Analyzing Inefficiency Using a Frontier Search Approach

Bernd Ebersberger [*], Uwe Cantner[†], Horst Hanusch [‡]

December 6, 2000

Abstract

Efficiency measurement naturally requires the definition of a frontier as a benchmark indicating efficiency. Usually a measure reflecting the distance of a data point to the frontier indicates the level of efficiency. One of the crucial characteristics to distinguish efficiency measurement tools is the way in which they construct the frontier.

The class of deterministic and non parametric tools of constructing the frontier mainly comprises of tools associated with Data Envelopment Analysis. Coming in various flavors all DEA frontiers suffer of their piecewise construction giving rise to numerous vertices. Those vertices do not allow convenient analysis of the frontier properties such as computing elasticities and the like.

In this paper we want to contribute to the class of deterministic and non parametric tools of constructing the frontier in an one output and n input setting. We suggest a new empirical approach drawing on functional search in the fashion of Koza's (1992) genetic programming. The frontier search algorithm employed evolves the functional form of the frontier and the parameters simultaneously. The frontier exhibits the neat property that it is smooth and differentiable enabling the computation of elasticities for example.

In particular we introduce both the idea and the algorithm of the frontier search procedure. We discuss the advantages and shortcomings with respect to empirical problems. The arguments brought forth in the preceding sections are illustrated by the investigation of an artificial example.

1 Introduction

Any effort to determine the efficiency of several DMUs characterized by a given data set starts with the decision of the researcher how to construct the frontier that defines the efficient combinations of inputs and output. If one looks at the different methodologies that can be used one has to trade off smoothness of the frontier with information required to construct the frontier. The question

[*]Department of Economics, University of Augsburg, 86135 Augsburg, Germany, e-mail: bernd.ebersberger@wiso.uni-augsburg.de.

[†]Department of Economics, Friedrich-Schiller-University, 07743 Jena, Germany

[‡]Department of Economics, University of Augsburg, 86135 Augsburg, Germany,

We thank Rolf Färe, Laurens Cherchye and Thierry Post for helpful comments on an earlier version of the paper.

(among others) is whether to employ a parametric method to yield a smooth and differentiable frontier of an assumed shape or to use non-parametric methods and get a frontier that is not differentiable due to several edges and vertices. The latter clearly has an informational advantage, whereas the former has an appeal due to smoothness.

In the following we introduce a frontier search algorithm that tries to combine both advantages. The frontier constructed by the algorithm is smooth and differentiable and the construction of the frontier needs - we would claim - even less assumptions than the construction of a data envelopment frontier in the vein of Charnes, Cooper and Rhodes (1978) or Banker, Charnes and Cooper (1984) and others.

2 Construction of a frontier as a search problem

Inefficiencies of DMUs are assessed on the bases of their distance to the *best practice* set of DMUs or to the efficient frontier. As the true efficient frontier is unknown a key methodological issue arises in constructing the frontier from a given data set. Hence, all techniques used for efficiency measurement, search for an appropriate frontier to be able to asses the inefficiencies. Those techniques can be discriminated by two features:

- the assumptions made concerning the frontiers
- the procedures generating the frontier.

The first discriminating characteristic, referring to stochastic versus deterministic frontiers has been discussed extensively (e.g. Greene (1993), Lovell (1993)). With the latter we want to distinguish whether the frontiers are generated by a deterministic procedure or whether they are created by a randomized algorithm. This characteristic, however, has not yet caught any attention in the literature, as major efficiency measurement techniques, data envelopment analysis and stochastic frontier estimation in particular, employ deterministically generated frontiers for efficiency measurement. The differences of creating the frontiers by a deterministic procedure or by a randomized procedure becomes obvious if we look at the process of creating a frontier as being a search problem. The frontier then can be interpreted as the solution to the search problem. The different methodologies for efficiency measurement are equivalent to different strategies to finally arrive at the desired solution, frontier that is. If the strategy starting with identical situations, i.e. using the same data set and the same model with identical specification, always produces the same path through the search space to arrive at the solution then we call the procedure a deterministic one. In this case the final frontiers do not differ. If, however, the path through the search space varies even if the starting point is the same, we will speak of a randomized procedure. The procedure might not come up with the same solutions after repeated runs using the same starting conditions.

To illustrate these points we briefly turn to DEA and stochastic frontier in the following paragraphs before introducing the genetic frontier search algorithm.

2.1 Data envelopment analysis

A basic model

In data envelopment analysis the efficiency measure for any DMU is optimized, minimized or maximized depending on the orientation of the model, subject to some constraints reflecting the assumption posed on the technology as well as the production possibilities represented by the data. Take for example the linear program in (1), the dual of the output oriented BCC DEA model (Banker et al. (1984)), where the index z_0 for firm 0 has to be maximized:

$$\max_{\phi,\lambda_0,} \qquad z_0 = \phi_0 \tag{1}$$

subject to

$$\begin{aligned} \phi_0 y_0 - \mathbf{y}\lambda_\mathbf{0} &\leq 0 \\ \mathbf{X}\lambda_\mathbf{0} &\leq \mathbf{x_0} \\ \vec{1}\lambda_0 &= 1 \\ \lambda_0 &\geq 0 \end{aligned}$$

where ϕ denotes the efficiency measure, $\mathbf{y}$ is the D-vector of outputs y_i and $\mathbf{X}$ is the matrix and inputs. $\mathbf{x_i}$ indicates the vector associated with the DMU i and the index 0 points to the DMU under inspection. Computing the efficiency of all D DMUs in the data set requires D linear programs in the fashion of (1).

