MULTI-OBJECTIVE SELF-OPTIMIZATION OF RECONFIGURABLE DESIGNS WITH MACHINE LEARNING

Maciej Kurek, Tianchi Liu, Wayne Luk [*]

Department of Computing
Imperial College London
180 Queen's Gate, London SW7 2BZ, England
email: mk306@imperial.ac.uk, tianchi.liu12@imperial.ac.uk, wl@imperial.ac.uk

ABSTRACT

Optimizing reconfigurable designs is a complex task that usually involves manual design analysis and subsequent tweaking. We present a new Multi-Objective Machine Learning Optimizer (MOMLO) which supports self-optimization of reconfigurable designs through automatic analysis and adaptation of design parameters. From a number of benchmark executions our tool automatically derives the characteristics of the parameter space and creates a surrogate model covering the multiple objectives of the design. The resulting Pareto fronts of possible design configurations can be used for self-optimization at run time. For example, we can switch between a fast but power hungry design and a relatively slow but low power alternative. We evaluate the algorithm using a multi-objective example consisting of power and throughput benchmarks.

1. INTRODUCTION

In previous work [1, 2] we have demonstrated automatic optimization for reconfigurable designs by constructing surrogate models of fitness functions which represent the design quality of parameterized designs. We now extend our work to optimize for multiple competing design aspects such as power efficiency, performance and accuracy. Our new Multi-Objective Machine Learning Optimizer (MOMLO) aims to discover a set of balanced solutions with respect to several objectives and represent them in a Pareto optimal front. A surrogate model of all the design objectives is constructed, which brings substantial savings since its evaluation is orders of magnitude faster than generation of bitstreams and code execution of benchmarks. Our MOMLO approach results in a substantially reduced design effort compared to traditional approaches which require the designer to manually analyze the application, create models and benchmarks, and subsequently optimize the design [3, 4, 5, 6]. Furthermore, we support self-optimization at run time where an optimal design variant can be reconfigured based on dynamically changing operating conditions or environments by extracting the most suitable design from the discovered Pareto optimal front. The contributions of this paper are:

- A new MOMLO. We show how multiple Bayesian regressors, classifiers and multi-objective meta-heuristics can be interlinked (Section 3).
- An evaluation of the extended MOMLO approach using a case study where a quadrature based financial application with varied precision is optimized for throughput and power consumption (Section 4).

2. BACKGROUND

When developing reconfigurable applications, designers are often confronted with a very large parameter space. As a result parameter space exploration can take an immense amount of time. A number of researchers approach the problem of high-cost fitness functions and large design spaces in various fields by having fitness functions combined with fast-to-compute Gaussian Process (GP) surrogate models for decreasing evaluation time [7, 8, 9, 10, 11]. However most current surrogate models only consist of a regressor and rarely take into account invalid configurations within the design space. Surrogate models approximating fitness functions by substituting lengthy evaluations with estimations based on closeness in a design space have been investigated in reconfigurable computing [12]. The work covers surrogate models for circuit synthesis from higher level languages, rather than parameter optimization. In previous work [1, 2] we have shown it useful to construct surrogate models of fitness functions representing the design quality of reconfigurable parameterized designs. The optimization approach we developed involves the following steps:

1. Build application and a benchmark returning design quality metrics.

[*]This work is supported by the European Union Seventh Framework Programme under grant agreement number 248976, 257906, 287804 and 318521, by UK EPSRC, by Maxeler University Programme, and by Xilinx.

2. Specify search space boundaries and optimization goal.

3. Create analytical models for the design.

4. Create tools to explore the parameter space.

5. Use the tools to find optimal configurations, guided by the models in step 3.

6. If result is not satisfactory, redesign.

When using Machine Learning Optimizer (MLO) the user supplies a benchmark along with constraints and goals, and the MLO automatically carries out the optimization. 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.

2.1. Gaussian Process Regression

GP is a machine learning technology based on strict theoretical fundamentals and Bayesian theory [13]. GP does not require a predefined structure; can approximate arbitrary function landscapes including discontinuities, and includes a theoretical framework for obtaining optimum hyper-parameters [10]. An advantage of GP is that it provides a predictive distribution, not a point estimate.

A Gaussian process is a collection of random variables, a finite set of which have a joint Gaussian distribution. A Gaussian process is completely specified by its mean function $m(\mathbf{x})$ and the covariance (kernel) function $k(\mathbf{x}, \mathbf{x}')$. The goal is regression: $\hat{f}(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$

The function $k(\mathbf{x}, \mathbf{x}')$ describes 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, and 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.

