On the Incorporation of Data Envelopement Analysis in Evolutionary Algorithms for Multiple Objective Optimization

J.V. Lill, Garrison Greenwood and Timothy R. Anderson

Abstract— **We present the first results of applying Data Envelopment Analysis (DEA) to compute fitness in a multiple objective Evolutionary Algorithm (EA). Results are presented for the multiple objective 0/1 knapsack model of Zitzler and Thiele, and for an additional 0/1 knapsack model that we introduce. The implications for solving Multiple Objective Optimization problems (MOP) in a general and flexible manner by generating Pareto optimal solutions using DEA are discussed.**

Keywords— **multiple objective, Data Envelopment Analysis, linear program, knapsack problem**

I. Introduction

Problems in multiple objective optimization (MOP) naturally arise in many fields of science, engineering and operations research. For example, MOPs in electrical and computer engineering are ubiquitous. Computer architects strive to build increasingly faster computers. But there are trade-offs among various attributes: when processing speed increases, so do the dollar cost and power consumption. The optimal design should have maximum processing speed, but at minimum cost and minimum power consumption. Other areas with difficult trade-offs include power transmission, control system design, and wireless broadcast networks. Note that these attributes are *non-commensurate*, that is, they are measured in different units.

The desire to compare the relative efficiencies of different industrial and economic activities in an unambiguous manner led economists to develop the basic mathematics of Pareto optimality early in the twentieth century. A solution to a MOP is *non-dominated* if all of its attribute values are at least as good as (or better than) all other corresponding attributes of any other solution; moreover, there is at least one attribute value that is strictly better. Such a non-dominated solution is *Pareto optimal*, and the set of all such solutions forms a surface in parameter space called the *Pareto efficiency frontier.*

Numerically, Pareto optimality achieved its most highly developed application in the latter twentieth century in Data Envelopment Analysis (DEA).[1] Here the mathematical technique of linear programming is used to compute a numerical efficiency of each solution to a MOP, as well as its projection onto the efficiency frontier. This efficiency denotes a normalized distance from the Pareto frontier, and serves as an unambiguous measure of the degree to which one solution is dominated by the others. The use of DEA in operations research has grown exponentially in the past twenty years, and a repository for DEA test problems has recently been established at PSU.[2] However DEA does not provide any method for *generating* solutions to MOPs.

The field of evolutionary computation uses the Darwinian principles of natural selection found in nature to find solutions to difficult optimization problems;[3][4] MOPs constitute a primary focus area.[5][6] Historically, evolutionary computations have a rich past, being independently developed by at least three independent research efforts, which ultimately produced three distinct paradigms: genetic algorithms, evolutionary programming and evolution strategies. All EAs share the same basic organization: iterations of competitive selection and random variation. More specifically, each generation (iteration of the EA) takes a population of individuals (solutions) and modifies the genetic material (problem parameters) to produce new individuals. Both the parents and offspring are evaluated, but only the most fit individuals survive over multiple generations.

Over the last 15 years evolutionary algorithms (EAs) have proven to be highly effective in searching for good solutions to difficult optimization problems in general, and MOPs in particular. Indeed, MOP is one of the most extensively studied areas in the evolutionary computation field. Fonseca and Fleming[7], and more recently Zitzler and Thiele[8][9][10], have provided excellent reviews of the various approaches to solving MOPs using EAs, and we may distinguish three broad categories: those that assign different weights or conversion factors to the various non-commensurate attributes in order to form a single objective function, those that switch between distinct objective functions for the different attributes during the course of evolution, and those that make use of the idea of Pareto optimality to define fitness. Our approach falls into this latter category.

We present here the first incorporation of DEA into EAs to compute fitness functions for solutions to MOPs. There are many different DEA models, and numerous distinct EAs and possible refinements, including *crossover*, *mutation*, *fitness sharing* and *elitism*. In this brief article we cannot hope to be exhaustive; rather, we will present the

James Lill and Timothy Anderson are in the Department of Engineering and Technology Management, Portland State University, Portland, Oregon. E-mail: tima@etm.pdx.edu .

