8-1-2015

Multiple Objective Evolution Strategies Using Data Envelopment Analysis

James V. Lill
Department of Defense Supercomputing Resource Center

Timothy R. Anderson
Portland State University, tim.anderson@pdx.edu

Citation Details

Lill, J. V., & Anderson, T. (2015, August). Multiple objective Evolution Strategies using Data Envelopment Analysis. In Management of Engineering and Technology (PICMET), 2015 Portland International Conference on (pp. 1969-1977). IEEE.

Multiple Objective Evolution Strategies Using Data Envelopment Analysis

James V. Lill[1], Timothy Anderson[2]
[1]Engility Corporation, Air Force Research Laboratory / Department of Defense Supercomputing Resource Center, Wright-Patterson Air Force Base, Ohio, USA
[2]Portland State University, Department of Engineering and Technology Management, Portland, Oregon, USA

Abstract--**Often in science and engineering we are faced with complicated nonlinear problems in optimization that involve simultaneously minimizing or maximizing various non-commensurate quantities. For example, a basic task in design engineering or technology management is to balance suitable measures of performance against the cost. We present a simplified approach for performing multiple objective optimization by combining standard single objective Evolution Strategies with Data Envelopment Analysis. This latter method employs linear programming to compute an L1 distance of a given solution from the Pareto frontier defined by the evolving population of solutions, or from a related frontier defined by DEA. This quantity is then used in a fitness function. Real variable linear programs must be solved for the optimization of convex problems, while the solution of mixed integer linear programs is required to optimize general non-convex problems. This hybrid method yields highly converged results with good coverage of the Pareto frontier when applied to a standardized suite of multiple objective problems. Several current applications will be discussed that employ a massively parallel program (MOES) written in C and MPI that runs on supercomputers. This material was assigned a clearance of CLEARED, Case Number 88ABW-2015-0638.**

I. INTRODUCTION

Often in science and engineering we are faced with complicated nonlinear problems in optimization that involve simultaneously minimizing or maximizing various non-commensurate quantities. For example, in design engineering one typically seeks to minimize the cost but maximize several measures of performance; in aircraft design these could include the maximum speed, range, altitude and payload, as well as the rates of climb, turn and roll. In practice one is given a set of physical parameters that characterize each competing design – e.g., the total weight, the thrust of the engine, the length and thickness of the fuselage, and the dihedral and sweepback angles of the wings, etc. Then one solves a series of complicated nonlinear problems to determine each *objective* – i.e., the cost and the performance measures. This process is repeated for each competing design. The difficulty then becomes: How do we judge the "goodness" of a particular design against the others when we must compare multiple objectives that have different physical dimensions – e.g., weight, length, velocity, etc.? Furthermore, how do we use these comparisons to alter the physical parameters of the designs and obtain better solutions?

One standard approach would be to develop some arbitrary weights that relate deviations of a given design's performance measures from their desired goals: for example, one could say that a deviation of 100 knots from the desired maximum speed was worth a deviation of 1000 meters from the desired maximum altitude, etc. A numerical score could then be assigned to each design as a *single objective* to be minimized. There are many standard algorithms available for single objective optimization – e.g., simulated annealing [12] and various evolutionary algorithms [3][5] – and these could then be used to adjust the physical parameters of the competing designs in order to evolve the "best" solution as judged by this single objective. However the effects of imposing such arbitrary constraints on the design process through the introduction of arbitrary weights to compose a single objective – and in particular, the limitations this may impose on the final design itself – are in general unknown.

An alternative is to use a *multiple objective* approach in which the deviations of all the non-commensurate quantities from their desired goals are minimized, or in which each attribute of a design is minimized (cost) or maximized (performance) without a specific goal. The difficulty now becomes how to balance the values of each objective of a given design with the corresponding objectives of the competing designs when trying to identify an optimal solution, and then manipulate the physical parameters of the designs to obtain a better solution. There are many proposed methods for multiple objective optimization [10][24] [25][26] but they are not as well-developed as are the algorithms for single objective optimization.

A resolution of this difficulty is provided by Data Envelopment Analysis (DEA) [9][1]. This is a specialized application of linear programming [8] that can be used to assign an unambiguous numerical score (the DEA *efficiency*) to each candidate solution to a multiple objective optimization problem. Furthermore, this efficiency can then be used as a fitness function in a *single objective* evolutionary algorithm to solve the original *multiple objective* optimization problem. In particular, we have combined DEA with a very powerful and flexible evolutionary algorithm for parametric optimization, Evolution Strategies (ES) [3]. The resulting multiple objective optimization method yields significant algorithmic simplifications and is quite general.

This brief report provides a qualitative discussion of the combination of DEA with ES for the non-specialist. In Section 2 we discuss the idea of Pareto dominance and its relation to DEA. In Section 3 we describe ES and its combination with DEA, as well as the concept of a *training set*. Next in Section 4 we outline our implementation of the combination of DEA with ES in the program MOES: *Multiple Objective Evolution Strategies*. MOES is written in standard C/C++ and parallelization is performed using the

standard Message Passing Interface (MPI). Finally in Section 5 we present a numerical example of the application of MOES to a standardized test problem and demonstrate the importance of convex assumptions. Mathematical details of the algorithm along with numerical comparisons of MOES to other algorithms on a standardized suite of multiple objective optimization problems [25][26] are presented elsewhere [16]. Also, additional details can be found in an application of MOES to a difficult problem in computational chemistry that has just been accepted for publication [14].