2.2. Support Vector Machines Classification

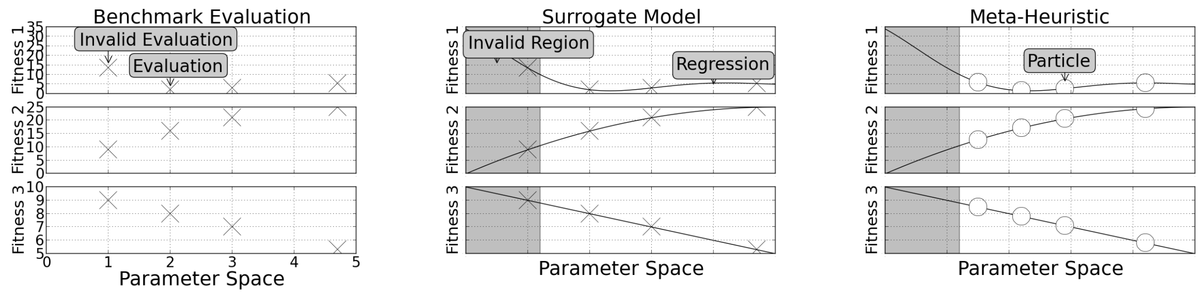
Support Vector Machine (SVM) is a maximum margin classifier, which constructs a hyperplane used for classification (or regression) [14]. SVMs use kernel functions $k(\mathbf{x}, \mathbf{x}')$ to transform the original feature space to a different space with a linear model used for classification. SVMs are a class of decision machines and 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.

2.3. PSO

Particle Swarm Optimization (PSO) is a population-based meta-heuristic based on the simulation of the social behavior of birds within a flock [15]. 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, with each particle displaced based on its velocity v_{id}. The criteria for termination of the PSO algorithm can vary, and usually are determined by a time budget. The variable x_{id} represents the dth coordinate of particle i from the set X_* of N particles, where each particle is a point within $\mathcal{X}$. Multi-objective optimization is usually approached by finding a Pareto optimal set of the underlying fitness functions. The original PSO algorithm was designed to cope with single-objective optimization problems, multiple different flavors have been developed to cope with multi-objective optimization [16, 17]. Many more sophisticated multi-objective meta-heuristic algorithms have been developed [18]. The designer has to access his requirements in terms of performance and robustness when making a decision which algorithm to choose. 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 compromise among the objectives. In other words, we wish to find the Pareto optimal set $\mathcal{P}*$ which is an approximation of the Pareto Front $\mathcal{PF}*$ [19].

3. OPTIMIZATION APPROACH

The optimization approach is inspired by that of MLO. The idea of multi-objective surrogate modeling is illustrated in Fig. 1. The MOMLO algorithm explores the parameter space by evaluating different benchmark configurations as presented in Fig. 1a. Seen in Fig. 1b the results obtained during evaluations are used to build surrogate model which provide regressions of the fitness function multiple-metrics and identifies invalid regions of the parameter space. A multi-



(a) (b) (c)

Fig. 1: Benchmark evaluations, surrogate model and model guided search for a problem with three conflicting objectives.

objective PSO guides the exploration of the parameter space using the surrogate model, as shown in Fig. 1c. The main novelty is that the result of optimization is a Pareto optimal set of designs $\mathcal{P}*$, allowing the design to self-adapt when processing circumstances.

3.1. Fitness Function

The parameter space $\mathcal{X}$ of a reconfigurable design is spanned by discrete and continuous parameters determining both the architecture and physical settings of Field programmable gate array (FPGA) designs. Given a parameter setting $\mathbf{x} \in \mathcal{X}$, the benchmark consists of a vector of fitness metric $[y_1, y_2, .., y_i] = \mathbf{y}$ and t, the exit code of the application. A function $b_i(\mathbf{x}) = y_i$ represents one of the K objectives. Execution time and power consumption are examples of possible objectives. Vector $\mathbf{y}$ consists of multiple fitness measures when the designer wants to find an optimal design defined in terms of a number of qualities. For example the $\mathbf{y}$ vector could constitute of execution time and power usage, if the aim is to find the set of power efficient designs. There are many possible exit codes t, with 0 indicating valid $\mathbf{x}$'s. The designer can choose to extend the benchmark to return additional exit codes depending on the failure cause, such as configurations producing inaccurate results or failing to build.

