ASYNCHRONOUS MASTER SLAVE PARALLELIZATION OF EVOLUTIONARY OPTIMIZATION IN AIRFOIL SHAPE DESIGN *

DANIELA ZAHARIE , MIRCEA DRĂGAN , SILVIU PANICA , DANA PETCU†, AND MARIUS STOIA-DJESKA‡

Abstract. The aim of this work is to analyze an asynchronous master-slave parallelization strategy for evolutionary algorithms used in solving multiobjective optimization problems arising in airfoil shape design. Numerical tests proved the superiority of this strategy over the classical synchronous one both in terms of time efficiency and with respect to the solutions quality.

1. Introduction. Optimal airfoil design means finding wing sections (profiles) which simultaneously optimize several criteria and satisfy some constraints. The characteristics to be optimized (e.g. aerodynamic coefficients like lift and drag) are not easy to estimate making difficult the application of gradient-based methods. Moreover the multiobjective character of the optimization problem makes the problem even more difficult. As derivative-free methods which work with populations of candidate solutions, the evolutionary algorithms (EAs) proved to be adequate methods in solving multiobjective optimization problems arising in engineering. Unfortunately EAs are stochastic iterative methods which need the evaluation of each set of design variables at each iteration. In the case of the optimal airfoil shape design this means to estimate for each candidate profile the aerodynamic characteristics by simulating the flow around it. This leads to high computational costs which can be reduced by using surrogate models (meta-models) for the evaluation of the aerodynamic characteristics [11] and/or by evaluating in parallel the elements of the population [8, 9]. This simultaneous evaluation of several elements corresponds to the so-called master-slave parallelization model of evolutionary algorithms [1] where the evolutionary operators used to construct the population of new candidates are executed by the master while the slaves are used only to evaluate the objective function(s).

Most of master-slave parallelization variants of EAs are based on a synchronous communication between the master and slave processes, i.e. at each generation of the evolutionary algorithm the master waits for answers from all slaves before starting to construct new elements. The synchronous strategy is not efficient when there are significant differences between the times needed by different slaves to accomplish their task. Such a situation arises when different evaluation models are used or because of the heterogeneous character of the computational infrastructure (e.g. as in the case of the grid enabled frameworks [9]). To deal with such situations in [3] is proposed a general template to design master-slave parallel algorithms based on the idea of using a pool of tasks and asynchronous communication protocols. This parallelization tool was successfully applied for adaptive integration of multidimensional integrals [3] and to solve global optimization problems arising in civil engineering [2]. In the context of evolutionary multiobjective optimization most master-slave parallel variants are based on a synchronous communication between the master and slaves. Notable exceptions are some recent works by Durillo et al. [6] and Szymanski et al. [10].

In this paper we analyze the behavior of an asynchronous parallelization strategy for multiobjective evolutionary algorithms, in the context of optimal design of

*This work is supported by the project RO-CEEX 65/ 31.07.2006 (SIAPOM)

†Department of Computer Science, West University of Timisoara and e-Austria Institute, blvd. V. Parvan, 4, 300223 Timisoara ({dzaharie,dragan,silviu,petcu}@info.uvt.ro).

‡Politehnica University of Bucharest, Romania(marius.stoia@rosa.ro).

airfoils. The problem of airfoil shape design is described in Sect. 2 while the evolutionary approach is presented in Sect. 3. Previous work on parallelization strategies of evolutionary algorithms is shortly reviewed in Sect. 4. The parallelization strategy and numerical results are presented in Sects. 5 and 6, respectively.

2. The Airfoil Shape Design Problem. The shape of an airfoil can be described by the parameters of two interpolation curves, one corresponding to the upper side and the other one to the lower side. In our analysis we used the following equations to describe these curves:

$$y^{u,l}(x) = 5v_0^{u,l}(v_1^{u,l}\sqrt{x} + x(v_2^{u,l} + x(v_3^{u,l} + x(v_4^{u,l} + v_5^{u,l}x)))), \qquad x \in [0,1] \tag{2.1}$$

where y^u defines the upper curve while y^l defines the lower one. Thus each profile is described by 12 design variables (v_i^u and v_i^l for $i \in \{0,\ldots,5\}$) satisfying the following constraints:

$$\sum_{i=1}^{5} v_i^u = 0 \qquad \text{and} \qquad \sum_{i=1}^{5} v_i^l = 0. \tag{2.2}$$