The frontier

The nature of the DEA formulation is, that the frontier function is not explicitely given, rather is it constructed within the model. No particular functional form is assumed for the frontier, in that sense it is non-parametric. In a more narrowly defined sense it is not completely non-parametric, as the model bases on the implicit assumption of piece-wise linearity. Deviation from the efficient frontier is assumed to be due to inefficiency only, giving rise to the assumption of a deterministic frontier.

The shape of the frontier, however, reveals a great disadvantage. The piece-wise linear composition of the frontier gives rise to several edges and vertices. Hence the frontier is not differentiable in the whole domain. So unique elasticities can not be computed for the entire domain. What makes things even worse is the fact that unique elasticities cannot be determined for a large fraction of the efficient units as edges and vertices are created by those.

The search

Solving the D linear programming problems in equation (1) is the process of searching for the values of the parameters ϕ_0 and λ_0 for each of the D DMUs. As ϕ_0 is a scalar and λ_0 is an D-vector the space to be searched to find the appropriate values is $R^{(1+D)D}$. As a search algorithm the basic Simplex-algorithm can be used to obtain the optimal values of the parameters. The Simplex-algorithm can be interpreted as a deterministic search algorithm guiding the search on a deterministic path through the search space to arrive at the optimal solution.

2.2 Stochastic frontier estimation

A basic model

Stochastic frontier estimation was introduced simultaneously by Meussen and van den Broek (1977) and Aigner, Lovell and Schmidt (1977). As the term indicates, it maintains a different assumption about the frontier. Deviations from the production function are due to two different influences. The first one concerns random deviations whereas the second one is attributed to inefficient production, in (2) denoted by v_i and u_i respectively. As simple model of the stochastic frontier is

$$y_i = g(\mathbf{X}_i, \beta) + v_i - u_i \tag{2}$$

where $g(.,.)$ is a production function specified in functional form β is the s-vector of parameters. v_i is drawn from a symmetric distribution, usually normal and u_i is drawn from a one-sided distribution, usually half-normal.

The frontier

The functional form of $g(.,.)$ has to be supplied beforehand. Hence stochastic frontier estimation represents a parametric estimation method. Most of the times the functional form of $g(.,.)$ is selected to yield a differentiable frontier. With a differentiable frontier, however, no problem arises when computing elasticities, such as scale elasticity or elasticities of substitution between input factors.

The search

In (2) β is the s-vector of the parameters to be searched for. The parameters have to be chosen such that the probability of realizing the given data set is maximized. Hence the search space is R^s, that can be searched by MLE algorithm guiding the search deterministically to the desired parameter vector.

2.3 Genetic Frontier Search

At this point of the discussion it seems as if there existed a trade-off between flexibility of the frontier created by a non-parametric method, in the case of DEA, and differentiability, in the case of a stochastic frontier. With the genetic frontier search algorithm we propose below, we attempt to losen the trade-off. In particular we introduce a search algorithm to find a frontier function with the property of being non-parametric and differentiable at the same time. The methodology, however, that we call genetic frontier search differs considerably from both DEA and stochastic frontier estimation as it is a randomized procedure for frontier construction.

In its primal representation DEA constructs the non-parametric frontier by searching for the appropriate weights. This burns down to pasting together linear pieces of the frontier. The stochastic frontier approach takes a given functional form and adjusts the parameters accordingly.

The idea of our frontier search algorithm is to combine both mechanisms. The frontier function does not have a predefined functional form, rather is it

composed of the parameters and several nested and combined primitive functions, such as `Plus`, `Minus`, `Times`, `Power`, `Exp` , ... The functional form and the parameters are searched for simultaneously. The next section will introduce the Genetic Frontier Search algorithm in more detail.

3 The Genetic Frontier Search Algorithm

3.1 The basic idea

The bases of the genetic frontier search algorithm can be found in the genetic programming approach recently introduced by Koza (1992) to solve complex optimization problems. A small collection of genetic programming applications in economics can be found in Koza (1992) and Schmertmann (1996).

The genetic frontier search algorithm constructs a sequence of populations each containing a number of candidate frontiers. Applying Darwinian ideas and simulating evolution on the populations the algorithm increases their overall quality – what we mean by quality of the population will be talked about in section 3.4 below. The best candidate frontier in the final population of the evolution is the result of the genetic frontier search.[1]

To start the search process the algorithm first generates a random population of a large number of individual frontier-functions. Those frontiers are evaluated on how good they envelop the given data cloud and are assigned a real number accordingly. This number indicates the fitness of the individual frontier function. The next generation is created by selecting the candidate frontiers depending on their fitness and applying genetic operators before they enter the new generation. Any individual frontier in this generation is again evaluated on how good it is in wrapping the data cloud. The best frontiers are selected and transformed to enter the successive generation .... and so forth. By repeating this procedure for several generations the fitness of the best individual frontier in the generation improves and a solution to the frontier search problem emerges.

The idea ruling the algorithm is the paradigm of evolutionary computing that good individuals are built from good components (building blocks, schema) and the recombination of those good building blocks leads to even better building blocks, creating ever improving individuals in the course of time.

Assumptions

Before we begin with setting up the algorithm we have to make a some assumptions for the frontier search to work.

Assumption 1 *We assume that there exists a best practice production frontier specified as* $y^0 = g(\mathbf{x}, \mathbf{z})$*, where* y^0 *represents the efficient level of production of a single output.* $\mathbf{x} = (x_1, x_2, \ldots, x_\nu)$ *is the vector of inputs and* $\mathbf{z}= (z_1, z_2, z_3, \ldots, z_k)$ *is a vector of parameters. Additionally* $g(\mathbf{x}, \mathbf{z})$ *is differentiable and concave in* $\mathbf{x}$*.*

[1] As the algorithm uses mechanisms that are analogous to concepts in evolutionary biology it has become common to use of the corresponding terminology as a metaphor only (see e.g. Koza (1992)) that might not be in perfect accordance with the features biologists attach to the terminology. Talking about a final generation of evolution is clearly flawed in terms of the biological idea of evolution. However in the case of the simulated evolution we have to stop the process at one point that will be discussed in section 3.6.