The first use of DEA in an evolutionary algorithm for multiple objective optimization appears to have been by Arakawa *et al.* [2] in 1998. Since then Yun, Nakayama, Arakawa and colleagues have continued this research [21][22][23]. (See also [17] and references therein.) A common thread in this work has been use of a generalized DEA (GDEA) model. This method includes an additional adjustable parameter that in a sense allows GDEA to interpolate between several basic DEA models [23]: the Charnes-Cooper-Rhodes (CCR), Banker-Charnes-Cooper (BCC) and the Free Disposal Hull (FDH) models [9]. MOES employs these basic DEA models and we will offer a few comments on this difference in approach later.

MOES began as a student project in the Engineering and Technology Management Department at Portland State University [15]. Its development continued under the Department of Defense High Performance Computing Modernization Program (http://www.hpc.mil/index.php) as part of the User Productivity, Enhancement, Technology Transfer and Training (PETTT) initiative (http://www.hpc.mil/index.php/2013-08-29-16-03-23/software-application-sas-overview/user-productivity-enhancement-technology-transfer-and-training). MOES is currently being used at the Army Research Laboratory (ARL) at Aberdeen Proving Grounds in Aberdeen, MD, and at the Air Force Research Laboratory (AFRL) at Wright-Patterson Air Force Base in Dayton, OH.

II. DEA AND THE PARETO FRONTIER

Comparing solutions to a single objective optimization problem is easy: In the case of minimization, one simply orders the solutions according to their objectives and picks the solution having the lowest objective. When more than one objective is involved, the situation would appear to require some arbitrary weights be introduced to form an appropriate sum to be minimized. An alternative is provided by the theory of Pareto dominance [16]. There are several ways to formulate the idea of Pareto dominance; we find it convenient to cast our definitions in the context solving of a multiple objective optimization problem. In this case one is presented with a population of solutions to the problem each characterized by a set of objectives. How do we rank these solutions without introducing arbitrary weights?

A solution to a multiple objective minimization problem is called *Pareto dominant* if each of its objectives is less than or equal to the corresponding objectives of every other solutions in the population, and furthermore there exists at least one objective that is strictly less than the corresponding objective(s) of every other solution. We can similarly define Pareto dominant solutions for multiple objective optimization problems involving maximization, as well as for problems that include some objectives that must be maximized and ad other objectives that must be minimized.

In principal one can determine the set of all Pareto dominant solutions in a population – the *Pareto frontier* – through exhaustive comparisons [24], for example, but is there an easier and more systematic algorithm that can scale to higher dimensions? The answer is yes, and that algorithm – or, rather, set of algorithms – is DEA [9][1].

DEA is a specialized application of linear programming, that is, the algebraic solution of a set of equations that takes the following canonical form [8]:

$$\begin{aligned}
&\max\left(c_1x_1 + c_2x_2 + c_3x_3 + \cdots c_Nx_N\right)\\
&a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots a_{1N}x_N \le b_1\\
&a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots a_{2N}x_N \le b_2\\
&a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots a_{3N}x_N \le b_3\\
&\cdots\\
&a_{M1}x_1 + a_{M2}x_2 + a_{M3}x_3 + \cdots a_{MN}x_N \le b_M\\
&\cdots\\
&x_1, x_2, x_3, \cdots x_N \ge 0
\end{aligned} \tag{1.1}$$

The problem consists of finding the N variables $\{x_n : 1 \le n \le N\}$ such that the linear expressions with the coefficients $\{c_n : 1 \le n \le N\}$ is maximized while the M constraints characterized by the coefficients $\{a_{mn} : 1 \le m \le M : 1 \le n \le N\}$ and $\{b_m : 1 \le m \le M\}$ constants are satisfied. *On average* the time required to solve such a problem scales as $N \times M$, but one can encounter pathological cases where the time required for solution is exponential in the dimensions [8] or even when the algorithm can cycle endlessly in rare pathological examples. When applying DEA to problems in multiple objective optimization using evolutionary algorithms, the number of rows (M) is approximately equal to the number of objectives, and the number of columns (N) is approximately equal to the number of solutions in the evolving population. Generally $N \gg M$. Details are provided elsewhere [16].

A numerically more challenging linear program includes the additional restriction that the variables be integers:

$$
\begin{aligned}
&\max\left(c_1x_1 + c_2x_2 + c_3x_3 + \cdots c_Nx_N\right)\\
&a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots a_{1N}x_N \le b_1\\
&a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots a_{2N}x_N \le b_2\\
&a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots a_{3N}x_N \le b_3\\
&\cdots\\
&a_{M1}x_1 + a_{M2}x_2 + a_{M3}x_3 + \cdots a_{MN}x_N \le b_M\\
&\cdots\\
&x_1, x_2, x_3, \cdots x_N \ge 0\\
&x_1, x_2, x_3, \cdots x_N \in \text{ integers}
\end{aligned}
\tag{1.2}
$$

It turns out that solving integer linear programs requires generally requires orders of magnitude more computation than solving linear programs with only continuous variables. Although it is not immediately obvious, the problem of determining the Pareto frontier in a population of solutions to a multiple objective optimization problem can be cast in such a numerical form [8]. The actual linear programs involved are only slightly more complicated than the above examples and we will not go into any details here – a rigorous mathematical discussion is given elsewhere [16] – but we can provide a qualitative discussion of what the DEA algorithm does by citing a simple two-dimensional example.

Imagine randomly sticking pins into a cork bulletin board within a region where x- and y-axes have been drawn. Each pin represents the solution of a two-objective minimization problem. The DEA algorithm can be envisioned as stretching a string around the perimeter of the lower left-hand portion of the collections of pins. The string then traces out the Pareto frontier and the pins it actually touches are the "best" solutions the population that minimize the x- and y-values simultaneously. Qualitatively, this is the "envelopment" in Data Envelopment Analysis. Several comments are now in order.

