Computers & Operations Research 36 (2009) 1813 – 1824

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

A procedure for large-scale DEA computations

Wen-Chih Chen*, Wei-Jen Cho

Department of Industrial Engineering and Management, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 300, Taiwan

ARTICLE INFO

Available online 17 May 2008

Keywords:
Data envelopment analysis
Computational efficiency
Large-scale LP problems

ABSTRACT

Data envelopment analysis (DEA), a performance evaluation method, measures the relative efficiency of a particular decision making unit (DMU) against a peer group. Most popular DEA models can be solved using standard linear programming (LP) techniques and therefore, in theory, are considered as computationally easy. However, in practice, the computational load cannot be neglected for large-scale—in terms of number of DMUs—problems. This study proposes an accelerating procedure that properly identifies a few "similar" critical DMUs to compute DMU efficiency scores in a given set. Simulation results demonstrate that the proposed procedure is suitable for solving large-scale BCC problems when the percentage of efficient DMUs is high. The computational benefits of this procedure are significant especially when the number of inputs and outputs is small, which are most widely reported in the literature and practices.

1. Introduction

Data envelopment analysis (DEA), first introduced by Charnes et al. [1], is a well-established method for relative performance evaluation. This method considers multiple inputs and outputs simultaneously without assigning a priori weights or assumptions on the functional form. In the past three decades, DEA has demonstrated itself to be a highly effective means of evaluating relative performance.

DEA evaluates relative performance for a decision making unit (DMU) against its peer group S. To accomplish this, the most widely used DEA models formulate and solve linear programming (LP) problems. Standard approaches solve $|S|$ LP problems, each with all members. Although an LP problem is theoretically polynomial-time solvable, in practice, solution time increases significantly for large cases. Barr and Durchholz [2] summarize this effect by stating, "Not only did the memory requirements of larger problems limit the amount of usable parallelism, but run times grew exponentially in n" where n is the number of DMUs. Intensive computation engenders numerous practical application challenges. For example, to evaluate the efficiency of 4796 Brazilian municipalities, Sampaio de Sousa and Stosic [3] take over 17 days to detect outlying points from the 4796 records. Therefore, methods for reducing solution time for DEA problems are practically beneficial, especially for large DEA problems.

Two major issues are associated with solving LP problems. The first issue is associated with how to solve the problem rapidly given a particular problem size (e.g., the simplex method, interior point method, algebraic method and graphic method). The second issue is related to reducing problem size using a specific technique or software package. Relevant literatures on DEA computational issues has focused on the second issue, and [4,5] provide comprehensive discussions and reviews.

Inefficient DMUs cannot affect the efficiency scores of other DMUs; identifying and removing these inefficient DMUs during computations can reduce LP problem size. There are some works presenting a way to flag dominated DMUs that are always inefficient (e.g., [6,7]). A DMU is also always inefficient overall when it is inefficient in comparison with a small subgroup. Barr and Durchholz [2] utilize this observation and partition a data set into small subsets to identify inefficient DMUs. Other studies attempt to utilize efficient DMU characteristics. For example, Chen and Ali [8] point out that a DMU is efficient when it has the highest value in any form of an output-to-input ratio.

Another stream of literature links computational geometry and DEA (e.g., Dulá and Venugopal [9]). The concept of frame, which is a set of minimum extreme points constructing a polyhedral cone, is the foundation of these studies. The data points in the frame corresponding to DEA models are efficient. Dulá and Trall [10] propose a two-stage approach based on this idea. During the first stage, points in the frame are identified using specific well-established algorithms (e.g., [11,12]). Points in the frame are used to compute efficiency scores for those not in the frame in the second stage. The LP problem size is therefore reduced in DEA computations, and computational time is decreased.

Three factors related to computational performance are [4]: (1) "Dimension" (number of inputs and outputs); (2) "Scale" (total number of DMUs) and (3) "Density" (percentage of efficient DMUs).

* Corresponding author.
E-mail address: wenchih@faculty.nctu.edu.tw (W.-C. Chen).

0305-0548/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2008.05.006

Early studies accelerate large-scale DEA computation by classifying efficient and inefficient DMUs such that a downsized data set is used. These techniques serve low-density cases effectively, as reported by Dulá [4] and Barr and Durchholz [2]. However, they may not be sufficient for all large cases, particularly the high-density situations. Considering the worst cases in which all DMUs are efficient, no inefficient DMUs can be identified; thus, the LP's problem size cannot be reduced. Therefore, this work presents a novel accelerating procedure for solving large-scale DEA BCC problems. The proposed procedure is suitable for solving large-scale problems with high-efficiency density data. The computational benefits of the procedure are significant especially for low-dimension cases, which is typical in practices.

The rest of this paper is organized as follows. Section 2 introduces the fundamentals of DEA, particularly its managerial and geometric interpretations. Section 3 then addresses the implementation of the proposed acceleration method. Finally, Section 4 demonstrates the effectiveness of the proposed method. Conclusions are finally drawn in Section 5, along with recommendations for future research.

2. Data envelopment analysis

Consider an input set I and output set O. Let $x \in \Re_+^{|I|}$ be the input vector, $y \in \Re_+^{|O|}$ be the output vector, and (x^k, y^k) be the input–output bundle for any DMU $k \in S$. A popular DEA model for evaluating the efficiency score by comparing k with S is the input-oriented BCC model proposed by Banker et al. [13]:

$$\begin{aligned} E(k,S) = \min_{\theta,\lambda_r} \quad & \theta \\ \text{s.t.} \quad & \sum_{r\in S} x_i^r \lambda_r \leqslant \theta x_i^k \quad \forall i \in I, \\ & \sum_{r\in S} y_j^r \lambda_r \geqslant y_j^k \quad \forall j \in O, \\ & \sum_{r\in S} \lambda_r = 1, \\ & \lambda_r \geqslant 0 \quad \forall r \in S. \end{aligned} \tag{1}$$

Model (1) attempts to proportionately minimize usage of x^k by θ while maintaining at least the same output level. A feasible input–output bundle for this minimization process, $(\theta x^k, y^k)$, should be within the empirical production possibility set (EPPS) constructed by set S. Ray and Mukherjee [14] describe the EPPS as "an inner approximation to the true production possibility set" and "the free disposal convex hull of the observed points." In model (1), the last two constraints and the left-hand side of the first two constraint types (input/output constraints) are associated with the convex hull constructed by S, i.e., $\{(\sum_{r\in S} x^r \lambda_r, \sum_{r\in S} y^r \lambda_r) | \sum_{r\in S} \lambda_r = 1; \lambda_r \geqslant 0, r \in S\}$. The inequalities represent the free disposal property, revealing that using more or equal resources ($\leqslant$) to produce fewer or equal outputs ($\geqslant$) than any known feasible bundle, i.e., the left-hand side of inequalities (the convex hull), is always achievable. Therefore, constraints of (1) define the feasibility relationship between EPPS and $(\theta x^k, y^k)$. The optimal value of model (1), $E(k,S)$, is the input efficiency or efficiency score for (x^k, y^k). This is because the optimal solution $(\sum_{r\in S} x^r \lambda_r^*, \sum_{r\in S} y^r \lambda_r^*)$ is feasible using only θ^*times of x^k to produce no less than y^k, i.e., $\sum_{r\in S} y^r \lambda_r^* \geqslant y^k$, and is the comparison basis. Any DMU k with $E(k,S) = 1$ is efficient and on the efficient frontier.

$(\sum_{r\in S} x^r \lambda_r^*, \sum_{r\in S} y^r \lambda_r^*)$ represents the "virtual DMU" composed of the peers for DMU k. DMU r is a reference point, or simply the reference, of DMU k if it contributes to the composed virtual DMU, i.e., $\lambda_r^* > 0$. The optimal solution $\lambda_r^* = 0$ indicates that r does not contribute to the virtual DMU and thus cannot affect the efficiency score for k. Hereafter, this study refers to DMU k in model (1) as the target DMU or target. Moreover, $E(k,S)$ is notably a radial measure that tries to move x^k toward the origin by minimizing θ; $\sum_{r\in S} x^r \lambda_r^*$ lies on the segment between x^k and the origin in most cases, and has mix identical to x^k. This feature implies that an inefficient DMU is compared against DMUs with similar input–output mix on the frontier. For example, suppose two inputs exist—capital and labor. A labor intensive inefficient DMU will be compared with efficient DMUs that are also labor intensive, not capital intensive. The following model is equivalent to the dual of model (1), also called ratio form in the DEA literature, and it provides a clear explanation on this argument.

$$\begin{aligned} E(k,S) = \max_{v_i,u_j,u_0} \quad & \frac{\sum_{j\in O} u_j y_j^k + u_0}{\sum_{i\in I} v_i x_i^k} \\ \text{s.t.} \quad & \frac{\sum_{j\in J} u_j y_j^r + u_0}{\sum_{i\in I} v_i x_i^r} \leqslant 1 \quad \forall r \in S, \\ & v_i \geqslant 0 \quad \forall i \in I, \\ & u_j \geqslant 0 \quad \forall j \in O. \end{aligned} \tag{2}$$

In model (2), DMU k selects its favorite weights according to its input–output vector for comparison. The reference DMUs are those performing well given the selected weights, and thus they should have similar input–output mix in response to the weights.

