Parallel and Hierarchical Decomposition Approaches for Solving Large-Scale Data Envelopment Analysis Models*

RICHARD S. BARR[1] AND MATTHEW L. DURCHHOLZ[2]

[1] *Department of Computer Science and Engineering*
Southern Methodist University
Dallas, Texas 75275
E-mail: barr@seas.smu.edu

[2] *i2 Technologies*
Dallas, Texas
E-mail: matthew_durchholz@i2.com

Accompanying the increasing popularity of DEA are computationally challenging applications: large-scale problems involving the solution of thousands of linear programs. This paper describes a new problem decomposition procedure which dramatically expedites the solution of these computationally intense problems and fully exploits parallel processing environments. Testing of a new DEA code based on this approach is reported for a wide range of problems, including the largest reported to date: a 8,700-LP banking-industry application.

Keywords: Parallel computing, data envelopment analysis, decomposition, mathematical programming

Surprisingly little has been published on the computational aspects of Data Envelopment Analysis [1, 2, 3, 4, 5, 23, 31]. Since DEA typically involves the solution of a large number of linear programs (LPs), many practitioners and researchers assume that the repeated use of standard optimization codes is sufficient for an analysis. Unfortunately this is not the case. Specialized codes are needed to correctly handle the preemptive prioritized multiple objectives (reflecting the models' non-Archimedian infinitesimal) and to coordinate and expedite the solution of the large number of interrelated LPs.

This study was motivated by large applications we have encountered: franchise analysis (e.g., over 8,000 McDonald's restaurants, 6,500 Century 21 real-estate offices, and approximately 5,000 H&R Block tax-preparation service centers), the Federal Reserve Bank's efficiency study of 8,700 U.S. banks, a V.A. study of over 20,000 hospitals, and U.S. Postal Service evaluations of over 30,000 branches. These problem sizes are clearly beyond the limits of current DEA codes.

In this paper, we describe a new code for solving large-scale DEA problems in a reasonable amount of time, demonstrate its performance on a real-world application, and

*This work was supported by National Science Foundation grant DMII-9313346 and the United States Air Force report on its ability to exploit parallel processing to further accelerate solution time. We also introduce a new decomposition algorithm that streamlines the solution of problem sets and provide in-depth computational testing of the code on small- and large-scale problems, including the largest DEA problems reported to date.

1 Computational DEA

Key to the development of software for putting DEA into practice are the mathematical underpinnings and associated means of exploiting domain-specific structure for computational gain. We now briefly summarize relevant DEA concepts, describe efficient implementation techniques, and present the results of computational testing of a new DEA research code.

1.1 DEA Fundamentals

Data Envelopment Analysis is a family of models for assessing and analyzing the relative transformational efficiency of similar decision-making units [21]. All DEA models have the same data requirements: for each *decision-making unit* (DMU) j, a nonnegative observed value for each of the s outputs (Y_j) and m inputs (X_j), with at least one positive input or output. A DMU's *efficiency* θ ($0 < \theta \leq 1$), is defined as a weighted ratio of the total output produced to the total input consumed, with weights determined by a separate linear program for each DMU in the analysis.

The various DEA models separate the set of n DMUs (D) into efficient (E^*) and inefficient (I^*) sets. A DMU is *efficient* if $\theta = 1, \mathbf{s}^o = \mathbf{0}$, and $\mathbf{s}^i = \mathbf{0}$, where $\mathbf{s}^o$ and $\mathbf{s}^i$ are slacks in the associated LP's input and output constraints, respectively. Mathematically the members of E^* are extreme points, extreme rays, or lie on a convex surface which, when taken with their associated facets, form a piecewise-linear empirical production surface, or *efficient frontier*. Otherwise, the DMU is deemed *inefficient* and the observation lies within, not on, the efficient frontier. For each DMU $d \in I^*$, the corresponding linear program's solution suggests a set of inputs and outputs that would make d efficient. This *virtual DMU* is obtained by a projection onto the efficient frontier, using the vector $\boldsymbol{\lambda}^* \in \Re^n$ to form a linear combination of d's *reference set*—those efficient DMUs with which d is being directly compared.

Although there are many formulations of DEA models (see Appendix A), the useful results in each case are similar in nature and are to be determined in practice by a DEA code. A complete data envelopment analysis produces, for each DMU: θ; a set of weights, $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$, for the outputs and inputs, respectively; and, for inefficient units, $\mathbf{s}^o$, $\mathbf{s}^i$, $\boldsymbol{\lambda}$, and a reference set. Section 2 describes some of the techniques that have been used to expedite the calculation of these items.

In the upcoming discussion, the following notation will be used. We define: $D = \{1, \ldots, n\}$, the index set of DMUs; B_j^* ($\overline{B}_j^*$) is the set of basic (nonbasic) variables associated with an optimal basis for an envelopment-form DEA model applied to DMU $j \in D$; $\theta_j^*, \mathbf{s}_j^{o*}$, and $\mathbf{s}_j^{i*}$ are the values of $\theta, \mathbf{s}^o$, and $\mathbf{s}^i$, respectively, in B_j^*; $\Lambda_j = \{i \in D | \lambda_i \in B_j^*\}$; $\overline{\lambda}_i$ is the reduced cost of DMU weight λ_i in a given optimal solution; $E^* = \{j \in D | \theta_j = 1, \mathbf{s}_j^{o*} = \mathbf{0}$, and $\mathbf{s}_j^{i*} = \mathbf{0}\}$; and $I^* = D - E^*$.

2 Survey of Computational Efficiency Techniques

A naive implementation of the DEA calculations can lead to unreasonable solution times for all but the smallest problems. This section surveys the variety of acceleration techniques that have been devised to exploit the special characteristics of DEA models. Proofs of the theorems can be found in [9].

2.1 Choice of Model Form

For each DEA model (CCR, BCC, NDRS, NIRS, and Additive) either the envelopment or the multiplier forms may be used to compute solutions. Since typically $(s+m) \ll n$, there will be fewer constraints and a smaller basis inverse to be maintained with the envelopment form, used by virtually all DEA codes. This choice of model becomes an increasingly favorable one as the size of the data set expands. This advantage can be further exploited as the analysis reveals the memberships of E^* and I^*.

2.2 Early Identification of Efficient DMUs

It has been observed in [3, 18] that if $\lambda_i, i \in D$, is a member of any $B^*_j, j \in D$, then DMU i is efficient. The following theorem holds for all standard DEA models.

Theorem 1
For any $j \in D$, $\Lambda_j \subseteq E^*$.

The early identification of members of $E \subset E^*$ permits: (1) bypassing unsolved subproblems associated with members of E, if values of the weights and slacks are not needed; (2) near-optimal advanced starting solutions for known efficient DMUs; and (3) pricing strategies that give priority to members of E. This can then reduce the number of LP subproblems to be solved and expedite the solution of subproblems that must be optimized.

In software implementations, the status of each DMU j, and its associated λ_j, can be easily maintained as: member of $E \subseteq E^*$, member of $I \subseteq I^*$, or unknown (member of $U = D - E - I$). When the subproblem associated with DMU j is solved, j can be added to E or I. In addition, B^*_j may reveal new members of E—from the reference set, if DMU j is inefficient, or from degenerate basic variables, if it is efficient.

Corollary 2
If, at optimality, $\overline{\lambda}_i = 0, i \in D$, then $i \in E^*$.

This corollary supports an extended search for members of E associated with the nonbasic variables. Our experience is that there are often a substantial number of alternate optimal solutions and nonbasic variables with zero reduced costs.

2.3 Restricted Basis Entry

Since any λ_j associated with an optimal basis corresponds to an efficient DMU j, we need only consider those variables associated with $D - I$ when pricing. Hence as inefficient DMUs are identified, their corresponding decision variables are dropped from subsequent problems. This has a ratchet-like effect on the size (number of variables) of the LPs being solved, causing "later" subproblems in a series to solve more quickly than the "earlier" ones.

2.4 Candidate List

Candidate lists, and other multi-pricing schemes, are standard computational procedures for expediting the solution of linear programs [29, 30]. These basically involve pricing a set of nonbasic variables to identify a short list of promising basis-entry candidates, from which incoming variables are repeatedly selected before replenishment.

This heuristic can be specialized for DEA models. First, the restricted basis entry procedure eliminates from consideration variables associated with I. Further, priority for list inclusion can be given to members of E over U, since any basis will contain only slacks and members of E^* [3].

2.5 Degeneracy and Anti-Cycling Logic

Note that if DMU j is efficient, the optimal basic solution for the envelopment model can be comprised of $\theta = \lambda_j = 1$, with all other variables equal to zero. When such degeneracy exists, so does the possibility of cycling. Although dismissed by many as unlikely, we find that cycling in DEA codes is not only common but likely in the absence of countermeasures.

The standard methods for avoiding cycling within the simplex method—lexicographic ordering, perturbation [17], and Bland's rule [15]—can be computationally demanding. Charnes, Rousseau, and Semple [23] observed an absence of stalling and substantially fewer pivots when lexicographic ordering was added to a DEA code. A specialized ratio test is proposed in [3] that gives preference to $\boldsymbol{\lambda}$ variables over slacks when breaking ties for zero minimum ratios, and yields large decreases in the number of pivots and solution times on reported test problems.

Our experience indicates that, for many problems, simple scaling of the problem data—by variables and by constraints, normalizing by averages [14]—is most effective in lowering the incidence of stalling and cycling. In those cases where a lack of progress is detected, the invocation of the lexicographic ordering procedure quickly remedies the situation.

2.6 Non-Archimedean Infinitesimal, ε

When DEA codes use small positive values (e.g., 10^{-6}) for the non-Archimedean infinitesimal, ε, the results of CCR and BCC analyses are dependent on the value chosen, and are incorrect in many cases [5]. Such data dependencies can be avoided and correctness ensured by recasting envelopment-model objective functions in a preemptive priority form: minimize $P_1\theta + P_2(-\mathbf{1s}^i - \mathbf{1s}^o)$. Proposed implementation schemes include two-stage optimization [1], and a multi-objective pricing form of the simplex method [23].

2.7 Preprocessing of Observations

An analysis of the DMUs' observations can lead to identification of members of I and E prior to the application of optimization. A DMU i is inefficient if it is dominated by any DMU j, that is if $\mathbf{X}_j \leq \mathbf{X}_i$ and $\mathbf{Y}_j \geq \mathbf{Y}_i$. In the BCC and additive models, DMU j is efficient if one of its outputs (inputs) has a unique maximum (minimum) value, or it has the largest ratio $O_j = \mathbf{1Y}_j/\mathbf{1X}_j$. Processing DMUs in O_j order is reported to identify members of E^* earlier than random order.

2.8 Advanced Starting Bases and Reoptimization

In many algorithms that involve the solution of a series of related linear programs, or subproblems, the use of *reoptimization* can lead to substantial time savings. This is accomplished by using B_i^* as an initial basic feasible solution for the LP associated with DMU j, as is possible when subproblems differ by one or more parameters (objective function or right-hand-side values) but have the same mathematical structure.

DEA subproblems can employ reoptimization, but its effectiveness is dependent on the mathematical "closeness" of the two solution points, B_i^* and B_j^*. Ali suggests the use of this technique only for $j \in \{\Lambda_i - E\}$ [3].

3 PIONEER Code

We have developed and tested PIONEER, a new DEA code which embodies most of the above efficiency techniques[7]. The code was designed for testing and evaluating alternative solution approaches to large-scale problems[8].

PIONEER's optimization kernel is based on the XMP library, a collection of portable, reliable, and widely used Fortran subroutines for the solution of linear programs using the primal and dual simplex methods [28]. The LP basis is maintained in LU-factorized form with pivoting routines from the Harwell library. Our customization includes data scaling and the invocation of lexicographic ordering when stalling is detected.