First, this method of ranking the solutions is *relative*. DEA cannot determine the best solution that is possible, it can only identify the best possible solutions within the population at a given time. If we have an evolving populations of solutions we would expect that from generation to generation the pins in the bulletin board would move to the lower left to minimize x and y, and DEA can be used within each generation to determine a relative ranking of the solutions based on Pareto Dominance.

Secondly, DEA not only identifies the solutions on the Pareto frontier, it assigns a numerical value to each solution that is based on a certain L1 distance from the frontier. The details are unimportant here, but this so-called DEA efficiency (solutions on the frontier are typically assigned unit efficiency) can then serve as a fitness function when we are solving a multiple objective optimization problem using an Evolutionary algorithm.

Thirdly, the regions on the frontier on our bulletin board between the pins, where our string has been stretched from pin to pin, includes an assumption of "convexity" which is built into several DEA models. Convexity indicates convex combinations of each "pin" are possible but as yet unrealized. Convexity is assumed in the Charnes-Cooper Rhodes (CCR) and Banks-Charnes-Cooper (BCC) models [9], but in general complex nonlinear multiple objective optimization problems are not convex [25][26]. There exists a more general formulation of DEA that does not assume convexity, the so-called "Free Disposal Hull (FDH). This is numerically unfortunate because the CCR and BCC models can be formulated using real linear programs whereas the FDH demands that we solve mixed integer linear programs.

The GDEA of Yun and colleagues [23] interpolates between the CCR, BCC, and FDH through the addition of an extra parameter. We have not seen the GDEA used with an evolutionary algorithm to solve explicitly non-convex test problems in multiple objective optimization [25][26]. However when we use the BCC model to evaluate the fitness of such problems we do not recover the entire Pareto frontier; the algorithm only converges to the convex hull of the true Pareto frontier. When using the FDH model to compute fitness, we can converge to the true Pareto frontier and achieve results consistent with another application – specifically, the Strength Pareto Evolutionary Algorithm (SPEA) of Zitzler [24] – that computes fitness using a completely different method. The absence of any assumptions of convexity in the FDH model thus appears crucial for MOES to converge to the true Pareto frontier and to compare our results with those of other Pareto-based methods when the problem is not convex.

By default we therefore employ the FDH models in MOES to evolve solutions to arbitrary problems in multiple objective optimization. We have found that when addressing complicated problems even solving the mixed integer linear programs associated with the FDH models is typically much faster than evaluating the training set. One more technical detail is that DEA demands that there be both *input* variables to be minimized and *output* variables to be maximized; otherwise the linear programs are ill-posed. The typical problem we have addressed involves minimizing a set of RMS (root mean square) errors. In this case we perform DEA using the input-oriented FDH model [9] with the RMS errors as inputs and a single additional "dummy" output variable that is held constant. Under these conditions we can compare the Pareto frontiers we obtain directly with those in the evolutionary literature.

III. EVOLUTIONARY ALGORITHMS AND EVOLUTION STRATEGIES

The category of "Evolutionary Algorithms" includes a very wide variety of mathematical methods inspired, to one degree or another, by biology. However they all include at least two aspects of Darwinian evolution: *descent with modification* and *natural selection*. We must first prepare an initial population of solutions to our problem. (When using

MOES, each solution is represented by a set of real and/or integer parameters.) We must then produce a new generation of solutions while introducing a degree of randomness in the process, the *descent with modification*. (In MOES this randomness can be due to either a random mutation of the parameters of a single solution, or a random mixing – or "crossover" – of parameters from multiple solutions.) We must then evaluate the fitness of the solutions in the population, including the possibility that solutions can die off with old age. (In MOES the fitness is an explicitly relative measure of the population at a specific time as evaluated using DEA.) We must then perform *natural selection* and eliminate bad solutions while retaining good solutions. (In MOES this can be accomplished using a strictly deterministic ordering of the solutions according to their DEA fitness and killing off a specified number of the lowest scoring parameters sets; or we can introduce a degree of randomness in selection as well and choose individuals at random and compare their fitness in a tournament to decide who survives and who dies.) We then iterate this process until some kind of termination criterion is reached, for example, until a maximum number of generations have been evolved.

MOES employs an additional termination criterion that we have found to be very effective in reducing the overall number of generations that need to be evolved in order to obtain a given degree of convergence. Typically, convergence of Evolutionary Algorithms begins rapidly and then slows down drastically – it is said to "plateau". Many generations can evolve after the plateau has been reached with little further improvement to the best solutions. We keep a list of the best N_{elite} solutions obtained so far in the evolution – the so-called "elite" solutions – and keep running averages of each objective of these elite solutions over the past, say, 50 generations. When the *changes* in the averages for each objective drop below a specified tolerance we consider the convergence to be stalled and the evolution is terminated. Finally, such evolutions can be repeated many times with different initial conditions and the final results averaged.

While Evolutionary Algorithms have become part of the standard repertoire of applied mathematics and have now been employed in a wide number of disciplines [3][5] it is still true to say that implementations to solve problems in multiple objective optimization [10][24] [25] [26] are less developed than are the algorithms for single objective optimization. The use of DEA in multiple objective optimization gives us the freedom to choose whichever single objective algorithm we feel is most appropriate and apply it directly to our particular multiple objective problem. Since we are primarily interested in *parameter optimization* we have chosen a particular Evolutionary Algorithm called "Evolution Strategies" [3]. However, it must be emphasized that if you have a multiple objective problem where some other form of Evolutionary Algorithm is more appropriate, DEA can still be used to compute fitness. Which Evolutionary Algorithm you choose is largely dictated by the form of the desired solution. For example, if the solution to your problem is more easily represented by a computer code than by a parameter set, then the use of Genetic Programming [13] would be indicated. However DEA could still be used to compute the fitness of individual computer codes as judged by some multiple objective criteria within an alternative Evolutionary Algorithm.