In addition to the radial efficiency measurement, classifying DMUs as efficient or inefficient is an important topic in DEA studies. The variance of model (1) for this purpose is as follows:

$$\begin{aligned} \min_{\theta,\lambda_r,s_i^+,s_j^-} \quad & \theta - \varepsilon\left(\sum_{i\in I} s_i^+ + \sum_{j\in O} s_j^-\right) \\ \text{s.t.} \quad & \sum_{r\in S} x_i^r \lambda_r + s_i^+ = \theta x_i^k \quad \forall i \in I, \\ & \sum_{r\in S} y_j^r \lambda_r - s_j^- = y_j^k \quad \forall j \in O, \\ & \sum_{r\in S} \lambda_r = 1, \\ & \lambda_r \geqslant 0 \quad \forall r \in S. \end{aligned}$$

where ε is the non-Archimedean infinitesimal, and s_i^+ and s_j^- are input/output slacks. The DMUs with unity scores and all slacks being zero are called strong efficient, whereas the DMUs with unity scores and non-zero slacks are referred to as weak efficient. Although the main objective is to compute radial efficiency, the proposed method can accelerate DEA classification as well. Such classification can be done using a two-phase procedure. Phase one solves model (1) for all DMUs. The scores obtained from phase one are parameters in phase two that checks for slacks. The optimal solutions obtained in model (1) can reduce the size of the LP problem for phase two problem solving.

3. Acceleration procedure

This section presents a novel procedure for solving large DEA problems. Particularly, the proposed method accelerates the cases in which most DMUs are efficient but this is unknown to analysts. The acceleration procedure presented here is for input-oriented BCC models, and can easily be extended to other radial models, such as CCR models [1] or different orientations with minor modifications.

3.1. Idea

Calculating BCC efficiency scores solves many different LP problems with structures identical to model (1). Model (1) has $|S| + 1$ variables and $|I| + |O| + 1$ constraints, typically with $|S| \geqslant |I| + |O| + 1$. The optimal solution of the corresponding LP problem has at most

```
Procedure FastDEA($S$, $\alpha$, $\beta$)
begin
    [$H$, $L$, $D$]=Initialization($S$, $\beta$);
    for each DMU $k$ in $S$ do
        $N$=FindNeighbor($k$, $\alpha$, $S \backslash D$);
        repeat
            [$\theta^k$, $\lambda$, $shadow\_price$]=SolveSmallLP($N$);
            $isKKT$=CheckKKT($\theta^k$, $\lambda$, $shadow\_price$, $S$);
            if $isKKT$ is FALSE do
                set larger $\alpha$;
                $N$=FindNeighbor($k$, $\alpha$, $S \backslash D$);
        until $isKKT$ is TRUE
    return $\theta^k$ for $k$ in $S$;
end
```

Fig. 1. The pseudocode of the procedure *FastDEA*.

$|I|+|O|$ λ's in the basis that are possibly non-zero, and the last slot in the basis is always left for θ. This observation reveals that for a single standard DEA (SDEA) computation, regardless of problem size, at most $|I|+|O|$ DMUs are related to the efficiency score, and only corresponding DMUs are needed for problem solving. Furthermore, as addressed in Section 2, the $|I|+|O|$ reference DMUs are similar to the target DMU. The proposed acceleration procedure, *FastDEA*, consequently selects as few DMUs (columns) as possible to construct the corresponding LP problem. The primary idea in finding this reduced set of DMUs to compute the score for a particular DMU k is to properly define the "similar" DMUs related to k. Hereafter these "similar" DMUs are referred to as k's neighbors.

Fig. 1 presents the pseudocode of *FastDEA*. Notably, *FastDEA* requires the peer group data S, a range parameter α and threshold β as inputs, and returns efficiency scores for all DMUs in S. The first part, *Initialization*, is the data preprocess that transforms data (x,y) to a polar coordinate system and categorizes DMUs into several subsets (H, L and D) representing different possibilities of being efficient. Data in D are dominated, i.e., at least one other DMU exists that uses relatively fewer resources but producing relatively more outputs. Thus data in D are clearly inefficient. Two major parts exist in the loop that computes scores for each DMU. In the first part, *FindNeighbor* identifies a small set of neighbors, N ($N \subseteq S$), for target DMU k, and *SolveSmallLP* solves model (1) using only reduced set N to obtain $\theta^k = E(k,N)$. The second part determines whether the score obtained by N is identical to that using S—the full-scale problem—via procedure *CheckKKT*, namely, whether $E(k,N)=E(k,S)$ holds is determined. A "no" answer enlarges N and *FindNeighbor* is repeated. This process is repeated until solutions based on N and S are identical. The following sections describe the sub-procedures in detail.

3.2. Finding neighbors

This section introduces the idea of procedure *FindNeighbor* and its implementation, which is key to downsizing LP problems. Identifying an appropriate small set of DMUs related to the target DMU, such that elements in this particular set are "similar" to the target, is desirable. Via proper selection, at most $|I|+|O|$ reference points are needed for efficiency computations. In reality, identifying the corresponding reference points is computationally intensive. Therefore, instead of locating exact references, this work determines a reasonably small set of neighbors that likely contains reference points.

Two aspects, mix and magnitude, are geometrically associated with input and output vectors x and y. Model (1) shows the radial downscaling of x with a fixed mix until the virtual DMU on the efficient frontier is reached. References, which compose the virtual DMU, depend only on mix and not on the magnitude of inputs as regardless of the magnitude of x, it will be reduced. The most likely references are thus the ones with an input mix similar to that of the target.

The original DEA data (x,y)'s are represented in the rectangular coordinate system. The polar coordinate system is another coordinate system that represents a point by its direction (angle or mix) ϕ and magnitude γ separately with a one-to-one mapping between these two systems [15]. For instance, in a two-dimension case, a rectangular system, (x_1,x_2), can be transformed to (ϕ,γ) in a polar coordinate system with the mappings $x_1=\gamma\cos\phi$ and $x_2=\gamma\sin\phi$, where $\gamma=\sqrt{x_1^2+x_2^2}$ and $0 \leqslant \phi \leqslant 2\pi$. Generally, n-dimension cases contain $n-1$ angles and 1 scale index (refer to [15], for high-dimension mappings). Transforming data to the polar coordinate system can separate the information into magnitude and mix. Hence, mix similarity can be easily identified using only angle information.

Denote $\phi_x^k=(\phi_1^k,\ldots,\phi_{|I|-1}^k)$ as the input angle vector of DMU k represented in the polar coordinate system. The coordinate transformation is only applied to the input vectors since only the input mix is of interest here. Given two angle vectors, ϕ_x^k and ϕ_x^r, for DMU k and r, respectively, one simple way, but not limited to, for determining the difference between two vectors is based on the two-norm. Thus, the input mix dissimilarity index between DMUs k and r can be defined as

$$IG(k,r)=\sqrt{\sum_{i=1}^{|I|-1}(\phi_i^k-\phi_i^r)^2}. \tag{3}$$

Greater $IG(k,r)$ indicates a higher input mix dissimilarity between k and r. $IG(k,r)=0$ indicates that k and r are identical in the input mix. Let $\alpha \in [0,1]$ be the predetermined tolerance parameter representing the acceptable proportion $(100\times\alpha\%)$ of most similar DMUs corresponding to the target. Set $S_x(k,\alpha) \subseteq S$ can be determined according to (3), where elements in $S_x(k,\alpha)$ are the top $100\times\alpha\%$ DMUs similar to k in terms of input mix, i.e., $\forall r \in S_x(k,\alpha)$ has the $100\times\alpha\%$ smallest value of $IG(k,r)$.

Rather than checking mix for defining input vector similarity, both mix and magnitude are needed for the output because, in contrast to inputs, no target output scaling is required in model (1) and magnitude does matter in this case. The dissimilarity index of any two DMUs based on [16] combining both mix and magnitude aspects can be used for the output:

$$AD(k,r)\equiv\frac{1}{|O|}\left(\sum_{j\in O}\left(\ln\frac{y_j^r}{y_j^k}\right)^2\right). \tag{4}$$

Higher $AD(k,r)$ indicates greater dissimilarity between DMUs k and r regarding outputs. In the same manner, the similar set related to k in terms of output can be denoted as $S_y(k,\alpha)$ according to (4).

This work is looking for DMUs similar to k in terms of inputs and outputs. Consequently, $N(k,\alpha)=S_x(k,\alpha)\cap S_y(k,\alpha)$ is the set of neighbors for the target k with respect to tolerance α. Additionally, the proportion of $N(k,\alpha)$ to the overall candidate set is notably around α^2, not α.