Assumption 2 *We assume that deviation from y^0 is only due to inefficient production. For the time being, there are no measurement errors to be considered.*

From assumption 2 we can see that $\epsilon_i = g(\mathbf{x}_i)/y_i$ is the a inefficiency measure for an observation i ($i = 1 \dots D$) where $\mathbf{x}_i$ is the vector of inputs and y_i is the output of DMU i.

Assumption 3 *We further assume that $g(.,.)$ is the function that gives rise to the minimal sum of the inefficiencies.*

Task

The task of the frontier search algorithm is to find a function $s^*(.,.)$ that approximates $g(.)$. This is searching for a function s^* that minimizes the overall inefficiency in the sample of D DMUs subject to the constraints imposed by assumption 1 and assumption 2.

The information available to the search process is the data set $(\mathbf{y}, \mathbf{X})$ containing the output and input information of D the DMUs. No prior knowledge about the functional form of $s^*(.,.)$ and both the number k and the size of the parameters $(z_1, z_2, z_3, \dots, z_k) = \mathbf{z}$ is available.

Instead the frontier search algorithm is supplied with two sets of components: $\mathcal{F} = \{f_1, f_2, f_3, \dots, f_k\}$ is the set of primitive functions and $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \dots, \tau_l\}$ is the set of terminals such as variables and constants. The frontier search tries to create s^* from the components of $\mathcal{F} \cup \mathcal{T}$ hence the search space S of the genetic frontier search contains any possible function s composed from elements of $\mathcal{F} \cup \mathcal{T}$.

This yields the following problem formulation:

$$\min \sum_{i=1}^{D} s(\mathbf{x}_i, \mathbf{z})/y_i \tag{3}$$

subject to

$$s(\mathbf{x}_i, \mathbf{z}) - y_i \text{ is not negative } \forall i$$
$$s(\mathbf{x}, \mathbf{z}) \text{ is concave in } \mathbf{x}$$
$$s(\mathbf{x}, \mathbf{z}) \text{ is differentiable in } \mathbf{x}$$
$$s(\mathbf{x}, \mathbf{z}) \text{ can be composed from elements of } \mathcal{F} \cup \mathcal{T}$$

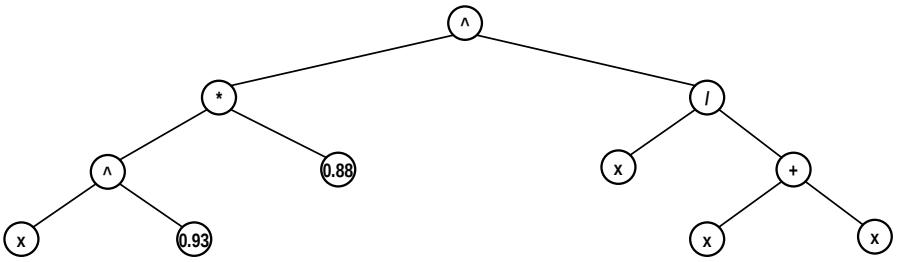
Note that the set of primitive functions and the set of terminals has to be defined such that their union complies with the requirement of *sufficiency* (Koza 1992). This demands that there exists a function built from the components of $\mathcal{F} \cup \mathcal{T}$ that complies with the constraints in (3). As we assumed that there exists a function $g(.,.)$ which is real valued by the nature of the problem, the Weierstrass' lemma (e.g. (Gamkrelidze 1990)) or Yao (1999) guide to sufficient $\mathcal{F}$ and $\mathcal{T}$. As one of the constraints in problem 3 points to differentiability of the solution the elements in $\mathcal{F}$ have to be differentiable as to yield a differentiable composite function s^*.

3.2 The Representation

As the frontier search algorithm searches for a function s by successively changing the representation of s, the frontier function has to represented as to allow

for easy modification within the algorithm. Therefore the candidate frontier functions are represented in a tree like structure. An example tree is displayed in figure 1.

Figure 1: An example tree



The frontier function displayed in figure 1 reads in standard mathematical notation: $(x^{0.93}0.88)^{\frac{x}{x+x}}$. The nodes of the trees are represented by primitive functions drawn from $\mathcal{F}$ whereas the leaves are elements drawn from $\mathcal{T}$. As we will see later the tree like representation of the frontier function enables easy manipulation of the frontier function.

3.3 The Population

The population $\mathcal{P}_t$ at time t is a set of N individuals each being a candidate frontier $s_{i,t}$ where $s_{i,t} \in S$, hence $\mathcal{P}_t \in \mathcal{S}^N$

$$\mathcal{P}_t = \{s_{1,t}, s_{2,t}, s_{3,t}, \ldots, s_{N,t}\}. \tag{4}$$

3.4 The Fitness Function

The measure how well each individual candidate frontier does its job in enveloping the data cloud defines the shape of the algorithm's result in the same way as a particular ecological niche shapes its inhabitants through natural evolution. Therefore all desired frontier properties should be taken account of while designing the fitness function.

As the task of the individual function is to represent a 'good' frontier to the data set the fitness function Φ assigns the candidate functions a real number, the fitness, indicating how good the candidate frontier envelopes the data cloud $(\mathbf{y}, \mathbf{X})$.

$$\Phi : \mathcal{S} \longrightarrow R \tag{5}$$

The fitness consists of at least two components. The first component depends on the sum of the distances ϵ_i between the frontier and the data cloud and serves as a measure of how close the frontier wraps the data cloud. It is this component of the fitness that directs the search towards frontiers that minimize the overall

distance between the frontier and the data cloud. Hence, it minimizes the overall inefficiency.

