objective optimization problems can be done in several ways and the one we use in this paper is the Multi-objective PSO.

In the multi-objective PSO, it is hard to find a global optimum and then we are now facing this problem:

$$\min_{\mathbf{x}} \overrightarrow{f_*(x)} = [f_1(\overrightarrow{x}), f_2(\overrightarrow{x}), \ldots, f_k(\overrightarrow{x})] \tag{4}$$

where $\overrightarrow{x} = [x_1, x_2, \ldots, x_n]$ is the vector of decision variables, there are a total of k objectives, among which $\mathbf{f_i} : \mathbb{R}^{\mathbf{n}} \to \mathbb{R}, i = 1, 2, ..., k$ are the objective functions.

Definition 1. Given two vectors, $\overrightarrow{x}, \overrightarrow{y} \in \mathbb{R}^{\mathbf{k}}$, we say that $\overrightarrow{x} \leq \overrightarrow{y}$ if $x_i \leq y_i$ for $i = 1, \ldots, k$, and that $\overrightarrow{x}$ **dominates** $\overrightarrow{y}$ (denoted by $\overrightarrow{x} \prec \overrightarrow{y}$ and $\overrightarrow{x} \neq \overrightarrow{y}$).

Definition 2. We say that a vector of parameters in parameter space of decision variables $\overrightarrow{x} \in \chi \subset \mathbb{R}^n$ is $\mathbf{non-dominated}$ with respect to χ, if there does not exist another $\overrightarrow{x'} \in \chi$ such that $\overrightarrow{f(\overrightarrow{x'})} \prec \overrightarrow{f(\overrightarrow{x})}$.

Definition 3. We say that a vector of decision variables $\overrightarrow{x*} \in \mathcal{D} \subset \mathbb{R}^n$ ($\mathcal{D}$ is the feasible region) is $\mathbf{Pareto-optimal}$ if it is non-dominated with respect to $\mathcal{D}$.

Definition 4. The **ParetoOptimalSet** $\mathcal{P}*$ is defined by:

$$\mathcal{P}* = \{\overrightarrow{x} \in \mathcal{D} | \overrightarrow{x} \text{ is Pareto-optimal}\}$$

Definition 5. The **ParetoFront**$\mathcal{PF}*$ is defined as:

$$\mathcal{PF}* = \{\overrightarrow{f}(\overrightarrow{x}) \in \mathbb{R}^k | \overrightarrow{x} \in \mathcal{P}*\}$$

.

In such problems, the objectives to be optimized are normally in conflict with respect to each other, which indicates that there is no single solution for all of these problems. Instead, we aim to find "trade-off" solutions that achieves the best possible compromises among these objectives. In other words, we wish to find the Pareto Optimal Set $\mathcal{P}*$ which is an approximation of the Pareto Front $\mathcal{PF}*$ [10].

2.2 Literature Review

Since the first proposed Multi-Objective Particle Swarm Optimizer (MOPSO) developed by Moore and Chapman in 1999 [11], more than thirty different MOPSOs have been reported in the specialized literature and usually they are mainly classified as the following methods [10].

- Aggregating approaches: combine (or "aggregate") all the objectives of the problem into a single one, then the multi-objective problem is simply transformed into a single-objective version [12, 13]. The drawback is obviously that it is hard to find a optimum coefficient to represent the different objectives.
- Lexicographic ordering: Rank the objectives in order of their importance. The optimum solution is then obtained by minimizing the objective functions separately, starting with the most important one and proceeding according to the predefined order of importance of the objectives [14]. Again, the drawback is obvious that the importance is hard to be measured and therefore the design lose its precision.

- Sub-Population approaches: involve the use of several subpopulations as single-objective optimizers. Then, the subpopulations of the PSO explore the parameter space separately and somehow exchange information or recombine among themselves aiming to produce trade-offs among the different solutions [15, 16]. This solution is somehow better but still suffer the problem of balancing the importance of several objectives.
- Pareto-based approaches: These approaches use leader selection techniques based on Pareto dominance and select as leaders to the particles that are non-dominated with respect to the swarm [17–21]. This is currently best solution for the Multi-objective optimization, however, the leader selection and non-dominate retaining method should be chosen carefully in order to reach great performance. Nowadays, several variations of the leader selection scheme are proposed and most authors adopt additional information to select leaders (e.g., information provided by a density estimator) in order to avoid a random selection of a leader from the current set of non-dominated solutions.
- Combined approaches: Combination of the solutions above [22, 23].

2.3 MOMLO Algorithm

Algorithm 1 MOMLO

for $\mathbf{x}_* \in X_*$ **do**
  for $\mathbf{i} \in \mathbf{1}, \mathbf{2}, ..., \mathbf{N}$ **do**
    $\mathbf{x}_*.\overrightarrow{fit_i} \leftarrow \overrightarrow{f_i}(\mathbf{x}_*)$ ▷ Initialize with a uniformly randomized set for every objectives f_i* in the fitness function.
  end for
end for
repeat
  for $\mathbf{x}_* \in X_*$ **do**
    $\overrightarrow{y_*}, \rho_{max} \leftarrow regressor(\mathcal{D}_r, \mathbf{x}_*)$
    $t_* \leftarrow classifier(\mathcal{D}_c, \mathbf{x}_*)$
    if $\rho_{max} < min_\rho$ and $t_* = 0$ **then**
      $\mathbf{x}_*.\overrightarrow{fit} \leftarrow \overrightarrow{y_*}$
    else
      if $t_* = 0$ **then**
        $\mathbf{x}_*.\overrightarrow{fit} \leftarrow \overrightarrow{f(\mathbf{x}_*)}$
      else
        $\mathbf{x}_*.\overrightarrow{fit} \leftarrow \mathbf{max_{val}}$ or $\mathbf{min_{val}}$
      end if
    end if
  end for
  $X_* \leftarrow Meta(X_*)$ ▷ Iteration of the meta-heuristic
until Termination Criteria Satisfied

We present our MOMLO in Algorithm 1. The algorithm's main change for the previous MLO [1] is the fitness function now return a vector $\overrightarrow{f_i}$ of fitness values and as a result, the model is trained for each fitness values. We initialize the meta-heuristic of our choice with N particles X_* uniformly randomly scattered across $\mathcal{X}$. Each particle has an associated fitness vector $\mathbf{x}.\overrightarrow{fit}$ and a position $\mathbf{x}$. Now after training the models, the predictor now get a vector of output for the prediction of each fitness functions as well as the variance. For simplicity, we only retain one variance which is the biggest value ρ_{max} after the prediction and the further judgement on how good the fitness vector is fully based on this single value. Similarly, for the predict code from the classifier, a vector of predictions from all the objective models is simplified by only one value t_*(any $t_i = 1$ means the model prediction is invalid). For all $\mathbf{x}_*$ predicted to lie in $\mathcal{V}$ we proceed as follows. Whenever ρ returned by the regressor is smaller than the minimum required confidence min_ρ we use the y_*; otherwise we assume the prediction to be inaccurate and evaluate $f(\mathbf{x}_*)$. The meta-heuristic will avoid $\mathcal{I}$ and $\mathcal{F}$ regions as they are both assigned unfavorable $\mathbf{max_{val}}$ or $\mathbf{max_{val}}$ vectors in respect to the fitness functions. We construct the training sets $\mathcal{D}_c$ and $\mathcal{D}_r$ as described in Algorithm 1. Whenever $b(\mathbf{x}_*)$ is evaluated, $(\mathbf{x}_*, t_*)$ is included within the classifier training set $\mathcal{D}_c$. If exit code is valid ($t_* = 0$), then $(\mathbf{x}_*, y_*)$ is added to $\mathcal{D}_r$.

In order to apply the meta-heuristic strategy for solving multi-objective optimization problems of the FPGA, it is obvious that the original scheme has to be modified. As we saw in Section II, the solution set of a problem with multiple objectives does not consist of a single solution (as in global optimization). Instead, in multi-objective optimization, we aim to find a set of different solutions (the so-called Pareto Optimal Set $\mathcal{P}*$) in order to have a better approximation of the Pareto Front. The pseudo code is defined as follows, First, the swarm is initialized. This initialization includes both positions and velocities. The corresponding *pbest* of each particle is initialized and the leader is located (usually the *gbest* solution is selected as the leader). Then, for a maximum number of iterations, each particle flies through the search space updating its position (using (4) and (5)) and its *pbest* and, finally, the leader is updated too.