We distinguish three different types of exit codes. The first type is exit code 0 indicating a valid design. The second type of exit codes indicate configurations that produce results yet fail at least one constraint making them undesirable. The third type of exit codes are used for configurations that fail to produce any results. The region of $\mathcal{X}$ that defines configurations $\mathbf{x}$ that produce $\mathbf{y}$ and satisfy all constraints is defined as valid region $\mathcal{V}$, regions with designs failing at least one constraint yet producing $\mathbf{y}$ are part of failed region $\mathcal{F}$, and the region with designs failing to produce $\mathbf{y}$ is the invalid region $\mathcal{I}$. If $\mathbf{x}_*$ does not produce a valid result, we assign a value that the designer assumes to be the most disadvantageous. Depending on whether we face a minimization or a maximization problem for a given objective function f_i either a ∞ or $-\infty$ value will be assigned as presented in Eq. 1.

$$f_i(\mathbf{x}) = \begin{cases} y_i & \mathbf{x} \in \mathcal{V} \\ \pm\infty & otherwise \end{cases} \tag{1}$$

3.2. MLMO Algorithm

We integrate a GP regressor $\hat{f}$ and an SVM classifier to create a novel surrogate model of fitness function f. As illustrated in Fig 1, the problem we face is regression of f over $\mathcal{V}$ and $\mathcal{F}$ as well as classification of $\mathcal{X}$. We make use of GP to access the standard deviation estimate $\sigma(\mathbf{x}_*)$ of non-examined parameter configurations $\mathbf{x}_*$. We use SVMs to predict exit codes of X_* across $\mathcal{X}$. Regression $\hat{f}_i$ for a function f_i is created using the training set obtained from benchmark execution $\mathcal{D}_{Ri}$, while classification is done using the training set $\mathcal{D}_C$. We invoke regressions $\hat{f}_i(\mathbf{x}_*)$ for every particle in $\mathbf{X}_*$ and for every function f_i and aggregate the results to obtain the regression $[\hat{f}_1(\mathbf{x}_*), \hat{f}_2(\mathbf{x}_*), .., \hat{f}_n(\mathbf{x})] = \hat{f}(\mathbf{x}_*) = \mathbf{y}_*$ and its uncertainty vector $[\sigma_1(\mathbf{x}_*), \sigma_2(\mathbf{x}_*), .., \sigma_n(\mathbf{x})] = \sigma_*(\mathbf{x})$, which is the standard deviation estimate. Exit code t_* of particle $\mathbf{x}_*$ is predicted by the classifier.

In our MOMLO algorithm, we adopt 1) density measure [20] (indicates the closeness of the particles within the swarm) as the criterion to choose the leader for particles, i.e. guide the population to spread out along real Pareto frontier as fast as possible; 2) "ϵ-dominance" method [21] to retain a non-dominated solution to the Pareto Front, which is believed to be able to generate well-formed Pareto optimal set as well generate the front evaluating fewer fitness functions. We present the MOMLO in Algorithm 1. The algorithm integrates a classifier to account for invalid regions of $\mathcal{X}$. We initialize the meta-heuristic of our choice with N particles X_* randomly distributed across the parameter space. Each particle has an associated fitness $\mathbf{x.fit}$ and a position $\mathbf{x}$. For all $\mathbf{x}_*$ predicted to lie in $\mathcal{V}$ we proceed as follows: whenever $\sigma_{max}(\mathbf{x}_*)$, the largest value out of all σ_i, returned by the GP is below a credible interval min_σ we

Algorithm 1 MOMLO

```
 1: for $\mathbf{x}_* \in X_*$ do
 2:     $\mathbf{x}_*$.fit $\leftarrow f(\mathbf{x}_*)$  ▷ Initialize with a uniformly randomized set for every objectives $f_i*$ in the fitness function.
 3: end for
 4: repeat
 5:     for $\mathbf{x}_* \in X_*$ do
 6:         if $\sigma_{max}(\mathbf{x}_*) < min_\sigma$ and $t_* = 0$ then
 7:             $\mathbf{x}_*$.fit $\leftarrow \mathbf{y}_*$
 8:         else
 9:             if $t_* = 0$ then
10:                 $\mathbf{x}_*$.fit $\leftarrow f(\mathbf{x}_*)$
11:             else
12:                 for $i \in 1, 2, ..., K$ do  ▷ Depending on the objective of each of the fitness function either $\infty$ or $-\infty$ is assigned
13:                     $\mathbf{x}_*.fit_i \leftarrow \pm\infty$
14:                 end for
15:             end if
16:         end if
17:     end for
18:     $X_* \leftarrow Meta(X_*)$  ▷ Iteration of the meta-heuristic
19: until Termination Criteria Satisfied
```