The PIONEER code implements all varieties of DEA models described in Appendix A and all of the efficiency techniques in Section 1.2 are employed, except for preprocessing and reoptimization. (Our testing showed that reoptimization was of uneven value—sometimes reducing, sometimes increasing, run times.) The candidate list is not prioritized, and the two-stage optimization approach to the non-Archimedean-infinitesimal issue is employed.

Auxiliary data structures maintain the E, I, or U status of each DMU, as indicated by the following outline of the code's logic:

PIONEER DEA CODE SOLUTION OUTLINE

```
Step 0   Initialize $U = D, E = \emptyset, I = \emptyset$.
Step 1   While $U \neq \emptyset$ :
             1. Select $i \in U$.
             2. Solve the subproblem for DMU $i$.
```

3. If DMU i is efficient, $E = E \cup \{i\}$, else $I = I \cup \{i\}$.
4. Update: $E = E \cup \Lambda_i \cup \{j|\overline{\lambda}_j = 0\}$, and $U = D - E - I$.

The strength of step 1.4 depends on Theorem 1, Corollary 2, and the distribution of observations in $\Re^{(s+m)}$, relative to the efficient frontier.

It should be noted that while the XMP data structures are designed for sparse problems, DEA subproblems are almost totally dense. Hence there is unnecessary processing in the code and the reported execution times should be considered conservative measures of performance, relative to a fully customized implementation.

4 Baseline Computational Testing

To evaluate the efficiency of the PIONEER research code, and to provide a set of baseline measurements from which to compare algorithmic and implementational enhancements, a series of test runs were made on medium- and large-scale problems. Testing with and without individual efficiency techniques lent insight into their impact on solution speed.

4.1 Test Data

Although the code was validated for solution accuracy on small problems from the open literature, of primary interest was its effectiveness on large-scale problem sets. Data from three sources were used in the testing.

First, the Federal Reserve Bank of Dallas provided a data set of 8,748 banks from its Southwest district, with variables consisting of the six inputs and three outputs described in [12, 13]. This challenging problem from industry was clearly beyond the state of the art of DEA codes, and was a prime motivator for this research. Testing was performed on smaller subsets by selecting the first n observations from the unordered file.

Randomly generated observations from a multinormal population provided a second source of test data. Using the DRNMVN routine from the IMSL library and the variance-covariance matrix given in Appendix B, large data sets could be easily created. The variables were partitioned into 3 inputs and 4 outputs so as to observe positive correlations between the "input" and "output" sets.

Finally, a random problem generator, DEA-GEN, was written to create observations based on the classic Cobb-Douglas form of production surfaces. As detailed in Appendix C, the program gives the user a measure of control over the proportion of efficient points, and creates data sets that more closely approximate realistic economic processes than the multinormal generator.

4.2 Test Environment

The PIONEER code was tested on Southern Methodist University's Sequent Symmetry S81B with 32MB of internal storage and processing units consisting of 16-MHz Intel 80386s with Weitek coprocessors. The software is written entirely in Fortran and executed under Dynix 3.0.12, a BSD-Unix-based operating system. While the processors are rated at 4 million operations per second, in terms of current technology they are equivalent to relatively slow personal computers.

Table 1: PIONEER Solution Times on Test Set A

Source	s	m	DMUs	No RBE Time	RBE Time	Ratio
FR Bank	3	6	1000	33.29 min	17.04 min	0.526
FR Bank	3	6	2000	179.91 min	95.59 min	0.532
FR Bank	3	6	8000	44.21 hour	19.80 hour	0.448
Multinormal	4	6	1000	31.96 min	18.72 min	0.586
Multinormal	4	6	2000	106.90 min	61.65 min	0.577
DEA-GENa	4	6	1000	57.51 min	33.13 min	0.576
DEA-GENa	5	3	2000	130.02 min	75.94 min	0.584
DEA-GENb	7	4	1000	61.49 min	37.96 min	0.617
DEA-GENb	7	2	2000	151.27 min	92.42 min	0.611
DEA-GENc	6	6	1000	72.83 min	43.29 min	0.594
DEA-GENc	6	5	2000	213.61 min	126.50 min	0.592
Average						0.567

4.3 Experimental Results

The PIONEER code was applied to problems from each of the three sources.

- Federal Reserve banking data: the first 1,000, 2,000, and 8,000 DMUs of the 8,748-bank data set.
- Multinormal-data models: two problems with $n = 1,000$ and $n = 2,000$, generated using the variance-covariance matrix and DEA interpretation given in Appendix B.
- DEA-GEN: problem sets with $n = 1,000$ and $n = 2,000$, created using and the parameters given in Appendix C.

Solution times for these problems with the PIONEER code and the CCR^o model are given in Table 1. The times are "wall-clock" or elapsed real execution times, exclusive of problem input and output. Unsolved subproblems associated with members of E (as determined by Step 1.4) were bypassed.

The code was run with and without the use of restricted basis entry (RBE) and early identification of efficient units (EIE) to examine its impact on solution time. In all cases, the RBE procedure had a strong impact, cutting solution times roughly in half. The 17.04-minute time for the 1,000- and 2,000-DMU problems indicated that the PIONEER code is reasonably efficient for medium-sized problems. But, even with the help of RBE, the 19.8-hour solution time for the 8,000-DMU problem is excessive for practical usage.

A closer examination of the 8000-DMU bank problem (BANK-8) solution process gives insight into the sources of the speed improvements. Figure 1 gives the time to solve each set of 1,000 subproblems in the 8,000 total, both with and without RBE logic. Note that when RBE is not used, the time to solve each set of 1,000 subproblems is roughly the same (around 4.1 hours). But when RBE is employed, the last group of 1,000 subproblems is solved almost four times faster than the first group. By restricting the known inefficient

Figure 1: Solution Time for Subsets of BANK-8

DMUs from entering the basis, the later subproblems are smaller, and easier to solve. In addition, the early identification of efficient DMUs results in fewer subproblems to be solved. Typically, fewer than 20% of all DMUs are efficient, so the faster solution time can be attributed mainly to restricted basis entry.

Figure 2 shows the cumulative effect of RBE on solution time for the BANK-8 problem. While the performance improvement is less pronounced in the earlier subproblem groups, the 2:1 ratio becomes evident in the last few groups, and trends indicate an even greater disparity might result for larger problem sets.

Although PIONEER's solution times are encouraging and indicate that the code may be comparable to others described in the literature, other performance improvements are possible. The next section describes the use of parallel processing to further decrease solution times for these and other problems.

5 Parallel PIONEER Code

A computational advance that holds great promise for expediting the solution of difficult problems is application-level parallel processing. Parallel processing is the simultaneous manipulation of data by multiple computing elements working to complete a common body of work. With this, the power of many processing elements can be brought to bear on a single problem. If an algorithm's steps can be properly subdivided and assigned to separate processors for simultaneous execution, opportunities for dramatic reductions in solution times arise.

Of the numerous varieties of parallel machine architectures, the most prevalent commercial design is multiple-instruction, multiple-data (MIMD) [10, 26]. Each such com-

Figure 2: Cumulative Solution Time for Subsets of BANK-8

puting system contains multiple independently executing processors that can operate simultaneously on different data sets. Processors communicate either via a shared memory accessed through a central switch, or by messages passed through an interconnection network in a distributed system. Shared-memory multiprocessors are called tightly coupled if the time required to access a particular memory location is the same for all processors, as opposed to being proximity dependent or loosely coupled, as in distributed-memory systems.

As with traditional single-processor (serial) machines, solution efficiencies are directly tied to how well the algorithmic steps match the architecture of the underlying machine. Because a DEA problem involves the optimization of many separate linear-programming subproblems, the use of MIMD-style parallelism to speed solution appears, on the surface, to be a "natural" one. In fact, the mapping of the DEA solution process to a tightly coupled MIMD architecture turns out to be an ideal example of the use of parallel processing.

5.1 Parallel Code Design

The application of parallel processing to DEA problems was first reported in [31], where four transputers were run from a Macintosh IIcx on a 54-DMU problem. Times were reduced by a factor of three in this loosely coupled MIMD implementation. The next section describes a very different computing environment and how the PIONEER code was modified to use this form of parallelism.

As with software designed for vector processors, parallel codes must be structured to match the architecture of the target machine. Our test machine was the same Sequent Symmetry S81B that was employed for serial testing, but which can be programmed for parallel processing. The system has a tightly coupled MIMD design, with 20 16-MHz

80386 processors, Weitek coprocessors, and 32MB of sharable memory.

The Sequent's operating system permits the programmer to create multiple, independent processes which it schedules on the available processors. The processes can have both private and shared data-spaces. The shared spaces can be read and written by all designated processes, and can be used for interprocess communication and to avoid duplication of common data.

This type of parallel machine is designed to be most effective with work that can be decomposed into large, independent tasks. The DEA solution process can be organized in this manner through the use of domain decomposition, where multiple identical processes apply the same program steps to different data. By considering each DMU's LP subproblem to be a separate task, such large-granularity work decomposition is possible.

In the parallel PIONEER code, a self-scheduling approach is used, where processes select and execute tasks from a shared work list, on a first-come-first-served basis. Although incurring a minor amount of coordination overhead, such self-scheduling permits a balanced distribution of the workload across processes, an important characteristic when individual task times vary.

Each process solves its LP subproblems in its private memory; shared memory stores the original problem data and the status of each DMU. Since a DMU's status—in terms of membership in E, I, or U—may change, restricted basis entry becomes dynamic and time-based. At one moment, a given λ_j variable may be part of the $E \cup U$ pricing set and, an instant later, be found to be inefficient and ineligible for basis entry. The shared status array automatically communicates this information to all processes. This is an instance of a race condition, or timing-dependent code, which can result in stochastic solution statistics when the order of events differ from run to run due to minute timing differences..

5.2 Testing Parallel PIONEER

To test the parallel implementation of the PIONEER code, a variety of problems were chosen for analysis. Table 1 describes the characteristics of the problems chosen. The original bank data consisted of 8,742 DMUs. For each of the bank problems, the DMUs composing the data set were randomly chosen from the original data set. The multinormal and DEA-GEN data sets are described in Appendices B and C. These problems were generated to simulate real life data of large-scale DEA problems. The CCR^i model was used for all runs.

The "wall clock" time, measured in minutes, to solve each DEA problem, exclusive of input output times, are given in Table 2. For the problems consisting of 8,000 and 4,000 DMUs, the number of processors used was limited by the available memory on the Sequent computer.

The reported *relative speedup*, $S(p)$, is the ratio of the solution time of the PIONEER code using a single processor to its solution time using p processors. *Efficiency* is defined as $E(p) = S(p)/p$. The goal of any parallel implementation is to achieve a linear speedup where $S(p) = p$ and $E(p) = 1$.

Tables 3 and 4 give the speedups and efficiencies for the parallel testing. In all cases, relative speedup was nearly linear. Isolated examples indicate that superlinear speedup may be possible with the use of RBE/EIE in the PIONEER code. Since PIONEER is asynchronous, i.e., each LP can be solved independently of all other LPs, the paralleliza-

Table 2: Test Problems' Parallel Run Times (min.)

No.	Bank	Bank	Bank	Multinorm	DEA-GENa	DEA-GENb	DEA-GENc
Procs	1000	4000	8000	4000	2000	2000	2000
1	17.04	283.22	1189.81	242.96	75.94	92.42	126.50
2	8.70	151.23	589.22	122.35	38.20	46.52	63.63
3	5.68	101.42	389.46	81.81	25.58	31.06	42.48
4	4.27	73.26		61.39	19.19	23.39	31.91
5	3.40	61.27		49.14	15.35	18.71	25.57
6	2.86				12.82	15.60	21.32
7	2.45				11.02	13.38	18.28
8	2.15				9.65	11.74	16.02
9	1.96				8.57	10.45	14.26
10	1.72				7.72	9.40	12.80
15	1.17				5.19	6.33	8.61

tion is highly effective at improving solution times.