The second component maintains the assumption of a deterministic frontier as posed in assumption 2. Any negative deviation from the frontier is regarded as inefficient production. Positive deviation from the frontier will not be allowed as indicated in the constraints of problem (3). Any candidate frontier with data points 'above' it is not considered a valid frontier to the data set. Thus it is assigned a prohibitively bad fitness to prevent the frontier from contributing to the next generation.

At this point of the discussion we have to stress the flexible nature of the genetic frontier search algorithm. Any other property (e.g. certain scale properties etc.) that one wants the frontier to exhibit can be implemented in the fitness function by adding another component in the same fashion as component two.[2] Violation of the desired property will result in a prohibitively bad fitness.

We can now rank the elements of the population $\mathcal{P}_t$ according to their fitness. The indexing function $\omega(i)$ does the job. $\omega(1)$ is the index of the best individual in population $\mathcal{P}_t$, $\omega(2)$ is the index of the second best individual, $\omega(N)$ the index of the worst individual as measured by the fitness function. Hence $s_{\omega(1)}$ is the best individual in population $\mathcal{P}_t$.

3.5 The Intergenerational mapping

The crucial part in a genetic programming based algorithm is the intergenerational mapping G transforming generation $\mathcal{P}_t$ into $\mathcal{P}_{t+1}$.

$$G : \mathcal{S}^N \longrightarrow \mathcal{S}^N \tag{6}$$

The intergenerational mapping G includes the genetic operations that – together with the population approach – create the evolutionary nature of the algorithm. One can think of G being the composition of several primitive genetic operators: selection for parenthood (G_p), crossover (G_c), mutation (G_m) and selection to maintain a constant population size (G_s). .

3.5.1 Selection

G_p selects π parent individuals from the present population, those are going to be transformed to build offsprings in the subsequent population.

$$G_p : \mathcal{S}^N \longrightarrow \mathcal{S}^\pi \tag{7}$$

The selection criterion serves as the basic Darwinian mechanism to improve the overall fitness of the populations in the course of time. To implement the *survival of the fittest* it chooses the individuals randomly where the rule holds that the more favorable the fitness of an individual the more likely it is selected. Koza (1992) suggests three types of selection: *rank order selection*, *tournament selection*, and *fitness proportionate selection*. Rank order selection produces a weaker selection pressure more favorable to unfit candidate frontiers. It is

[2] In some cases, however, it might be computationally more convenient to specify a property that has to be fulfilled by the result of the algorithm but not by the best individuals of all preceding generations. If this is the case one can check the property in a post-processing step after termination of the algorithm and rerun the algorithm in case the property is violated.

often used when large differences in fitness within the population are present. Although we have large differences in the fitness created by the second component of the fitness function, we want them to exert their influence on selection. Tournament selection is the artificial analogon to two bulls fighting, the winner mating with a given cow and the offspring immediately being member of the present generation. This selection criterion does not support the notion of discrete generations. Hence we choose fitness proportionate selection for the genetic frontier search algorithm. The probability of a candidate frontier to be selected for parenthood is proportionate to the individual's normalized fitness which is an increasing function of the individual's quality.

3.5.2 Crossover

G_c creates one offspring from the π parents by applying the crossover operator.

$$G_c : \mathcal{S}^{\pi} \longrightarrow S \tag{8}$$

A subtree of each of the π parents is selected randomly, where each node of the tree has the same probability of being chosen as the root node of the subtree. Then the parents exchange the subtrees to create actually π offsprings. One of the offsprings is selected to be the offspring of the π parents.[3]

An example crossover involving two tree structures is displayed in figure 2.

3.5.3 Mutation

G_m takes the offspring generated by G_c and mutates it with a certain probability.

$$G_m : \mathcal{S} \longrightarrow \mathcal{S} \tag{9}$$

Subtree mutation chooses the root of a subtree randomly from the nodes and leafs of the tree being mutated. Then the adjacent subtree is replaced by a randomly generated subtree. If a leaf is selected as the root of the subtree and the randomly generated subtree only consists of a leaf we have the special case of a point mutation. Hence point mutation is a subclass of subtree mutation. Subtree mutation is depicted in figure 3.

3.5.4 Maintaining the population size

G_s is the genetic primitive function that maintains the population size at a level of N members. G_s is designed in a way as to select the N best individuals from an overpopulated population of ϱ members.

$$G_s : \mathcal{S}^{\varrho} \longrightarrow \mathcal{S}^{N} \tag{10}$$

The overpopulated population of ϱ members consists both of ϱ^O offsprings created by the genetic operations and of the ϱ^P best members of the parent generation indicated by the indexing function $\omega(i)$ where $i \in \{1, 2, \ldots \varrho^P\}$. The best candidate frontier in population $\mathcal{P}t$ is $s_{varrho(1),t}$, the second best candidate frontier in the population is $s_{varrho(1),t}$ and so forth.

[3]Let us assume that within the selection process following immediately after the crossover all π offsprings have the same probability of being selected. If we set out to choose the fittest among the offsprings, which implies the fitness function having been applied to each of the offsprings, we have a crossover similar to the brood crossover suggested by Tackett (1994).