Garrison Greenwood is in the Department of Electrical and Computer Engineering at Portland State University, Portland, Oregon. E-mail: greenwd@ee.pdx.edu .

most simple application, and address only a few of the practical difficulties. It is our hope that this work will inspire further cross-disciplinary collaborations that combine techniques from different areas of engineering and operations research to address difficult problems.

In the next section, we describe the two test MOPs to be solved, and the genetic algorithm (GA) employed. In the following section we present a brief desccription of DEA and discuss the particular models used in the GA to compute fitness. Numerical results re presented in the succeeding section. We conclude with a brief discussion, and summarize the LPs that define some of the more common DEA models in an appendix.

II. Theory

A. 0/1 Multiple Objective, Single Knapsack Model

The test first MOP we have chosen is the 0/1 knapsack problem of Zitzler and Thiele.[8][9] Although originally described in terms of a number of different knapsacks each assigning different "values" and "weights" to every item, and each having different limits on the total "weight," it is perhaps more easily understood in terms of a single knapsack. Here the different "weights" could correspond to the physical weight or volume, or to any extensive property of the items; there are then several distinct capacities of the single knapsack, given in terms of their total weight, total volume, *etc.* The different values assigned to each item could correspond to the different possible barter systems in which the items could be traded; for example, one compass might be worth three pigs in the jungle or two goats in the mountains. The different values must be non-comensurate, otherwise the problem would reduce to that of a single objective optimization.

A single subset of the total number of items is to be is to be chosen so that all the constraints (*e.g.*, those on volume and on weight) are satisfied and each value (*e.g.*, pigs or goats) is maximized; these items are then put into the single knapsack. In effect, in this model, we don't know if we are going to the jungle or to the mountains, or if we will visit both; however, we can only take a single knapsack. In summary, this MOP tries to find a *single* set of items that will be valued highly in *any* of the scenarios envisioned. A single solution to the example of this MOP having a total of M items could be represented by a bit string having M bits, representing the presence or absence of each item in the knapsack. We will numerically solve examples of this multiple objective, single knapsack model, and compare our results with those obtained by Zitzler and Thiele in in order to verify that the correct Pareto frontier is obtained using DEA to compute fitness.

B. 0/1 Multiple Objective, Multiple Knapsack Model

A distinct MOP can be derived by supposing that there are separate knapsacks for each set of "weights" and "values," and that a finite number of items must be distributed among all the knapsacks, subject to a single constraint for each knapsack. In effect, each item has a different value and weight in each knapsack, each knapsack has a different capacity expressed in terms of the corresponding weight, and no item can appear in more than one knapsack in any given solution. In this model, we have the freedom to pack separate knapsacks for trips to the jungle or to the mountains, but we only have a single example of each item to pack. In summary, this MOP tries to determine *multiple, mutually exclusive* sets of items, each of which will be valued highly under *one* of the scenarios envisioned. A single solution to an example of this MOP having a total of M items and K knapsacks could be represented by K bit strings each having a total of M bits, representing the presence or absence of a given item in a given knapsack. The K bit strings can share no common items.

This same representation can be used to solve the multiple objective, single knapsack model if we demand that the bit strings in each sack be *identical*. We will numerically solve examples of this multiple objective, multiple knapsack model below, and compare and contrast these results to those obtained with the multiple objective, single knapsack model.

C. Evolutionary Algorithm

Because of the discrete nature of the test MOP, we have chosen to use a simple genetic algorithm. Bit strings are constructed as discussed above; thus each solution to the single knapsack problem is represented by M bits and each solutions to the multiple knapsack problem is represented by $M * K$ bits. Initialization is accomplished by looping over each item and inserting it in a randomly into either the single knapsack, or into a randomly chosen knapsack.