ES is a particularly flexible and powerful method for parameter optimization in part because it is *self-adaptive*; that is to say, the algorithm changes the manner in which it searches through parameter space as the evolution proceeds [3]. This is accomplished by adding additional parameters to the problem. ES includes one additional *strategy parameter* for every *physical parameter* that describes the actual problem being solved. These strategy parameters are standard deviations that describe the sizes of the (Gaussian) random mutations of the corresponding physical variables that are currently allowed. Furthermore, the strategy parameters themselves evolve along with the physical parameters. The standard deviations are themselves subject to *log-normal* mutations so the subsequent mutations of the physical parameters remains Gaussian. At first this sounds crazy: How can doubling the size of the parameter space help? It turns out that in most cases this is a very good idea because the strategy parameters can help guide the algorithm through parameter space to find optima more efficiently.

For converged evolutions, the typical observed behavior of the strategy parameters is that they are initialized at some small value; in the early stages of the evolution they grow as the algorithm examines more and more of parameter space using larger mutations; then in the latter stages of the evolution the strategy parameters decrease rapidly as the evolving population becomes concentrated upon some minimum in parameter space. If you are familiar with simulated annealing [12] one way to look at a "mutation-only" ES algorithm is that is like simulated annealing only each parameter has its own temperature and follows its own annealing schedule. However ES is in fact much more general and flexible because of the many possible mutation and crossover (mixing) operations that are possible in addition to the fact that each parameter is treated independently.

For convergence to occur in ES these standard deviations must ulitmately be driven towards zero. This is because if the population is indeed concentrated at some minimum, then any small mutation of the parameters will cause the solution to try to climb out of that minimum and yield a worse fitness. Evolution will then drive the standard deviations towards zero. This is a great numerical advantage of the algorithm because we can monitor the standard deviations of all the physical paramters and determine with confidence if convergence is actually occurring.

A representative example of the convergence of an ES evolution is shown in Figure 1. This is taken from a publication that applied MOES to fit a subset of the parameters for a reactive force field (ReaxFF) describing

energetic materials at the atomic and molecular level [14]. These classical approximations are useful in large scale molecular dynamics simulations of material properties, including crystal structure. This is a very complicated problem having more than 500 ReaxFF parameters total; we choose to modify approximately 50 of these parameters dealing with the long range contributions of the intermolecular forces with the goal of describing the structure of the molecular crystals of RDX more accurately. In fact this graph represents the average of the standard deviations of this subset of ReaxFF parameters as a function of the generation of the evolution. (In thse calculations we optimized 5 objectives – charges, geometric variables, heats of formation, bond energies, and non-bonded dimer energies – evolving sets of 46 real parameters that contrl the long-range portion of the ReaxFF potential.) In order to be certain of convergence we must examine the evolutions of each individual standard deviation, but their average is useful as a quick and dirty indication of convergence. We can rationalize the behavior of the average standard deviation in this graph by saying that the standard deviations start at some very small pre-assigned values and then grow as the evolutionary algorithm explores more and more of parameter space. When the algorithm settles upon some minimum the standard deviations decrease rapidly – because any mutation then tends to increase the error – indicating convergence. The evolution in fact resulting in a substantial improvement of the density of RDX as computed using ReaxFF [14].

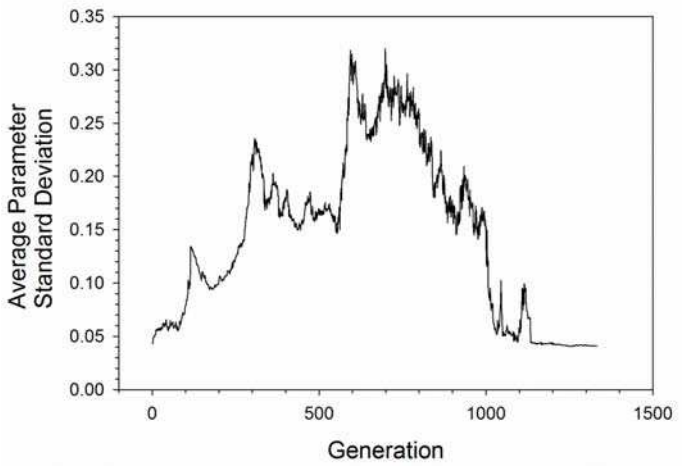

Figure 1: Trajectory for the averaged standard deviation during an evolution of ReaxFF parameters

ES has been demonstrated to be a powerful and flexible algorithm and had been applied to a wide variety of problems. We have incorporated nearly all the features of ES as discussed by Bäck in his comprehensive treatment [3]. We have also included features to allow for finite lifetimes of solutions [4]. Typical implementations of ES allow for optimization of problems described by only real parameters but we have also included extensions to integer parameters [18]. In this case the strategy variables corresponding to the integer parameters are *mutation probabilitites* rather than standard deviations; these control the likelihood of an integer parameter making a discrete jump to another value.