Algorithm 2 MOPSO

```
Initialize swarm
Initialize leaders in an external archive
g ← 0
while i ≤ 10 do
    for particle ∈ Population do
        Select leader
        Update Position (Flight)
        Mutation
        Evaluation
        Update pbest
    end for
    Update leaders in the external archive
    Quality(leaders)
    g ← g + 1
end while
Report Pareto Front in the external archive
```

In fact, the criterion of how good a Pareto Optimal Set $\mathcal{P}*$ varies a lot since there is no global best to be measured. In general, when solving a multi-objective problem, three are the main goals to achieve [24]:

- Maximize the number of elements of the Pareto Optimal Set $\mathcal{P}*$ found.
- Minimize the distance of the individuals Pareto Optimal Set $\mathcal{P}*$ produced by our algorithm with respect to the true (global) Pareto Optimal Set $\mathcal{P}*$ (assuming we know its location).
- Maximize the spread of solutions found, so that we can have a distribution of vectors as smooth and uniform as possible.

Given the population-based nature of PSO, it is desirable to produce several (different) non-dominated solutions with a single run. So, as with any other evolutionary algorithm, the three main issues to be considered when extending PSO to multi-objective optimization are [25]:

- How to select leaders for each particle to guide the particle to find non-dominated solution and then have a better approximation on the Pareto Front?
- How to give a criterion on whether to retain a non-dominated solution to the Pareto Optimal Set $\mathcal{P}*$, which means the found solutions should be well spread along the Pareto Front and the total amount of which should be restricted on a reasonable limit.
- How to maintain diversity in the swarm in order to avoid premature convergence to a single solution?

For our version of Multi-Objective MLO, we design carefully in respect of the performance and feasibility of different method that have proposed by the others. Since we have two criterions to select the method:

- The method should give good performance on selection and updating of leaders as well as creation of new solutions.
- It must be easy implemented and compatible for our previous solution.

3 Design

3.1 Leaders in Multi-Objective Optimization

Although a few researches have avoided the problem of defining a new concept of leader for multi-objective problems by adopting aggregating functions (i.e., weighted sums of the objectives) or approaches that optimize each objective separately [10], they are not suitable for the FPGA design since the weight of the different objectives vary a lot on different FPGA applications.

In our solution, the different objectives are controlled by the weights but they are only for the flight of the particles. In this way, a quality measure that indicates how good is a leader is very important. Obviously, such feature can be defined in several different ways. One possible way of defining such quality measure can be related to density measures. Promoting diversity may be done through this process by means of mechanisms based on some quality measures that indicate the closeness of the particles within the swarm. Nearest neighbor density estimator [26]. The nearest neighbor density estimator gives us an idea of how crowded are the closest neighbors of a given particle, in objective function space.

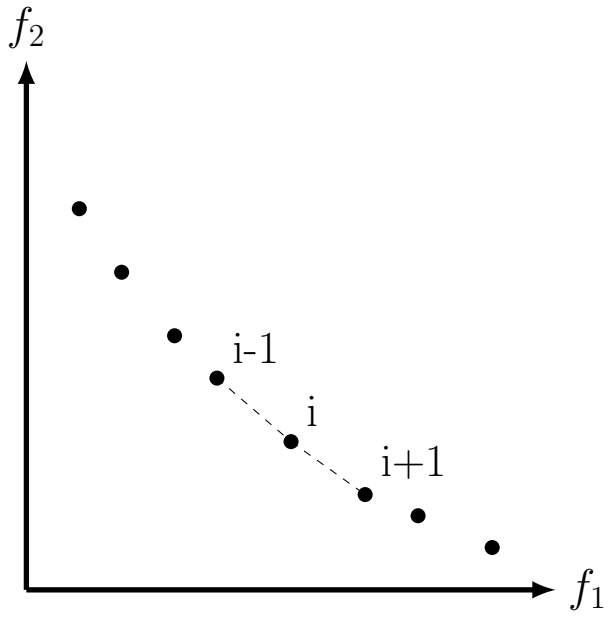


Fig. 2: The neighbor density estimator for and example with two objective functions. Particles with a large distance from the neighbors is preferred.

3.2 Retaining and Spreading Non-dominated Solutions

As we mentioned before, it is important to retain the non-dominated solutions found along all the search process so that we can report at the end those solutions that are non-dominated with respect to all the previous populations.

The most straightforward way of retaining solutions that are non-dominated with respect to all the previous populations (or swarms) is to use an external archive [10]. Such an archive will allow the entrance of a candidate solution only if: (a) it is non-dominated with respect to the current stored solution in the archive or (b) it dominates any of the solutions within the archive (in this case, the dominated solutions have to be deleted from the archive) [10]. This approach has, however, the drawback of increasing the size of the archive exponentially.

Recently, other researchers have proposed the use of relaxed forms of dominance. The main one adopted in PSO has been "ϵ-dominance" [27], which concept is illustrated in the definitions 6 [28]. The ϵ-dominance solution is believed to be able to generate well-formed Pareto Optimal Set as well as perform a much quicker converge speed.

Definition 6(ϵ- Dominance) : Let $f, g \in \mathbb{R}^m$. Then f is said to $\epsilon - dominante$ g for some $\epsilon > 0$, denoted as $f \succ_\epsilon g$, iff for all $i \in \{1, \ldots, m\}$

$$(1 + \epsilon) \cdot f_i \geq g_i. \tag{5}$$

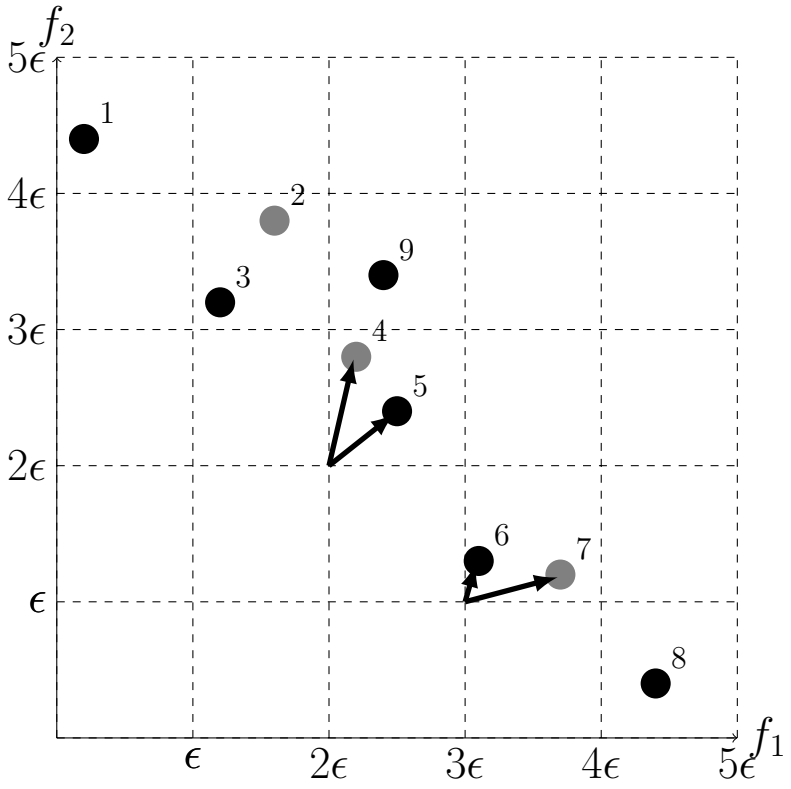


Fig. 3: A example of the use of ϵ-dominance in an external archive. Solution 3 dominates solution 2, therefore solution 1 is preferred. Solutions 4 and 5 are incomparable. However, solution 5 is preferred over solution 4, since solution 5 is closer to the lower lefthand corner represented by point (2ϵ, 2ϵ).Solution 6 is preferred compared to solution 7 for the same reason. Solution 9 is not accepted since its box, represented by point (2ϵ, 3ϵ) is dominated by the left box.