We use only the following mutation operators in reproduction: complementation of a specified maximum number $m_{\max}$ of bits; swapping two segments of consisting of a specified maximum length $m_{\max}$ of the bit string of an individual with probability P_{swap}; attempting to insert a specified maximum number n of items into the randomly chosen knapsacks of an individual with probability P_{insert}. If neither the swap nor insertion operations are performed on a given individual, then a uniform variate on $[1, m_{\max}]$ is chosen, and this number of bits is complemented; the knapsacks for the complemented bits are chosen at random. If complementation results in addition of an item to the knapsack, then the other knapsacks are checked to insure that the item does not appear twice; if the item is found in a second knapsack, it is removed from this second knapsack.

We also employ the "greedy" repair mechanism of Zitzler and Thiele:[8][9][10] If a particular knapsack is over its capacity, we remove items from it in increasing order of the ratio of their value to their weight, as evaluated in that particular knapsack. When solving the multiple objective, multiple knapsack model, we must include a check to insure that no items are repeated in different knapsacks in the same individual; this check is not required with the mutation operators just described, but is necessary if other forms of reproduction (*e.g.*, crossover) are employed. When solving the multiple objective, single knapsack model, we only perform mutations within a single sack, whose bit string is then replicated exactly in all the other sacks. If an item must be removed from one sack to satisfy capacity constraints, then this same item must be removed from all sacks.

DEA demands that we identify all the parameters in the LP as being either "inputs" or "outputs." These are defined so that increasing the value of an input decreases the fitness of a solution, while increasing the value of an output increases the fitness of a solution. At first glance, the obvious associations would be to define the weights of the items as inputs and their values as outputs; however, we will see later that this view must be modified. We have also found it necessary to identify all the clones in the population and remove the "extra" copies prior to evaluating the DEA efficiencies. Otherwise, numerical instabilities in the routine used to solve the linear program can result. We could identify clones by bit-wise comparisons of their genomes, but we have found it faster to compute phenotypic distances, including both the weights (x_{nk}) and values (y_{nk}) of individual n evaluated in knapsack k:

$$d_{n'n} = \sqrt{\sum_{k=0}^{K} \left[(x_{n'k} - x_{n'k})^2 + (y_{nk} - y_{n'k})^2 \right]} \qquad (1)$$

Finally, we have adopted a $(\lambda + \mu)$ strategy with decimation to perform selection. In addition, we follow Zitzler and Thiele and maintain an elite population of ε Pareto optimal individuals; these efficient solutions can contribute to the mating pool. Other possibilities will be considered in a more lengthy publication. An outline of the algorithm is as follows:

1. Initialize the parent population of μ individuals.
2. Loop over the λ parents and the ε elite individuals.
 (a) Choose one parent and one elite solution at random.
 (b) Mutate each parent and elite individual to form new children.
 i. Choose either the *swap* or *insertion* operations at random.
 ii. Generate a uniform variate u; if $u \le P_{swap}$ (or, if $u \le P_{insert}$), perform the *swap* (or *insertion*) operation.
 iii. If neither swap nor insertion is performed, then generate a uniform variate $1 \le u \le m_{\max}$. Choose u bits in knapsacks at random, and complement each bit.
 iv. If *iii* resulted in an item being added to a knapsack, loop over the remaining knapsacks and check if the same item appears in a *second* knapsack; if so, delete the item from the *second* knapsack.
 (c) Repair each child.
 i. Verify that the total weight of each knapsack is less than or equal to its total capacity. If a knapsack is over capacity, remove items from the knapsack in order of *increasing* ratio of value to weight.
 ii. Verify that no item appears in more than one knapsack in the same individual. If a given items appears in k knapsacks, choose $k-1$ of these knapsacks at random and remove the item from each.
 (d) If λ children have been produced, *break*.
3. Identify the clones in the population, and remove the extra copies.
 (a) Compute the phenotypic distances between the solutions according to 1 and identify the clones in the population.
 (b) Eliminate all but one copy of each clone.
4. Compute the DEA efficiencies for the combined $(\lambda + \mu + \varepsilon)$ population.
5. Update the elite population.
 (a) Remove items that are no longer Pareto optimal, and introduce new efficient individuals.
 (b) If the number of efficient individuals exceeds ε, reduce the elite population size to ε by clustering.[8][9][10]
6. Decimate the inefficient individuals to form the parent population for the next generation.
7. If the maximum number of generations has been reached, *stop*; if not, *return to step* 2.