The notation (μ,λ,κ) has been adopted to describe evolutions using ES [3][4]. Here μ is the number of parents, λ is the number of children, and κ is the maximum lifetime in generations. As a rule of thumb we try to keep the number of children λ four or five times the number of parents μ, and the total size of the evolving population $\mu+\lambda$ at least an order of magnitude larger than the number of parameters that characterize the solutions to be optimzied. Thus for a probem of 50 parameters we would want to have an evolving population of at least $\mu+\lambda = 500$ and approximately $\mu = 100$ and $\lambda = 400$, however this is only a rough guide.

Before using MOES to optimize a particular problem one must first construct the "training set". This is a code that can be called from a C function and accepts a set of mutated parameters. The training set then performs a series of calculations and computes all the quantities that are relevant to the "goodness" of the parameter set. For example, if the parameters characterize some model calculation that must be compared against experimental results the training set would compute a series of errors between the predictions of the emodel and the corresponding experimental results. All the errors having the same physical units could then be combined, for example, as RMS errors, and these would then be returned to MOES as the objectives to be minimized. If we carry through with the biological analogy, MOES only knows about the genotype and the training set only knows about the phenotype. The interface between the MOES and the training set is typically a C function of at most a few hundred lines.

Finally, to avoid any possible confusion, we say we are evolving solutions to complex nonliear problems in parameter optimization because the training set typically involves evaluating complicated nonlinear functions of the parameters. However we are employing a technique from linear programming (DEA) to assign the fitness of the solutions – i.e., the parameter sets – by comparing their objectives. So the calculations we are performing are nonlinear in parameter space and linear in objective space.

IV. MOES: MULTIPLE OBJECTIVE EVOLUTION STRATEGIES

MOES is a large body of massively parallel C/C++/MPI code we have written to optimize parameters that govern complex nonlinear models. Ideally, if you have a model governed by up to about 100 parameters and can provide a serial code that computes all the errors you wish to minimize, we can incorporate it into our process by writing a single C function as an interface. This interface takes a set of mutated parameters from our evolutionary algorithm, passes them to your code (i.e., the "training set"), and then returns all the errors or "objectives" that are to be minimized by the evolutionary algorithm. The "evaluation of the training set"

must include all the calculations involving the model that you wish to compare with known results in order to judge the accuracy of the model. The minimum requirements for our algorithm to get started are the training set, the interface function, and a set of strict [a,b] bounds for each parameter.

We employ a master-slave paradigm, in which the evolution is performed on the master processor, and all the objectives are computed on the slave processors; each slave processor gets one set of mutated parameters at a time, computes the objectives for that particular parameter set, and then moves on to the next task. Parallelization is performed using MPI thus assuring compatibility modern massively parallel machines running Linux.

Our implementation of the master-slave approach uses the "bag of tasks" technique [19] in which each processor announces when it is available to compute the objectives for the next available set of mutated parameters within a simple "for" loop; load balancing is accomplished automatically. Since there are typically many more sets of parameters in the evolving population than available processors, on each pass through the loop the slave processors are queried to determine who among them has finished evaluating his last set of objectives and are therefore available to receive the next set of mutated parameters in the queue. Typically the communication overhead associated with querying the slave processors, sending the parameter sets and receiving the objectives can be completely masked by the computational load of evaluating the training sets.

In practice we have found that while evolving large populations (e.g., thousands of parameter sets) we can obtain good scaling on HPC systems when the "evaluation of the training set" takes several minutes on a single processor. The only time we have encountered difficulties with this approach occurred in the evolution of reactive force fields [14] when certain "pathological" parameter sets caused the time required to evaluate the training set to increase by several orders of magnitude. This would destroy load balancing and in such cases the calculation of the objectives were terminated early and bad fitness values were assigned to the troublesome parameter sets; the evolution then continued normally. A simplified representation of MOES and the master-slave paradigm is shown in Figure 2.

In practice DEA is also performed in parallel, although for simplicity this is not indicated in the figure. Since DEA requires that we solve one linear program for each parameter set in the evolving population to evaluate its fitness, and since each such linear program is independent of the others, the calculation labeled "Compute Fitness" in Figure 2 is also "naturally parallel" – see [6] and references therein for other parallel DEA implementations – and the individual linear programs for distinct parameter sets are assigned to different processors.