In the case of symmetric profiles the parameters corresponding to the lower and upper sides are related as follows: $v_0^u = -v_0^l$ and $v_i^u = v_i^l$ for all $i \in \{1,\ldots,5\}$. For instance, by considering a symmetric profile characterized by the following values for the coefficients corresponding to the upper side: $v_0^u = 0.12$, $v_1^u = 0.2969$, $v_2^u = -0.126$, $v_3^u = -0.3537$, $v_4^u = 0.2843$, $v_5^u = -0.1015$ one obtains a slightly modified version of the NACA0012 reference profile. This profile was used in our tests as starting point for generating new profiles satisfying optimization criteria related to some aerodynamic characteristics. In our analysis we computed for each profile the lift and the drag coefficients (C_L and C_D respectively) given by $C_L = 2L/(\rho c\|V\|^2)$ and $C_D = 2D/(\rho c\|V\|^2)$, where ρ is the density of the fluid, $\|V\|$ is the norm of the airplane velocity vector and c is the profile chord (e.g. $c = 1$). L and D are components of the integral of the pressure on the profile border. The optimization task we consider consists in minimizing the drag coefficient (C_D) under the constraint that the lift coefficient (C_L) is as close as possible to a given value, C_L^* . This constrained optimization problem can be transformed into a bi-objective one which minimizes both C_D and $|C_L - C_L^*|$. Such optimization criteria correspond to a compressible transonic/supersonic flow (Mach number$\geq$ 1) for a zero angle of attack. In order to analyze the impact of the complexity of the evaluation model on the efficiency of the parallel implementation we used two types of evaluation of the aerodynamic coefficients. The first variant is based on a rough estimation of C_D which does not use a flow simulation but the following approximation:

$$C_D^0 = \frac{2}{cb}\sum_{j=1}^{n-1}\frac{(y^u(x_{j+1}) - y^u(x_j))^2 + (y^l(x_{j+1}) - y^l(x_j))^2}{x_{j+1} - x_j} \tag{2.3}$$

where

$$x_j = 0.5c(1 + \cos(\pi(1 - (j-1)/(n-1)))), \quad j = \overline{1,n} \tag{2.4}$$

and $b = \sqrt{\text{mach}^2 - 1}$. In the above equations n denotes the number of discretization points taken on each side of the profile. The rough estimation, C_D^0, of the drag

coefficient is used just to decide if a profile is promising and if it deserves to be further evaluated by using a flow model.

The second variant used for estimating C_L and C_D is based on the simulation of the flow by using a computational fluid dynamics module (CFD). The CFD module is based on a cell-centered, second order Godunov type finite volume discretization of the partial differential equation describing the flow. The final steady-state solution is obtained starting from an arbitrary initial state by using an explicit marching in time procedure. To avoid solving a large linear algebraic system, at each time step a dual-time approach combined with a fourth-order Runge-Kutta scheme is used. The boundary conditions are fixed in time and the convergence criterion is the reduction of the residuals under a predefined threshold.

3. The Multi-objective Evolutionary Algorithm. There currently exist a large set of evolutionary algorithms designed for multi-objective optimization (MOEAs) on continuous domains. They differ either by the variation operators (crossover and mutation) or by the selection operator. The selection operator is used to decide how are assimilated the new elements into the population. Depending on this assimilation process there are two main variants: generational and steady state MOEAs. In the first case the assimilation process is activated only when a set of new elements is generated and is usually based on a sorting process applied to the joined population containing both the old elements (parents) and the newly generated elements (offsprings). This is the case of the generational NSGA-II algorithm [4] which uses both a primary sorting criteria (the non-dominance rank) and a secondary one (a crowding factor) in order to enhance the diversity of the approximated Pareto optimal set. In the steady state variants the new elements are assimilated sequentially as they are generated. In the case of single objective optimization this sequential assimilation of elements is easy to implement being based on replacing the worst element of the population with the new one, if this new element is better. In the case of MOEAs the concept of worst element is not so easy to define, thus the steady state variants are less used and studied than their generational counterparts.