The main use of this concept in multi-objective PSO has been to filter solutions in the external archive. Mostaghim and Teich [28] have found that when comparing "ϵ-dominance" against existing clustering techniques for fixing the archive size, the "ϵ-dominance" method can find solutions much faster (computationally speaking) than the clustering technique with a comparable (and even better in some cases) convergence and diversity.

3.3 Promoting Diversity while Creating New Solutions

It is well known that one of the most important features of the PSO algorithm is its ability to find global optimum instead of blocked at a local one.This is a positive feature as long as we don't have premature convergence (i.e., convergence to a local optimum). Premature convergence is caused by the rapid loss of diversity within the swarm. When adopting PSO for solving multi-objective optimization problems, it is possible to promote diversity through the selection of leaders. In this way, we choose to use a mutation operator to help the flight of the particle is very important in order to escape from local optima and to improve the exploratory capabilities of PSO.

4 Implementation

As I illustrated in the previous part, there are 5 most important functionalities to be implemented in order to maintain the Pareto Optimal Set:

- Initialize the population of the particle swarm;
- Retrieve best global leader for each of the particles before it update its position;
- Measure the density of a new found non-domininate particle;
- Retain the new non-dominated solution based on the new fitness values;
- Remove dominated solution from Pareto Optimal Set $\mathcal{P}*$;

4.1 Initialize

- At first we create an empty Pareto Optimal Set dictionary $\mathcal{P}*$, set the size of the Dictionary as the maximum amount of Pareto Optimal Set points we want. Every entry of the $\mathcal{P}*$ is another dictionary including the particle's position and fitness values.
- Then for every initialized particle in the population, evaluate the fitness tuple $\mathcal{T}$ using the multi-objective fitness function.
- Check if this $\mathcal{T}$ dominate every points in the Pareto Set $\mathcal{P}*$ and decide whether to store this fitness tuple $\mathcal{T}$ into the external Pareto Optimal Set $\mathcal{P}*$ or not.
- Retrieve the Pareto Optimal Set $\mathcal{P}*$ and assign a leader for every particle.

4.2 Retrieve

For every Pareto Point in the Pareto Optimal Set $\mathcal{P}*$, a nearest neighbor density (crowding radius) estimator is chosen to maintain it's suitability as leader and thus to promote diversity. Three methods have been implemented to select leader based on this density measure.

- Choose the Least density particle (one has the $density_{min}$): For every member of Pareto non-dominate particles (stored in a external achieve), a neighbor density is computed and then chooses the least density one.
- By means of Tournament: For every member of Pareto non-dominate particles (stored in an external achieve), a neighbor density is computed and then the top k (usually a small positive integer) particles are chosen then select one randomly within them.
- By means of a Roulette selection: That is for every member of Pareto non-dominate particles (stored in a external achieve), a neighbor density is computed and then chooses it with roulette. A particle with a small neighbor density has more probability to be chosen since the neighbor is less density.

4.3 Neighbor Density Estimator

Euclidean Distance is chosen as the measure of each two particles in the Pareto Optimal Set achieve, however, there still remaining one question that how we represent the distance for a particle in the Pareto Optimal Set $\mathcal{P}*$, namely the neighbor density. Two methods are implemented to calculate the neighbor density.

- Nearest k neighbors: That is for every member of Pareto non-dominate particles (stored in a external achieve), a Euclidean distance is computed between it and every others and then bottom k with less distance than others are chosen to add up as the neighbor density. In my program the k is set to be 2, namely to calculate it's nearest two neighbors (left one and right one in 2-D space).
- Niche neighbors: That is for every member of Pareto non-dominate particles (stored in a external achieve), a Euclidean distance is computed between it and every others, and then only the ones within that niche count are retained to add up as the neighbor density. The size of niche may influence the result a lot, so it should be carefully chosen for each run.

4.4 Retain

Every time when a non-dominated solution is discovered, a new question raise for whether it should be added to the Pareto Optimal Set $\mathcal{P}*$ as discussed in the previous part to prevent premature converge and over-sized achieve. Two methods are implemented to calculate the neighbor density.

- Simple Retain with replacement: After the non-dominated property of the given particle has been proved, the particle is treated as a candidate for entering the Pareto Optimal Set $\mathcal{P}*$. If the set is not full (the size is set by the configuration), the particle gets its right to be retained. However, if the $\mathcal{P}*$ is full, the most density one (computed by the previous density measure) has to be replace by this new particle or the new particle won't be able to be retained.
- ϵ-dominate retain measure: In this case, the simple dominate property won't be enough for the particle to be retained. After the simple non-dominated property of the given particle has been proved, the particle is again treated as a candidate for entering the Pareto Optimal Set $\mathcal{P}*$. Then as discussed in previous part, the Pareto Optimal Set $\mathcal{P}*$ is divided to ϵ boxes, the one in the bottom left of each box is the implicit non-dominate point in this box. Then if the particle belongs to a certain box and the box is empty, it can be retained without competing. Otherwise, if a particle already exists in this particle, the candidate should compete with this particle, if one of the two wins, it will be retained and if both of them cannot dominate each other, the one near the implicit non-dominate point in the bottom left will be chosen. Again, the value of ϵ should be carefully chosen as it may place a tremendous influence on the result. In my implementation, ϵ starts with a minimal $\epsilon = 0.0075$, which is systematically increased every time the number of archived vectors exceeds a predetermined maximum size.

4.5 Auxiliary Functionalities

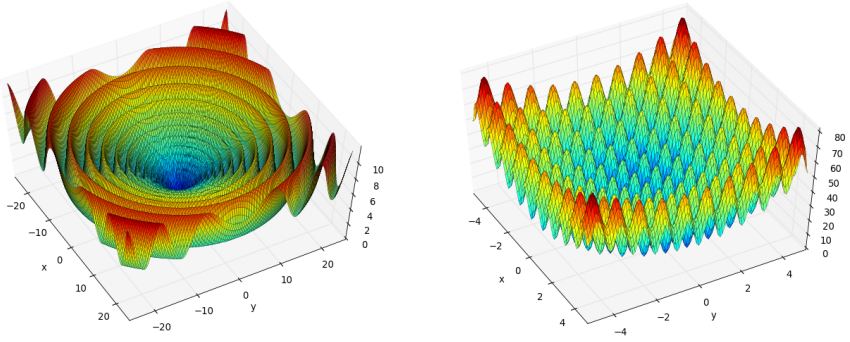
- Fitness Values Tuple $\mathcal{T}$: For the multi-objective optimization problem, the fitness is no longer one value but a tuple of the values from different fitness values. Thus the existing data structure for the particle fitness should be change to tuples $(f_1, f_2, \ldots, f_n)$.
- Train and Predict: Each objective has its own fitness function, so does the surrogate model. In the problem of multi-objective optimization, the number of models equals to the number of objective functions. During the implementation, the training functions train surrogate model for every fitness functions and then a list of models are stored instead of just one model. The predict procedure are almost the same, however, some more implementation should be done since there are a list of results from different models. For the final result, only the predictions of the different fitness values should be retained in a tuple but the classification and variances should be only one value or a vector ($t_{t_i=0}$ and ρ_{max}) in respect to the parameter space $\mathcal{X}$.
- Non-dominate Criterion: The non-dominate property of each particle is simple and which can be specialized for every maximization/minimization problem. For every single fitness function in the multi-objective optimization, the particle should give better performance than others and this particle is defined dominate others. However, if only one fitness value is not better, it cannot dominate the others and if it has at least one fitness value is better, it is a non-dominated point.
- Termination Condition: The Termination Condition now is different from the single-objective one, that is there is no best fitness value we can decide as the termination condition like we do in the single objective one. Now other than the generation budget and number of times we evaluate the fitness functions, another criterion on how good the Pareto Optimal Set $\mathcal{P}*$ is has been implemented: when the Pareto Optimal Set $\mathcal{P}*$ has reach the size we pre-defined in the configuration, a variance value of them is computed and if the variance is well below a predefined criterion. In that condition, it is believed that the Pareto Optimal Set is an appropriate approximation of the Pareto Optimal Set.
- Multi-objective Plot function: The plot function for multi-objective one is again different since now a few of fitness functions should be shown and the same as the surrogate models and most importantly, the Pareto Optimal Set.