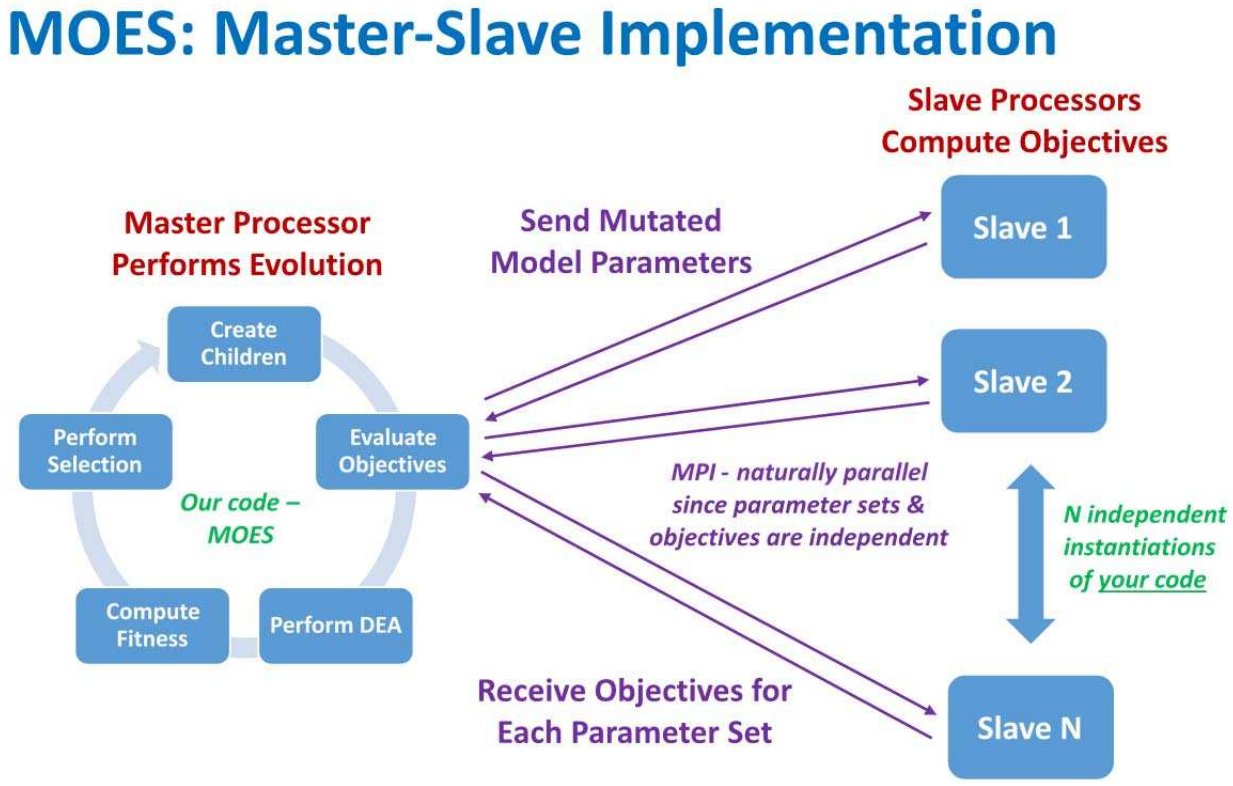

Figure 2: Schematic outline of the MOES program; in the actual code the DEA calculation is also parallelized

The code we use to solve the real and mixed integer linear programs in DEA is the freely available serial program *lpsolve 5.5* [7]. We have found this program to be stable; it is well-maintained and has an active user group, and the documentation for the programming interface is excellent. We wrote C code to compute the possible basic DEA models (the input- and output-oriented CCR, BCC, and FDH models) by initializing the corresponding linear programs and then solving them in parallel using separate instantiations of *lpsolve* on each processor. We therefore had to make no alterations to the *lpsolve* source code; after experimentation we hardwired most of the numerical switches controlling the options we used in *lpsolve*. The raw data from MOES (principally the objectives computed in the training set) must first be scaled so that the entries in the tableau constructed in *lpsolve* are all of the same magnitude. We have found a logarithmic scaling to be most effective. In our experience the time required to perform DEA and evaluate the fitness of the evolving population has been trivial compared to the time required to evaluate the objectives of each solution, but this will depend on the particular problem being solved and on the number of calculations performed in the training set.

Typically millions of linear programs must be solved during an evolution and there is no guarantee that a solution exists for each. Even with logarithmic scaling of the objectives we monitor the return code provided by *lpsolve* and check to see if a solution was actually found. In those rare cases where *lpsolve* fails to converge within a specified time limit – well under 1% of the calculations in our experience – we assign bad fitness values to the solutions whose objectives caused *lpsolve* to fail, and the evolution continues.

In 1997 Barr and Durchholz [6] presented an alternative DEA scheme (PIONEER) designed to solve large envelopment models whose tableaux have many columns. The improvements include both serial and parallel components. The principal improvement involves the early identification of inefficient solutions followed by the use of "restricted basis entry" to reduce the size of the subsequent linear programs to be solved. Once a solution has been identified as inefficient – that is, it is seen to be dominated in the Pareto sense by another solution – it can be eliminated from the tableau for subsequent calculations. Thus as the calculations proceed the linear programs that must be solved become smaller. In their parallel implementation each processor (solving independent linear programs) must be given the "current" list of inefficient solutions, and this list changes as more independent linear programs are solved. They relied on shared memory (on the machine they used in 1997) to communicate the (changing) list of inefficient solutions to each processor while the (independent) linear programs were solved on local memory. Recent trends in high performance computing have emphasized increasing the core count per CPU and moving to distributed memory architectures [20]. While the algorithm in PIONEER could be implemented on distributed memory systems using MPI, the fundamental loop in the bag-of-tasks paradigm would now have to include MPI broadcasts of the current list of inefficient solutions to all the processors. The efficiency of the resulting algorithm would be strongly machine-dependent as well as problem-dependent. Finally PIONEER would also have to be generalized to include the FDH model.

The use of the early identification of inefficient solutions and restricted basis entry were studied again in 2006 by Dulá who solved large envelopment models with 10,000 to 100,000 columns [11]. The sophisticated commercial LP solver CPLEX was used within a serial formulation of the DEA problem. Additionally Dulá considered the effects of using "hot starts". Like restricted basis entry, this involves modifications of the tableaux for the solution of subsequent linear programs based on the results obtained when solving earlier linear programs in the DEA protocol. Substantial speedups were observed with the combination of "hot starts" and restricted basis entry, and the savings in time became dramatic as the numbers of columns in the tableaux were increased. Again, a parallel implementation on a distributed memory machine would require the broadcast of all the information required to perform the "hot starts" to all the processors. Additionally the methods of Barr and Durchholz and of Dulá would have to be generalized to include the FDH model. However as we look at larger and larger problems we may have to revisit issues involving the efficiency of performing DEA to compute fitness within MOES. Whether the increase in communication and algorithmic complexity can be offset by the speedups in computing fitness within MOES will depend upon the problem, the size of the evolving population and the computer architecture.