D. DEA Models and Pareto Efficiency

DEA is a set of protocols for using linear programming to compute Pareto efficiencies for a population of individuals that can be characterized as possessing two types of parameters, "inputs" and "outputs." In typical applications from operations research, each individual could represent a factory, an operating unit of a corporation, or an industrial process. The "inputs" are distinguished from the "outputs' by the sense in which an individual's efficiency is affected by an increase in the variable: if an arbitrary increase in a variable x *decreases* the individual's efficiency, then x is an "input;" conversely, if an arbitrary increase in a variable y *increases* the individual's efficiency, then y is an "output." DEA begins with a fractional definition of efficiency of individual i as being the ratio of weighted sums of inputs to outputs:

$$\eta_n = \frac{\sum_{j=1}^{J} y_{nj} u_{nj}}{\sum_{i=1}^{I} x_{ni} v_{ni}} \qquad (2)$$

where we assume there are J outputs and I inputs. The task is now to determine sets of weight for the for the inputs $\{v_{ni}\}$ and outputs $\{u_{nj}\}$ of solution n; note that each individual has a distinct set of weight for its inputs and outputs. The basic idea is to assign to each individual n a set of weights that maximizes its efficiency η_n. There are several protocols for doing so, and these give rise to different DEA models. However, they all share the feature that the *fractional* program - *i.e.*, the problem of maximizing the ratio in 2 subject to some specified constraints - becomes a *linear* program.

For example,**??** we can maximize the efficiency η_i by minimizing the weighted sum of inputs:

$$\min \left[\sum_{i=1}^{I} x_{ni} v_{ni} \right] \qquad (3)$$

subject to the constraints that weighted outputs for individual i sum to unity

$$\sum_{j=1}^{J} y_{nj} u_{nj} = 1 \ \cdots \ 1 \leq n \leq N \tag{4}$$

and that the weight for the inputs and outputs be non-negative:

$$u_{nj}, v_{ni} \geq 0 \tag{5}$$

Finding the extremum of 3 subject to the constraints 4 and 5 constitute a *linear program*, the solution of which lies at the heart of DEA. Before proceeding, it is customary to transform this linear program (LP) into its *dual* form. The dual of linear program is obtained essentially by substituting maximizations for minimizations, changing constraints into variables, and *vice versa*.[12] In the present case, the dual of the linear program defined by 3, 4 and 5 becomes: [1]

$$\max\,[\eta_n] \tag{6}$$

subject to the constraints:

$$\eta_{n'} y_{n'j} - \sum_{n=1}^{N} \mu_{n'n} y_{nj} \leq 0 \ \cdots \ 1 \leq j \leq J \tag{7}$$

$$x_{n'i} - \sum_{n=1}^{N} \mu_{n'n} x_{ni} \geq 0 \ \cdots \ 1 \leq i \leq I \tag{8}$$

in addition to the non-negative constraints.5 In the parlance of DEA, 3, 4 and 5 represent the LP in the *multiplier* form, while 6, 7 and 8 represent the LP in the *envelopment* form. This particular DEA model is called the *Charnes-Cooper-Rhodes* (CCR) model - in particular, the *output-oriented* CCR model, or CCRO.