The main concepts behind the original MLO and the mutli-objective version are the same and as such we are not going to present them.

5 Evaluation

The evaluation takes on four different multi-objective functions by combining the artificial continuous functions provided by deap evolutionary algorithm package in python(See Fig. 4 for examples):

1. Two single-objective functions (minimizaiton functions): sphere and schaffer, see table 1 for details of the functions.
2. Two single-objective functions (maximizaiton functions): sphere and rastrigin, see table 2.
3. One bi-objective function (minimizaiton functions): fonseca, see table 3
4. Three-objective function (minimizaiton functions): One single objective function (sphere) and one bi-objective function (fonseca), see table 4.



(a) schaffer (b) rastrigin

Fig. 4: Single-objective artificial continious examples.

Table 1: Multi-objective function 1

Attributes	**Function1 : sphere**	**Function2 : schaffer**
Type	minimization	minimization
Range	none	$x \in [-100, 100]$
Global optima	$x_i = 0, \forall x \in \{1, ..., X\}, f(x) = 0$	$x_i = 0, \forall x \in \{1, ..., X\}, f(x) = 0$
Function	$f(x) = \sum_{i=1}^{N} x_i^2$	$f(x) = \sum_{i=1}^{N-1} (x_i^2 + x_{i+1}^2)^{0.25} \cdot \sin^2(50 \cdot (x_i^2 + x_{i+1}^2)^{0.1} + 1)$

Table 2: Multi-objective function 2

Attributes	**Function1 : sphere**	**Function2 : rastrigin**
Type	maximization	maximization
Range	none	$x \in [-5.12, 5.12]$
Global optima	$x_i = 0, \forall x \in \{1, ..., X\}, f(x) = 0$	$x_i = 0, \forall x \in \{1, ..., X\}, f(x) = 0$
Function	$f(x) = \sum_{i=1}^{N} x_i^2$	$f(x) = 10N \sum_{i=1}^{N} x_i^2 - 10 \cos(2\pi x_i)$

Table 3: Multi-objective function 3

Objectives	**fonseca multi − objective function**
Function 1	$f_{Fonseca1}(\boldsymbol{x}) = 1 - e^{-\sum_{i=1}^{3}(x_i - \frac{1}{\sqrt{3}})}$
Function 2	$f_{Fonseca2}(\boldsymbol{x}) = 1 - e^{-\sum_{i=1}^{3}(x_i + \frac{1}{\sqrt{3}})}$

Table 4: Multi-objective function 4

Objectives	**fonseca multi − objective function**
Function 1	$f(x) = \sum_{i=1}^{N} x_i^2$
Function 2	$f_{Fonseca1}(\boldsymbol{x}) = 1 - e^{-\sum_{i=1}^{3}(x_i - \frac{1}{\sqrt{3}})}$
Function 3	$f_{Fonseca2}(\boldsymbol{x}) = 1 - e^{-\sum_{i=1}^{3}(x_i + \frac{1}{\sqrt{3}})}$

In all runs of the examples, we use GPs utilizing an isotropic exponential kernel with additive Gaussian noise. We choose SVMs as our classifier with a Radial Basis Function (RBF) kernel. Due to its simplicity and effectiveness we use a velocity clamping version of PSO with c_1 and c_2 set to 2.0. All presented results are averaged over 10 trials.

Additionally, For the purpose of evaluating the performance on different approaches to select leaders and retain non-dominate point in the MOMLO, we run a series of trials based on the same fitness function with different approaches and give comparison by providing the graphs of Pareto Front $\mathcal{PF}$ and tracing the velocity of the particles.

5.1 Results

The results for all of the four different evaluations are plotted by the view function of MOMLO (Fig. 5, Fig. 7, Fig. 8 and Fig. 6). For each image:

- The top-left is the display of the Pareto Optimal Set $\mathcal{P}$;
- The right two of it are the view of original fitness function and the two below them are the corresponding training models by the Gaussian regressors (the view may seen a little bit different from the original one because the models from regressors currently are not work perfectly especially for complicated fitness functions: it only can draw a sketch of the fitness function but not accuracy enough as the original one);
- The lower two under them are the view of the design space (the green cross shows the original sample place for the particles and the white circles are the particles' position after the algorithm terminates) and the bottom-left sub-image is a plot of the average particle velocity within the current generations.

As we can see in the first 3 images (Fig. 5, Fig. 7 and Fig. 8), which are all bi-objective optimization problem, the shape of the Pareto Optimal Set $\mathcal{P}$ are all denote a reasonable approximation of the Pareto Front $\mathcal{PF}$.

Notice that the plot of the design space shows the particles converge but not exactly in the same position, which is because the multi-objective optimization won't guarantee a best solution but a series of Non-dominated solutions. At last image (Fig. 6) for the 3D plotting of the 3-objective fitness functions, each axis represents a single fitness function and the results gives the Pareto Optimal Set $\mathcal{P}$ (now it is a surface instead of a line in bi-objective problems), which indicates the trade-off between these objectives.

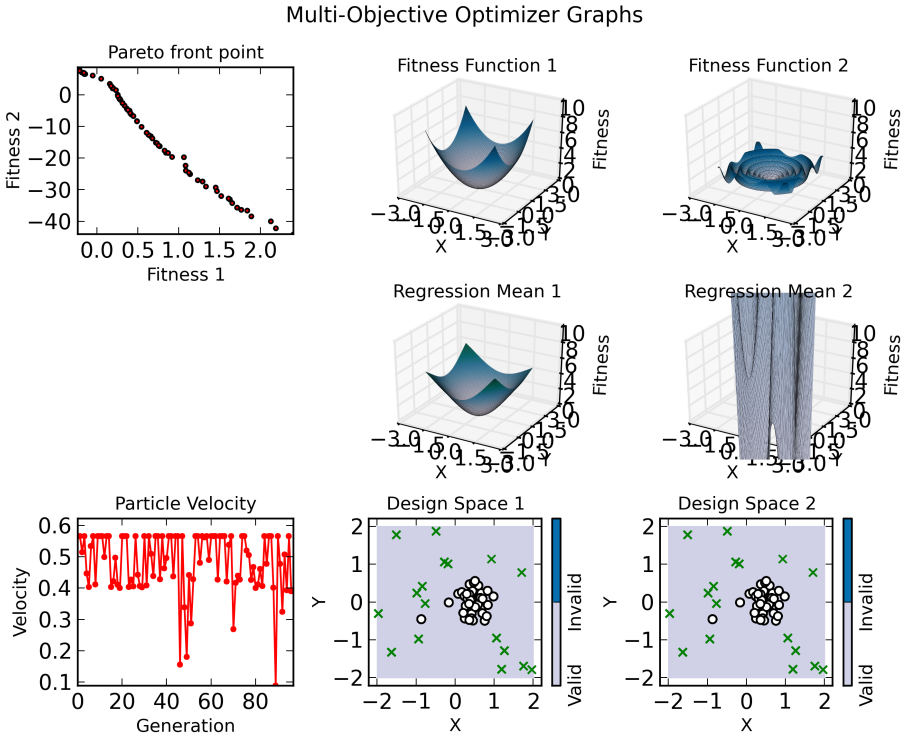


Fig. 5: Plot of the run results image for two minimization single-objective artificial continious functions.