The evolutionary algorithm starts with an initial population of design vectors, $(v_1, \ldots, v_m)$ $(v_i \in [-1, 1]^{12})$ obtained by random perturbations of the design variables corresponding to the NACA0012 profile, described in Sect. 2. At each generation a population of offsprings is created by applying a reproduction operator typical to differential evolution algorithms [7]. This means that for each v_i an offspring w_i is obtained by randomly mixing components from v_i and from a trial vector computed by using three randomly selected but distinct elements (v_{r_1}, v_{r_2} and v_{r_3}) from the population. The rule to construct a new element, $w_i = (w_i^1, \ldots, w_i^n)$, is:

$$w_i^j = \begin{cases} v_{r_1}^j + F \cdot (v_{r_2}^j - v_{r_3}^j) & \text{if } U_{ij}(0,1) < CR \text{ or } j = j_0 \\ v_i^j & \text{otherwise} \end{cases}, j = \overline{1,n} \tag{3.1}$$

where $F \in (0, 2)$ is a scaling factor, $U_{ij}(0, 1)$ is a random value uniformly generated in $(0, 1)$, $CR \in (0, 1]$ is a crossover probability and j_0 is a random index. In order to satisfy the constraints imposed on the coefficients of profile curves, a repairing procedure is applied for each newly generated element. The repairing procedure consists in modifying w_i^j for $j \in \{6, 12\}$ such that $w_i^6 = -\sum_{k=2}^{5} w_i^k$ and $w_i^{12} = -\sum_{k=8}^{11} w_i^k$. After new profiles are generated they are evaluated and then a selection procedure is applied. From the joined population of old and new profiles a new population of m elements is constructed by applying a non-dominated sorting strategy as in [4]. At

each iteration of the evolutionary process the evaluation of the profiles is the most costly step.

4. Parallelization of Evolutionary Algorithms and Related Work. There are several parallelization models for evolutionary algorithms [1]: master-slave, island, cellular and hierarchical (combining for instance the island model with the cellular or with the master slave one). When the cost of elements evaluation is significantly higher than the cost of other evolutionary operators the appropriate model is the master-slave one. In this model the master process controls the entire evolution of the population by executing the evolutionary operators and by distributing evaluation tasks to the slave processes. Depending on the way the communication between the master and the slaves is conducted there are two main implementation variants: a synchronous and an asynchronous one.

The synchronous master-slave model consists in independent evolution and evaluation epochs: a new population is constructed only after the master received the results from all slaves. Thus the time efficiency of the algorithm is at each step influenced by the longest evaluation time. In the asynchronous case the generation of new elements and the evaluation are interleaved, therefore the master can send for evaluation a new element as soon as a slave is free.

In the case of evolutionary algorithms there are at least two reasons for using an asynchronous communication between master and the slave evaluators: (i) when the execution environment is a heterogeneous one (e.g. a grid environment); (ii) when the evaluation of different elements has different complexity due to the use of different evaluation models (e.g. surrogate and exact models).

The synchronous variants have been analyzed before and in the case of homogeneous evaluation tasks there exist mathematical models for predicting the speedup [5]. On the other hand, the behavior of the asynchronous versions was significantly less studied and there are only a few results [10],[6]. The results presented in [10] refers to a master slave parallelization of an EA for single objective optimization and is based on the idea of asynchronously updating the population in order to adapt to the particularities of an heterogeneous computing infrastructure.

On the other hand, the results presented [6] refer to three parallel versions of the NSGA-II [4] algorithm for multi-objective optimization: (i) a synchronous parallelization of the generational NSGA-II; (ii) an asynchronous parallelization of the generational NSGA-II characterized through the fact that the master process sends elements for evaluation to all free slave processes and construct a new generation, by applying the NSGA-II sorting procedure, only when the offspring population is filled; (iii) an asynchronous parallelization of the steady-state NSGA-II in which as soon as an element is evaluated it is assimilated into the population by applying ranking and crowding. Both asynchronous versions need significantly less time to obtain similar results with the synchronous one, especially in the case when the number of available processors is significantly larger than the population size (the synchronous parallelization cannot benefit from a number of processors larger than the population size).

The asynchronous versions of NSGA-II analyzed by Durillo et al. [6] came to our attention after we developed the asynchronous strategy described in the next section. The main difference between our approach and that described in [6] relies on the starting motivation. Our main reason was to deal with situations when there are significant differences in time between the evaluations of different elements while in [6] the aim was to develop a strategy which can take advantage of the available

processing resources.