3.3. Categorizing DMUs

A potential problem exists regarding the effectiveness of catching true references by selected neighbors in $N(k,\alpha)$. Only efficient DMUs in $N(k,\alpha)$ are proper true reference candidates. More efficient candidates in $N(k,\alpha)$ have an increased chance of catching the right

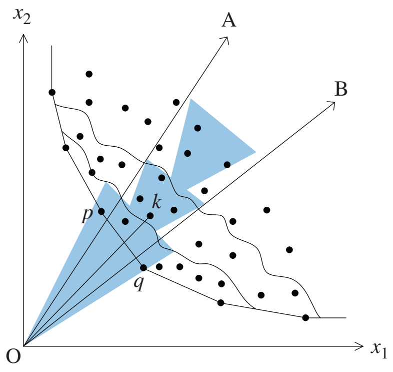

Fig. 2. How to find the neighbors.

references given the same size of neighbors. Therefore, density of $N(k,\alpha)$ determines the possibility of including true references and, hence, its effectiveness.

Set $S_X(k,\alpha)$ is determined by an angle and is a cone-shaped region. It is more likely to include inefficient DMUs than efficient DMUs because of a large transection located far from the origin. Fig. 2 presents a simple two input equal output example illustrating this issue. This work evaluates the efficiency of k with underlying but unknown references p and q. Suppose $N(k,\alpha)$ is the region between rays OA and OB, with ten total points within it. Eight of the ten are worse than k, and $N(k,\alpha)$ does not contain references p and q due to the data distribution. The region between rays OA and OB widens far from the origin; hence, it will contain more uncritical points than true references. Therefore, noise needs to be filtered out (e.g., inefficient DMUs) when using certain bullets (neighbors) to shoot targets (references).

First, dominated DMUs can be flagged as inefficient to resolve this problem without solving any LP problems [17]. By pairwise comparisons, dominated DMUs can be easily identified and collected in set D. Only DMUs in $S\backslash D$ are possible as a reference and require further processing such as being transformed to a polar coordinate system.

Classifying $S\backslash D$ into different classes according to potential efficiency and selecting neighbors from each class separately can further increase the possibility of $N(k,\alpha)$ containing true references. If DMUs are classified into three classes, one can have different angle ranges in each class. The piece near the frontier should be wider than without classification and thus more likely to contain true references for k.

Fig. 3 presents the pseudocode of the *Initialization* procedure, which contains a DMU data set S and threshold β as inputs and returns three disjoint subsets of S (H, L and D). Via paired comparisons, the *FindDominated* procedure identifies points dominated by any point and classifies them in D. Only DMUs in $S\backslash D$ are neighbor candidates and require that vector (x,y) be transformed into the polar coordinate system $(\phi_x,\phi_y,\gamma_x,\gamma_y)$ via the *TransformToPolar* procedure. This work transforms the input and output vectors independently.

Categorization partitions $S\backslash D$ into H, L and others, where H and L represent sets of DMUs being relatively more or less efficient, respectively. Instead of solving model (1), the sets are roughly determined by the possible dominance relationship between DMUs. Given $(\phi_x,\phi_y,\gamma_x,\gamma_y)$ for each DMU, the DMUs with the larger γ_y and smaller γ_x are more likely to be efficient. Comparison of DMUs p and q with very a "similar" input–output mix indicates that p is very likely more efficient than q if $\gamma_y^p > \gamma_y^q$ and $\gamma_x^p < \gamma_x^q$. Thus, this work divides $S\backslash D$ into regions, in which DMUs are very "similar" in the input–output mix according to (ϕ_x,ϕ_y). Ranking γ_x and γ_y separately within region g, H_x^g collects DMUs whose γ_x are the bottom $100\times\beta\%$, where β is a prespecified threshold, and L_x^g represents those with γ_x that are the largest $100\times\beta\%$. Similarly, H_y^g and L_y^g can be defined according to γ_y, where H_y^g and L_y^g are collections of DMUs with γ_y being the top and bottom $100\times\beta\%$, respectively. Classes H and L are obtained as

$$H=\bigcup_g (H_x^g\cap H_y^g) \quad \text{and} \quad L=\bigcup_g (L_x^g\cap L_y^g).$$

Various means can define region g. Section 4 presents one example. Applying *FindNeighbor* with α to H, L and $S\backslash\{H\cup L\cup D\}$ separately, $N(k,\alpha)$ is the union of selected neighbors from three classes.

3.4. Checking optimality

Set $N(k,\alpha)$ determined by the *FindNeighbor* procedure contains neighbors likely to be the reference. For simplicity, $N(k,\alpha)$ is denoted as N in this section. The efficiency score for k using set N can be computed by model (1) as $E(k,N)$; $E(k,N)=E(k,S)$ is not necessary since $N\subseteq S$. Hence, this section addresses how to determine whether $E(k,N)=E(k,S)$ holds.

Given the optimal solution related to $E(k,N)$, the solution is also optimal for model (1) using S, i.e., $E(k,N)=E(k,S)$, if it satisfies the Karush–Kuhn–Tucker (KKT) optimality condition corresponding to S. The dual of model (1) is as follows:

$$\begin{aligned} E(k,S)=\max_{v_i,u_j,u_0}\quad & \sum_{j\in O} u_j y_j^k + u_0 \\ \text{s.t.}\quad & \sum_{i\in I} v_i x_i^k = 1, \\ & \sum_{j\in J} u_j y_j^r - \sum_{i\in I} v_i x_i^r + u_0 \leqslant 0 \quad \forall r\in S, \\ & v_i \geqslant 0 \quad \forall i\in I, \\ & u_j \geqslant 0 \quad \forall j\in O. \end{aligned} \tag{5}$$

Suppose $(\hat\theta;\hat\lambda_r, r\in N)$ is the optimal solution related to $E(k,N)$ obtained by model (1) and $(\hat u_0;\hat u_j, j\in O;\hat v_i, i\in I)$ is the optimal solution related to $E(k,N)$ obtained by (5). This work determines whether the solution $(\hat u_0;\hat u_j, j\in O;\hat v_i, i\in I;\hat\theta;\hat\lambda_r, r\in N;\hat\lambda_t=0, t\in S\backslash N)$ can yield $E(k,S)$ from models (1) and (5) by checking the KKT optimality condition. Notably, $\hat\lambda_t=0$ for $t\in S\backslash N$ are artificially added, not solved; $\hat\lambda_t=0$ because they are not neighbors and not in the basis with respect to the optimal solution for $E(k,S)$. The difference in model (5) regarding S and N is the number of constraints, not variables. The KKT optimality condition regarding S is, thus, the satisfaction of the following constraints.

Primal feasibility:

$$\sum_{t\in S\backslash N} x_i^t\hat\lambda_t + \sum_{r\in N} x_i^r\hat\lambda_r \leqslant \hat\theta x_i^k \quad \forall i\in I, \tag{6.1}$$

$$\sum_{t\in S\backslash N} y_j^t\hat\lambda_t + \sum_{r\in N} y_j^r\hat\lambda_r \geqslant y_j^k \quad \forall j\in O, \tag{6.2}$$

$$\sum_{t\in S\backslash N} \hat\lambda_t + \sum_{r\in N} \hat\lambda_r = 1, \tag{6.3}$$

$$\hat\lambda_r \geqslant 0 \quad \forall r\in N. \tag{6.4}$$

$$\hat\lambda_t \geqslant 0 \quad \forall t\in S\backslash N. \tag{6.5}$$

Dual feasibility:

$$\sum_{i\in I} \hat v_i x_i^k = 1, \tag{6.6}$$

$$\sum_{j\in J} \hat u_j y_j^t - \sum_{i\in I} \hat v_i x_i^t + \hat u_0 \leqslant 0 \quad \forall t\in S\backslash N, \tag{6.7}$$

```
Procedure Initialization($S$, $\beta$)
begin
    $D$=FindDominated($S$);   /* $D$ is the set of dominated DMUs identified by checking data directly.
    TransformToPolar($S\backslash D$);
    [$H$, $L$]= Categorization($S$,$\beta$);
    return [$H$, $L$, $D$];
end
```

Fig. 3. The pseudocode of the procedure *Initialization*.

$$\sum_{j\in J}\hat{u}_j y_j^r - \sum_{i\in I}\hat{v}_i x_i^r + \hat{u}_0 \leqslant 0 \quad \forall r \in N, \tag{6.8}$$

$$\hat{v}_i \geqslant 0 \ \forall i \in I \quad \text{and} \quad \hat{u}_j \geqslant 0 \ \forall j \in O. \tag{6.9}$$

Complementary slackness:

$$\hat{v}_i\left(\sum_{t\in S\backslash N} x_i^t\hat{\lambda}_t + \sum_{r\in N} x_i^r\hat{\lambda}_r - \hat{\theta}x_i^k\right) = 0 \quad \forall i \in I, \tag{6.10}$$

$$\hat{u}_j\left(\sum_{t\in S\backslash N} y_j^t\hat{\lambda}_t + \sum_{r\in N} y_j^r\hat{\lambda}_r - y_j^k\right) = 0 \quad \forall j \in O, \tag{6.11}$$

$$\hat{\lambda}_t\left(\sum_{j\in J}\hat{u}_j y_j^t - \sum_{i\in I}\hat{v}_i x_i^t + \hat{u}_0\right) = 0 \quad \forall t \in S\backslash N, \tag{6.12}$$

$$\hat{\lambda}_r\left(\sum_{j\in J}\hat{u}_j y_j^r - \sum_{i\in I}\hat{v}_i x_i^r + \hat{u}_0\right) = 0 \quad \forall r \in N. \tag{6.13}$$

The constraints (6.1)–(6.6) and (6.8)–(6.13) are clearly satisfied because $\hat{\lambda}_t = 0$, $t \in S\backslash N$. Only (6.7) is required for further assessment, which is easily done by procedure *CheckKKT*. If constraint (6.7) holds, the solution satisfies the KKT condition and thus is the optimal solution for model (1) using the full data set S. Therefore, $E(k,N) = E(k,S)$ is obtained and the loop is terminated; otherwise, a relatively larger neighborhood is required.