Figure 2: Subtree crossover

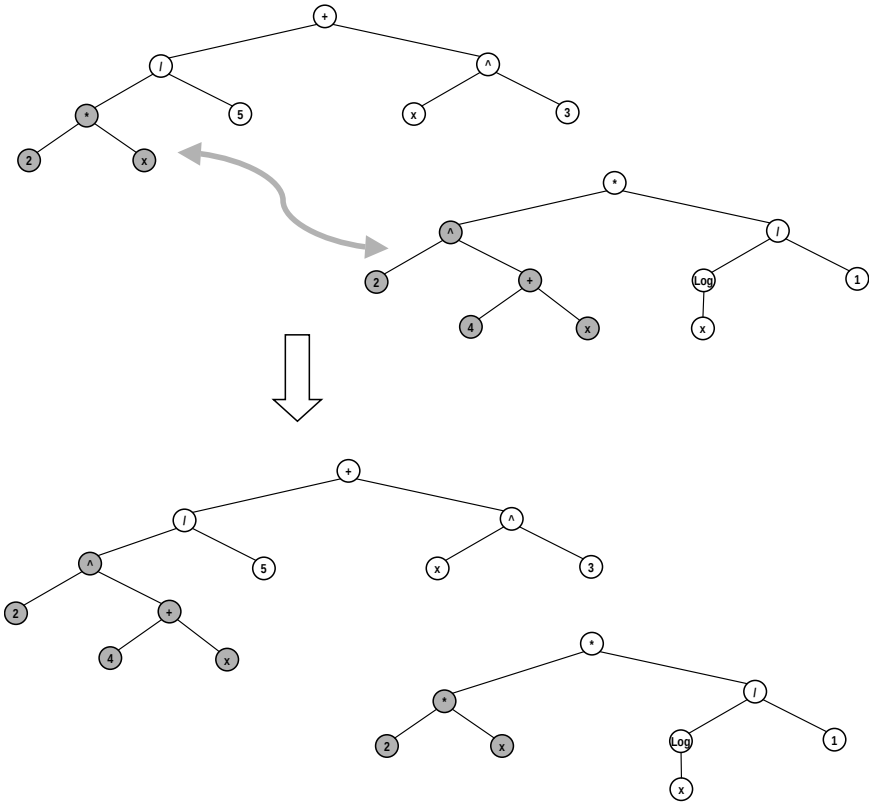
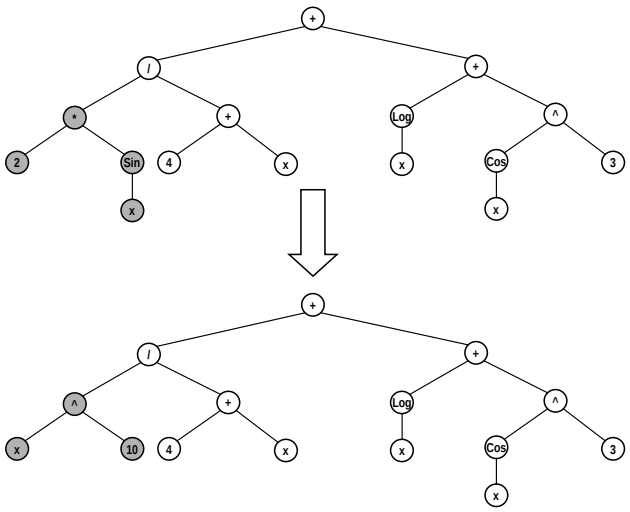


Figure 3: Subtree mutation



3.5.5 Composition of the intergenerational mapping

From the above components of the intergenerational mapping function G one can basically compose two different configurations that can constitute a frontier search routine. The first and simpler configuration is composed only of G_p, G_c and G_s which amounts to a genetic programming algorithm only consisting of selection, reproduction and crossover.

$$G_1(.) = G_s(\{\underbrace{G_c \circ C_p(.), G_c \circ G_p(.), \ldots, G_c \circ G_p(.)}_{\varrho^P}\} \cup \{\underbrace{s_{\omega(1),t}, s_{\omega(2),t} \ldots}_{\varrho^O}\}) \quad (11)$$

The second configuration G_2 of the intergenerational mapping function G can be composed of all primitive functions G_p, G_c, G_m and G_s.

$$G_2(.) = G_s(\{\underbrace{G_m \circ G_c \circ C_p(.), \ldots, G_m \circ G_c \circ G_p(.)}_{\varrho^P}\} \cup \{\underbrace{s_{\omega(1),t}, s_{\omega(2),t} \ldots}_{\varrho^O}\}) \quad (12)$$

It can be shown that the intergenerational mapping G_1 is not sufficient to guarantee that the solution s^*, even though it exists, will be found (Ebersberger 2000). However configuration G_2 secures that the solution s^* will be found by the search algorithm within a finite number of iterations regardless of the initial population $\mathcal{P}_0$ (Rudolph 1998, Ebersberger 2000). This leads us to abandoning G_1 and to use configuration G_2 of the intergenerational mapping for the frontier search.

3.6 The termination condition

In many applications of the genetic programming approach it might be useful to stop the evaluation once a certain level of approximation is achieved. As the fitness of each candidate frontier indicates a measure of the overall inefficiency being present in the sample and there is no a priori knowledge on its size, we cannot implement a criterion terminating the run in case the fitness falls below a certain threshold. So the run terminates after a maximum number of γ generations.

Although there is no general rule that can be applied to figure out the optimal number of generations, it can be concluded that the maximum number of generations should increase with the complexity of the situation analyzed. Hence it should grow with the number of DMU's and with the number of inputs. In our example application here we set the maximum number of generations γ to 50.

At this point of the discussion it becomes obvious, what purpose G_s serves in the intergenerational mapping function. G_s allows to transfer the best individual of the parent generation to the next generation if the offsprings do not surpass the best fitness in the parent generation. By doing this we can make sure that the fitness of the best individual does not decrease over time. Hence a too large maximum number of generations γ does not harm the fitness of best individual, merely it increases computation time.