5. The Parallelization Strategy. Since the proposed asynchronous strategy is compared with a synchronous one let us first describe the synchronous variant (see Algorithm 1). In the case of p slave processes and $m = k \cdot p$ elements in the population, the master process sends packages of k elements to all slave processes. The master process starts a new generation only after the results from all slaves are received.

Let us turn now to the problem of aerodynamic coefficients evaluation. In order to limit the number of calls of the CFD solver one can first compute a rough approximation of the drag coefficient (C_D^0) and compare it with the approximation of the same coefficient computed for the reference profile (e.g. NACA0012).

The CFD solver is then called only if C_D^0 is smaller than the value corresponding to the reference profile (in this case the element is considered to be promising). Thus there can be a significant difference between the tasks different slaves have to do. Such differences can appear also when surrogate models are used or even when the evaluation is based on the CFD solver since for different profiles the CFD simulation can involve less or more computational costs. To deal with these differences we propose the asynchronous strategy in Algorithm 2. This strategy can be applied both in the case of generational and in the case of steady state MOEAs.

Algorithm 1 The synchronous parallelization strategy in the case of p slave processors and $m = k \cdot p$ elements in the population

1: {Master:}
2: Population initialization and evaluation;
3: **for** $g \in \{1, \ldots, g_{max}\}$ **do**
4: Generate m offsprings
5: **for** $j \in \{1, \ldots, p\}$ **do**
6: Pack elements with indices in $\{(i-1) \cdot p + j | i = \overline{1,k}\}$
7: Send the package for evaluation to process j;
8: **end for**
9: **repeat**
10: Receive a package from a process;
11: Unpack the values of the corresponding elements
12: **until** all processes returned the evaluation results
13: Select m survivors for the next generation
14: **end for**

1: {Slave:}
2: **for** $l \in \{1, \ldots, g_{max}\}$ **do**
3: Get a package of k elements from the master process
4: Evaluate the received elements; Pack the results
5: Send the package containing the values of the evaluated elements to the master process
6: **end for**

The only difference between these two cases arises in Step 21 which corresponds to the assimilation of the new evaluated element in the current population. In the case of the generational MOEA, in order to limit the impact of the asynchronous evaluation strategy on the dynamics of the MOEA, the selection step is activated only after the evaluation of m elements. In both cases the selection step is based on a sorting procedure using the nondominance relationship but in the generational

Algorithm 2 The asynchronous parallelization strategy

```
{Master:}
Population initialization and evaluation;
g ← 1; Mark all processes as free
while g ≤ m g_max do
   Generate a new offspring v
   while (no free process) do
      j ← 1
      while (j ≤ p and j is not free) do
         if there is a message from j then
            get the message; mark j as free
         else
            j ← j + 1
         end if
      end while
   end while
   Send v for evaluation to process j; mark j as busy
   for each process i do
      if there is a message from process i then
         Get the result from process i
         if the result is promising then
            Assimilate the corresponding element into the offspring population
            g ← g + 1
         end if
         Mark i as free
      end if
   end for
end while
Send a termination message to all processes
```

```
{Slave:}
while not a termination message do
   Get an element v for evaluation
   Estimate C_D^0(v) by using a rough approximation
   if C_D^0(v) < C_D^0(NACA0012) then
      Estimate C_L(v), C_D(v) by using the CFD solver
   end if
   Send the evaluated element to the master; Check for a termination message
end while
```

case the sorting is applied to $2m$ elements (the joined old and new populations) while in the steady-state case it is applied to $m + 1$ elements (the population of old elements plus the newly generated element). However the dynamics of such a parallelized version of the generational MOEA is slightly different than the dynamics of the corresponding sequential version. The difference consists in the fact that some elements generated at a given generation are not necessarily involved in the selection step of that generation but, because of their slow evaluation, their assimilation is postponed for further generations. However no significant impact on the final results was observed.

6. Numerical Results and Discussion. The proposed asynchronous evaluation strategy was compared with a synchronous one(Algorithm 1). Both the synchronous and the asynchronous variants were implemented in Java using MPIJava as communication library. The flow simulation was made by calling some external codes: FreeFEM++ 2.22 (http://www.freefem. org/) for mesh generating and a Fortran executable code for the computing of the drag and lift coefficients. The tests were conducted on a cluster with 14 computing elements Intel Core 2 Duo 1.6Ghz (corresponding to 28 processing units). Each computing element is characterized by 2MB L2 cache per core, 2GB RAM, 160GB HDD and the communication speed is of 1000Mbps.