4. Case study

This section demonstrates the effectiveness of the proposed method by using simulated cases. The *SDEA* computational procedure for solving full-scale LP problems and the method proposed by Barr and Durchholz [2] (*HDEA*) are used as benchmarks for comparisons. The domination procedure *FindDominated* is also applied in *HDEA* implementation, but not in *SDEA*. The modified procedure, *SDEAwD*, which is *SDEA* with a domination procedure, is also implemented for comparison. No additional fine-tuning processes are implemented in all methods for fair comparisons. The *SDEA* procedure is utilized as the benchmark for computation time and solution accuracy; the other three methods must have solutions identical to those provided by *SDEA*. Trade-offs between the number of neighbors and number of computation iterations are also discussed.

4.1. Implementation

Simulated data are generated via a two-stage process. First, efficient data, (x^k, y^{k*}) $\forall k \in S$, specifying the ideal efficient input–output relationship, are generated by the software package FEAR [18]. Inefficiency components are added by the following function in the second stage:

$$y^k = \mathrm{e}^{-U_k} y^{k*} \quad \forall k \in S, \tag{7}$$

where U_k for $k \in S$ are independent and identical random variables distributed exponentially with mean μ. The data (x^k, y^k) $\forall k \in S$ are used for various analyses in this section.

The *SDEA*, *SDEAwD*, *HDEA* and *FastDEA* are coded and executed using Matlab on the same machine. This study sets α=0.5 in *FastDEA*, where $N(k,\alpha)$ contains roughly 25% (0.5^2) of points of $S\backslash D$, and α is set at 0.7, 0.9 and 1 to obtain a larger $N(k,\alpha)$ if necessary. In *Categorization*, a base point is randomly selected from unclassified DMUs in $S\backslash D$. Using the same idea as described in Section 3.2, the top 10% ($\alpha = 0.1$) of the remaining data that are close to the base point are identified separately according to input and output mix aspects by Eq. (3); the intersection of the two sets defines a region. This procedure for defining a region is repeated until the remaining data size is less than an arbitrarily selected threshold, say, 10% of $S\backslash D$. Points not belonging to any region are grouped and flagged in one additional region. Both H and L are defined using $\beta = 50\%$ as the threshold. On the implementation of *HDEA*, the block sizes are 250, 150 and 100 DMUs for 5000-DMU, 3000-DMU and 1000-DMU cases, respectively. These numbers are used by Barr and Durchholz [2, p. 355]. Gauging tolerance is $r = 0.7$ and the incremental ratio is 1.5.

Table 1 lists 72 scenarios representing different combinations of density, dimension and scale. Four blocks represent different dimensions. Cases of 1000, 3000 and 5000 DMUs represent different problem scales. Density is the result of instance data randomly generated by setting parameter μ of random variable U_k in (7). Density columns list average density of 10 instances for a particular dimension and scale; the corresponding μ values are also listed next to the density. This work sets average density in certain ranges, namely, around 10%, 20%, 30%, 40%, 50% and 100%, such that comparison results will be more meaningful. Hence, the values of μ are properly selected, and may vary from case to case based on different scales. For example, average density of 31% is obtained when $\mu = 5$ in 1000-DMU cases; however, $\mu = 15$ yields 34.5% and 30% DMUs that are efficient on average for the other two problem scales. Notably, as dimension increases, the density generated by μ decreases. This phenomenon is not surprising since higher input–output dimensions increase DMU efficiency in relative comparisons.

Ten instances are generated for each scenario. The same data set is tested for different methods under that particular scenario. Table 1 lists the average of 10 instances for average LP problem scale and total LP problems solved with respect to *HDEA* and *FastDEA*. Additionally, Table 1 lists the average number of dominated DMUs. Average LP size increases as dimension, density and/or scale increase (Table 1). *FastDEA* has a slightly smaller average LP size than *HDEA* under most scenarios. Total number of LP problems solved grows in response to increases in dimension and scale for both *FastDEA* and *HDEA*. However, as density increases, *FastDEA* solves fewer LP problems than *HDEA*.

Notably, results in Table 1 demonstrate that the total number of LP problems solved in *HDEA* is around three times the number of DMUs in high-density cases. This is because inefficient DMUs are seldom found in early stages; thus, LP size cannot be reduced to offset the additional effort associated with inefficient DMU identification.

Table 1
Average size and total number of LP problems.

DMUs	μ	Efficient density (%)	Avg. no. of dominated DMUs	Avg. LP size		Avg. Tol. LP solved	
				HDEA	FastDEA	HDEA	FastDEA
3 inputs 3 outputs							
1000	0.1	6	856	54	49	1160	1544
	1	15	615	131	485	1672	2118
	5	31	267	234	217	2677	1971
	10	41	123	276	172	2724	1796
	15	48	69	306	292	2888	1673
	–	100	0	414	330	3000	1257
3000	1	9.5	2087	251	179	4655	6257
	5	21	1048	472	452	6673	5623
	15	34.5	320	710	701	8401	4639
	25	49	61	921	787	9262	4057
	55	58	18	1019	796	9461	3808
	–	100	0	1121	882	9000	3416
5000	1	8	3640	354	234	7183	9934
	5	17	1930	659	637	10 439	8981
	15	30	616	1020	1043	13 368	7339
	35	43	124	1386	1208	15 031	6451
	55	52	38	1581	1244	15 539	6123
	–	100	0	1869	1430	15 000	5544
3 inputs 6 outputs							
1000	0.1	7	803	74	56	1347	2218
	0.5	17	572	146	585	1768	2278
	1	25	428	194	172	2045	2261
	2	35	269	248	247	2441	2247
	4	48	129	303	320	2798	2130
	–	100	0	414	377	3000	1541
3000	0.5	12	1967	303	237	4846	6850
	1	18	1549	418	375	5723	6857
	3	32	746	656	696	7467	6553
	5	41	426	788	861	8329	6235
	9	53	175	946	989	9018	5753
	–	100	0	1121	1047	9000	4260
5000	0.5	10	3439	432	326	7653	11 050
	2	20	1882	724	845	10 349	11 150
	4	33	1028	1094	1213	12 717	10 489
	7	43	507	1341	1470	14 189	9731
	11	53	238	1541	1603	15 042	9069
	–	100	0	1869	1673	15 000	6750
6 inputs 3 outputs							
1000	0.01	12	573	128	113	1825	3133
	0.5	25.5	386	220	921	2389	2454
	1	28	301	226	228	2446	2374
	2	39	206	298	277	2799	2279
	4	49.5	111	341	327	3021	2106
	–	100	0	414	366	3000	1420
3000	0.1	10	1953	305	236	5181	8064
	1	22	1234	564	486	6949	7148
	3	30.5	660	686	734	7964	6596
	5	41	411	889	854	8923	6164
	9	51	179	1003	963	9394	5544
	–	100	0	1121	1022	9000	3979
5000	0.1	8	3570	390	289	7913	13 229
	1	18	2353	788	663	10 724	11 756
	3	29	1302	1137	1074	13 205	10 837
	7	41.5	516	1450	1415	14 878	9419
	13	53	177	1662	1559	15 701	8357
	–	100	0	1869	1632	15 000	6333
6 inputs 6 outputs							
1000	0.01	12	588	126	109	1827	3145
	0.1	17	483	163	153	1978	2631
	0.5	32	292	262	241	2562	2509
	1	41	188	313	298	2782	2435
	2	52.5	95	353	357	3052	2331
	–	100	0	414	408	3000	1689
3000	0.01	7	2114	210	176	4785	9634
	0.3	18	1414	489	433	6532	7794
	1	31	786	737	714	8015	7313
	2	41.5	438	891	900	8905	6912
	3.5	52	223	1012	1033	9333	6495
	–	100	0	1121	1126	9000	4622

Table 1 (*Continued*)

DMUs	μ	Efficient density (%)	Avg. no. of dominated DMUs	Avg. LP size		Avg. Tol. LP solved	
				HDEA	FastDEA	HDEA	FastDEA
5000	0.1	10	3307	463	377	8530	13 674
	0.5	19	2203	828	755	11 132	12 545
	1.5	32	1111	1249	1247	13 742	11 781
	2.5	41	678	1456	1489	14 770	11 219
	5	55	269	1712	1744	15 617	10 208
	–	100	0	1869	1830	15 000	7461

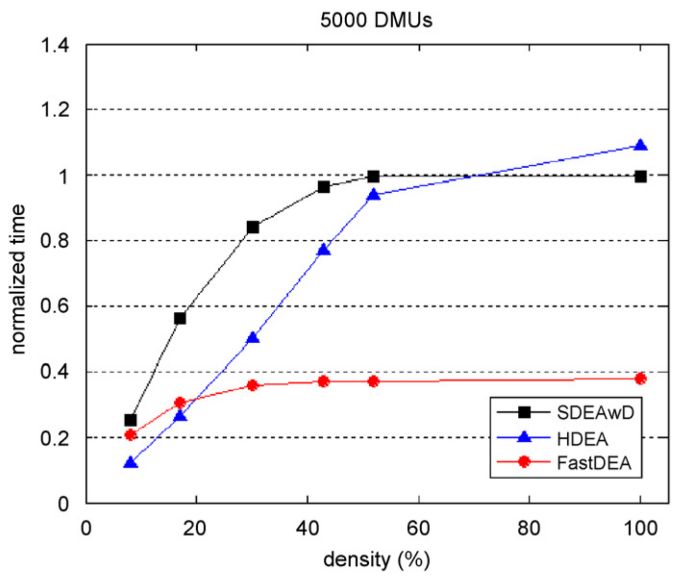

Fig. 4. Normalized average computational time for cases with 3 inputs and 3 outputs.