3.7 The algorithm

Having defined the necessary components and their properties we can now state the frontier search algorithm briefly:

Figure 4: The frontier search algorithm

Create $\mathcal{P}_0$ from $\mathcal{F}$ and $\mathcal{T}$
$t = 0$
Repeat
    $\mathcal{P}_{t+1} = G_2(\mathcal{P}_t)$
    $t = t + 1$
for γ generations
Return $s_{\omega(1),t}$

4 An artificial example

To illustrate the properties of the frontier search and to show how the genetic frontier search algorithm approximates a given data set, we simulated a one input one output data set with 20 DMUs.

Table 1: Data

DMU	Input	Output	Eff.Score
DMU1	0.384731	0.148735	1.00
DMU2	0.434705	0.685458	1.82
DMU3	0.608727	0.864081	1.45
DMU4	0.415110	0.201750	1.07
DMU5	0.616597	0.564039	1.17
DMU6	0.535586	0.667698	1.46
DMU7	0.544541	0.775683	1.54
DMU8	0.438632	0.537460	1.61
DMU9	0.398157	0.256440	1.25
DMU10	0.369565	0.405442	1.67
DMU11	0.603906	0.442009	1.06
DMU12	0.187088	0.114276	1.82
DMU13	0.330528	0.131704	1.10
DMU14	0.430582	0.664070	1.81
DMU15	0.717651	0.901329	1.25
DMU16	0.114361	0.035825	1.73
DMU17	0.802817	0.737389	1.02
DMU18	0.824814	0.755796	1.00
DMU19	0.786873	0.928470	1.16
DMU20	0.197487	0.053924	1.21

4.1 The artificial data set

Table 1 shows the input and output data of 20 artificial units for which the best practice frontier is generated by the genetic frontier search algorithm.[4] We simulated a production function $y = rx^{0.5}$, where x denotes the normalized inputs taken randomly from $]0, 1]$ and r is a random inefficiency term drawn from a uniform distribution in $[0.5, 1]$. Thus the best-practice frontier represented by our sample lies on or below the graph of $x^{0.5}$.

4.2 Specification of the frontier search algorithm

Table 2 shows the default specification of the frontier search algorithm for our example here.

Table 2: Specification of the search algorithm

Characteristic	default value
Objective	Find the best-practice frontier of the data set
Function set	Plus, Minus, Times, Divide, Power, Exp
Terminal set	x, Random constants from $[0.8, 1]$
Data set	Data set displayed in table 1
Fitness	Sum of errors + 10^{10} if data point is above the frontier
Genetic operators	Copy, reproduction, crossover, mutation
Halting condition	Terminate after generation 50.
No. of individuals	200.
Post processing	Check for concavity. Transform the function to standard mathematical notation.
Peculiarities	Crossover implemented as brood crossover (Tackett 1994). Adaptive parsimony pressure (Zhang and Mühlenbein 1995)

[4] Column 4 already contains the inefficiency measures as they will be computed for the frontier genterated by the genetic frontier search algorithm.

4.3 Evolution of the frontier

Starting with 200 randomly created candidate frontiers the algorithm evolves a highly fit frontier for the data within 17 generations. Figures 5 to 10 show the graphs of the best individual frontiers in generation 1, 2, 3, 4, 5 and 17.

We observe the frontier bending and moving towards the data cloud, finally wrapping it quite neatly. The frontier is $y = 0.9423\, x^{0.4692}$. It gives rise to the inefficiencies displayed in column 4 of table 1. This production frontier is differentiable on $[0, 1]$, hence it causes no trouble when it comes to calculating the scale elasticity. In this respect the frontier generated by the genetic frontier search algorithm has a clear advantage compared to the non-parametric DEA methodology.

4.4 Genotype and phenotype

The simulated evolution of the frontier can be traced in the graphs in figure 5 to figure 10. The features of the frontiers like their graphical representation or the overall inefficiency it gives rise to are the phenotype. It is the phenotype that determines whether the individual will be selected for parenthood and the genotype will be passed on to the next generation.

The genotype, the tree representation, of the best individuals is depicted in figures 11 to 16. It changes both structure (the shape of the trees) and content (the content of the nodes and leaves) simultaneously. Looking at the phenotype we see that the simultaneous change of structure and content of the genotype is the true cause for evolving the functional form and the parameters simultaneously.

4.5 Inventing necessary components of the frontier

The genetic frontier search exhibits a nice and desirable feature one would be inclined to call *creativity*. It creates components that are necessary for a frontier from the given function set and the given set of variables and constants. To illustrate this in our example we restricted the range of random constants from 0.8 to 1.0, knowing from the simulated data that a good candidate frontier would need an exponent less than or equal to 0.5.

Hence the algorithm has to find a way to accommodate this restriction in the supply of components. As we can see in figure 16 It sets out to take the square root of the constant 0.9385 to create the constant 0.4692. As square root is not supplied as a primitive function the algorithm used the power function. The necessary exponent 0.5, again not available as a constant, is created by $\frac{x}{x+x}$.

This feature, generating almost any function and constant from the given sets of functions, constants and variables, relaxes the need for the researcher to supply any possible component. The non parametric nature of the algorithm is underlined once again. This, however, does not mean that the researcher can supply what ever he feels like supplying and the algorithm will remedy any shortcoming. Any component that is not supplied and has to be created endogenously by the algorithm reduces its efficiency and increases the computational complexity considerably.