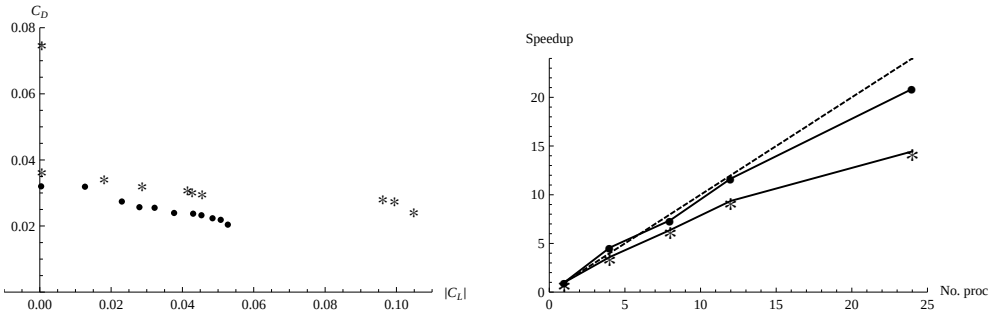

FIG. 6.1. *Comparison between the synchronous strategy (stars) and the asynchronous one (disks) (a) Pareto fronts (obtained after* 20 *generations,* $m = 24$*); (b) Speedup.*

As Fig. 6.1a illustrates, the use of a preliminary rough approximation of C_D in order to select the promising elements leads to better profiles (the solutions generated by the asynchronous variant dominates the solutions generated by the synchronous one). The profiles evolved by the evolutionary algorithm are characterized by smaller values of C_D than the reference NACA0012 profile (e.g. C_D of an evolved profile after 50 generations is around 10 times smaller than C_D(NACA0012)). Examples of evolved profiles characterized by different values of C_D and C_L are illustrated in Fig. 6.2. Even if the communication between master and slaves is more intensive in the case of the asynchronous variant (because of the unpromising elements which are not assimilated into population but still transferred between the master and the slaves) the speedup is higher than in the synchronous case (Fig. 6.1b). This can be explained by the fact that in the asynchronous case the waiting times are significantly reduced.

Another difference between synchronous and asynchronous variants is related with the averaged number of the elements evaluated by different slaves (in the synchronous case all slaves evaluate almost the same number of elements while in the asynchronous one this number is variable from one slave to another one). Moreover when the number of slaves is higher than the number of elements in the population both the average number of evaluations/slave and the total execution time decrease (see Fig. 6.3). We should also remark that using a number of slave processes larger than the population size does not significantly influence the quality of solutions.

7. Conclusions. Combining surrogate and exact models for aerodynamic coefficients estimation leads to significant differences in evaluation times of different profiles which limit the efficiency of synchronous parallelization strategies. The proposed asynchronous variant proved to be flexible (it can be applied both to generational and

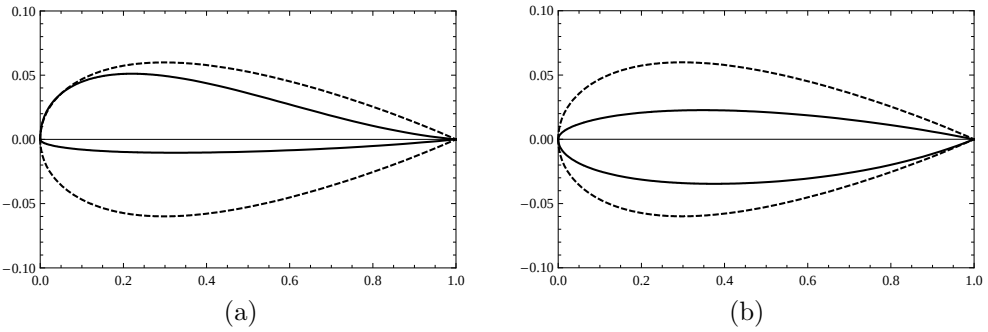

FIG. 6.2. *The reference profile (dashed line) and examples of evolved profiles (continuous line) (a) Profile with small* $|C_D|$*; (b) Profile with* C_L *close to* 0.05.