5.3 Limits of Parallelism

The computational testing indicated that the parallel PIONEER code was a highly scalable MIMD application that fully utilized the multiprocessing capability of the given computer system. We feel that the software would also exhibit much of the same efficiency if implemented in a loosely coupled MIMD environment. In the latter setting, changes in a DMU's status would have to be broadcast to all processors, thus incurring additional overhead and a latency in communicating the information. While unnecessary work (avoidable in a shared-memory environment) might result, the use of a larger number of processors could more than offset this disadvantage.

In our testing, memory size limited the dimensions of problems that could use the full parallelism of this system. We believe that additional internal storage would permit the same excellent speedups on the larger problems as was observed on the smaller ones. (This notion was verified by preliminary testing on a larger system.)

Even with these encouraging results, we felt that further improvements were needed and possible. In fact, a close examination of the parallel solution statistics led to a new procedure which further reduced all times—both serial and parallel—by up to an order of magnitude.

6 Hierarchical Decomposition

Experimentation with sets of problems that were identical except for the number of DMUs yielded solution times such as those given in Figure 3. Not only did the memory requirements of larger problems limit the amount of usable parallelism, but run times grew exponentially in n. Hence if a larger problem could be decomposed into a series of smaller ones, lower memory requirements, greater parallelism, and faster individual solution times

Table 3: Test Problems' Relative Speedups

No.	Bank	Bank	Bank	Multinorm	DEA-GENa	DEA-GENb	DEA-GENc
Procs	1000	4000	8000	4000	2000	2000	2000
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	1.96	1.87	2.02	1.99	1.99	1.99	1.99
3	3.00	2.79	3.06	2.97	2.97	2.98	2.98
4	3.99	3.87		3.96	3.96	3.95	3.96
5	5.01	4.62		4.94	4.95	4.94	4.95
6	5.96				5.92	5.92	5.93
7	6.96				6.89	6.91	6.92
8	7.93				7.87	7.87	7.90
9	8.69				8.86	8.84	8.87
10	9.91				9.84	9.83	9.88
15	14.56				14.63	14.60	14.69

Table 4: Test Problems' Efficiencies

No.	Bank	Bank	Bank	Multinorm	DEA-GENa	DEA-GENb	DEA-GENc
Procs	1000	4000	8000	4000	2000	2000	2000
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	0.979	0.936	0.979	0.993	0.994	0.993	0.994
3	1.000	0.931	1.000	0.990	0.990	0.992	0.993
4	0.998	0.966		0.989	0.989	0.988	0.991
5	1.002	0.924		0.989	0.989	0.988	0.989
6	0.993				0.987	0.987	0.989
7	0.994				0.984	0.987	0.989
8	0.991				0.984	0.984	0.987
9	0.966				0.985	0.983	0.986
10	0.991				0.984	0.983	0.988
15	0.971				0.975	0.973	0.979

Figure 3: Solution Time versus Problem Size

might offset any additional work that might be required.

6.1 DEA Decomposition Background

Consider a partitioning of the set of DMUs into a series of k mutually exclusive and collectively exhaustive subsets $D_i \subset D$, where $D = \bigcup_{i \in K} D_i, \bigcap_{i \in K} D_i = \emptyset$, and $K = \{1, \dots, k\}$. Define $E(D_i)$ and $I(D_i)$ to be the index sets of DMUs in D_i that are efficient and inefficient, respectively, relative to D_i (i.e., $D_i = E(D_i) \cup I(D_i)$), based on the meaning of efficiency in the DEA model of interest.

Theorem 3
If $D_i \subseteq D$, then $I(D_i) \subseteq I^*$ and $E^* \subseteq \bigcup_{i \in K} E(D_i)$.

Hence if a DMU is inefficient in a subproblem, it is inefficient for the full problem; if it is efficient for a subproblem, it may or may not be efficient overall. These relationships provide the foundation for the following decomposition procedure for DEA models.

6.2 The Hierarchical Decomposition Procedure

The following approach to DEA problems solves a series of subproblems which are arranged in a hierarchy. The subproblems are themselves DEA problems created from subsets of the DMUs in D. They can be solved independently, but information about set I membership generated from the subproblems can accelerate the solution of others.

First we define a procedure for creating and solving the subproblems. It partitions the set of DMUs whose efficiency is unknown (U) into a series of subsets, or *blocks*, of size b. Recall that $U = D - E - I$ and $|U|$ is the cardinality of U.

Figure 4: Hierarchical Decomposition Schematic

```
PROCEDURE SolveBlocks($b, \ell, I$)
Step 1   Partition $U$ into $k = \lceil |U|/b \rceil$ mutually exclusive, approximately equal-sized,
         blocks of DMUs.
Step 2   For each block $B_i, i \in K, K = \{1, \ldots, k\}$:
           1. Apply a DEA envelopment model to compute $E_i^\ell = E(B_i)$, using early
              identification of efficiency and restricted basis entry.
           2. Set $I = I \cup I(B_i)$.
```

This procedure is used in the following hierarchical decomposition algorithm (HDEA). A graphical representation of the process is given in Figure 4.

```
PROCEDURE HDEA($b, \beta, \gamma$)
Level 1  (Initial pass)
           1. $I = \emptyset, U = D, \ell \leftarrow 1$.
           2. SolveBlocks($b, \ell, I$).
Level 2  (Identify $E^*$ and $I^*$.)
         while ( $U \neq \emptyset$ ) do:
           1. $\ell \leftarrow \ell + 1$.
           2. $u \leftarrow |U|$.
           3. SolveBlocks($b, \ell, I$).
           4. if $|U|/u > \gamma$, then $b \leftarrow |U|$, else $b \leftarrow \beta b$.
```

Level 3 *(Resolution of I^*)*
Re-solve the DEA model (with basis entry restricted to E^*) for members of I^* to compute correct solution values.

User-supplied parameters β and γ control the blocksize b in Level 2. Specifically, b grows at each iteration by the β multiplier until it reaches $|U|$; b is set equal to $|U|$ when the fraction of U retained in an iteration exceeds γ.

6.3 Implementation Considerations

HDEA is a divide-and-conquer algorithm similar in nature to merge sorting and sorting networks [25, 27]: the original problem is broken into several smaller subproblems, which are solved recursively and the results combined to solve the original problem. The blocksize parameter, b, defines sets of linear programs with the same number of constraints as the original problem, but many fewer variables. This results in lower memory requirements and faster solutions for the subproblems, although more linear programs may be involved. Because of this increased speed and Theorem 3, these easier problems can eliminate inefficient-DMU variables earlier than with non-hierarchical approaches.

The HDEA procedure focuses initially on isolation of E (in levels 1 and 2), so as to expedite solution of the subproblems associated with I (in level 3). The method should be particularly effective when $|E| \ll |I|$, as is typically the case. The decomposition into subproblems with minimal data communication requirements is highly attractive from a parallel processing standpoint.

Memory requirements are a function of $\max\{b, |E|\}$. If primary storage is at a premium, its use can be minimized by paging in DMU data from external storage for each subproblem separately. In parallel implementations, participating processes might place in shared memory a copy of the $\mathbf{X}$ and $\mathbf{Y}$ data, plus a DMU-status array; each process would also need its own private storage for solver-specific information such as a simplex basis inverse and candidate list.

The choice of blocksize also affects overall solution time. Since b influences the tradeoff between the size and the number of subproblems solved, computational testing for an appropriate setting is required. Figure 5 shows the total solution time for various values of b on an example problem.

6.4 Computational Testing

For testing the HDEA procedure, real life data, randomly generated data, and data based on the economic Cobb-Douglas functional form were employed. For each test problem, an array containing the status of each DMU was maintained in shared memory. A private copy of the appropriate data was given to each processor in parallel so that each DMU could be solved independently. Since the status of each DMU's problem could be updated without conflict from other processors, no locks were needed during the status update and as a result, the parallel implementation is purely asynchronous.

All times reported are wall clock times, in minutes, to solve the problems exclusive of any input/output. The single processor times represent the best achieved serial times across the differing-sized DEA problems. The parallel case is a direct implementation of the serial formulation, requiring only a few modifications for the parallel environment to accomplish the self-scheduling parallel implementation. It is important to note that none

Figure 5: Total Solution Time vs. Blocksize for 8,000-DMU Problem

of the runs were made in a dedicated environment, and hence, times are subject to system load. However, precautions were taken to conduct the runs during non-peak hours so as to minimize the confounding of outside factors on the solution times.

For the test problems shown, a blocksize of 250 was used for the 8,000 and 4,000 DMU cases, 125 for the 2,000 DMU cases, and 100 for the 1,000 DMU cases. The effects of different blocksizes must be investigated, hence, these results should not be viewed as the best possible. During level 2 of the HDEA procedure, $\beta = 1.5$. As a result, each subsequent block at level 2 grows by 50% until all DMUs have been classified as either efficient or inefficient.

Table 5 shows that for all test runs the speedup was nearly linear. Larger more difficult problems resulted in better speedup than smaller problems. With the smaller problems, the overall solution time in parallel is so small that the overhead of maintaining the self-scheduling tasks, as well as possible workload imbalance between processors, becomes apparent. Consequently, although the parallel results for small problems are quite good, the parallel implementation will be more efficient for large, difficult problems.

Table 5 also gives the total number of pivots and pricing operations executed to find the DEA scores for each data set. These numbers vary with the number of processors because of the method used to update the status of each DMU. As expected, problems with a greater number of DMUs, and those with a larger density of efficient DMUs, require more pivots and pricing operations and consequently take longer to solve.

Tables 6 thru 10 contain the results of each test problem for each level of the HDEA procedure. The speedup remains relatively consistent across all problems across all levels and is nearly linear.

The effects of using early identification of efficient DMUs to reduce the number of linear programming problems that must be solved are shown in Tables 11 through 15.

Table 5: Hierarchical Test Problem Results

Table 6: Bank Hierarchical Level Test Problem Results

Table 7: Multinormal Hierarchical Level Test Problem Results

Table 8: DEA-GENa Hierarchical Level Test Problem Results

Table 9: DEA-GENb Hierarchical Level Test Problem Results

Table 10: DEA-GENc Hierarchical Level Test Problem Results

With the HDEA procedure, the total number of linear programs that must be solved is slightly more than twice the original number of DMUs contained in each data set. However, since the HDEA linear programs are smaller than other DEA methods, overall performance is improved. EIE improves performance further by reducing the number of linear programs at level 1 and level 2 by 10-25%. Early identification has no effect at level 3 since only DEA scores for inefficient DMUs are found at that level.

Table 16 shows the dramatic effect of the hierarchial procedure on reducing solution time for the 8,000 DMU cases over the best DEA approach reported to date. Only three processors are used in the parallel implementation because of memory limitations of the non-hierarchial procedures. The HDEA procedure results in a 6- to 12-fold increase in speed over the non-hierarchial procedure when both cases are run with the same number of processors. Because the HDEA procedure requires less memory, permitting use of more processors, the cases with 15 processors could be run and are 75 to 125 times faster than the serial non-hierarchial procedure.

Although the test problems were limited to 8,000 DMUs because of the availability of real life data, the HDEA procedure can accommodate much larger problems. A Cobb-Douglas test case (Appendix C) consisting of 25,000 DMUs with 3 inputs and 3 outputs was solved using 15 processors in 19.26 minutes. Again, memory size precluded solution of this problem with the non-hierarchial procedure.