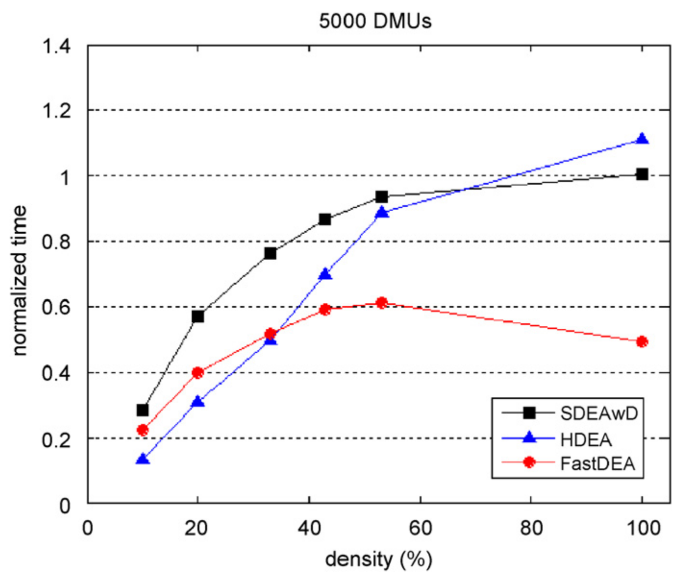

Fig. 5. Normalized average computational time for cases with 3 inputs and 6 outputs.

For cases with 1000 DMUs and 100% density, 100 LP problems are solved in the first iteration for each 10 blocks; however, no inefficient DMUs are flagged. According to the pseudocode in Barr and Durchholz [2, p. 353], block size increases to 150 and 1000 LP problems are solved during the second iteration. Due to no significant improvement, 1000 DMUs are pooled in one block in the terminating iteration. The observation demonstrates the shortcomings of *HDEA* when handling high-density cases.

4.2. Effectiveness

Fig. 4 displays the comparative relationship between average computational time and case density for large-scale cases with 5000 DMUs each of which has 3 inputs and 3 outputs. The x-axis represents the average density of 10 instances, and the y-axis is the normalized computational time with respect to *SDEA*. The normalized average time takes *SDEA* time in a particular scenario (density, dimension and scale) as the benchmark (time = 1). Squares, triangles and circles in Fig. 4 represent normalized average time for *SDEAwD*, *HDEA*, and *FastDEA*, respectively.

As shown in Fig. 4, *FastDEA* generally reduce computational time when compared with that of *SDEA*, and saves >60% for all densities. *FastDEA* has more significant time saving in low-density cases; it takes only 20% and 30% of *SDEA* time in densities 8% and 18%, respectively. As density increases, the normalized time slightly increases to around 0.4.

HDEA performs impressively for low-density cases. The *HDEA* has the most saving from *SDEA* at >70% for low-density cases. When density <50%, the edge of *HDEA* relative to *SDEA* diminishes linearly with the increase in density. Fig. 4 shows that *HDEA* is not significantly better (or is worse) than the *SDEA* method when efficient density is >40%. This is because *HDEA* uses additional steps to identify inefficient DMUs and, thus, computational time is reduced. However, only a few DMUs are inefficient in high-density cases; the extra computing load does not lead to significant savings. No gain is realized from flagging inefficient DMUs that consume additional time in 100% efficient cases; overall performance of *HDEA* is worse than that of the *SDEA* method.

SDEAwD reduces computational time in low-density instances, but the edge to *SDEA* diminishes faster than *HDEA* and *FastDEA* as density increases. Comparing *FastDEA* with the other two methods shown in Fig. 4, *FastDEA* outperforms *SDEAwD* for all densities; *FastDEA* has a significant edge over *SDEAwD* except density being 8%. *FastDEA* outperforms *HDEA* when density >20%, the performance edge increases when density grows. In the cases where the *FastDEA* method does not perform better than *HDEA*, the margin is limited and acceptable at <0.1 normalized time.

Figs. 5–7 have identical interpretation as that in Fig. 4, but with different case dimensions. *HDEA* and *SDEAwD* reveal similar trends and observations shown in Fig. 4, but *HDEA* loses its performance edge to *SDEAwD* and *SDEA* when density >50% as dimension increases. *FastDEA* reduces computational time when compared with that of *SDEA* for all densities. *FastDEA*'s normalized time increases across densities until approximately 40% to 60% and then decreases when density is 100% (Figs. 5–7). *FastDEA* outperforms *HDEA* when density >40% for all dimension cases and in many cases is better

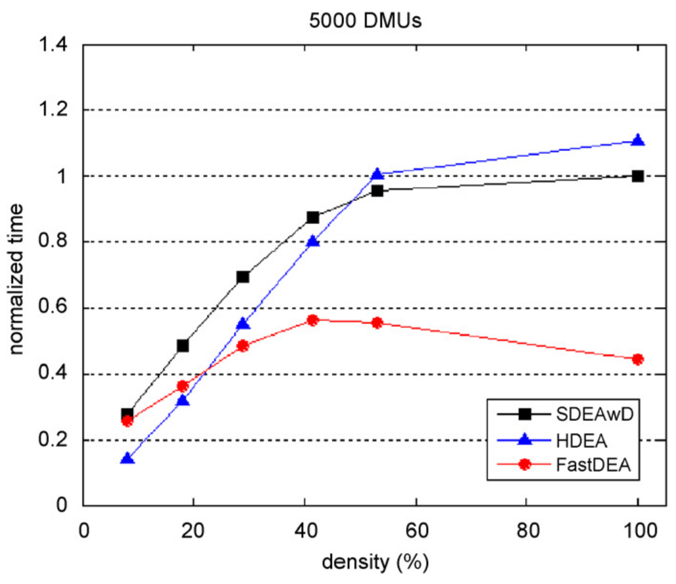

Fig. 6. Normalized average computational time for cases with 6 inputs and 3 outputs.

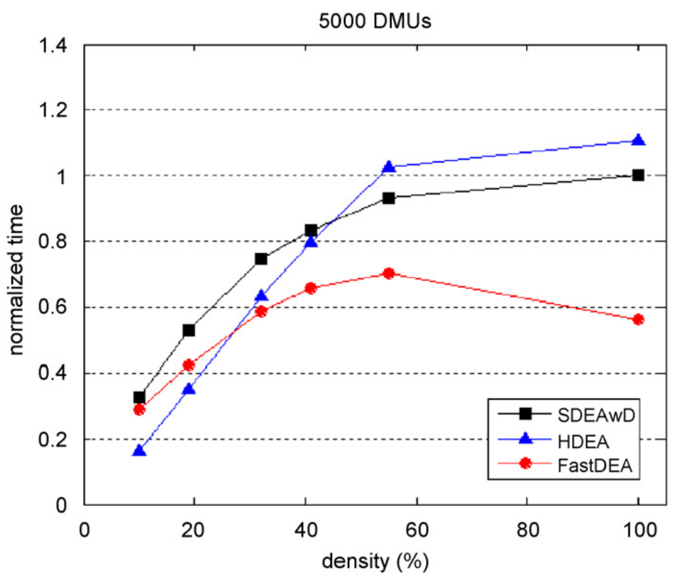

Fig. 7. Normalized average computational time for cases with 6 inputs and 6 outputs.

when density is 30%. *FastDEA* is better than *SDEAwD* for all densities (Figs. 5–7).

By comparing Figs. 4–7, *FastDEA* performs well for low dimension, the 3-input 3-output, cases. As dimension increases, performance of *FastDEA* may worsen, particularly around 50% density. The dimension effect is relatively insignificant for low density such as <20%, and the dimension has relatively serious impact for medium density around 40% to 60%. However, it outperforms the other three methods in high-density cases (>40%).