This name may at first appear incongruous - after all, in the multiplier form of the CCRO model we determine the weight for the outputs by a normalization constraint and the weight for the inputs by a minimization condition. However, the nomenclature becomes more clear when we consider that solution of the LP proceeds by first introducing the slack variables s^+_{ni} and s^-_{ni}:

$$\eta_{n'} y_{n'j} - \sum_{n=1}^{N} \mu_{n'n} y_{nj} = -s^+_{n'j} \ \cdots \ 1 \leq j \leq J \tag{9}$$

$$x_{n'i} - \sum_{n=1}^{N} \mu_{n'n} x_{ni} = s^-_{n'i} \ \cdots \ 1 \leq i \leq I \tag{10}$$

A solution $n\prime$ is Pareto optimal if and only if [1]

$$\eta_{n'} = 1 \tag{11}$$

and

$$s^+_{n'j} = 0 \ \cdots \ 1 \leq j \leq J \tag{12}$$

$$s^-_{n'i} = 0 \ \cdots \ 1 \leq i \leq I \tag{13}$$

The collection of such solutions forms the Pareto frontier. It is clear from 9 that the envelopment form of the output-oriented CCR model that the efficiency $\eta_{n'}$ measures the departure of an inefficient solution from the Pareto frontier only in those dimensions defined by the outputs. For inefficient solutions, the CCRO efficiency lies in the range $1 < \eta_n < \infty$.

One reason for working with the dual problem is numerical efficiency. Typically, the time required by the various *tableau* algorithms employed to solve an LP scales with the number of constraints, and $N >> I + J$. Note that a complete evaluation of all the efficiencies η_n requires solution of N envelopment LPs each having $I + J$ constraints and N variables ($\mu_{n'n}$) or solution of N multiplier LPs each having N constraints and $I + J$ variables (v_{ni} and u_{nj}). In the following section, we will use the envelopment CCRO model - and a slight variation of it - to compute fitness functions for our GA to solve the 0/1 multiple knapsack problem.

III. Numerical Results

In applying DEA to the 0/1 knapsack problem, the obvious choice of variables is to assign the total weights of the knapsacks as inputs and the total values of the knapsacks as outputs. Increasing the weight of a knapsack will *decrease* 2 while increasing its value will *increase* 2.

$$y_{nj} = \text{ value of knapsack } j \text{ in solution } n \tag{14}$$

$$x_{ni} = \text{ weight of knapsack } i \text{ in solutions } n \tag{15}$$

In the present problem with two knapsacks and 100 items, we have $I = J = 2$. Traditional fitness functions are defined so that large values describe individuals that are more fit than those whose fitness functions have a smaller value. Thus we define the fitness of individual n as

$$f_n = 1/\eta_n \tag{16}$$

We have implemented the simple tableau algorithm (SIMPLX) given in *Numerical Recipes in C*.[11] Slight modifications were necessary in order to make the code run reliably with the tableaux generated by the GA during the course of evolution. These minor changes are briefly described in the appendix.

Use of this fitness function in combination with the efficiencies calculated by the CCRO model yields the data displayed as filled circles in Figure 1. This graph summarizes the efficient solutions obtained after 10 separate evolutions having 100 parents, 200 children and 25 in the elite population. The results are *dysgenic*; that is, solutions become *worse* as evolution proceeds. It is not difficult to understand why this has occurred. A further problem is that solutions *inside* the apparent Pareto frontier are assigned to the elite population.

With the efficiency defined as in 2, the GA seeks to find the solution having the largest ration of value to weight; but the solution having the largest such ratio can not have a high total weight since, since inclusion of items having less than the maximum ratio will *decrease* the overall efficiency of the individual. The fact that we see solutions inside the apparent Pareto frontier in the two-dimensional space of the outputs is that we are really viewing in Figure 1 is a projection of the four-dimensional Pareto surface in the space of inputs and outputs onto the two-dimensional space of the outputs only.