As seen with the test problems, the HDEA procedure allows for the solution of large-scale DEA problems much faster than previously reported DEA methods. When coupled with a parallel environment, the HDEA procedure yields solutions to problems in a matter of minutes which previously, would have taken days. Additionally, the HDEA procedure allows for the solution of very large-scale DEA problems that remain unsolvable (in a practical sense) with any other reported approaches.

7 Further Computational Advantages of the HDEA Approach

Accompanying the HDEA method's advances in sheer solution speed are a variety of additional computational advantages in other settings. The hierarchial structure can be exploited when solving multiple DEA models and further streamlining opportunities await exploration.

7.1 Solving Multiple Models

There are numerous instances in which multiple models must be solved for each DMU under consideration. The HDEA approach can be easily modified and extended to efficiently determine all of the required values.

- To solve for technical, scale, and overall efficiency (inefficiency), both the BCC and CCR models must be solved [33]. With HDEA, We can use the BCC or ADD model through level 2 to find E^*_{bcc}. On only the E^*_{bcc} set we can apply the CCR model formulation to find the E^*_{ccr} ($\subseteq E^*_{bcc}$) set of DMUs. During level 3, while determining the inefficient scores, we can solve the values for I^*_{ccr}. Then, as an advanced starting basis, we can allow all E^*_{bcc} DMUs to enter the basis and solve for I^*_{bcc}. (Alternately, we could solve for the I_{bcc} values first, then relax the BCC

Table 11: Bank Data Results: Linear Programs Required

Table 12: Multinormal Data Results: Linear Programs Required

Table 13: DEA-GENa Results: Linear Programs Required

Table 14: DEA-GENb Results: Linear Programs Required

Table 15: DEA-GENc Results: Linear Programs Required

Table 16: Non-Hierarchical and Hierarchical Comparisons: 8,000-DMU Problems

$\mathbf{1}\boldsymbol{\lambda} = 1$ constraint, tighten the basis entry rules, and solve for the I_{ccr} scores). Once the BCC and CCR values are found, we can rapidly solve for the three efficiency values desired. Note that this prevents a duplication of effort to find the inefficient values.

- Byrnes, Färe, and Grosskopf [16] argue that a "no increasing returns to scale" (NIRS) model must be solved to determine if the DMU (or the virtual DMU if inefficient) is in a region of increasing or decreasing returns to scale. Knowing this may help indicate the direction the DMU should go to become efficient, either expand operations through output augmentation, or downsize through input contraction. We know that $E^*_{ccr} \subseteq E^*_{nirs} \subseteq E^*_{bcc}$. Consequently, the NIRS strategy could be added to the strategy above to find these scores.

- Besides solving for BCC, CCR, NIRS models, we can also apply assurance region restrictions [33] once the E^* for each model is found at level 2. The various assurance region approaches can be solved at level 2 then extended to level 3. In this way, a series of models can be solved rapidly without duplication of effort. We know that the set of efficient DMUs under assurance-region constraints is a subset of E^* for any of the above models. This same approach can be used for the cone-ratio model of [19, 20].

- For the CCR and BCC models, there is an associated orientation, either input or output. The inefficiency scores will vary depending on the orientation, but the E^*_{bcc} and E^*_{ccr} sets will remain the same regardless of the orientation. Consequently, if both orientations are needed, then only the level 3 values must be resolved to obtain both.

- Some versions of the model may provide faster solutions than others depending on the number of output and input variables that are used. For example, if the number of outputs exceeds the number of inputs, the output-oriented model solves faster. Also, the CCR^o model has an initial feasible solution, thus bypassing Phase I of the simplex method. Since the output model may solve faster, this can be used through level 2, then the input model applied in level 3 calculations if the input values are needed. HDEA provides the flexibility to incorporate the fastest model to identify E^* through level 2, then at level 3 can use the model of choice to determine the inefficiency scores.

7.2 Implementation Issues

- As noted above, any method can be used to find the set of DMUs belonging to E^* and I^* such as sorting, preprocessing, domination theory, etc. These can possibly enhance the HDEA procedure to expedite the level 2 and level 3 efficiency score values.

- Computational concerns have arisen over the two-phase or multi-objective approaches to the non-Archimedean infinitesimal issue, which can be decreased via HDEA. We know that the efficiency scores do not vary with the sum-of-slack solutions. So in HDEA, there is no need to solve for the sum of slacks during level 1 or most of level 2. Once potential E^* DMUs are identified, the sum of slacks must be solved only for these DMUs to determine if they are weakly efficient. If there are any weakly efficient DMUs, they are removed from E^*. During level 3, there is no need to solve for the sum of slacks for the inefficient DMUs since a proper projection to the efficient surface is obtained since no weakly efficient DMUs can enter the basis. Solving for the sum of slacks for the inefficient DMUs may simply identify possible alternate optimal solutions that offer no new information when weakly efficient DMUs are not in the basis.

- As the number of input/output variables grows, the likelihood of cycling also increases because of severe degeneracy involved with the solution of some DMUs (especially efficient DMUs). Anti-cycling procedures invoke a high computational cost. Also, the possibility of cycling can increase with a larger number of DMUs (variables) in the model. HDEA can help reduce the cost of cycling by maintaining smaller problems. Additionally, at level 1 and most of level 2 there is no need to invoke the anti-cycling rules. Any DMUs that exhibit cycling can be deferred. Only when all potential members of E^* are identified at level 2, must the anti-cycling rules be invoked. By passing over the DMUs that cycle, inefficient DMUs may later reveal the problematic DMUs as efficient. In this manner, the DMUs will not need to be solved and the cost of applying anti-cycling rules is reduced.

7.3 Exploiting the Levels

- As we progress through level 1 to level 2 to level 3, information at each level can be used to enhance the solutions at the next level. DMUs that frequently show up in efficient reference sets can be given priority during the pricing procedure to enter the basis. By choosing the most attractive variables to enter the basis, the number of pivots to solve the LPs may be reduced. This may also reduce the potential for cycling. This also indicates the possibility of using different heuristics for the entering and leaving variables for the different levels of the HDEA process.
- One advantage of the HDEA process over domination and other preprocessing techniques is that it can be used to find DEA scores for categorical variables and window analysis enroute to determining overall DEA scores. By blocking appropriately, the efficient DMUs for each category or window can be recorded along the way. In this way, solving overall scores as well as those within particular categories or across various categories can be accomplished without duplication of effort.

7.4 Parallel Implementation

- The HDEA procedure can expand the parallel approach across many platforms. For example, a SIMD distributed network can be used to solve each block of DMUs. Since the blocks are smaller, memory requirements are smaller. The advantage of updated restricted basis entry and early identification of DMU status within blocks will be lost, but the basic process will still lead to a solution. The basis entry and identification schemes will hold between levels.
- For the parallel case, Amdahl's law assumes that the same information is available for the single processor as for multiple processors. But with RBE and early

identification, the interaction between processors can enhance the solution process resulting in high levels of efficiency when multiple processors are used. This is not an HDEA-exclusive advantage, but it does hold for the HDEA process.

8 Conclusions

We have described a new hierarchial decomposition procedure for solving DEA problems that advances the state of the art for computational data envelopment analysis. As demonstrated with medium- and large-scale test sets, this approach can have dramatic benefits over traditional methods—in both single-processor and parallel settings—and permits enormous problems to be optimized in a modest amount of time. The ability to routinely solve problems with thousands of DMUs permits researchers and practitioners to be more ambitious in their application of this important class of models and, we hope, will encourage new and even more exciting applications of DEA.

A DEA Models

A variety of models have been proposed for data envelopment analysis, each with different assumptions and interpretations. This appendix provides a mathematical and notational summary of the most commonly used formulations: CCR [22], BCC [6], additive [18], non-decreasing returns-to-scale (NDRS) and non-increasing returns-to-scale (NIRS) [32]. Each basic model is shown in both envelopment and multiplier forms (which are duals of each other), and for all traditional orientations. Model taxonomy follows the form M_j^r, where M is the base model's abbreviation, r is the orientation indicator ($r = o$ for output-oriented, $r = i$ for input-oriented, and missing for non-oriented), and j is the index number of the DMU being evaluated by the model.

1.1 Envelopment Models

$$\begin{array}{rcl}
 & & \text{Input} - \text{oriented} \\
CCR_j^i & : & \min\limits_{\lambda,\mathbf{s}^i,\mathbf{s}^o\geq\mathbf{0},\theta_j} \{\theta_j - \varepsilon(\mathbf{1s}^i + \mathbf{1s}^o) | \mathbf{Y}\boldsymbol{\lambda} - \mathbf{s}^o = Y_j, \theta_j X_j - \mathbf{X}\boldsymbol{\lambda} - \mathbf{s}^i = \mathbf{0}\} \\
BCC_j^i & : & \min\limits_{\lambda,\mathbf{s}^i,\mathbf{s}^o\geq\mathbf{0},\theta_j} \{\theta_j - \varepsilon(\mathbf{1s}^i + \mathbf{1s}^o) | \mathbf{Y}\boldsymbol{\lambda} - \mathbf{s}^o = Y_j, \theta_j X_j - \mathbf{X}\boldsymbol{\lambda} - \mathbf{s}^i = \mathbf{0}, \mathbf{1}\boldsymbol{\lambda} = 1\} \\
NDRS_j^i & : & \min\limits_{\lambda,\mathbf{s}^i,\mathbf{s}^o\geq\mathbf{0},\theta_j} \{\theta_j - \varepsilon(\mathbf{1s}^i + \mathbf{1s}^o) | \mathbf{Y}\boldsymbol{\lambda} - \mathbf{s}^o = Y_j, \theta_j X_j - \mathbf{X}\boldsymbol{\lambda} - \mathbf{s}^i = \mathbf{0}, \mathbf{1}\boldsymbol{\lambda} \geq 1\} \\
NIRS_j^i & : & \min\limits_{\lambda,\mathbf{s}^i,\mathbf{s}^o\geq\mathbf{0},\theta_j} \{\theta_j - \varepsilon(\mathbf{1s}^i + \mathbf{1s}^o) | \mathbf{Y}\boldsymbol{\lambda} - \mathbf{s}^o = Y_j, \theta_j X_j - \mathbf{X}\boldsymbol{\lambda} - \mathbf{s}^i = \mathbf{0}, \mathbf{1}\boldsymbol{\lambda} \leq 1\} \\
 & & \text{Output} - \text{oriented} \\
CCR_j^o & : & \max\limits_{\lambda,\mathbf{s}^i,\mathbf{s}^o\geq\mathbf{0},\phi_j} \{\phi_j + \varepsilon(\mathbf{1s}^i + \mathbf{1s}^o) | \mathbf{X}\boldsymbol{\lambda} + \mathbf{s}^i = X_j, \phi_j Y_j - \mathbf{Y}\boldsymbol{\lambda} + \mathbf{s}^o = \mathbf{0}\} \\
BCC_j^o & : & \max\limits_{\lambda,\mathbf{s}^i,\mathbf{s}^o\geq\mathbf{0},\phi_j} \{\phi_j + \varepsilon(\mathbf{1s}^i + \mathbf{1s}^o) | \mathbf{X}\boldsymbol{\lambda} + \mathbf{s}^i = X_j, \phi_j Y_j - \mathbf{Y}\boldsymbol{\lambda} + \mathbf{s}^o = \mathbf{0}, \mathbf{1}\boldsymbol{\lambda} = 1\}
\end{array}$$

$$
\begin{array}{rcl}
NDRS^o_j & : & \max\limits_{\lambda,\mathbf{s}^i,\mathbf{s}^o\geq\mathbf{0},\phi_j} \{\phi_j + \varepsilon(\mathbf{1s}^i + \mathbf{1s}^o)|\mathbf{X}\boldsymbol{\lambda} + \mathbf{s}^i = X_j, \phi_j Y_j - \mathbf{Y}\boldsymbol{\lambda} + \mathbf{s}^o = \mathbf{0}, \mathbf{1}\boldsymbol{\lambda} \geq 1\} \\
NIRS^o_j & : & \max\limits_{\lambda,\mathbf{s}^i,\mathbf{s}^o\geq\mathbf{0},\phi_j} \{\phi_j + \varepsilon(\mathbf{1s}^i + \mathbf{1s}^o)|\mathbf{X}\boldsymbol{\lambda} + \mathbf{s}^i = X_j, \phi_j Y_j - \mathbf{Y}\boldsymbol{\lambda} + \mathbf{s}^o = \mathbf{0}, \mathbf{1}\boldsymbol{\lambda} \leq 1\} \\
 & & \text{Non} - \text{oriented} \\
ADD_j & : & \max\limits_{\lambda,\mathbf{s}^i,\mathbf{s}^o\geq\mathbf{0}} \{\mathbf{1s}^i + \mathbf{1s}^o|\mathbf{Y}\boldsymbol{\lambda} - \mathbf{s}^o = Y_j, \mathbf{X}\boldsymbol{\lambda} + \mathbf{s}^i = X_j, \mathbf{1}\boldsymbol{\lambda} = 1\}
\end{array}
$$