use the prediction y_*; otherwise we assume the prediction to be inaccurate and evaluate $f(\mathbf{x}_*)$. This step is required and happens in a situation when at least one of the underlying f_i function is not modeled accurately. Although individual $f_i(\mathbf{x}_*)$ could be evaluated, usually the cost of evaluation of a single f_i is marginally smaller then cost of evaluation of f. Based on our experience values within the range of 0.01 and 0.1 are most practical for min_σ. Larger credible interval will usually hinder MOMLO performance due to high admissible uncertainty which is especially problematic when the mean estimate is relatively small. The meta-heuristic will avoid $\mathcal{I}$ and $\mathcal{F}$ regions as they are both assigned unfavorable $\pm\infty$ values. Whenever $f(\mathbf{x})$ is evaluated, $(\mathbf{x}, t)$ is included within the classifier training set $\mathcal{D}_C$. If exit code is valid ($t = 0$), then $(\mathbf{x}, y_i)$ is added to $\mathcal{D}_{Ri}$.

4. EVALUATION

In [4] the designer explores trade-off between accuracy and throughput in a quadrature-based financial application with three parameters. The first two parameters are mantissa width m_w of the floating point operators and the number of computational cores $cores$. Larger number of m_w bits increases computation accuracy, but limits the maximum number of $cores$ that can be implemented on the chip due to the increased size of the individual core. The third parameter is the density factor d_f which specifies the density of quadratures used for integral estimation. It is a software parameter and is independent of the generated bitstream. Density factor d_f increases computation time per integration while improving the accuracy of the results due to finer estimation.

The optimization goal is to find the design offering the highest throughput of integrations per second ϕ_{int} (f_1) and the lowest power W (f_2) given a required minimum accuracy defined in terms of root mean square error ϵ_{rms}. The error is defined with respect to results obtained by calculating a set of reference integrals at the highest possible precision. MOMLO terminates when the globally optimal configuration for a given ϵ_{rms} is found. The $\mathcal{F}$ region contains the inaccurate result class. The design space $\mathcal{X}$ is defined as $m_w \times cores \times d_f$: $\{11-53\} \times \{1-16\} \times \{4-32\}$. We repeated the experiment for different error limits ϵ_{rms} 10 times; we found that in order for the approximate front to cover the real Pareto front we require around 158 ($\epsilon_{rms} = 0.1$), 116 ($\epsilon_{rms} = 0.05$) and 91 ($\epsilon_{rms} = 0.01$) fitness function evaluations. By coverage we understand that around 30% of designs within the approximate front will reside on the real front and around 35% will match it within a 5% performance limit. The approximate front includes more designs, and 50% of the designs from the real front reside within it. The rest of the designs have a higher discrepancy (around 10%) due to surrogate model inaccuracies. The coverage can be improved by increasing the number of fitness evaluations.

When comparing MOMLO to the single-objective MLO [2] the increase of the number of required fitness function evaluations to reach the termination criteria is noticeable and dependant on the size of valid area. The increase in fitness function evaluations is 15% (0.1), 73% (0.05) and 94% (0.01) for the evaluated error limits. Longer optimization time of multi-objective problems is expected as the problems complexity is increased with respect to single-objective optimization. Although the overhead can be significant, it seems to decrease as the size of valid area increases (increased ϵ_{rms}). As presented in [2] the manual optimization procedure requires 420 fitness function evaluations to find an optimal design for a given ϵ_{rms}. In best case scenario the number would not be increased for multiple objectives meaning MOMLO would still offer superior performance. The drawback of MOMLO is the lack of guaranty of finding the true Pareto optimal front.

5. CONCLUSIONS AND FUTURE WORK

Our MOMLO can be used to create a self-adaptive system which can constantly improve its knowledge of the designs Pareto optimal configuration set, and switch between design configurations depending on the current environment. The algorithm is showing a lot of promise, yet its evaluation is limited and requires further investigation. We are preparing a number of new multi-objective evaluation cases which should help us to access MOMLO robustness and performance. Furthermore we are investigating a distributed version of the algorithm, allowing for a parallelized approach and hence faster optimization when the compute resources are available. This allows for an optimization approach where the algorithm self-adapts the optimization strategy to balance its search speed and efficiency.