Clearly, we must employ a different definition of fitness or a different DEA model. Fortunately, a standard approach to such problems exists. We retain the basic structure of the CCRO model but define all the inputs to be unity. This output-only, or CCROO model, is defined by:

$$y_{nj} = \text{ value of knapsack } j \text{ in solution } n \tag{17}$$

$$x_{ni} = 1 \tag{18}$$

Using 16 but with the CCROO model yields the open circles in Figure 1. The solutions are no longer dysgenic, and the Pareto frontier now appears as a line in the two-dimensional space of outputs - as it should. Convergence is rapid.

IV. Discussion

In conclusion, we have demonstrated that appropriate DEA models can be used to compute fitness functions in a multiple objective EA. There is an enormous DEA literature in operations research, where various DEA models have been applied to a wide variety of problems. There is a correspondingly large numerical literature that provides many alternatives for solving the LPs that arise. DEA provides a flexible and numerically-proven method for computing Pareto efficiency in a manner that is independent of the units used to measure the different attributes, and is thus well-suited to computing solutions to MOPs.

V. Appendix: Common DEA Models and Numerical Details

Because this is the first application of Data Envelopment Analysis to Evolutionary Algorithms, we have included here a summary of some the more common DEA models. We merely display the primal and dual LPs that define each model, and offer a few comments on each. Broadly speaking, we may distinguish three categories of DEA models: the *Charnes-Cooper-Rhodes* (CCR) models, characterized by a *constant* return to scales; the *Banker-Charnes-Cooper* (BCC) models, characterized by a *variable* return to scales; and various *additive* models. These latter models assume some relationship between the variables that allows them to be added, *i.e.*, the variables are commensurate; we will not consider additive models here.

In general, we will assume that there are I inputs, J outputs and N solutions; the LPs summarized below solve for the efficiency of solution $n\prime$. Denote the input variables as $\{X_{mk}\}$ and the output variables as $\{Y_{mk}\}$, and assume:

$$X_{mk} \geq 0,\ Y_{mk} \geq 0 \tag{19}$$

The following table summarizes the LPs that define the common DEA models. We largely adher to the standard notation,[1] however minor departures have been introduced for greater clarity in the present context. The

Multiplier Form:	
objective:	$\max \left[\sum_{j=1}^{J} u_{n'j} Y_{n'j}\right]$
constraints:	$1 = \sum_{i=1}^{I} v_{n'i} X_{n'i}$
constraints:	$s_n = \sum_{i=1}^{I} v_{ni} X_{ni} - \sum_{j=1}^{J} u_{nj} Y_{nj}$
variables:	$u_{nj} \geq 0,\quad v_{ni} \geq 0,\quad s_n \geq 0$
Envelopment Form: Phase 1	
objective:	$0 \leq \min\left[\theta_{n'}\right] = \theta^*_{n'} \leq 1$
constraints:	$s^-_{n'i} = \theta_{n'} X_{n'i} - \sum_{n=1}^{N} \lambda_{n'n} X_{ni}$
constraints:	$-s^+_{n'j} = Y_{n'j} - \sum_{n=1}^{N} \lambda_{n'n} Y_{nj}$
variables:	$\lambda_{n'n} \geq 0,\ s^+_{nj} \geq 0,\ s^-_{ni} \geq 0$
Envelopment Form: Phase 2	
objective:	$\max\left[\sum_{i=1}^{I} s^-_{n'i} + \sum_{j=1}^{J} s^+_{n'j}\right]$