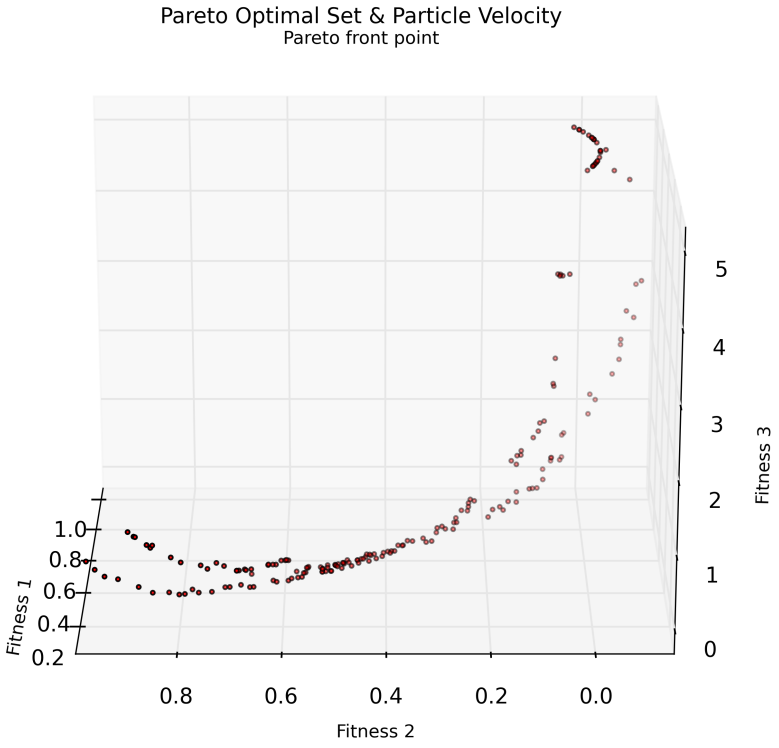


Fig. 6: Plot of the run results image for three-objective artificial continious function.

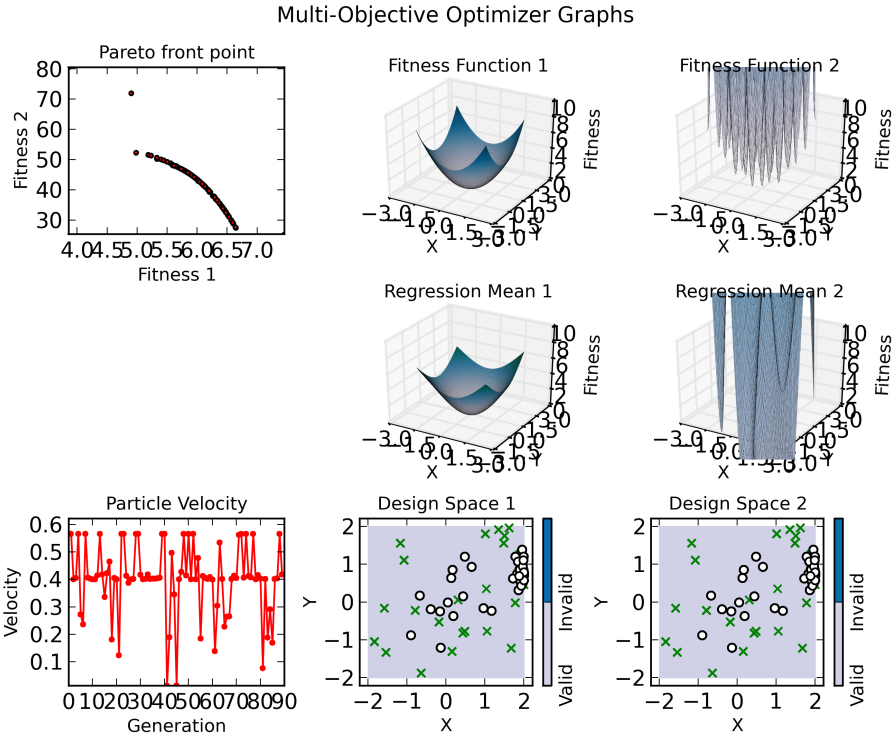


Fig. 7: Plot of the run results image for two maximization single-objective artificial continious functions.

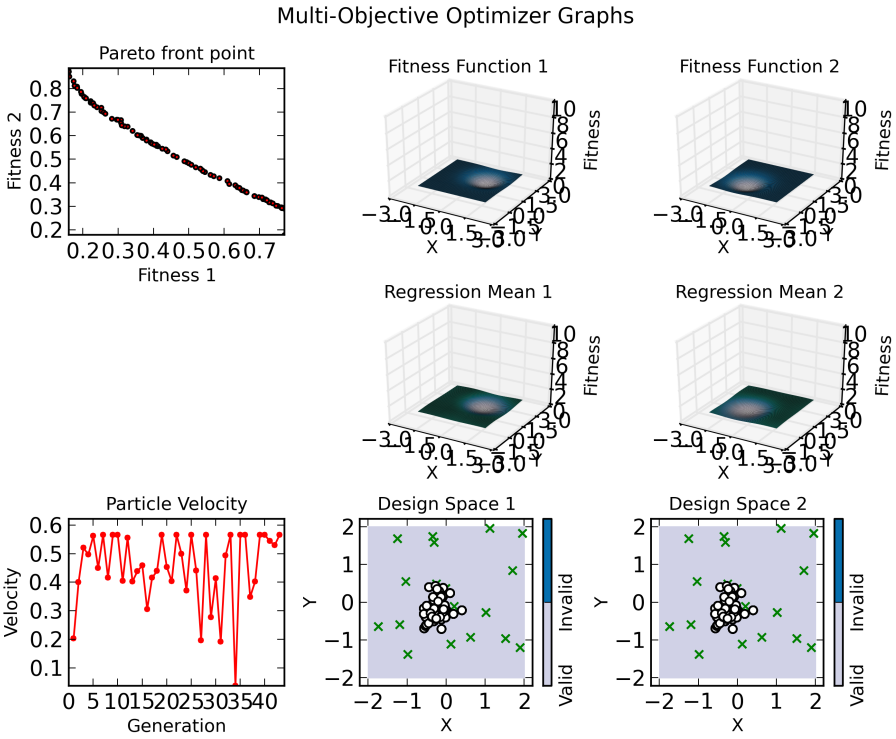
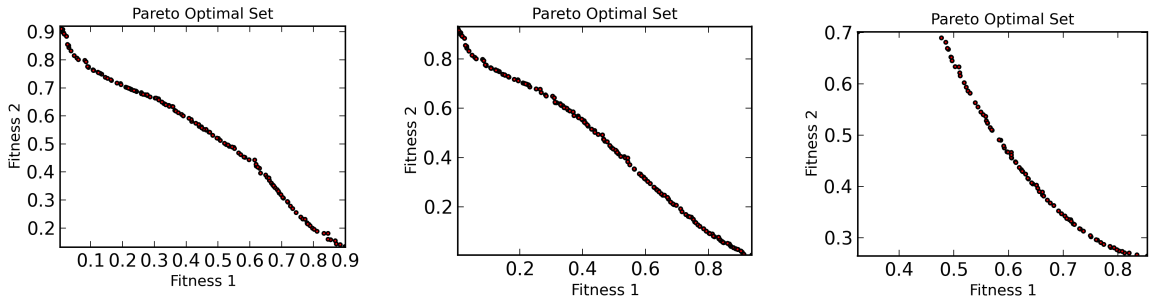


Fig. 8: Plot of the run results image for bi-objective artificial continious function.

5.2 Comparison of approaches

We compared the different approaches we used to design the MOMLO by giving several runs of the trials on the same bi-objective fitness function: fonseca. For each approach we present a graph for the Pareto Front obtained from the trial (the most representative one) to indicate how good the approach is in approximating the Pareto Front (we expect it to be similar as Figure). In addition, an average converge time (which obtained from over 10 trials each) has been presented for each approach to measure the converge property of them. Obviously, we prefer the one with less converge time and more accurate Pareto Front.

Leader-retrieve methods We can see only slightly difference from the converging time (Table. 5)for all of the three solutions and the roulette method is slightly better than the others. The roulette selection method shows great performance on the shape of the Pareto Front (see Fig. 9), which is because its half-randomness property and also it avoids the aimless of a particle tracing the worse solution in least density solution.