Fig. 8, similar to Fig. 4, displays the normalized time related to *SDEA* but with smaller-scale cases, namely 1000 DMUs and 3000 DMUs. Comparing Figs. 4 and 8 shows that *FastDEA* performs better, i.e., it has more significant edge to *SDEA*, for large-scale problems. For example, the normalized time at 100% density changes from 0.8 in 1000-DMU cases to 0.5 for 3000 DMUs and 0.4 for 5000 DMUs. *HDEA* has slightly shorter normalized time for the larger scales than small scales comparing to itself. *SDEAwD*'s normalized time is relatively independent on the scale in our experiments (Figs. 4 and 8). In the 1000-DMU cases, *FastDEA* does not perform well; it only outperforms *HDEA* for 100% density cases.

Figs. 9–11 have identical interpretation as that in Fig. 8, but with different case dimensions. Comparing them with Figs. 5–7 shows the impact due to scale and dimension. Again, increase in dimension limits the performance of *FastDEA*, especially for the medium density around 30% to 60%. In the small scale 1000-DMU *FastDEA* has poor performance for high dimension. *HDEA* performs well for large-scale, not for small-scale cases. *HDEA* is not impacted significantly by the dimension as *FastDEA* is. *SDEAwD* is relatively insensitive to both dimension and scale with normalized time roughly >0.8 for density >40%.

In summary, *FastDEA* significantly outperforms *SDEA*, *SDEAwD* and *HDEA* when density is high. The saving of time compared to that of *SDEA* is generally >50% for large problems. In large-scale problems, *FastDEA* is relatively insensitive on density comparing to other methods in the experiments, and its performance is superior as density increases. Scale and dimension also have an impact on *FastDEA* performance. The experiments show that *FastDEA* has more significant benefits as scale increases. However, dimension has negative impact on *FastDEA* performance. For the large-scale cases such as 3000 and 5000 DMUs, the performance advantages on *SDEA* and *SDEAwD* diminish as dimension increases. We thus conclude that *FastDEA* is suitable for solving large-scale DEA problems with high-efficiency density data, and the benefits are significant especially for low-dimension cases.

4.3. *Robustness*

Fig. 12 displays computational time for the three methods considering different dimensions. The *x*-axis represents problem sizes of 1000, 3000 and 5000 DMUs, and the *y*-axis is computational time. The three bars above each problem size represent *SDEAwD*, *HDEA* and *FastDEA*, from the left to the right, respectively. In addition to the worst (maximum) and best (minimum) computational time, the top and the bottom of the bars, respectively, average time is also marked in each bar. The computational time data are obtained and summarized across different densities (total 10 instances for each of six densities), but the detailed associations between time and density are not shown in Fig. 12. Therefore, one can interpret the range as expressed by the height of the bar for each method as a summary of the time required under different unknown data distributions.

The *FastDEA* has the best average computational time performance among three methods (Fig. 12). As expected, computational time increases as problem size increases for all three methods; however, time variation also increases. Time variation increases significantly for *SDEA-W* and *HDEA* as problem scale increases; in contrast, *FastDEA*'s variation is relatively stable for different scales. For large cases, the best case performance of *SDEA-W*, *HDEA* and *FastDEA* is impressive, particularly for 5000-DMUs cases. However, the *SDEA-W* and *HDEA* methods perform poorly under some circumstances. The empirical results reveal that although *HDEA* performs very well in some circumstances, it is not robust for all situations. Conversely, the *FastDEA* method has good and robust performance.

Computational time variation increases as case dimensions increase for all four methods. Although the time variation of the *FastDEA* method increases as problem dimension increases, it remains the best method in terms of variation. Thus, without knowing the underlying efficiency distribution, the *FastDEA* method generates a good and robust performance under all scenarios.

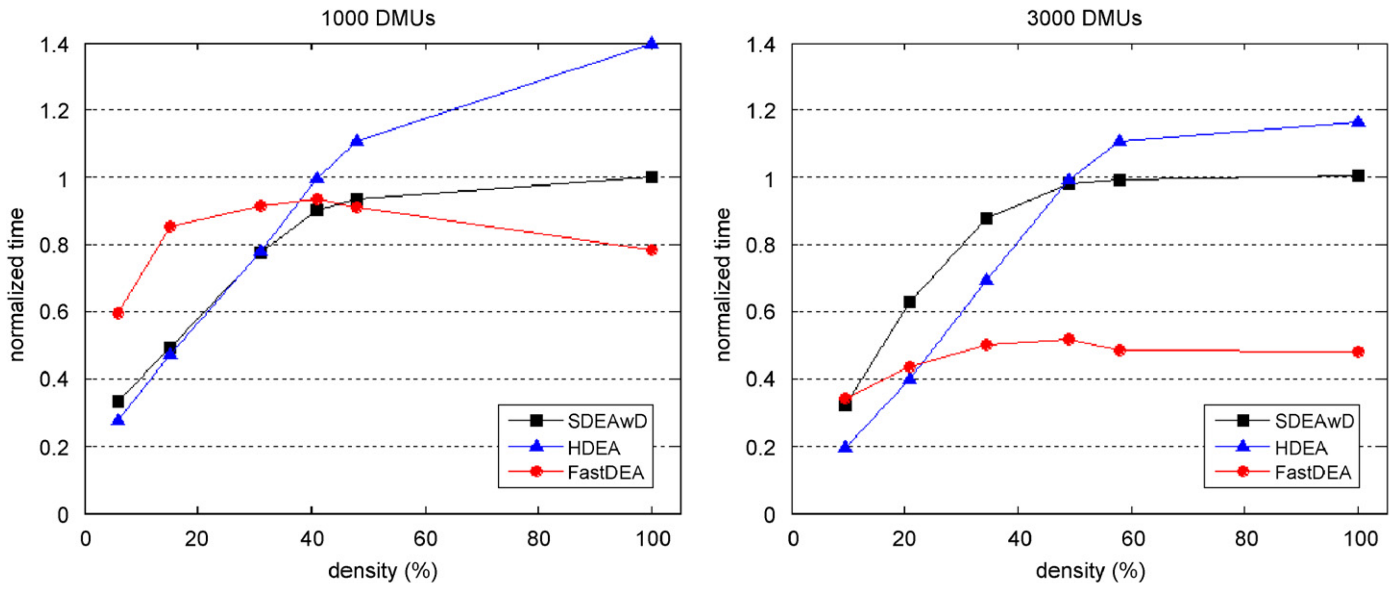

Fig. 8. Normalized average computational time for cases with 3 inputs and 3 outputs.

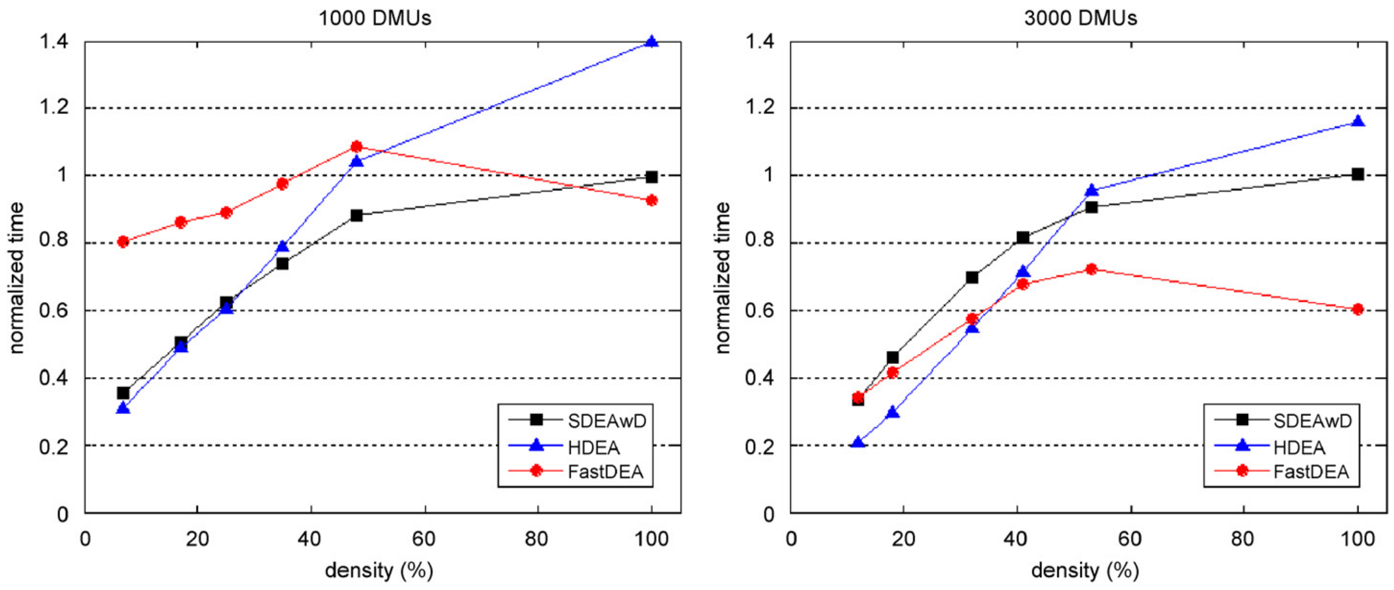

Fig. 9. Normalized average computational time for cases with 3 inputs and 6 outputs.