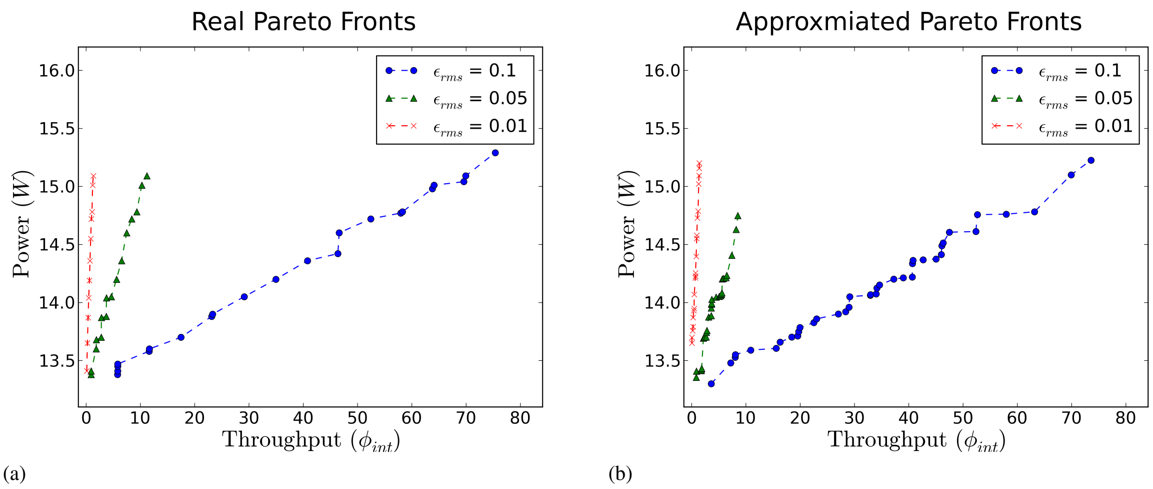


Fig. 2: The Real and Approximated Pareto Fronts for different ϵ_{rms} limits.

6. REFERENCES

[1] M. Kurek and W. Luk, "Parametric reconfigurable designs with machine learning optimizer," in *FPT*, 2012, pp. 109–112.

[2] M. Kurek, T. Becker, and W. Luk, "Parametric optimization of reconfigurable designs using machine learning," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7806, pp. 134–145.

[3] X. Niu and et al., "Exploiting run-time reconfiguration in stencil computation," in *FPL 2012*, 2012, pp. 173–180.

[4] A.H.T. Tse and et al., "Optimising performance of quadrature methods with reduced precisions," in *ARC*. Springer, 2012, pp. 251–263.

[5] T. Becker, W. Luk, and P. Y. Cheung, "Parametric design for reconfigurable software-defined radio," in *ARC*. Springer, 2009, pp. 15–26.

[6] Q. Jin and et al., "Optimising explicit finite difference option pricing for dynamic constant reconfiguration," in *FPL*, 2012, pp. 165–172.

[7] A. I. Forrester and D. R. Jones, "Global optimization of deceptive functions with sparse sampling," in *AIAA/ISSMO*. American Institute of Aeronautics and Astronautics, September 2008.

[8] Y. Jin, M. Olhofer, and B. Sendhoff, "A framework for evolutionary optimization with approximate fitness functions," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 481–494, 2002.

[9] Y. S. Ong, P. B. Nair, and A. J. Keane, "Evolutionary optimization of computationally expensive problems via surrogate modeling," *AIAA*, vol. 41, no. 4, pp. 689–696, 2003.

[10] 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.

[11] H.A.L. Thi, D.T. Pham, and N.V. Thoai, "Combination between global and local methods for solving an optimization problem over the efficient set," *EJOR*, vol. 142, no. 2, pp. 258–270, 2002.

[12] C. Pilato, A. Tumeo, G. Palermo, F. Ferrandi, P. L. Lanzi, and D. Sciuto, "Improving evolutionary exploration to area-time optimization of FPGA designs," *J. Syst. Archit.*, vol. 54, no. 11, pp. 1046–1057, 2008.

[13] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[15] F. Van Den Bergh, "An analysis of particle swarm optimizers," Ph.D. dissertation, University of Pretoria, South Africa, 2002.

[16] X. Hu and R. Eberhart, "Multiobjective optimization using dynamic neighborhood particle swarm optimization," in *IEEE CEC*, vol. 2, 2002, pp. 1677–1681.

[17] J. Liang, B. Qu, P. Suganthan, and B. Niu, "Dynamic multi-swarm particle swarm optimization for multi-objective optimization problems," in *IEEE CEC*, June, pp. 1–8.

[18] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evol. Comput.*, vol. 8, no. 2, pp. 173–195, Jun. 2000.

[19] M. Reyes-Sierra and C. 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.

[20] K. Deb and et al., "A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2000.

[21] M. Laumanns and et al., "Combining convergence and diversity in evolutionary multiobjective optimization," *Evol. Comput.*, vol. 10, no. 3, pp. 263–282, Sep. 2002.