Figure 5: Best candidate frontier $s_{\omega(1),1}$ in generation $\mathcal{P}_1$

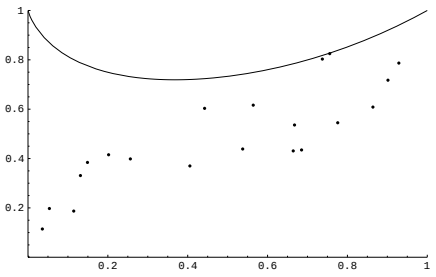


Figure 6: Best candidate frontier $s_{\omega(1),2}$ in generation $\mathcal{P}_2$

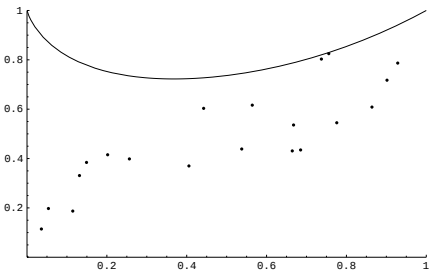


Figure 7: Best candidate frontier $s_{\omega(1),3}$ in generation $\mathcal{P}_3$

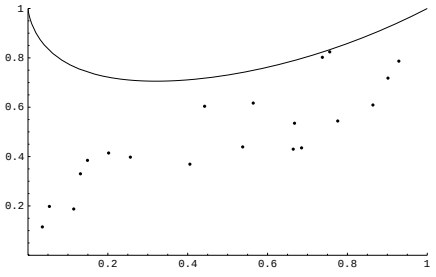


Figure 8: Best candidate frontier $s_{\omega(1),4}$ in generation $\mathcal{P}_4$

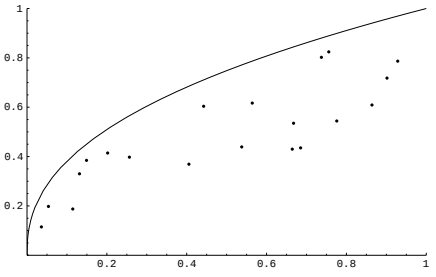


Figure 9: Best candidate frontier $s_{\omega(1),5}$ in generation $\mathcal{P}_5$

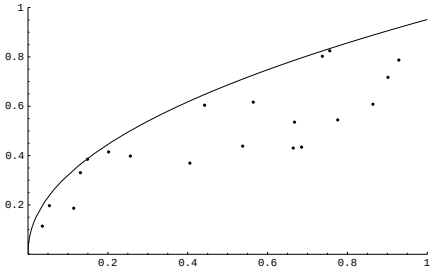


Figure 10: Best candidate frontier $s_{\omega(1),17}$ in generation $\mathcal{P}_{17}$

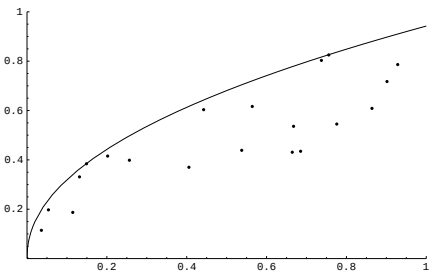
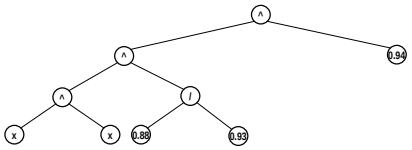


Figure 11: Best candidate frontier $s_{\omega(1),1}$ in generation $\mathcal{P}_1$

Figure 12: Best candidate frontier $s_{\omega(1),2}$ in generation $\mathcal{P}_2$

$x^{x^{0.896288}} = x^{x^{0.8962}}$



$x^{x^{0.8833}} = x^{x^{0.8833}}$

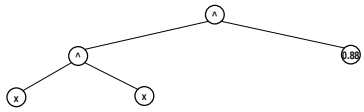
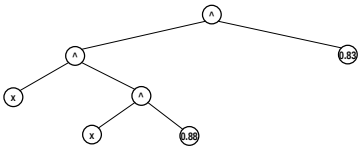


Figure 13: Best candidate frontier $s_{\omega(1),3}$ in generation $\mathcal{P}_3$

Figure 14: Best candidate frontier $s_{\omega(1),4}$ in generation $\mathcal{P}_4$

$x^{x^{0.8798}\,0.8342} = x^{x^{0.8797}\,0.8342}$



$x^{0.888\,\frac{x}{x+x}\,0.9428} = x^{0.4186}$

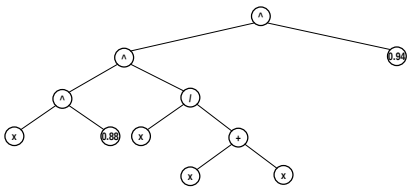
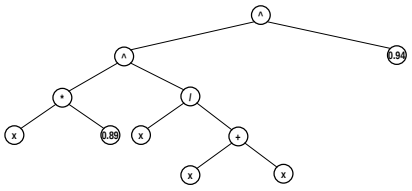


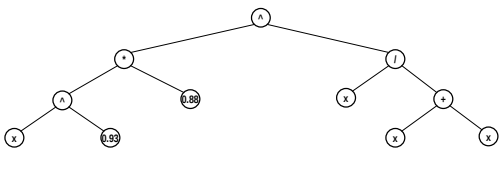
Figure 15: Best candidate frontier $s_{\omega(1),5}$ in generation $\mathcal{P}_5$

Figure 16: Best candidate frontier $s_{\omega(1),17}$ in generation $\mathcal{P}_{17}$