(a) Least Density Method (b) Tournament Method (c) Roulette Method

Fig. 9: Plot of Pareto Optimal Set from three Leader-retrieve methods

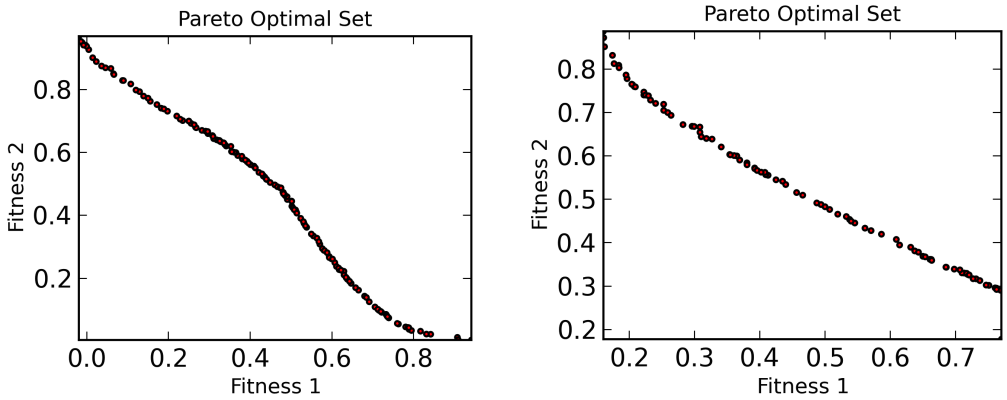
Table 5: Average converge time for different Leader-retrieve of methods

Methods	**Average Converge Time(ms)**
Least density particle	1290
Tournament	1360
Roulette selection	1123

Non-dominated Retain methods For the retain methods, the ϵ-dominate method is obviously better concerning both measurements as presented in the Table. 6 and the Fig. 10: the average converging time is two-thirds of the simple retain methods and the Pareto front of ϵ-dominate method shows a much better shape than another.

Neighbor Density Estimators At last, as we can see in the Table. 7 and Fig. 11, the two methods produce almost the same results on both the converge time and Pareto front approximation, so we declaim that both of them are suitable for this application and we will choose one randomly.

From these results, we can now claim that the (roulette or tournament) + ϵ-dominate + (Nearest-k neighbors or Niche neighbors) are the best combinations for the current application. Through all of the results, we can now decline that our design is correct and can efficiently solve the Multi-Objective

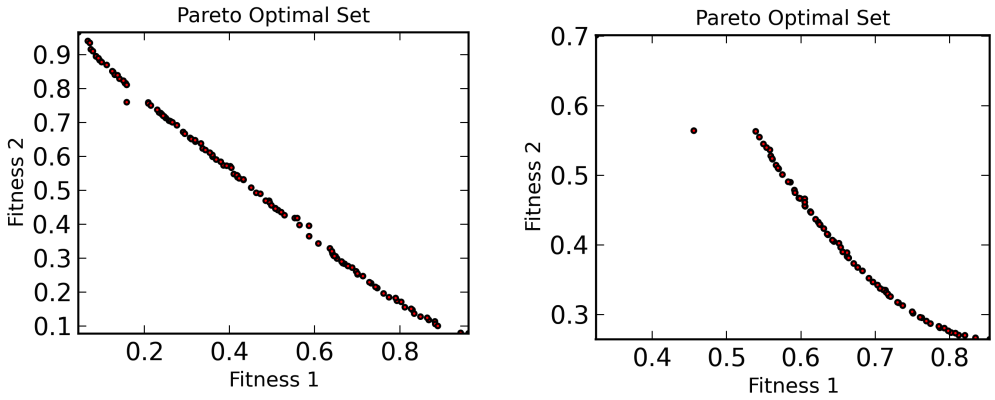


(a) Simple Retain Method (b) ϵ-dominate Method

Fig. 10: Plot of Pareto Optimal Set from two Non-dominated Retain methods.

Table 6: Average converge time for different Non-dominated Retain of methods

Methods	**Average Converge Time(ms)**
Simple Retain with replacement:	1330
ϵ-dominate retain	890



(a) Nearest-2 neighbors (b) Niche neighbors

Fig. 11: Plot of Pareto Optimal Set from two Non-dominated Retain methods.

Table 7: Average converge time for different Neighbor Density methods

Methods	**Average Converge Time(ms)**
Nearest-2 neighbors	1232
Niche neighbors	1295

Machine Learning Optimization problems on artificial continuous fitness functions. There are a little bit more work to do in the future to extend the current evaluation to discrete fitness functions which is actually the optimazation of the FPGA architecture want to achieve. The result is predictable to be good because in our previously work is successful on real FPGA problem such as Reconfigurable Software-defined Radio and Quadrature Method-based Application [1]. The further evaluation on the FPGA will be carried through the following steps:

1. For each objectives, construct a fitness function which receive parameters produces a single fitness value.
2. When apply the update of the particle's position, the new position must be dithered to a discrete parameter space.
3. Then run the system to get a Pareto Optimal Set of the parameter space.
4. Choose a best one or a best subset artificially within the Pareto Optimal Set.

By doing all of these, we can get an Pareto Optimal Set concerning several FPGA objectives, i.e. Speed, Accuracy, Power/Energy consumption, which can make a trade-off between different objectives. It is noticeable that this method successfully employs a completely new way of exploring the FPGA architecture within multi-objective scope and the result gain from it is unlikely to be achieved by a human FPGA designer within a small amount of time to consider an array of fitness functions and make best of them.

6 Regression Test

6.1 Regression Test Design

Regression testing is an integral part of the extreme programming software development method. In this method, design documents are replaced by extensive, repeatable, and automated testing of the entire software package throughout each stage of the software development cycle [29]. In the corporate world, regression testing has traditionally been performed by a software quality assurance team after the development team has completed work. Regression testing can be used not only for testing the correctness of a program, but often also for tracking the quality of its output. In our project, the regression test is very important since the quality on how good a MLO is partly on how long it takes to achieve the termination condition and how many generations it takes for the whole trial. One possible way to achieve this goal is to keep a golden result file in the same folder of configuration script and every time we run a trial based on this configuration file, it will be read and used to compare with the current result to see how good the current result is. Thus a golden file is created once and to be compared later with latter design. In every run of the trial, two main fields of criterion are measured:

counters

- Counter "g": Used for measuring how many generations in this run of MLO, obviously we want to find optimal solution to reduce the generations to reach the termination condition.
- Counter "it": Used for measuring how many times the trial evaluate the fitness funtions. Again this should be as less as possible because each evaluation of the fitness will occupy the computing capacity of the host machine it runs.

timers

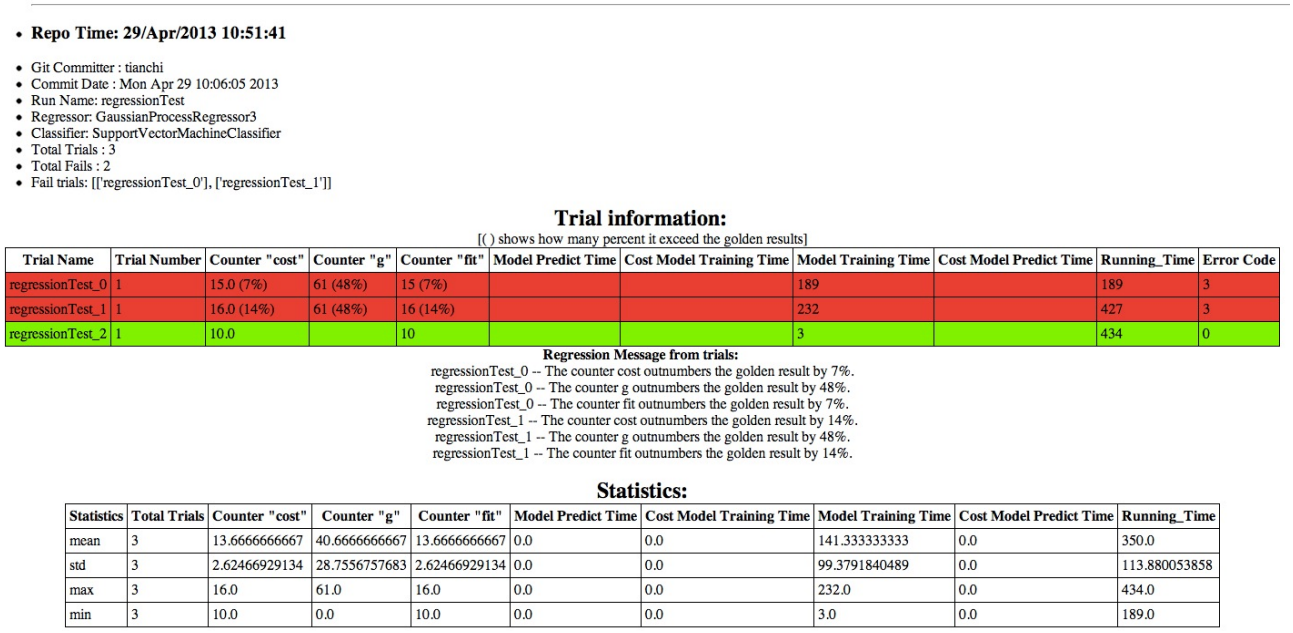
- Model Training Time: The model predict time is the time for the regressors to train the surrogate model. As this is the most time-consuming part in the whole program, it becomes the bottleneck for the improvement of the design. A substantial effort has been done to improve this part, so the time can be used as how good the current training parameters (including the kernel function, training algorithm, etc.).
- Model Predicting Time: The prediction time for the models also consume some time but not so much currently which is negligible but we still need to measure it for the future use.