TABLE I
Input-Oriented CCR Model

Multiplier Form:	
objective:	$\min \left[\sum_{i=1}^{I} p_{n'i} X_{n'i}\right]$
constraints:	$1 = \sum_{j=1}^{J} q_{n'j} Y_{n'j}$
constraints:	$t_n = \sum_{i=1}^{I} p_{nk} X_{nk} - \sum_{j=1}^{J} q_{nk} Y_{nk}$
variables:	$q_{nj} \geq 0,\quad p_{ni} \geq 0,\quad t_n \geq 0$
Envelopment Form: Phase 1	
objective:	$1 \leq \max\left[\eta_{n'}\right] = \eta^*_{n'} \leq \infty$
constraints:	$t^-_{n'i} = X_{n'i} - \sum_{n=1}^{N} \mu_{n'n} X_{ni}$
constraints:	$-t^+_{n'j} = \eta_{n'} Y_{n'j} - \sum_{n=1}^{N} \mu_{n'n} Y_{nj}$
variables:	$\mu_{n'n} \geq 0,\ t^+_{nj} \geq 0,\ t^-_{ni} \geq 0$
Envelopment Form: Phase 2	
objective:	$\max\left[\sum_{i=1}^{I} t^-_{n'i} + \sum_{j=1}^{J} t^+_{n'j}\right]$

TABLE II
Output-Oriented CCR Model

envelopment forms listed below constitute the first phase of a two-phase solution. The second phase involves maximization of the slacks and (or and) after substitution

There are many books[12][13] and web resources[14] that describe the numerical solution of linear programs and their application to various problems in economics and operations research. Typical applications of DEA in operations research involve solving few, large, sparse LPs. Much of the effort in numerically solving LPs that arise in such problems is devoted to performing pivoting in large sparse arrays. By contrast, the present analysis involves solution of many, small, dense LPs. We have therefore implemented the simple tableau algorithm with no sparse matrix technology given in *Numerical Recipes in C.*[11]

However, numerical instabilities would occasionally arise during evolution, generating solutions that violated various constraints to a significant degree. In addition to translating the code to double precision, we therefore wrote a driver for SIMPLX that would monitor the number of pivots executed within SIMPLX as well as the quality of the results it returned. If a specified number of pivots (*e.g.*, 250) was

Multiplier Form:	
objective:	$\max\left[\sum_{j=1}^{J} u_{n'j}Y_{n'j} - w_{n'}\right]$
constraints:	$1 = \sum_{i=1}^{I} v_{n'i}X_{n'i}$
constraints:	$s_n = \sum_{i=1}^{I} v_{ni}X_{ni} - \sum_{j=1}^{J} u_{nj}Y_{nj} + w_n$
variables:	$u_{nj} \geq 0, \quad v_{ni} \geq 0, \quad s_n \geq 0$
Envelopment Form: Phase 1	
objective:	$0 \leq \min\left[\theta_{n'}\right] = \theta^*_{n'} \leq 1$
constraints:	$0 \leq s^-_{n'i} = \theta_{n'}X_{n'i} - \sum_{n=1}^{N} \lambda_{n'n}X_{ni}$
constraints:	$0 \geq -s^+_{n'j} = Y_{n'j} - \sum_{n=1}^{N} \lambda_{n'n}Y_{nj}$
constraints:	$1 = \sum_{n=1}^{N} \lambda_{n'n}$
variables:	$\lambda_{n'n} \geq 0, \ s^+_{nj} \geq 0, \ s^-_{ni} \geq 0$
Envelopment Form: Phase 2	
objective:	$\max\left[\sum_{i=1}^{I} s^-_{n'i} + \sum_{j=1}^{J} s^+_{n'j}\right]$

TABLE III
Input-Oriented BCC Model

Multiplier Form:	
objective:	$\min\left[\sum_{i=1}^{I} p_{n'i}X_{n'i} - r_{n'}\right]$
constraint:	$1 = \sum_{j=1}^{J} q_{n'j}Y_{n'j}$
constraint:	$t_n = \sum_{i=1}^{I} p_{nk}X_{nk} - \sum_{j=1}^{J} q_{nk}Y_{nk} - r_{n'}$
variables:	$q_{nj} \geq 0, \quad p_{ni} \geq 0, \quad t_n \geq 0$
Envelopment Form: Phase 1	
objective:	$1 \leq \max\left[\eta_{n'}\right] = \eta^*_{n'} \leq \infty$
constraints:	$t^-_{n'i} = X_{n'i} - \sum_{n=1}^{N} \mu_{n'n}X_{ni}$
constraints:	$-t^+_{n'j} = \eta_{n'}Y_{n'j} - \sum_{n=1}^{N} \mu_{n'n}Y_{nj}$
constraints:	$1 = \sum_{n=1}^{N} \mu_{n'n}$
variables:	$\mu_{n'n} \geq 0, \ t^+_{nj} \geq 0, \ t^-_{ni} \geq 0$
Envelopment Form: Phase 2	
objective:	$\max\left[\sum_{i=1}^{I} t^-_{n'i} + \sum_{j=1}^{J} t^+_{n'j}\right]$