V. NUMERICAL EXAMPLE

We have applied MOES to a series of standardized multiple objective test problems [25][26]; details will be presented elsewhere [16]. Here we want to emphasize the effects of assumptions concerning convexity and how this relates to the choice of DEA models. We have chosen problem "T2", a non-convex problem, and followed the general protocol used in [26]: we performed a series of 30 evolutions, each of 250 generations, with an evolving population of 100, keeping 20 solutions in the external "elite" set. The evolution strategy used was $(\mu, \kappa, \lambda) = (20, 5, 80)$. The results of the SPEA of Zitzler and Thiel are shown in Figure 3.

The SPEA was demonstrated to perform better than number of competing algorithms for multiple objective optimization [25][26]. If we (mistakenly) assume the Pareto frontier to be convex and apply MOES using the BCC model we get the results in Figure 4. Note that coverage of the frontier is very poor.

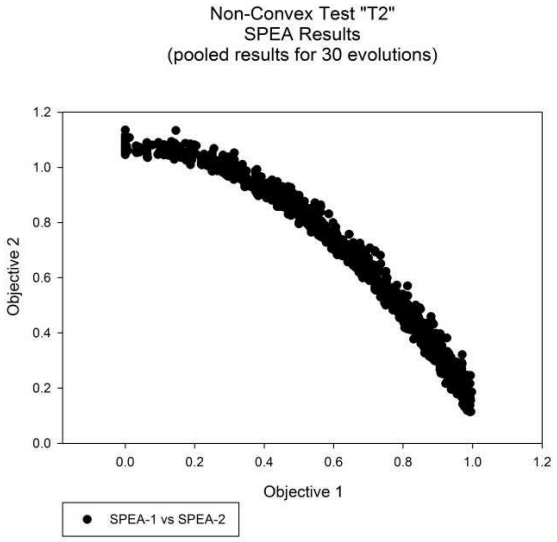

Figure 3: SPEA results for a non-convex test problem

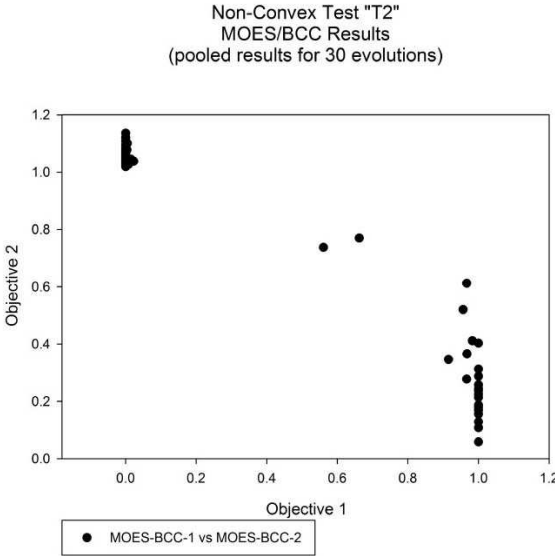

Figure 4: MOES results using the BCC model for a non-convex test problem

If we apply MOES using the FDH model we get the results in Figure 5. Using MOES with the FDH model is seen to yield results comparable to those obtained with the SPEA, even though the algorithms used to compute fitness are very different. Results for additional test problems and more details are presented elsewhere [16].

In general we do not know *a priori* whether the Pareto frontier for an arbitrary problem will be convex or not, in which case the FDH model should be employed. Otherwise it is clear that poor results can be obtained, especially in terms of the coverage of the Pareto frontier. Practically this means that solutions that might offer a particularly desirable balance between minimizing (and/or maximizing) the various objectives will not be obtained. To date we have encountered no difficulties in using the FDH model because the evaluation of the training set has completely dominated the calculations [14]. Furthermore, no failures of *lpsolve* were encountered in any of the BCC or FDH computations. We are nonetheless considering possibly more efficient alternatives. [16]

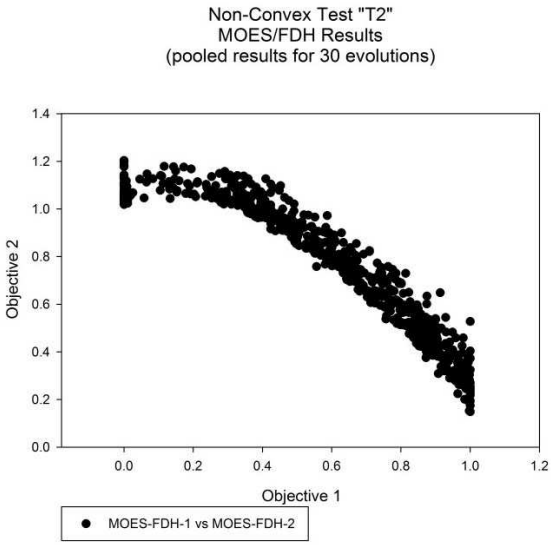

Figure 5: MOES results using the FDH model for a non-convex test problem

ACKNOWLEDGEMENTS

We gratefully acknowledge the support of the Department of Defense High Performance Computing Modernization Program and the PETTT initiative. We also gratefully acknowledge the support of Dr. Betsy Rice's MSRM (Multi-Scale Reactive Modeling) Software Institute at ARL/WMRD (Army Research Laboratory / Weapons and Materials Research Directorate). Dr. James Lill (Engility Corporation) is the principal author of MOES but over the years he has received substantial assistance from Dr. Shawn Brown (of the Pittsburgh Supercomputing Center), Dr. Anthony Yau (who is now working for another government agency), as well as Dr. James Larentzos and Dr. Ross Smith (both of Engility Corporation and the PETTT initiative).