- Running Time: The running time for the whole program is a important one since it tell us how long it takes to run this trial runs including the training, predicting and the meta-heuristic run time, this is a primary comparing method for measuring the trials. However, this is not always the case because two different trial with the same configuration and fitness function may have huge difference on the running time since the meta-heuristic is non-determinate. In that case, a good method to eliminate this is to have several trials running consecutively and then take the average of the running time.

6.2 Implementation

As discussed above, to develop a scheme for discovering how good current solution is when compared to the golden solution is very important. There are mainly three parts within this implementation:

- The maintain of the trial information: In the implementation a 'snapshot' function is used for take a virtual 'snapshot' for the trial, which contains all the information (name, budget, timers, counters, etc) for the trial. For the other class to get the trial information, one call to this snapshot is enough when the trial finishes and then passed to the processing function for the further comparison.



- **Repo Time: 29/Apr/2013 10:51:41**

- Git Committer : tianchi
- Commit Date : Mon Apr 29 10:06:05 2013
- Run Name: regressionTest
- Regressor: GaussianProcessRegressor3
- Classifier: SupportVectorMachineClassifier
- Total Trials : 3
- Total Fails : 2
- Fail trials: [['regressionTest_0'], ['regressionTest_1']]

Trial information:

[( ) shows how many percent it exceed the golden results]

Trial Name	Trial Number	Counter "cost"	Counter "g"	Counter "fit"	Model Predict Time	Cost Model Training Time	Model Training Time	Cost Model Predict Time	Running_Time	Error Code
regressionTest_0	1	15.0 (7%)	61 (48%)	15 (7%)			189		189	3
regressionTest_1	1	16.0 (14%)	61 (48%)	16 (14%)			232		427	3
regressionTest_2	1	10.0		10			3		434	0

Regression Message from trials:

regressionTest_0 -- The counter cost outnumbers the golden result by 7%.
regressionTest_0 -- The counter g outnumbers the golden result by 48%.
regressionTest_0 -- The counter fit outnumbers the golden result by 7%.
regressionTest_1 -- The counter cost outnumbers the golden result by 14%.
regressionTest_1 -- The counter g outnumbers the golden result by 48%.
regressionTest_1 -- The counter fit outnumbers the golden result by 14%.

Statistics:

Statistics	Total Trials	Counter "cost"	Counter "g"	Counter "fit"	Model Predict Time	Cost Model Training Time	Model Training Time	Cost Model Predict Time	Running_Time
mean	3	13.6666666667	40.6666666667	13.6666666667	0.0	0.0	141.333333333	0.0	350.0
std	3	2.62466929134	28.7556757683	2.62466929134	0.0	0.0	99.3791840489	0.0	113.880053858
max	3	16.0	61.0	16.0	0.0	0.0	232.0	0.0	434.0
min	3	10.0	0.0	10.0	0.0	0.0	3.0	0.0	189.0

Fig. 12: An example of Regression Test Html Report

- The comparison of the current solution with the golden result: For every measurement of the trial, the comparison with the golden result is done and the outnumber percentage over the golden result is calculated which will give a more straightforward way for further comparing between any two new trials. Furthermore, the generation information and the fitness counter are compared with the budget, which will used as a mark for the trial fails or not. In addition, the statistic information is calculated across all the trials in a single run, which contains the mean, standard deviation, max value and min value of all the attributes calculated above.
- The design and implementation of the details of the report and techniques to achieve this: We choose to use two different packages in python (HTML and pisa) to create the html file and the pdf report respectively. It is important to keep the trial information like the github version and the run configuration in the report, which is easy to implemented as list using the html package. The content in the head of the report is designed as follows:
  - Repo Time: 17/Apr/2013 20:49:52
  - Git Committer : tianchi

- Commit Date : Fri Apr 12 17:23:37 2013
- Run Name: example
- Regressor: GaussianProcessRegressor3
- Classifier: SupportVectorMachineClassifier
- Total Trials : 1
- Total Fails : 1
- Fail trials: 'example_0'

The next part in the report comes with the trial information on the counters and timers, and there are messages pop out for this trial under the table. It is worth to be mentioned that when the trial fails, the corresponding row will be marked red compared with the successful one in green. As discussed above, the statistics measure is very important to eliminate the randomization of the meta-heuristic so it is displayed under the trial information. The Fig. 12 is an example of the html report.

7 Conclusions and Future Work

Based on the previous successful work on the MLO, we present a new method on the Multi-objective Machine Learning Optimizer on the FPGA architecture design. The Multi-Objective MLO which mainly based on Pareto optimal solution can offer superior performance on the exploration on the parameter space in respect to several objectives and yielding "trade-off" between these objectives, which may including performance measure like power consumption and accuracy. Several approaches to achieve best performance of the approximation of the Pareto Front are done and the evaluation and comparison are shown, which indicate the good performance of using a density estimator and the ϵ-Dominance to manage the Pareto Optimal Set. Furthermore, the multi-objective optimization approach that we have presented is only tested on artificial examples, with additional work, may be simply extended to a multitude of FPGA applications which offer a discrete fitness functions.

Future works for the Multi-Objective MLO is mainly about the improvement of the performance: apply more density measures and adjust the parameters used in the algorithm to be fast to converge; extend the currently artificial continuous test function to discrete optimization problem on the real FPGA applications; improve the modeling algorithm to reduce modeling error and time because now modeling is the main bottleneck which consume a majority of time running on the application.

8 What I learned

8.1 FPGA

FPGA is not a familiar technique for me the only thing I know before is a single application often has multiple computationally intensive kernels that can benefit from acceleration using custom or reconfigurable hardware platforms, which is usually called field-programmable gate arrays (FPGAs). Then for the reconfigurable applications whose behavior can be modified by changing a few input parameters before execution are usually optimized manually, which on the contrary can be done by the optimizer automatically and this is completely a new area and it arouse my strong interest to research more on it.

8.2 Maxeler Platform for Acceleration of Dataflow programming

During the project I took part in the dataflow computing course operated by Maxeler Technology, which is extremely useful for me to apply my current Multi-Objective algorithm in the real FPGA applications. The dataflow computing was popularized by a number of researchers in the 1980's, in which an application is considered as a dataflow graph of the executable actions; as soon as the operands for the action are valid, the action is executed and the result is forward to the next action in the graph. In a Dataflow application, the program source is transformed into a Dataflow engine configuration file, which describes the operations, layout and connections of a Dataflow engine. Data can be streamed from memory into the chip where operations are performed and data is forwarded directly from one

computational unit ("dataflow core") to another, as the results are needed, without being written to the off-chip memory until the chain of processing is complete.There are three applications I was doing for exercise and they are written in Java:

- Graph brightness adjustment: use the dataflow engine (DFE) to adding value to each pixel;
- Gamma Correction: use DFE to transfer the linear input image to the output image;
- Stream Offsets: read data from different locations with a stream. An example of the dataflow programing is shown in Fig. 13