4.4. Sensitivity analysis

The selection of the initial tolerance parameter (α) is a trade-off between single LP problem solving time and number of iterations needed, where one iteration solves one LP problem. Higher tolerance results in fewer iterations needed since an increased number of neighbors identifies and increases likelihood of catching true references. However, a larger set of neighbors means that a large LP problem size consumes more time. Small initial tolerance, by contrast, leads to small problem size and possibly an increased number of iterations, i.e., an increased number of LP problems to solve. This section demonstrates the trade-offs using simulation cases.

This study uses 3-input 3-output and 6-input 6-output cases with 5000 DMUs as examples since these cases are large-scale and represent two extreme dimensions in our simulation studies. Four initial tolerance values, $\alpha = 0.3$, 0.5, 0.7 and 0.9, are tested, where α^2 of $S \backslash D$ are selected as neighbors for each iteration. The values of α also represent how the new parameter values are defined for the next iteration if necessary, e.g., $\alpha = 0.7$ is used if using $\alpha = 0.5$ does not meet the termination criterion. Moreover, if $\alpha = 0.9$ still cannot terminate the loop, the LP problem with $S \backslash D$ is solved ($\alpha = 1$).

Fig. 13 displays the relationship between initial tolerance and average time of 10 instances for solving one LP problem. The x-axis represents values of the initial α, and the y-axis represents average time for solving one LP problem, which is measured in 10^{-3} s. Triangles and circles represent 3-input 3-output (3×3, hereafter) and 6-input 6-output (6×6) cases, respectively. The computation results suggest that higher initial tolerance results in longer single LP problem

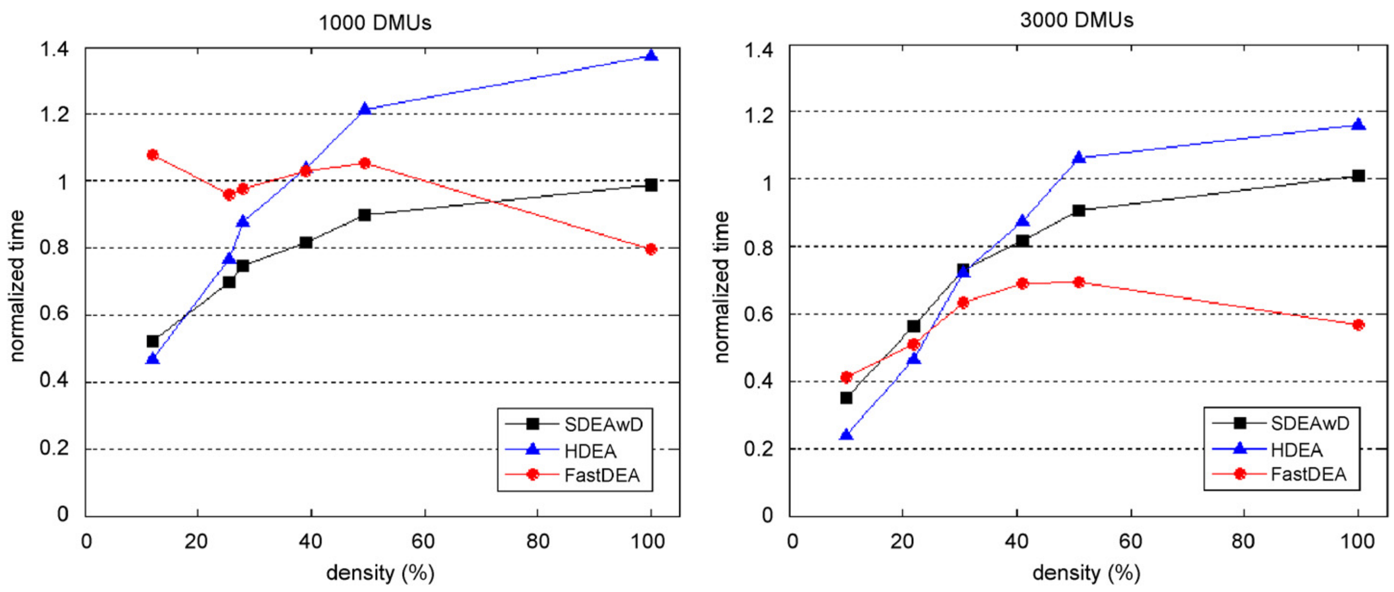

Fig. 10. Normalized average computational time for cases with 6 inputs and 3 outputs.

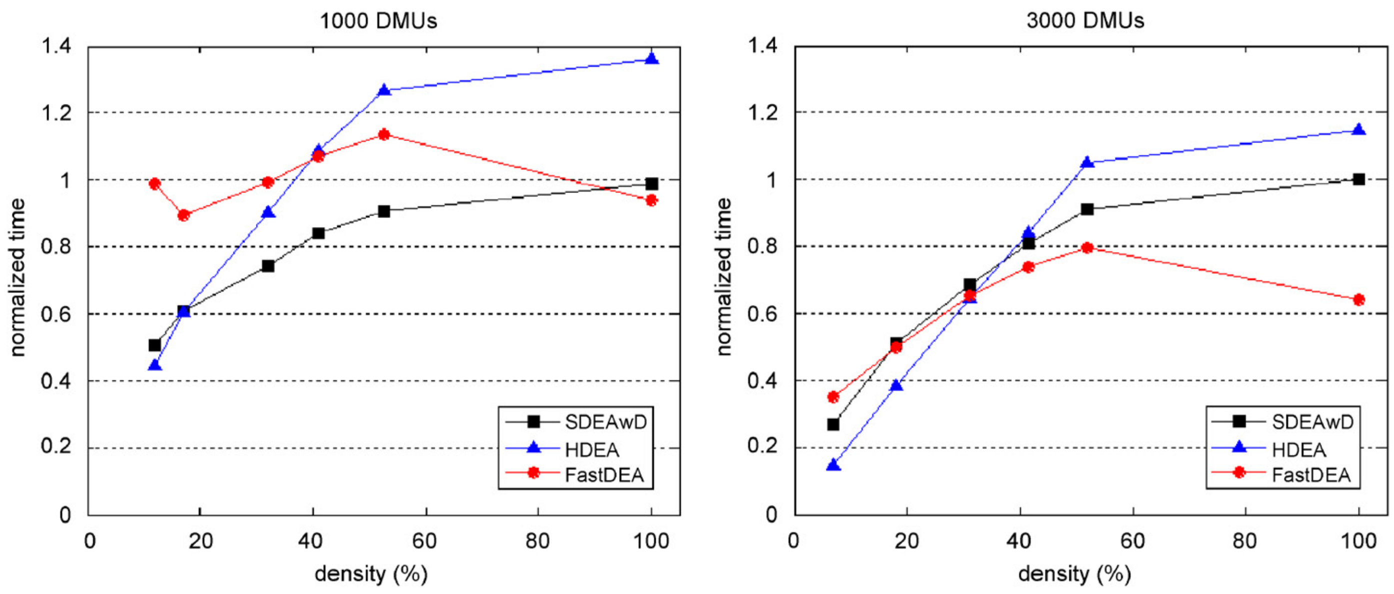

Fig. 11. Normalized average computational time for cases with 6 inputs and 6 outputs.

solving time. In addition, high-dimension (6×6) cases take longer time for single problem solving than low-dimension (3×3) cases. The differences between 3×3 and 6×6 cases are relatively similar across different α values.

Fig. 14 displays the relationship between initial tolerance (x-axis) and average number of iterations needed for solving one DMU (y-axis). Triangles and circles represent 3×3 and 6×6 cases, respectively. The computation results in the simulation show that iteration required reduces as initial α increases. Taking $3{\times}3$ cases as an example, $\alpha = 0.3$ requires solving two iterations on average for one DMU; one iteration is needed on average when $\alpha = 0.9$. Moreover, more iterations are needed for high-dimension cases than for low-dimension cases given the same initial α. The difference is particularly significant for $\alpha = 0.3$. The test results reveal that, on average, $\alpha = 0.7$ (49% of DMUs in $S \backslash D$) is needed to catch true reference for 6×6 cases irrespective of whether initial α is 0.3 or 0.5. This is because as dimension increases, it is hard to identify DMUs with similar mix.

5. Conclusions

Notably, DEA measures relative efficiency of a DMU by comparing it against a peer group. The most popular DEA models can be solved using standard LP techniques and, thus, are theoretically considered computationally easy. However, in practice, computation time increases significantly for large cases and yields challenges for many applications.