TABLE IV
Output-Oriented BCC Model

exceeded, of if the results returned by SIMPLX violated the imposed constraints by some specified numerical tolerance (*e.g.*, 10^{-6}), then the driver would randomly re-order the columns of the tableau and attempt solution by SIMPLX once again. This re-ordering could be repeated a specified number (*e.g.*, 250) of times. This strategy would be inappropriate for solution of a few large, sparse systems, but it improves numerical stability with relatively little additional computational cost when solving many small, dense systems, only a few of which require re-ordering. Care must be take to retain the map connecting the original and re-ordered columns so that the variables and slacks may be identified correctly after the LP has been solved.

After these modifications, the only numerical trouble encountered has been occasional cycling; that is, the specified number of re-orderings can occasionally be reached with no convergence having taken place. The most effect counter to this has been to identify and eliminate the clones in the population prior to computing DEA fitness. The subject of cycling is somewhat controversial in the DEA literature, and examples of genuine cycling are rare. We obviously do not exhaust all the possibile re-orderings of the columns; this would require solution of order 100! LPs for the current problem.

Acknowledgments

We wish to acknowledge financial support from Portland State University, and to express our gratitude to Dr. Eckart Zitzler of the Institut TIK in Zurich for providing us with additional data.

References

[1] William W. Cooper, Lawrence M. Seiford and Kaoru Tone, *Data Envelopment Analysis,* Kluwer (2000).

[2] Timothy R. Anderson and Keith B. Hollingsworth, *An Introduction to Data Envelopment Analysis in Technologh Management,* Proceedings of PICMET '97.

[3] Thomas Bäck, *Evolutionary Algorithms in Theory and Practice,* Oxford (1996).

[4] David B. Fogel, *Evolutionary Computation,* IEEE Press (1996).

[5] Yahya Rahmat-Samii and Eric Michielssen, *Electromagnetic Optimization by Genetic Algorithms.* Wiley (1999).

[6] G. Greenwood, Xiabo Hu and Joseph G. D'Ambrosio, *Fitness Functions for Multiple Objective Optimization Problems: Combining Preferences with Pareto Rankings,* Foundations of Genetic Algorithms 4, Morgan Kaufmann (1996).

[7] Carlos M. Fonseca and Peter J. Fleming, *Fitness Functions for Multiple objective Optimization Problems: Combining Preferences with Pareto Rankings,* Evolutionary Computation 3(1)1-16, Morga-kaufmann (1997).

[8] Eckart Zitzler and Lothar Thiele, *SPEA2: Improving the Strength Pareto Evolutionary Algorithm,* IEEE Transactions on Evolutionary Computing 3(4) 257-274 (1999).

[9] Eckart Zitzler, Marco Laumanns and Lothar Thiele, *Multiobjective Evolutionary Algorithms: A comparitive Case Study and the Strength Pareto Approach,* TIK Report 103 (2001).

[10] Eckart Zitzler, *Multiobjective Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications,* TIK SCHRIFTENREIHE Nr. 30 (1999).

[11] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes in C,* Cambridge (1996).

[12] Vašek Chvátal, *Linear Programming,* W. H. Freeman (1980).

[13] Robert J. Vanderbei, *Linear Programming: Foundations and Extensions,* Kluwer (1997).

[14] http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html