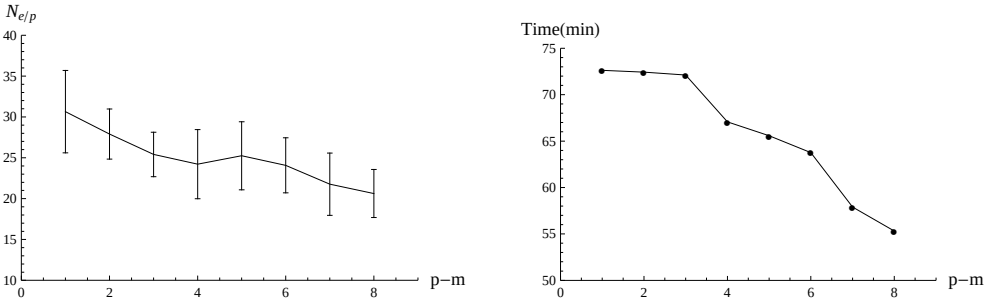

FIG. 6.3. *(a) The dependence between the mean number of evaluations per processor* ($N_{e/p}$) *and the number of slave processors exceeding the population size* ($p - m$)*; (b) The dependence between the total execution time (in minutes) and the number of slave processors exceeding the population size* ($p - m$).

steady state evolutionary algorithms) and led to benefits both in solutions quality and time efficiency. However the asynchronous communication can change the dynamics of the evolutionary process with respect to the sequential variant and its efficiency is more difficult to be theoretically analyzed than that of the synchronous one.

REFERENCES

[1] E. ALBA AND M. TOMASSINI, *Parallelism and evolutionary algorithms*, in IEEE Trans. Evol. Comput. 5 (6), 2002, pp. 443-462.

[2] M. BARAVYKAITĖ, R. BELEVIČIUS AND R. ČIEGIS, *One application of the parallelization tool of Master - Slave algorithms*, Informatica, 13(4), 2002, pp. 393-404.

[3] R. ČIEGIS, R., R. ŠABLINSKAS AND J. WASNIEWSKI, *Numerical Integration on Distributed-Memory Parallel Systems*, in M.Bubak et al. (eds), LNCS 1332, 1997, pp. 329-336.

[4] K. DEB, S. AGRAWAL, A. PRATAB AND T. MEYARIVAN, T., *A Fast Elitist Non-Dominated Sorting Genetic Algorithm For Multi-Objective Optimization: NSGA-II*, in M. Schoenauer, et al. (eds), LNCS, 1917, 2000, pp. 849-858.

[5] M.DUBREUIL, C. GAGNÉ AND M. PARIZEAU,*Analysis of a Master-Slave Architecture for Distributed Evolutionary Computations*, in IEEE Trans. on Systems, Man, and Cybernetics, 36 (1), 2006, pp. 229-235.

[6] J.J. DURILLO, A.J NEBRO, L. LUNA, L., AND E. ALBA, *A Study of Master-Slave Approaches to Parallelize NSGA-II* in Proceedings of IPDPS 2008, pp. 1-8.

[7] N.K. MADAVAN, *Multiobjective optimization using a Pareto Differential Evolution approach*, in Proceedings of CEC 2002, pp. 1145-1150.

[8] R.A.E MÄKINEN, P. NEITTAANMÄKI, J. PERIAUX, M. SEFRIOUI, AND J. TOIVANEN, *Parallel*

Genetic Solution for Multiobjective MDO, in A. Schiano et al. (eds), Algorithms and Results Using Advanced Computers, Proceedings of the Parallel CFD'96 Conference, Elsevier, 1997, pp. 352-359.

[9] H.-K. Ng, D. Lim, Y-S. Ong, B.-S. Lee, L. Freund, S. Parvez, and B. Sendhoff, *A multi-cluster Grid enabled evolution framework for aerodynamic airfoil design optimization*, in L. Wang, K. Chen and Y.S. Ong (Eds.): LNCS 3611, 2005, pp. 1112-1121.

[10] B.K. Szymanski, T. Desell, and C. Varela, *The Effects of Heterogeneity on Asynchronous Panmictic Genetic Search*, in Proceedings of PPAM 2007, LNCS 4967, 2008, pp. 457-468.

[11] Z. Zhou, Y.S. Ong, P.B. Nair, A.J. Keane, and K.Y. Lum, *Combining Global and Local Surrogate Models to Accelerate Evolutionary Optimization*, in IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 37 (1), 2007, pp. 66 - 76.