This work presents a novel accelerating procedure, *FastDEA*, for solving large DEA BCC problems according to geometric and managerial interpretations of radial DEA models. The *FastDEA* solves

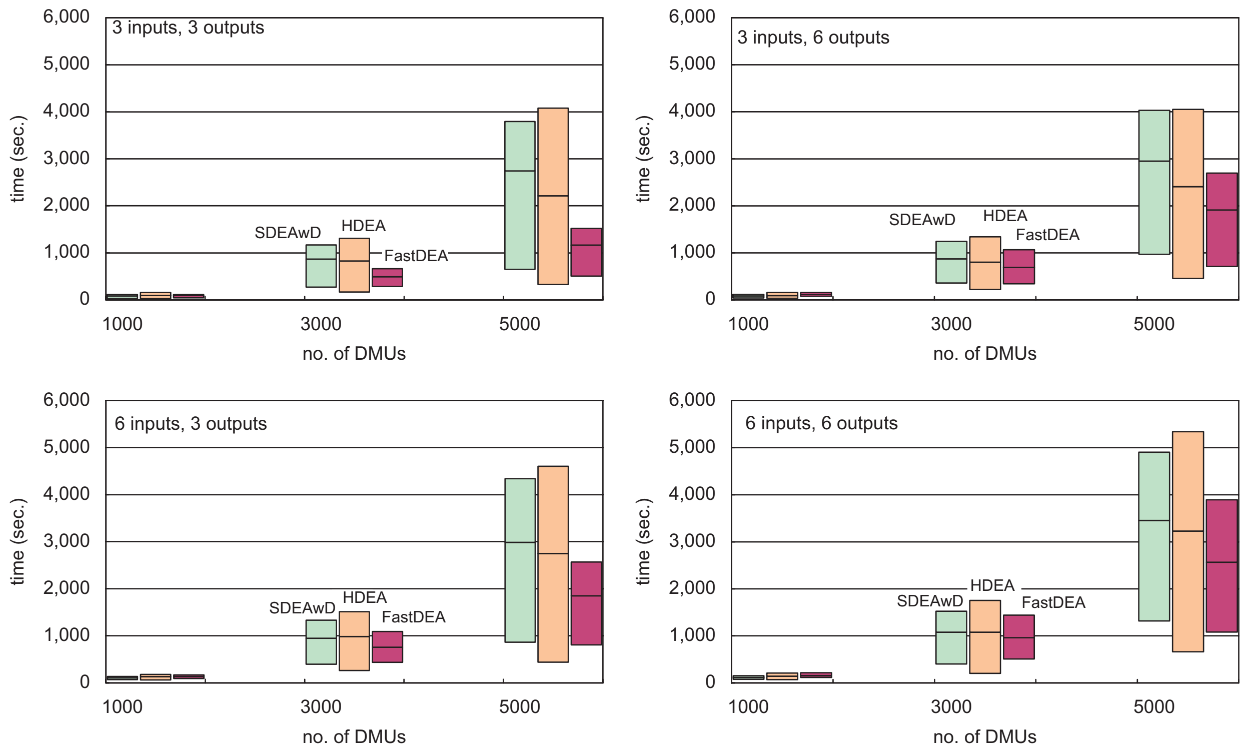

Fig. 12. Computational time for different dimensions.

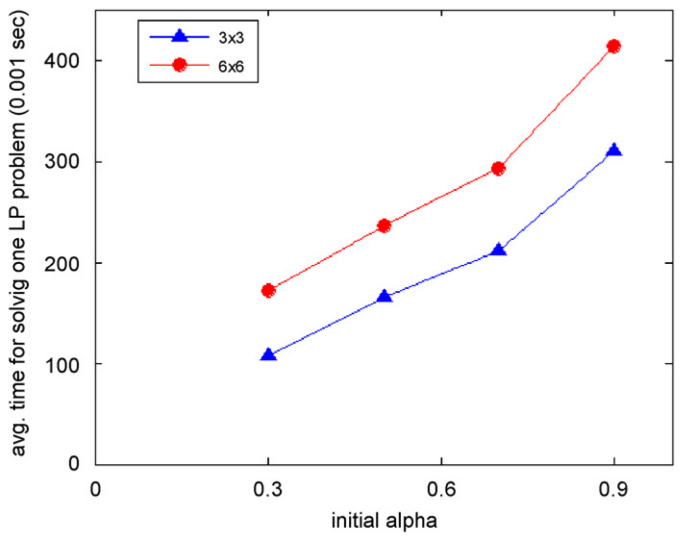

Fig. 13. Average time for solving one LP problem.

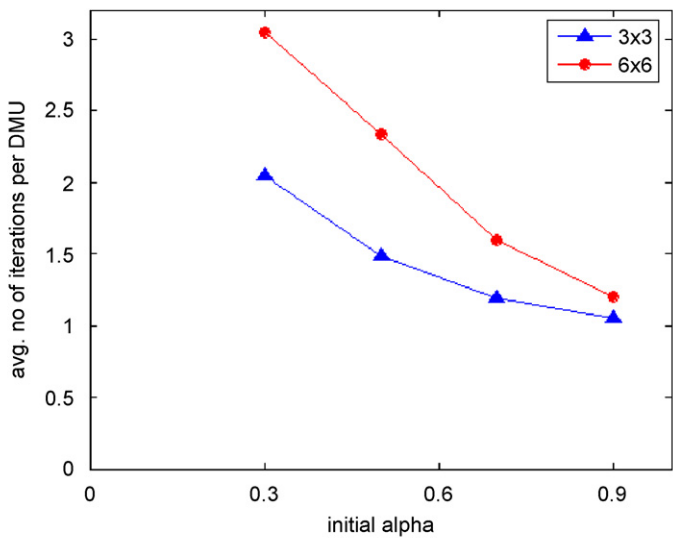

Fig. 14. Average number of iterations needed for solving one DMU.

reduced LP problems to obtain accurate efficiencies by selecting only a few DMUs. The *FastDEA* procedure is suitable for solving large-scale DEA BCC problems with high-efficiency density data. The benefits are significant especially for low-dimension cases, which is typical in practice. Simulation results demonstrate that the proposed procedure can reduce computational time by >50% for large cases. Moreover, the performance of *FastDEA* in this study can be considered as the worst bound. Additional fine-tuning processes can reduce computational time further. For example, recording all inefficient DMUs in solved problems and moving them to inefficient set D will reduce the number of neighbor candidates $S\backslash D$ significantly, thereby reducing solving time. Section 4.2 also shows the potential limitation for high-dimension cases. A better way to define the similarity or dissimilarity is also worth further investigation.

Acknowledgments

This research was supported in part by the National Science Council, Taiwan under Grant NSC 94-2213-E-009-078. The authors would like to thank the anonymous reviewers for their constructive comments and invaluable suggestions.

References

[1] Charnes A, Cooper WW, Rhodes E. Measuring the efficiency of decision making units. European Journal of Operational Research 1978;2:429–44.

[2] Barr RS, Durchholz ML. Parallel and hierarchical decomposition approaches for solving large-scale data envelopment analysis models. Annals of Operations Research 1997;73:339–72.

[3] Sampaio de Sousa MC, Stosic B. Technical efficiency of the Brazilian municipalities: correcting nonparametric frontier measurements for outliers. Journal of Productivity Analysis 2005;24:157–81.

[4] Dulá JH. Computations in DEA. Pesquisa Operacional 2002;22:165–82.

[5] Dulá JH. A computational study of DEA with massive data sets. Computers & Operations Research 2008;35:1191–203.

[6] Ali AI. Streamlined computation for data envelopment analysis. European Journal of Operational Research 1993;64:61–7.

[7] Ali A.I. Computational aspects of DEA. In: Charnes A, Cooper WW, Lewin A, Seiford LM, editors. Data envelopment analysis, theory, methodology and applications. 1994. p. 63–88.

[8] Chen Y, Ali AI. Output–input ratio analysis and DEA frontier. European Journal of Operational Research 2002;142:476–9.

[9] Dulá JH, Venugopal N. On characterizing the production possibility set for the CCR ratio model in DEA. International Journal of Systems Science 1995;26: 2319–25.

[10] Dulá JH, Thrall RM. A computational framework for accelerating DEA. Journal of Productivity Analysis 2001;16:63–78.

[11] Dulá JH, Helgason RV. A new procedure for identifying the frame of the convex hull of a finite collection of points in multidimensional space. European Journal of Operational Research 1996;92:352–67.

[12] Dulá JH, Helgason RV, Venugopal N. An algorithm for identifying the frame of a pointed finite conical hull. INFORMS Journal on Computing 1998;10:323–30.

[13] Banker RD, Charnes A, Cooper WW. Some models for estimating technical and scale inefficiencies in data envelopment analysis. Management Science 1984;30:1078–92.

[14] Ray SC, Mukherjee K. Decomposition of the Fisher ideal index of productivity: a non-parametric dual analysis of US airlines data. The Economic Journal 1996;106:1659–78.

[15] Kendall MG. A course in the geometry of *n* dimensions. New York: Hafner Pub. Co.; 1961.

[16] Fox KJ, Hill RJ, Diewert WE. Identifying outliers in multi-output models. Journal of Productivity Analysis 2004;22:73–94.

[17] Sueyoshi T, Chang Y-L. Efficient algorithm for additive and multiplicative models in data envelopment analysis. Operations Research Letters 1989;8(4):205–13.

[18] Wilson PW. FEAR: a package for frontier efficiency analysis with *R*, unpublished working paper, Department of Economics, University of Texas at Austin, Austin, Texas, 2005.