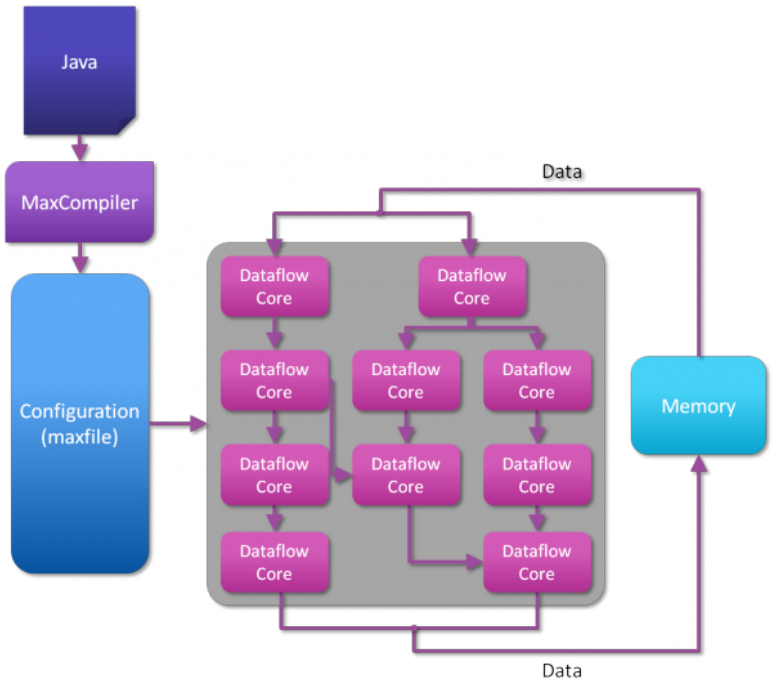


Fig. 13: Computing with dataflow cores

8.3 Machine Learning Optimizer

As discussed above reconfigurable applications whose behavior can be modified by changing a few input parameters before execution are usually optimized manually. This is a very tedious task, as the designer has to analyze the application, create benchmarks (fitness functions) and run the application with different combinations of input parameters .The objective of the MLO algorithm is to minimize the number of fitness function evaluations when solving a parameter optimization problem, which is clearly more beneficial than traditional approaches, when the fitness function takes a long time to return. The motivation for this was for optimizing parameters when designing heterogeneous computer systems, but it is useful in any parameter optimization problem with a time-consuming fitness function.

8.4 Meta-heuristic

The PSO and Multi-Objective PSO are the main knowledge I have learning during this project since the performance of the MLO is mainly based on them. PSO is a helpful method for any optimization problems which want to find an optimum as well as avoid premature converge to local minimum. My main work on the project lies on the Multi-Objective PSO and during the development I have to make my best to adjust the algorithm to fit our work on the MLO as well as improving the performance on it. Admittedly, it is worth to work several years on improving it but at least now I have learnt to adjust an combine different solutions to achieve my goal and this effort helps me make a further step in this area.

8.5 Regression Test

Regression Test is very helpful for any project that want to seek optimized performance during development, through which we can compare our result to the golden result to take a judgment of the current design. There is one thing to notice that the result of one trial is not enough for comparison in any project which has the non-deterministic characteristic, and the alternative way to do it is to take the statistical result of several trials which will eliminate the error.

8.6 Python Skills

Python is a very powerful programming language especially for any projects that need to deal with a mass of data. The library and package based on it is abundant which greatly reduce the work any programmers working on it. Although python is not new to me, I have learnt to use several python packages like the deap package for evolutionary algorithm, scikit-learn for machine learning and so on. Additionally, through working on the project which mainly programming on the object-orient python, I have developed my skills greatly on it.

8.7 Project Management

Project management is another primary skill what I learnt during this project, because before I've never participated a group project which uses the git version control. GitHub is the most well-known git repository host, with over 2.9 million people currently using it. At the beginning, I almost knew nothing about it and I have to study from scratch on how to use it. I have to admit that this process is tough and I've made some mistake during it, but at last I can manage it well.

Acknowledgements My Independent Study Option(ISO) is supervised by Professor Wayne Luk and his PhD student Maciej Kruek, they give me instructions and suggestions during this term. Without their efforts I won't have learnt so many things within only one term.

References

1. M. Kurek, T. Becker, and W. Luk, "Parametric optimization of reconfigurable designs using machine learning," in *ARC*. Springer, 2012.
2. M. Kurek and W. Luk, "Parametric Reconfigurable Designs with Machine Learning Optimizer," in *FPT*, 2012.
3. C. Pilato and et al., "Improving evolutionary exploration to area-time optimization of FPGA designs," *J. Syst. Archit.*, vol. 54, no. 11, pp. 1046–1057, 2008.
4. M. Seeger, "Gaussian processes for machine learning," *International Journal of Neural Systems*, vol. 14, pp. 69–106, 2004.
5. C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.
6. S. Guoshao and J. Quan, "A cooperative optimization algorithm based on gaussian process and particle swarm optimization for optimizing expensive problems," in *CSO*, vol. 2, 2009, pp. 929–933.
7. C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.
8. R. C. Eberhart and Y. Shi, "Comparison between comparison between genetic algorithms and particle swarm optimization." *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pp. 611–619, March 1998.
9. J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco,California,: Morgan Kaufmann, 2001.
10. M. Reyes-Sierra and C. A. C. Coello, "Multi-objective particle swarm optimizers: A survey of the state-of-the-art," *International Journal of Computational Intelligence Research*, vol. 2, pp. 287–308, 2006.
11. J. Moore and R. Chapman, "Application of particle swarm to multiobjective optimization." Department of Computer Science and Software Engineering, Auburn University, 1991.
12. *Nonlinear programming*, no. 481–492. University of California Press, 1951.
13. U. Baumgartner, C. Magele, and W. Renhart, "Pareto optimality and particle swarm optimization," *IEEE Transactions on Magnetics*, pp. 1172–1175, March 2004.
14. X. Hu and R. Eberhart, "Multiobjective optimization using dynamic neighborhood particle swarm optimization," *Congress on Evolutionary Computation (CEC'2002)*, vol. 2, pp. 1677–1681, May 2002.
15. *Multiobjective optimization using parallel vector evaluated particle swarm optimization*, vol. 2, February 2004.
16. *Autonomous agent response learning by a multi-species particle swarm optimization*, vol. 1, June 2004.
17. J. Moore and R. Chapman, *Application of particle swarm to multiobjective optimization*. Department of Computer Science and Software Engineering: Auburn University, 1999.

18. T. Ray and K. Liew, "A swarm metaphor for multiobjective design optimization," *Engineering Optimization*, pp. 141–153, March 2002.
19. J. E. Fieldsend and S. Singh, "A multiobjective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence." *Proceedings of the 2002 U.K. Workshop on Computational Intelligence*, pp. 37–44, September 2002.
20. C. A. C. Coello and M. S. Lechuga, "Mopso: A proposal for multiple objective particle swarm optimization," *Congress on Evolutionary Computation (CEC'2002)*, vol. 2, pp. 1051– 1056, May 2002.
21. G. T. P. Carlos A. Coello Coello and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, pp. 256–279, June 2004.
22. M. Mahfouf, M.-Y. Chen, and D. A. Linkens, "Adaptive weighted particle swarm optimisation for multi-objective optimal design of alloy steels," *Parallel Problem Solving from Nature - PPSN VIII*, September 2004.
23. M. H.-y. Zhang Xiao-hua and J. Li-cheng, "Intelligent particle swarm optimization in multiobjective optimization," *Congress on Evolutionary Computation*, pp. 714–719, September 2005.
24. K. D. Eckart Zitzler and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results." *Evolutionary Computation*, 2000.
25. C. A. C. Coello, D. A. V. Veldhuizen, and G. B. Lamont, "Evolutionary algorithms for solving multi-objective problems," New York, May. 2002.
26. S. A. Kalyanmoy Deb, Amrit Pratap and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga–ii," in *IEEE Transactions on Evolutionary Computation*, 2002.
27. K. D. Marco Laumanns, Lothar Thiele and E. Zitzler., "Combining convergence and diversity in evolutionary multi-objective optimization." in *Evolutionary Computation*, 2002.
28. S. Mostaghim and J. Teich, "The role of ϵ-dominance in multi objective particle swarm optimization methods," in *Congress on Evolutionary Computation (CEC'2003)*, vol. 3, Canberra, Australia, December 2003, pp. 1764–1771.
29. "Regression testing." [Online]. Available: http://en.wikipedia.org/wiki/Regression_testing