$(x\,0.8999)^{\frac{x}{x+x}\,0.9428} = 0.9515\,x^{0.4714}$



$\left(x^{0.9385}\,0.888\right)^{\frac{x}{x+x}} = 0.9423\,x^{0.4692}$



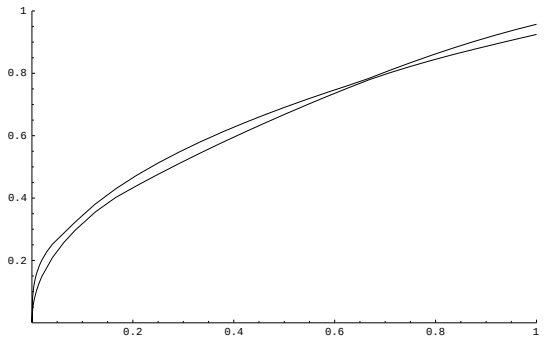
5 Properties of the frontier search algorithm

After having displayed an example to the algorithm we can summarize the properties of the genetic frontier search. The genetic frontier search algorithm creates a random initial population to start with. It further more selects the parent frontiers randomly, though based on their individual fitness. Hence, the algorithm generates a randomized path through the search space. With this the genetic frontier search markedly differs from both data envelopment analysis and stochastic frontier.

By nature of a randomized procedure one cannot guarantee that the algorithm generates identical results starting from the same point. The positive features of the algorithm, its non-parametric nature and the resulting frontier being supplied in a functional form are traded off by the potential disadvantages a randomized search creates – there is no free lunch in frontier construction.

To examine the extent of the randomized nature of the algorithm we run it on the data set of table 1 for another 22 times. The upper and the lower bounds of the corridor in which all solutions lay are depicted in figure 17.

Figure 17: Upper and lower bounds of the solutions found



This picture suggests that in our case the randomized nature of the algorithm does not cause large differences in the results. The algorithm produces comparable results in the different runs we performed on the same data set. To verify this result on a reliable bases monte-carlo simulation would be necessary.

Table 3 summarizes the properties of DEA, stochastic frontier and the frontier search algorithm introduced here.

6 Conclusion

This paper introduced a genetic programming based methodology to determine production frontiers. We argue that this procedure combines flexibility on the functional form with differentiability. The first being the result of the strict non-parametric nature of the method and the latter being caused by generating an explicit representation of the frontier. To accomplish this beneficial combination of characteristics we have to accept a randomized procedure to create the frontier.

Table 3: Properties of the frontier search

property	DEA	stochastic frontier	Genetic Frontier Search
frontier representation	implicit	explicit	explicit
frontier properties	not differentiable deterministic	differentiable stochastic	differentiable deterministic
search algorithm	deterministic	deterministic	randomized
assumption	piecewise linear frontier	functional form and distribution of the errors	primitive function set, domain of the constants
computational intensity	low	low	high

Having discussed the elements of the approach a simple artificial example has been investigated. The results show

1. that the Genetic Frontier Search comes considerably close to the (hidden) functional relationship and
2. that despite the randomized nature of the search several runs lead to results within a small range. This at least holds for the simple example.

We can conclude that the genetic frontier search algorithm presented here enhances the tool kit of productivity analysis with a methodology that constructs non-parametric, differentiable frontier functions in a n input and one output setting. Further research should enhance the algorithm to a n input and m output case. Ray production frontiers (see e.g. Kumbhakar (1996)) might be a good starting point in this regard.

References

Aigner, D. J., Lovell, C. A. K. and Schmidt, P.: 1977, Formulation and estimation of stochastic frontier production function models, *Journal of Econometrics* **5**, 21–38.

Banker, R. D., Charnes, A. and Cooper, W. W.: 1984, Some models for estimation technical and scale inefficiencies in data envelopment analysis, *Management Science* **30**, 1078–92.

Charnes, A., Cooper, W. W. and Rhodes, E.: 1978, Measuring the efficiency of decision making units, *European Journal of Operational Research* **2**, 429–444.

Ebersberger, B.: 2000, *Functional Search - Bound to fail?*, mimeo, University of Augsburg, Augsburg.

Gamkrelidze, R. V.: 1990, *Analysis II*, Vol. 14 of *Encyckopedia of Mathematical Sciences*, Springer, Heidelberg, New York.

Greene, W. H.: 1993, The econometric approach to efficiency analysis, *in* H. O. Fried, K. C. A. Lovell and S. S. Schmitt (eds), *The Measurement of Productive Efficiency*, Oxford University Press, New York, Oxford, pp. 68–119.

Koza, J.: 1992, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA.

Kumbhakar, S. C.: 1996, Efficiency measurement with multiple outputs and multiple inputs, *Journal of Productivity Analysis* **7**, 225–255.

Lovell, K. C. A.: 1993, Production frontiers and productive efficiency, *in* H. O. Fried, K. C. A. Lovell and S. S. Schmitt (eds), *The Measurement of Productive Efficiency*, Oxford University Press, New York, Oxford, pp. 3–67.

Meussen, W. and van den Broek, J.: 1977, Efficiency estimation from cobb-douglas production functions with composed error, *International Economic Review* **15**, 435–444.

Rudolph, G.: 1998, Finite markow chain results in evolutionary computation: A tour d'horizon, *Fundamentata Informaticae* pp. 1–22.

Schmertmann, C. P.: 1996, Functional search in economics using genetic programming, *Computational Economics* **9**, 275–298.

Tackett, W. A.: 1994, *Recombination, Selection, and the Genetic Construction of Computer Programs*, PhD thesis, University of Southern California, Department of Electrical Engingeering Systems.

Yao, X.: 1999, Universal approximation by genetic programming, *in* T. Haynes, W. B. Langdon, U.-M. O'Reilly, R. Poli and R. Rosca (eds), *Foundations of Genetic Programming*, Orlando, Florida, USA.

Zhang, B. T. and Mühlenbein, H.: 1995, Balancing accuracy and parsimony in genetic programming, *Evolutionary Computation* **3**, 17–38.