1.2 Multiplier Models

$$
\begin{array}{rcl}
 & & \text{Input} - \text{oriented} \\
\overline{CCR}^i_j & : & \max\limits_{\mu,\nu}\{z = \boldsymbol{\mu}Y_j|\boldsymbol{\mu}\mathbf{Y} - \boldsymbol{\nu}\mathbf{X} \leq \mathbf{0}; \boldsymbol{\nu}X_j = 1; \boldsymbol{\mu},\boldsymbol{\nu} \geq \mathbf{1}\varepsilon\} \\
\overline{BCC}^i_j & : & \max\limits_{\mu,\nu,u}\{z = \boldsymbol{\mu}Y_j + u|\boldsymbol{\mu}\mathbf{Y} - \boldsymbol{\nu}\mathbf{X} + \mathbf{1}u \leq \mathbf{0}; \boldsymbol{\nu}X_j = 1; \boldsymbol{\mu},\boldsymbol{\nu} \geq \mathbf{1}\varepsilon; u\ free\} \\
\overline{NDRS}^i_j & : & \max\limits_{\mu,\nu,u}\{z = \boldsymbol{\mu}Y_j + u|\boldsymbol{\mu}\mathbf{Y} - \boldsymbol{\nu}\mathbf{X} + \mathbf{1}u \leq \mathbf{0}; \boldsymbol{\nu}X_j = 1; \boldsymbol{\mu},\boldsymbol{\nu} \geq \mathbf{1}\varepsilon; u \geq 0\} \\
\overline{NIRS}^i_j & : & \max\limits_{\mu,\nu,u}\{z = \boldsymbol{\mu}Y_j + u|\boldsymbol{\mu}\mathbf{Y} - \boldsymbol{\nu}\mathbf{X} + \mathbf{1}u \leq \mathbf{0}; \boldsymbol{\nu}X_j = 1; \boldsymbol{\mu},\boldsymbol{\nu} \geq \mathbf{1}\varepsilon; u \leq 0\} \\
 & & \text{Output} - \text{oriented} \\
\overline{CCR}^o_j & : & \min\limits_{\mu,\nu}\{z = \boldsymbol{\nu}X_j|\boldsymbol{\mu}\mathbf{Y} - \boldsymbol{\nu}\mathbf{X} \leq \mathbf{0}; \boldsymbol{\mu}Y_j = 1; \boldsymbol{\mu},\boldsymbol{\nu} \geq \mathbf{1}\varepsilon\} \\
\overline{BCC}^o_j & : & \min\limits_{\mu,\nu,v}\{z = \boldsymbol{\nu}X_j + v|\boldsymbol{\mu}\mathbf{Y} - \boldsymbol{\nu}\mathbf{X} + \mathbf{1}v \leq \mathbf{0}; \boldsymbol{\mu}Y_j = 1; \boldsymbol{\mu},\boldsymbol{\nu} \geq \mathbf{1}\varepsilon; v\ free\} \\
\overline{NDRS}^o_j & : & \min\limits_{\mu,\nu,v}\{z = \boldsymbol{\nu}X_j + v|\boldsymbol{\mu}\mathbf{Y} - \boldsymbol{\nu}\mathbf{X} + \mathbf{1}v \leq \mathbf{0}; \boldsymbol{\mu}Y_j = 1; \boldsymbol{\mu},\boldsymbol{\nu} \geq \mathbf{1}\varepsilon; v \leq 0\} \\
\overline{NIRS}^o_j & : & \min\limits_{\mu,\nu,v}\{z = \boldsymbol{\nu}X_j + v|\boldsymbol{\mu}\mathbf{Y} - \boldsymbol{\nu}\mathbf{X} + \mathbf{1}v \leq \mathbf{0}; \boldsymbol{\mu}Y_j = 1; \boldsymbol{\mu},\boldsymbol{\nu} \geq \mathbf{1}\varepsilon; v \geq 0\} \\
 & & \text{Non} - \text{oriented} \\
\overline{ADD}_j & : & \max\limits_{\mu,\nu,\omega}\{\boldsymbol{\mu}Y_j - \boldsymbol{\nu}X_j + \omega|\boldsymbol{\mu}\mathbf{Y} - \boldsymbol{\nu}\mathbf{X} + \mathbf{1}\omega \leq \mathbf{0}, \boldsymbol{\mu} \geq \mathbf{1}, \boldsymbol{\nu} \geq \mathbf{1}; \omega\ free\}
\end{array}
$$

B Multinormal Population Parameters

Since few large-scale DEA problems are currently available, randomly generated data sets were needed to simulate realistic applications. One procedure for generating such data is to draw the samples from a multinormal distribution. To this end, the DRNMVN routine from the IMSL library was used to generate observations for 10,000 DMUs based on the variance-covariance matrix of Table 17. The code used to create this data is described in [11].

From the randomly generated data, the variables representing inputs and outputs needed to be carefully chosen. To coincide with sound economic theory, all outputs should be positively correlated with all the inputs. By eliminating variable x_4, variables x_1, x_3, and x_7 could represent outputs since they would be positively correlated with all the other remaining variables that would represent inputs. In this way, each DMU would be comprised of 3 outputs and 4 inputs. Positively correlated inputs would be compliments in the production process; those negatively correlated are substitutes. Since the input variables cover a wide range of cases of compliment and substitute relationships, the randomly generated values further simulate realistic data.

Table 17: Variance-Covariance Matrix 1 for Multinormal Data

X_1	303.4							
X_2	2.5	2.5						
X_3	12493.9	6093.9	112750625.0					
X_4	4062.6	484.2	-2890405.7	31033992.2				
X_5	265.2	45.3	-1346076.8	-455158.7	7629765.5			
X_6	19597.2	-195.0	3436081.9	-270405.4	-674642.7	86492323.0		
X_7	36418.8	6428.4	111950224.4	27415022.3	5153887.3	88983356.8	233502490.7	
X_8	3647.0	1001.0	10815783.1	112293.8	-64824.7	-319883.0	10543369.3	7812141.0

C DEA-GEN, Cobb-Douglas Problem Generator

While random data generators can provide large problems, a more systematic approach is needed to represent realistic scenarios. To this end, economic theory was used to generate large-scale DEA problem sets.

In production economics, the most widely used functional form is known as the Cobb-Douglas production function [24]. This function is written as:

$$y_j = a_o \prod_{i=1}^{m} x_{ij}^{\alpha_i}, \; x_{ij} > 0, \; j = 1, \ldots, n$$

Here, y_j is the single aggregate output produced by DMU$_j$, the x_{ij}s are values of the input variables used by DMU$_j$ in the production process, α_i is the factor elasticity for input i, and a_o is a constant scale factor. If $\sum_{i=1}^{m} \alpha_i = 1$ then only constant returns to scale exist in the production process. For $\sum_{i=1}^{m} \alpha_i < 1$ decreasing returns to scale are present, while $\sum_{i=1}^{m} \alpha_i > 1$ indicates increasing returns to scale. In the DEA studies, increasing returns to scale are not used because the function results in only a few DEA efficient points. Although this does not pose a problem, it does not realistically represent realistic data.

For the single output model, this production function has many desirable properties. If the inputs are randomly generated, the function generates output values that will always lie on the production possibility frontier (i.e., they will be DEA efficient for $\sum_{i=1}^{m} \alpha_i \leq 1$). This frontier is central to the theory of economic growth and measures the rate of technological progress. The Cobb-Douglas frontier represents the best use of technology as well as the best management practices to achieve efficient production. This coincides with the practical use of DEA. Unlike DEA, the quest in economic studies is to attempt to estimate the values of α_i by fitting the function to the observed data. By choosing the α_i values a priori, then randomly generating the input values, the output values can be determined so that they coincide with widely accepted economic theory. To insure that not all generated data sets fall on the efficient frontier surface, the a_o scale factor can be randomly generated; the efficient DMUs will consist of those for which a_o takes on it maximum value. Control over the number of DMUs that are efficient in the data set can be maintained by limiting the number of DMUs generated with $\max a_o$.

Table 18: Parameters for DEA-GEN Problems

Type	#DMUs	α Values	Mean Values	Standard Dev.
a	1,000	.3,.2,.2,.3	.12,.13,.13,.14,.11	.03,.04,.02,.01,.01
a	2,000	.3,.2,.2,.2,.1	.20,.30,.15	.03,.04,.02
a	4,000	.3,.3,.2,.2	.12,.12,.15,.14	.03,.04,.02,.01
a	8,000	.4,.2,.1,.1,.1	.5	.1
b	1,000	.03,.07,.12,.03,.12,.1,.02	.2,.3,.15	.01,.05,.01
b	2,000	.1,.1,.1,.1,.1,.1,.1	.52,.13	.03,.04
b	4,000	.4,.3,.1	.4,.3,.15	.13,.08,.02
b	8,000	.3,.17,.12,.03,.12,.2	.2,.3,.15,.14	.01,.05,.01,.01
c	1,000	.18,.2,.14,.1,.1,.1	.2,.12,.14,.11,.11	.03,.04,.02,.01,.01
c	2,000	.13,.18,.2,.14,.1,.1	.2,.18,.15,.19	.03,.04,.02,.01
c	4,000	.13,.2,.12,.13	.2,.3,.15,.14,.11	.03,.04,.02,.01,.01
c	8,000	.08,.12,.21,.07,.11	.3,.3,.3,.1	.04,.04,.1,.03
d	25,000	.3,.3,.3	.3,.3	.01,.01

In DEA analysis, the single-output, multiple-input scenario is not of primary interest. DEA was developed to analyze the case of multiple-output, multiple-input studies. Färe and Grosskopf [51] point out that the existence of a joint production function has not been established. That is, the multiple-output, multiple-input model does not produce values strictly on the efficient production frontier. To do so, would require strict assumptions that would not provide the desired realism for the DEA study. However, a joint model without the strict assumptions would simulate economically sound production processes. Consequently, a joint model was developed for the DEA-GEN problem generator.

For each of the problem sets, the input values were generated from a uniform distribution. The α_i values were chosen to simulate constant returns to scale processes as well as a variety of cases representing different levels of decreasing returns to scale. The a_o value was randomly generated, but with a modest control of the number of efficient DMUs that could be present in the problem data.

Once the single aggregate output level was calculated, the individual output levels were determined by assigning each individual output as a percentage of the aggregate. The percentages for each individual output were drawn from normal distributions with predetermined means and standard deviations. The means of the normal distributions were chosen so that the percentages sum to one. Table 18 lists the α_i values as well as the means and standard deviations that were used to generate twelve different Cobb-Douglas data sets.