REFERENCES

[1] Anderson, Timothy, *A Data Envelopment Analysis (DEA) Home Page* http://www.emp.pdx.edu/dea/homedea.html [Accessed 03/26/2015]

[2] Arakawa, Masao, H. Nakayama, I. Hagiwara and H. Yamakawa, "Multiple Objective Optimization Using Adaptive Range Genetic Algorithms with Data Envelopment Analysis", Multidisciplinary Analysis and Optimization 3, 2074-2082, 1998.

[3] Bäck, Thomas, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*; Oxford University Press, New York, 1996.

[4] Bäck, Thomas, and H-P. Schwefel, *Evol. Comput.,* 1993, *1*, 1-23.

[5] Bäck, Thomas, D.B. Fogel and Z. Michalewicz (Editors), *Evolutionary Computation 1: Basic Algorithms and Operators* and *Evolutionary Computation 2: Advanced Algorithms and Operations*; Institute of Physics Publishing, Philadelphia, 2000.

[6] Barr, Richard S., and M. L. Durchholz, "Parallel and hierarchical decomposition approaches for solving large-scale Data Envelopment Analysis models", Annals of Operations Research 73, 339-372, 1997.

[7] Berkelaar, Michel, K. Eikland, and P. Notebaert, *lpsolve5.5, Open source (Mixed-Integer) Linear Programming system* 2014. http://sourceforge.net/projects/lpsolve/ [Accessed 03/26/2015]

[8] Chvátal, Václav, *Linear Programming*; W.H. Freeman and Company: New York, 1983.

[9] Cooper, W.W., L.M. Seiford, and K. Tone, *Data Envelopment Analysis: A Comprehensive Text with Models, Applications, References and DEA-Solver Software (Second Edition)*, Kluwer: Boston, 2000.

[10] Deb, Kalyanmoy, *Multi-Objective Optimization using Evolutionary Algorithms*; Wiley: New York, 2001.

[11] Dulá, J. H., "A computational study of DEA with massive data sets", Computers & Operations Research 35 (4), 1191-1203, 2006.

[12] Kirkpatrick, Scott, C.D.Gelatt, M.P. Vecchi, Science, 1983, 220, 671-680.

[13] Koza, John, *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*, MIT Press, Cambridge (1992).

[14] Larantzos, J.P., B.M. Rice, E.F.C. Byrd, N.S. Weingarten and J.V. Lill, "Parameterizing complex reactive force fields using Multiple Objective Evolutionary Strategies (MOES): Part 1, ReaxFF models for cyclotrimethylene trinitramine (RDX) and 1,1-diamino-2,2-dinitroethene (FOX-7)" (*manuscript to appear in The Journal of Chemical Theory and Computation*).

[15] Lill, James, "On the Incorporation of Data Envelopment Analysis in Evolutionary Algorithms for Multiple Objective Optimization", MS project submitted to the Department of Engineering and Technology Management, Portland State University, Portland, Oregon, 2001.

[16] Lill, James, T. Anderson, et al. (*manuscript in preparation*).

[17] Nakayama, Hirotaka, Y. Yun, M. Yoon, *Sequential Approximate Multiobjective Optimization Using Computational Intelligence (Vector Optimization)*, Springer-Verlag, Berlin, 2009.

[18] Schwefel, Hans-Paul, and G. Rudolph, "Contemporary Evoluation Strategies", In *Advances in Artificial Life: Third European Conference on Artificial Life, Granada, Spain, June 4-6, 1995 Proceedings*; Morán, F., Moreno, A., Merelo, J.J., Chacón, P., Eds.; Lecture Notes in Computer Science 929; Springer: Berlin, 1995; pp 891-907.

[19] da Silva, F.A.B., H. Senger, Parallel Comput. 35, 57-71, 2009.

[20] http://www.top500.org/statistics/overtime/ Interested readers can visit the "TOP500" website and make a plot of "Architecture – Systems Share" for themselves: choose "Architecture" under "Category" and "Systems share" under "Type". The entries labeled "Cluster" and "MPP" (Massively Parallel Processors) represent distributed memory architectures. The entries labeled "Single Processor" and "SMP" (Symmetric Multi-Processing) and "SIMD" (Same Instruction Multiple Data, e.g. vector machines) all represent shared memory architectures. The entries labeled "Constellations" represent a kind of hybrid architecture in which the "compute nodes" are composed of shared memory multiprocessors; computations involve shared memory within nodes and message passing between nodes. [Accessed 03/26/2015]

[21] Yun, Yeboon, H. Nakayama, T. Tanino, M. Arakawa: "Generation of efficient frontiers in multi-objective optimization problems by generalized data envelopment analysis", European Journal of Operational Research 129(3), 586-595, 2001.

[22] Yun, Yeboon, M. Arakawa, H. Nakayama: "Fitness evaluation using generalized data envelopment analysis in MOGA", in IEEE Congress on Evolutionary Computation , 464-471, 2004.

[23] Yun, Yeboon, H. Nakayama and T. Tanino, "A generalized model for data envelopment analysis", European Journal of Operational Research 157, 87-105, 2004.

[24] Zitzler, Eckart, *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications.* Ph.D. Dissertation, Swiss Federal Institute of Technology (ETH), Zurich, November 1999. http://www.tik.ee.ethz.ch/sop/publications/

[25] Zitzler, Eckart, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms on Test Functions of Different Difficulty." In *Proceedings of the GECCO-1999 Genetic and Evolutionary Computation Conference,* Orlando, 1999; Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E., Eds.; Morgan Kaufmann, 1999.

[26] Zitzler, Eckart, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results", *Evol. Comput.,* 2000, *8*, 173-195.