Because of the economic foundations of the DEA-GEN code, the data generated resembles a class of problems that should more closely simulate realistic economic data than what would be possible from the data sampled from a multinormal distribution.

References

[1] A. I. Ali, IDEAS: integrated data envelopment analysis system, technical report, Department of General Business and Finance, University of Massachusetts, Amherst, MA, 1989.

[2] A. I. Ali, Data envelopment analysis: computational issues, *Comput. Environ. and Urban Systems* 14 (1990) 157–165.

[3] A. I. Ali, Streamlined computation for data envelopment analysis, *European Journal of Operational Research* 64 (1993) 61–67.

[4] A. I. Ali, Computational aspects of data envelopment analysis, in A. Charnes, W. W. Cooper, A.Y. Lewin, and L.M. Seiford, *DEA: Theory, Methodology, and Applications*, Kluwer Academic Publishers, Boston, 1994, pp. 63–88.

[5] A. I. Ali, L. Seiford, Computational accuracy and infinitesimals in data envelopment analysis, *INFOR* 31 (1989) 290–297.

[6] R. D. Banker, A. Charnes, W. W. Cooper, Some models for estimating technical and scale inefficiencies in data envelopment analysis, *Management Science* 30 (1984) 1078–1092.

[7] R. S. Barr, M. L. Durchholz, Parallel software for large-scale data envelopment analysis, presented at the Joint National Meeting of ORSA and TIMS, San Francisco, CA, 1992.

[8] R. S. Barr, M. L. Durchholz, Parallel and hierarchical decomposition approaches for solving large-scale DEA models, presented at the Joint National Meeting of ORSA and TIMS, Chicago, IL, 1993.

[9] R. S. Barr, M. L. Durchholz, Parallel and hierarchical decomposition approaches for solving large-scale DEA models, technical report 94-CSE-6, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, 1994

[10] R. S. Barr, B. L. Hickman, Reporting computational experiments with parallel algorithms: issues, measures, and experts' opinions, *ORSA Journal on Computing* 5 (1993) 2–18.

[11] R. S. Barr, B. L. Hickman, J. S. Turner, Statistical file merging: a new, constrained network model and parallel solution approach, technical report, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, 1997

[12] R. S. Barr, L. Seiford, T. F. Siems, An envelopment-analysis approach to measuring the managerial quality of banks, *Annals of Operations Research* 42 (1993) 1–19.

[13] R. S. Barr, T. F. Siems, Predicting bank failure using DEA to quantify management quality, in R. Barr, R. Helgason, J. Kennington eds., *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Techniques*, Kluwer Academic Publishers, Boston, 1997, pp. 341–365.

[14] R. E. Bixby, Implementing the simplex method: the initial basis, *ORSA Journal on Computing* 4 (1992) 267–284.

[15] R. G. Bland, New finite pivoting rules for the simplex method, *Mathematics of Operations Research* 2 (1977) 103–107.

[16] P. Byrnes, R. Färe, S. Grosskopf, Measuring productive efficiency: an application to illinois strip mines, *Management Science* 30 (1984) 671–681.

[17] A. Charnes, W. W. Cooper, *Management Models and Industrial Applications of Linear Programming*, John Wiley, New York, 1961.

[18] A. Charnes, W. W. Cooper, B. Golany, L. Seiford, J. Stutz, Foundations of data envelopment analysis for pareto-Koopmans efficient empirical production functions, *Journal of Econometrics* 30 (1985) 91–107.

[19] A. Charnes, W. W. Cooper, Z.M. Huang, Polyhedral Cone-Ratio DEA Models with an Illustrative Application to Large Commercial Banks, *Journal of Econometrics* 46 (1990) 73–91.

[20] A. Charnes, W. W. Cooper, Z.M. Huang, D.B. Sun, Relations between half-space and finitely generated cones in polyhedral cone-ratio DEA models, *International Journal of Systems Science* 22 (1991) 2057–2077.

[21] A. Charnes, W. W. Cooper, A. Y. Lewin, L. M. Seiford, *Data Envelopment Analysis: Theory, Methodology, and Application*, Kluwer Academic Publishers, Boston, 1994.

[22] A. Charnes, W. W. Cooper, E. Rhodes, 1978. Measuring the efficiency of decision making units, *European Journal of Operational Research* 2 (1978) 429–444.

[23] A. Charnes, J. Rousseau, J. Semple, An effective non-Archimedean anti-degeneracy/cycling linear programming method especially for data envelopment analysis and like models *Annals of Operations Research* 47 (1993), 271–278.

[24] C. W. Cobb, P. H. Douglas, A theory of production, *American Economic Review* March (Suppl.) (1928) 139–165.

[25] T. H. Cormen, C. E. Leiverson, R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[26] M. J. Flynn, Very high-speed computing systems, *Proceedings of the IEEE* 54 (1966) 1901–1909.

[27] D. E. Knuth, *The Art of Computer Programming Volume 3 / Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[28] R. Marsten, The design of the XMP linear programming library, *ACM Transactions on Mathematical Software* 7 (1981) 481–497.

[29] J. M. Mulvey, Pivot strategies for primal-simplex network codes, *Journal of the ACM* 25 (1978) 206–270.

[30] W. Orchard-Hays, *Advanced Linear Programming Computing Techniques*, McGraw-Hill, NY, 1968.

[31] F. Phillips, R. G. Parsons, A. Donoho, Parallel microcomputing for data envelopment analysis, *Comput., Environ., and Urban Systems* 14 (1990) 167–170.

[32] L. M. Seiford, R. M. Thrall, Recent development in DEA: the mathematical programming approach to frontier analysis, *Journal of Econometrics* 46 (1990) 7–38.

[33] R. G. Thompson, L. N. Langemeier, C.-T. Lee, E. Lee, R. M. Thrall, The role of multiplier bounds in efficiency analysis with application to Kansas farming, *Journal of Econometrics* 46 (1990) 93–108.

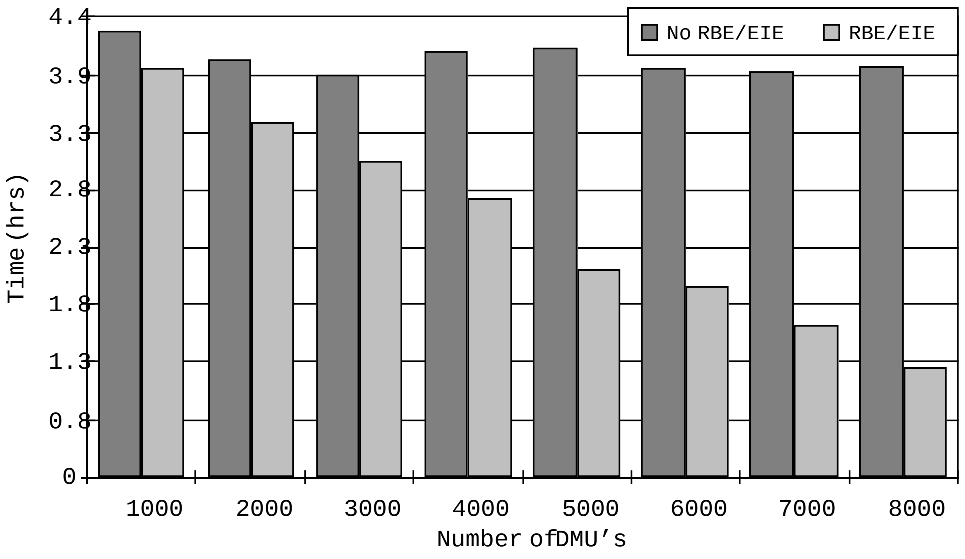

Figure 1: Solution Time for Subsets of BANK-8

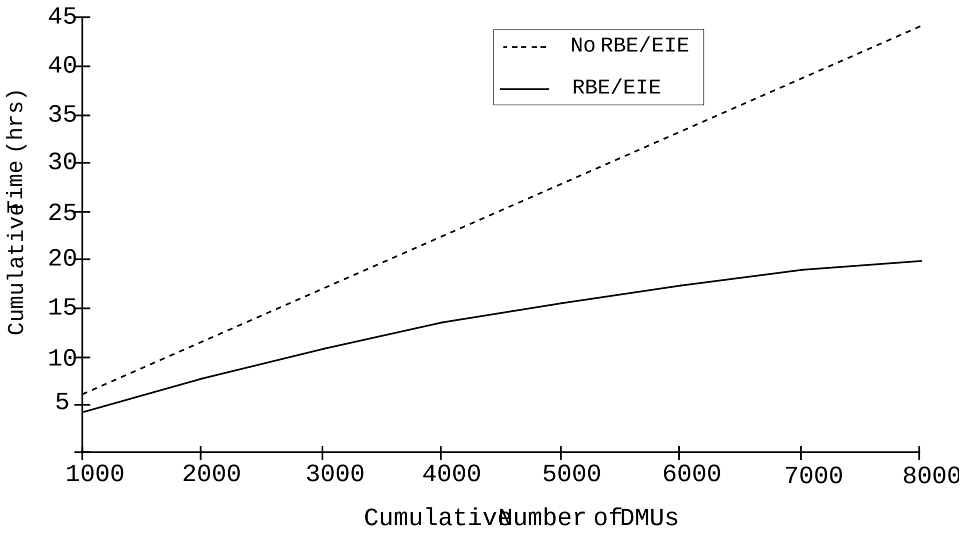

Figure 2: Cumulative Solution Time for Subsets of BANK-8

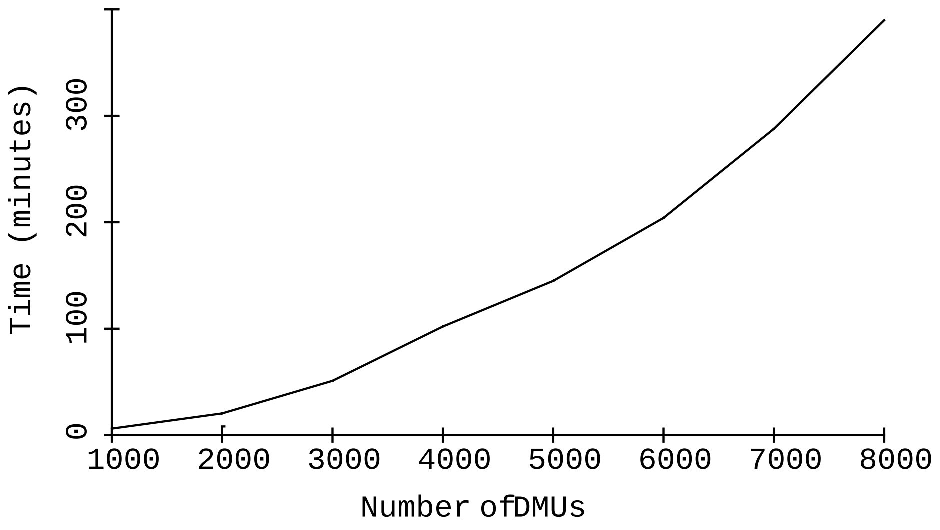

Figure 3: Solution Time versus Problem Size

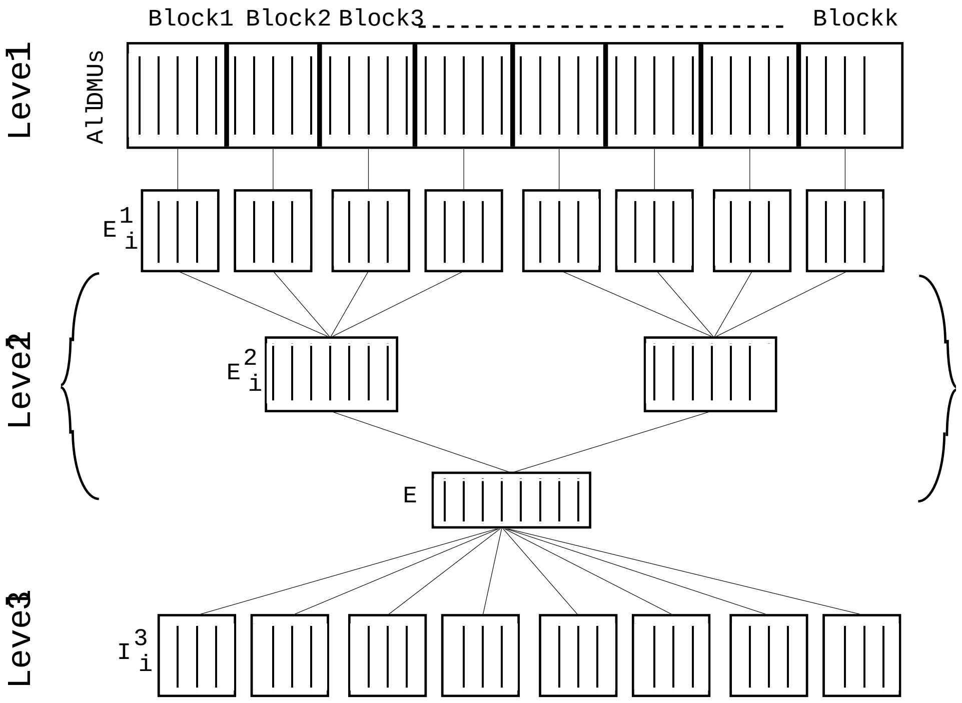

Figure 4: Hierarchical Decomposition Schematic

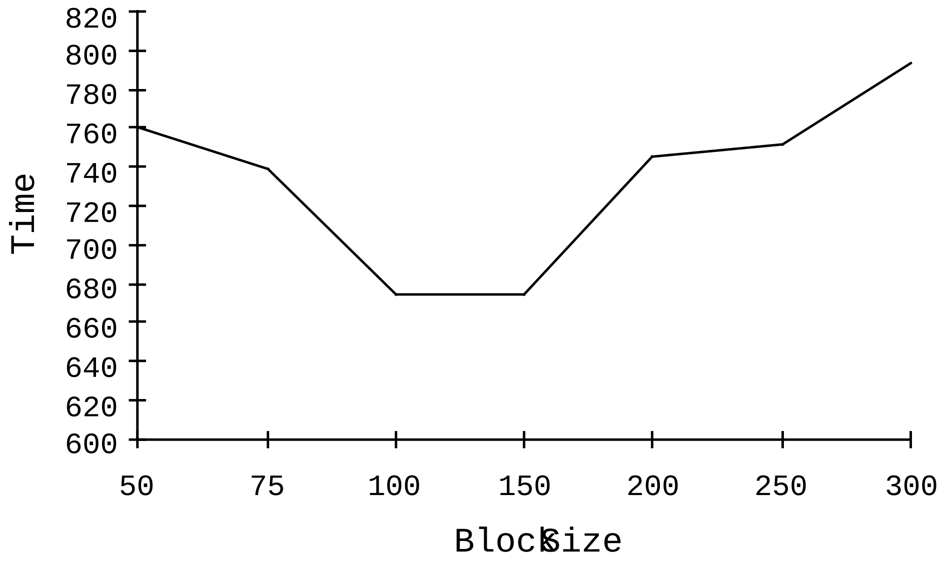

Figure 5: Total Solution Time vs. Blocksize for 8,000-DMU Problem

Table 5.-- Hierarchial Test Problem Results

	DMUs	Variables		Eff.	Number of Pivots			Number Priced			Time to Solve(min.)			Speedup		Efficiency	
		Out	In		1	5	10	1	5	10	1	5	10	5	10	5	10
Bank	1000	3	6	67	43992	44532	45207	1590581	1632351	1680000	12.41	2.78	1.70	4.46	7.30	0.89	0.73
	2000	3	6	70	92374	93154	94219	4572332	4959926	4757618	32.81	7.17	4.19	4.58	7.83	0.92	0.78
	4000	3	6	87	189784	190340	190987	12093187	12232628	12242949	85.31	17.64	9.40	4.84	9.08	0.97	0.91
	8000	3	6	77	382755	374088	383872	22890453	26946497	23429491	166.00	37.68	18.18	4.41	9.13	0.88	0.91
Multi-Normal	1000	3	4	79	43992	44532	45207	1590581	1632351	1680000	12.41	2.78	1.70	4.46	7.30	0.89	0.73
	2000	3	4	89	62418	62502	62674	4984732	5029902	5089368	28.03	5.85	3.14	4.81	8.92	0.96	0.89
	4000	3	4	85	132884	133149	133566	10490470	10595619	10723294	59.41	12.36	6.66	4.79	8.93	0.96	0.89
	8000	3	4	104	282679	283219	284001	22740407	22934235	23190802	128.10	26.64	14.30	4.81	8.96	0.96	0.90
DEA-GENa	1000	6	4	96	51102	51645	52392	1922161	1964649	2019596	15.81	3.50	2.04	4.52	7.75	0.90	0.78
	2000	3	5	147	87709	88084	88746	5211167	5275631	5364496	33.64	7.12	4.00	4.72	8.41	0.94	0.84
	4000	5	4	82	168813	169281	169856	12682231	12816378	12986010	83.38	17.18	9.11	4.85	9.15	0.97	0.92
	8000	2	5	70	233108	233585	234293	18772046	18962220	19215932	107.60	22.40	12.08	4.80	8.91	0.96	0.89
DEA-GENb	1000	4	7	214	72961	73706	74457	3530627	3583495	3638872	30.60	6.43	3.51	4.76	8.72	0.95	0.87
	2000	2	7	275	119211	113755	114479	8040444	8143142	8239181	55.37	11.54	6.21	4.80	8.92	0.96	0.89
	4000	3	3	64	98087	99237	99378	7397992	7474611	7560169	39.77	8.29	4.49	4.80	8.86	0.96	0.89
	8000	4	6	153	16418	16443	16493	36671079	36943789	37360152	248.90	51.63	27.53	4.82	9.04	0.96	0.90
DEA-GENc	1000	6	6	201	79180	79976	81034	3693590	3749286	3824666	33.97	7.19	3.91	4.72	8.69	0.94	0.87
	2000	5	6	265	151704	152727	154118	10210540	10327905	10479963	79.73	16.58	8.82	4.81	9.04	0.96	0.90
	4000	4	4	66	127473	127684	127827	9348034	9440561	9547832	57.72	11.97	6.38	4.82	9.05	0.96	0.90
	8000	4	5	216	400772	401413	402133	37030392	37277393	37572989	231.20	48.24	25.57	4.79	9.04	0.96	0.90

Table 6.-- Bank Hierarchial Level Test Problem Results

DMUs		Number of Pivots			Number Priced			Time to Solve(min.)			Speedup		Efficiency		Fraction of Time		
		1	5	10	1	5	10	1	5	10	5	10	5	10	1	5	10
1000	Level 1	14739	14968	15458	504349	528431	562621	3.92	0.87	0.53	4.51	7.40	0.90	0.74	.32	.31	.31
	Level 2	7865	8176	8361	322718	340406	353865	2.41	0.53	0.30	4.55	8.03	0.91	0.80	.19	.19	.18
	Level 3	21388	21388	21388	763514	763514	763514	6.08	1.38	0.87	4.41	6.99	0.88	0.70	.49	.50	.51
2000	Level 1	30770	31234	31679	1493465	1558862	1628509	10.53	2.30	1.31	4.58	8.04	0.92	0.80	.32	.32	.27
	Level 2	14486	14702	14922	820049	842246	870291	5.82	1.22	0.67	4.77	8.69	0.95	0.87	.18	.18	.16
	Level 3	47218	47218	47218	2258818	2558818	2258818	16.46	3.65	2.21	4.51	7.45	0.90	0.74	.50	.50	.53
4000	Level 1	80874	81060	81454	5347913	5438858	5441015	36.49	7.52	3.97	4.85	9.19	0.97	0.92	.43	.43	.42
	Level 2	20475	20845	21098	1614961	1663457	1671621	10.95	2.28	1.20	4.80	9.13	0.96	0.91	.13	.13	.13
	Level 3	88435	88435	88435	5130313	5130313	5130313	37.87	7.84	4.23	4.83	8.95	0.97	0.90	.44	.44	.45
8000	Level 1	165189	165432	165996	10774504	11051147	11232790	73.65	16.53	8.05	4.46	9.15	0.89	0.91	.44	.44	.44
	Level 2	41814	41985	42124	3252093	3295040	3332845	22.26	4.81	2.34	4.63	9.51	0.93	0.95	.13	.13	.13
	Level 3	175752	175752	175752	8863856	8863856	8863856	70.10	16.34	7.79	4.29	9.00	0.86	0.90	.43	.43	.43

Table 7.-- Muli-normal Hierarchial Level Test Problem Results

DMUs		Number of Pivots			Number Priced			Time to Solve(min.)			Speedup		Efficiency		Fraction of Time		
		1	5	10	1	5	10	1	5	10	5	10	5	10	1	5	10
1000	Level1	14739	14968	15458	504349	528431	562621	3.92	0.87	0.53	4.51	7.40	0.90	0.74	.32	.31	.31
	Level2	7865	8176	8361	322718	340406	353865	2.41	0.53	0.30	4.55	8.03	0.91	0.80	.19	.19	.18
	Level3	21388	21388	21388	763514	763514	763514	6.08	1.38	0.87	4.41	6.99	0.88	0.70	.49	.50	.51
2000	Level1	27897	27969	28102	2239921	2279134	2330107	12.36	2.57	1.37	4.81	9.02	0.96	0.90	.44	.44	.44
	Level2	3928	3940	3979	366038	371995	380488	1.98	0.41	0.22	4.83	9.00	0.97	0.90	.07	.07	.07
	Level3	30593	30593	30593	2378773	2378773	2378773	13.69	2.87	1.55	4.77	8.83	0.95	0.88	.49	.49	.49
4000	Level1	55307	55535	55790	4420400	4511812	4610712	24.29	5.09	2.71	4.77	8.96	0.95	0.90	.41	.41	.41
	Level2	12400	12437	12599	1131057	1144794	1173569	6.31	1.30	0.70	4.85	9.01	0.97	0.90	.11	.11	.11
	Level3	65177	65177	65177	4939013	4939013	4939013	28.81	5.97	3.25	4.83	8.86	0.97	0.89	.48	.48	.48
8000	Level1	112154	112564	113192	8900632	9068521	9286181	49.50	10.28	5.48	4.82	9.03	0.96	0.90	.39	.39	.38
	Level2	23633	23763	23917	2149424	2175363	2214270	12.17	2.49	1.30	4.89	9.36	0.98	0.94	.10	.09	.09
	Level3	146892	146892	146892	11690351	11690351	11690351	66.40	13.87	7.52	4.79	8.83	0.96	0.88	.51	.52	.53

Table 8.-- DEA-GENa Hierarchial Level Test Problem Results

DMUs		Number of Pivots			Number Priced			Time to Solve(min.)			Speedup		Efficiency		Fraction of Time		
		1	5	10	1	5	10	1	5	10	5	10	5	10	1	5	10
1000	Level1	16360	16749	17042	538284	569692	599536	4.54	1.02	0.60	4.45	7.57	0.89	0.76	.29	.29	.29
	Level2	7960	8114	8568	340961	352041	377144	2.66	0.58	0.34	4.59	7.82	0.92	0.78	.17	.17	.17
	Level3	26782	26782	26782	1042916	1042916	1042916	8.61	1.90	1.10	4.53	7.83	0.91	0.78	.54	.54	.54
2000	Level1	25295	25565	26010	1331525	1377928	1442319	8.58	1.86	1.08	4.61	7.94	0.92	0.79	.26	.26	.27
	Level2	16902	17007	17224	1072520	1090581	1115055	6.77	1.39	0.75	4.87	9.03	0.97	0.90	.20	.20	.19
	Level3	45512	45512	45512	2807122	2807122	2807122	18.29	3.87	2.17	4.73	8.43	0.95	0.84	.54	.54	.54
4000	Level1	69334	69591	69879	5397968	5501412	5623544	33.95	7.08	3.72	4.80	9.13	0.96	0.91	.41	.41	.41
	Level2	18130	18341	18628	1935924	1966627	2014127	12.07	2.47	1.29	4.89	9.36	0.98	0.94	.14	.14	.14
	Level3	81349	81349	81349	5348339	5348339	5348339	37.36	7.63	4.10	4.90	9.11	0.98	0.91	.45	.45	.45
8000	Level1	93421	93782	94278	7692497	7842514	8035761	42.46	8.90	4.79	4.77	8.86	0.95	0.89	.39	.40	.40
	Level2	26747	26863	27075	3274462	3314619	3375084	17.28	3.53	1.84	4.90	9.39	0.98	0.94	.16	.16	.15
	Level3	112940	112940	112940	7805087	7805087	7805087	47.87	9.97	5.45	4.80	8.78	0.96	0.88	.45	.44	.45

Table 9.-- DEA-GENb Hierarchial Level Test Problem Results

DMUs		Number of Pivots			Number Priced			Time to Solve (min.)			Speedup		Efficiency		Fraction of Time		
		1	5	10	1	5	10	1	5	10	5	10	5	10	1	5	10
1000	Level1	14439	14815	15240	556774	584561	615245	4.92	1.09	0.65	4.51	7.57	0.90	0.76	.16	.17	.19
	Level2	24853	25222	25548	1302084	1327165	1351858	10.76	2.25	1.21	4.78	8.89	0.96	0.89	.35	.35	.34
	Level3	33669	33669	33669	1671769	1671769	1671769	14.92	3.09	1.65	4.83	9.04	0.97	0.90	.49	.48	.47
2000	Level1	24953	25337	25788	1404447	1456216	1514152	9.64	2.09	1.20	4.61	8.03	0.92	0.80	.17	.18	.19
	Level2	32561	33021	33294	2456025	2506954	2545057	16.21	3.36	1.79	4.82	9.06	0.96	0.91	.29	.29	.29
	Level3	55397	55397	55397	4179972	4179972	4179972	29.52	6.09	3.22	4.85	9.17	0.97	0.92	.54	.53	.52
4000	Level1	44515	45676	45831	3647181	3720602	3801151	18.56	3.86	2.08	4.81	8.92	0.96	0.89	.47	.47	.46
	Level2	7166	7155	7141	674808	678006	683015	3.47	0.70	0.36	4.96	9.64	0.99	0.96	.09	.08	.08
	Level3	46406	46406	46406	3076003	3076003	3076003	17.74	3.73	2.05	4.76	8.65	0.95	0.87	.44	.45	.46
8000	Level1	136962	137406	138254	11300447	11493854	11764096	75.94	15.67	8.28	4.85	9.17	0.97	0.92	.31	.30	.30
	Level2	66459	66735	67432	7522680	7601983	7748104	50.08	10.19	5.24	4.91	9.56	0.98	0.96	.20	.20	.19
	Level3	215839	215839	215839	17847952	17847952	17847952	122.90	25.77	14.02	4.77	8.76	0.95	0.88	.49	.50	.51

Table 10.-- DEA-GENc Hierarchial Level Test Problem Results

DMUs		Number of Pivots			Number Priced			Time to Solve (min.)			Speedup		Efficiency		Fraction of Time		
		1	5	10	1	5	10	1	5	10	5	10	5	10	1	5	10
1000	Level1	16890	17438	17904	632567	668930	705190	5.87	1.36	0.76	4.32	7.72	0.86	0.77	.17	.19	.19
	Level2	26070	26318	26910	1310594	1329927	1369047	11.53	2.40	1.31	4.80	8.80	0.96	0.88	.34	.33	.34
	Level3	36220	36220	36220	1750429	1750429	1750429	16.57	3.43	1.84	4.83	9.01	0.97	0.90	.49	.48	.47
2000	Level1	30545	31123	31848	1663487	1732443	1815061	13.02	2.81	1.59	4.63	8.19	0.93	0.82	.16	.17	.18
	Level2	41269	41714	42380	2975857	3024266	3093706	22.37	4.63	2.45	4.83	9.13	0.97	0.91	.28	.28	.28
	Level3	79890	79890	79890	5571196	5571196	5571196	44.34	9.14	4.78	4.85	9.28	0.97	0.93	.56	.55	.54
4000	Level1	59940	60099	60225	4627122	4708855	4805916	27.30	5.66	2.99	4.82	9.13	0.96	0.91	.47	.47	.47
	Level2	9627	9679	9696	979096	989890	1000100	5.82	1.19	0.61	4.89	9.54	0.98	0.95	.10	.10	.10
	Level3	57906	57906	57906	3741816	3741816	3741816	24.60	5.12	2.78	4.80	8.85	0.96	0.88	.43	.43	.43
8000	Level1	128400	128880	129358	10534301	10730605	10964250	65.30	13.66	7.24	4.78	9.02	0.96	0.90	.28	.28	.28
	Level2	54664	54825	55067	5780407	5831104	5893055	35.75	7.30	3.75	4.90	9.53	0.98	0.95	.15	.15	.15
	Level3	217708	217708	217708	20715684	20715684	20715684	130.10	27.28	14.58	4.77	8.92	0.95	0.89	.57	.57	.57

Table 11.-- Bank Data Results – Linear Programs Required

DMUs		LPs No Early Identification 1	 5	 10	LPs With Early Identification 1	 5	 10
1000	Level1	1000	1000	1000	795	807	827
	Level2	408	408	408	294	304	312
	Level3	922	922	922	922	922	922
	Total	2330	2330	2330	2011	2033	2061
2000	Level1	2000	2000	2000	1649	1670	1694
	Level2	653	653	653	509	519	530
	Level3	1918	1918	1918	1918	1918	1918
	Total	4571	4571	4571	4076	4107	4142
4000	Level1	4000	4000	4000	3552	3562	3577
	Level2	779	779	779	638	648	653
	Level3	3902	3902	3902	3902	3902	3902
	Total	8681	8681	8681	8092	8112	8132
8000	Level1	8000	8000	8000	7092	7116	7149
	Level2	1487	1487	1487	1278	1288	1289
	Level3	7923	7923	7923	7923	7923	7923
	Total	17410	17410	17410	16293	16327	16361

Table 12.-- Multi-Normal Data Results – Linear Programs Required

DMUs		LPs No Early Identification			LPs With Early Identification		
		1	5	10	1	5	10
1000	Level1	1000	1000	100	795	807	827
	Level2	408	408	408	294	304	312
	Level3	922	922	922	922	922	922
	Total	2330	2330	2330	2011	2033	2061
2000	Level1	2000	2000	2000	1781	1783	1788
	Level2	282	282	282	211	212	213
	Level3	1911	1911	1911	1911	1911	1911
	Total	4193	4193	4193	3903	3906	3912
4000	Level1	4000	4000	4000	3530	3542	3549
	Level2	709	709	709	569	570	579
	Level3	3915	3915	3915	3915	3915	3915
	Total	8024	8624	8624	8014	8027	8043
8000	Level1	8000	8000	8000	7106	7120	7145
	Level2	1295	1295	1295	1074	1080	1091
	Level3	7896	7896	7896	7896	7896	7896
	Total	17191	17191	17191	16076	16096	16132

Table 13.-- DEA-GENa Results – Linear Programs Required

DMUs		LPs No Early Identification			LPs With Early Identification		
		1	5	10	1	5	10
1000	Level1	1000	1000	1000	774	791	797
	Level2	435	435	435	288	291	300
	Level3	904	904	904	904	904	904
	Total	2339	2339	2339	1966	1986	2001
2000	Level1	2000	2000	2000	1520	1533	1566
	Level2	972	972	972	682	687	692
	Level3	1853	1853	1853	1853	1853	1853
	Total	4825	4825	4825	4055	4073	4111
4000	Level1	4000	4000	4000	3522	3530	3540
	Level2	729	729	729	612	616	623
	Level3	3918	3918	3918	3918	3918	3918
	Total	8647	8647	8647	8052	8064	8081
8000	Level1	8000	8000	8000	7029	7050	7079
	Level2	1465	1465	1465	1271	1276	1285
	Level3	7930	7930	7930	7930	7930	7930
	Total	17395	17395	17395	16230	16256	16294

Table 14.-- DEA-GENb Results – Linear Programs Required

DMUs		LPs No Early Identification 1	5	10	LPs With Early Identification 1	5	10
1000	Level1	1000	1000	1000	670	688	709
	Level2	1105	1105	1105	716	729	740
	Level3	786	786	786	786	786	786
	Total	2891	2891	2891	2172	2203	2235
2000	Level1	2000	2000	2000	1436	1455	1475
	Level2	1785	1785	1785	1173	1190	1201
	Level3	1725	1725	1725	1725	1725	1725
	Total	5510	5510	5510	4334	4370	4401
4000	Level1	4000	4000	4000	3636	3644	3650
	Level2	438	438	438	384	384	383
	Level3	3936	3936	3936	3936	3936	3936
	Total	8374	8374	8374	7956	7964	7969
8000	Level1	8000	8000	8000	6678	6698	6507
	Level2	2326	2326	2326	1893	1898	2506
	Level3	7847	7847	7847	7847	7847	7886
	Total	18173	18173	18173	16418	16443	16899

Table 15.-- DEA-GENc Results – Linear Programs Required

DMUs		LPs No Early Identification 1	5	10	LPs With Early Identification 1	5	10
1000	Level1	1000	1000	1000	684	704	726
	Level2	1052	1052	1052	670	684	700
	Level3	799	799	799	799	799	799
	Total	2851	2851	2851	2153	2187	2225
2000	Level1	2000	2000	2000	1415	1446	1480
	Level2	1806	1806	1806	1181	1194	1213
	Level3	1735	1735	1735	1735	1735	1735
	Total	5541	5541	5541	4331	4375	4428
4000	Level1	4000	4000	4000	3611	3616	3624
	Level2	462	462	462	413	413	413
	Level3	3934	3934	3934	3934	3934	3934
	Total	8396	8396	8396	7958	7963	7971
8000	Level1	8000	8000	8000	6730	6757	6782
	Level2	2181	2181	2181	1712	1716	1724
	Level3	7784	7784	7784	7784	7784	7784
	Total	17965	17965	17965	16226	16257	16290

Table 16.-- Non-Hierarchial and Hierarchial Comparisons: 8,000 DMU Problems

	Solution Times (min.)						Speed Improvement				
	No Hierarchial		Hierarchial								
	1	3	1	3	10	15	1:1	3:3	1:3	1:10	1:15
Bank	1294.07	434.58	166.00	61.73	18.18	14.27	7.80	7.04	20.96	71.18	90.68
Multi-Normal	931.98	313.15	128.10	43.72	14.30	10.54	7.28	7.16	21.32	65.17	88.42
DEA-GENa	1088.50	366.51	107.60	31.92	12.08	8.83	10.12	11.48	34.10	90.11	123.27
DEA-GENb	2091.20	704.13	248.90	84.47	27.53	19.89	8.40	8.34	24.76	75.96	105.14
DEA-GENc	1437.32	482.84	231.20	78.88	25.57	18.55	6.22	6.12	18.22	56.21	77.48