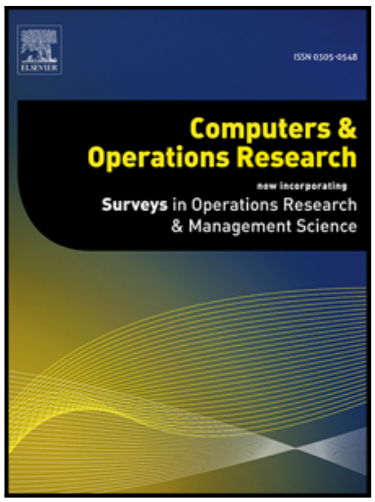

Efficiency evaluation based on data envelopment analysis in the big data context

Qingyuan Zhu , Jie Wu , Malin Song

PII: S0305-0548(17)30155-7
DOI: 10.1016/j.cor.2017.06.017
Reference: CAOR 4272

To appear in: *Computers and Operations Research*

Received date: 6 August 2016
Revised date: 19 June 2017
Accepted date: 21 June 2017

Please cite this article as: Qingyuan Zhu , Jie Wu , Malin Song , Efficiency evaluation based on data envelopment analysis in the big data context, *Computers and Operations Research* (2017), doi: 10.1016/j.cor.2017.06.017

Highlights

- Novel algorithms are proposed to accelerate the computation process in the big data environment.
- An easy algorithm is developed to divide the large scale DMUs into small scale and identify all strongly efficient DMUs.
- We only need to select two reference points as the sample in the situation of just one input and one output.
- A variant of the algorithm is then presented to handle cases with multiple inputs or multiple outputs.

Efficiency evaluation based on data envelopment analysis in the big data context

Qingyuan Zhu[a b]; Jie Wu[a]; Malin Song[c*]

a. School of Management, University of Science and Technology of China, Hefei, Anhui Province, 230026, P. R. China

b Department of Business Administration, University of Illinois at Urbana-Champaign, Champaign, Illinois, 61822, USA

c. School of Mathematics and Finance, Chuzhou University, Chuzhou, 239000, P. R. China

Abstract

Data envelopment analysis (DEA) is a self-evaluation method which assesses the relative efficiency of a particular decision making unit (DMU) within a group of DMUs. It has been widely applied in real-world scenarios, and traditional DEA models with a limited number of variables and linear constraints can be computed easily. However, DEA using big data involves huge numbers of DMUs, which may increase the computational load to beyond what is practical with traditional DEA methods. In this paper, we propose novel algorithms to accelerate the computation process in the big data environment. Specifically, we firstly use an algorithm to divide the large scale DMUs into small scale and identify all strongly efficient DMUs. If the strongly efficient DMU set is not too large, we can use the efficient DMUs as a sample set to evaluate the efficiency of inefficient DMUs. Otherwise, we can identify two reference points as the sample in the situation of just one input and one output. Furthermore, a variant of the algorithm is presented to handle cases with multiple inputs or multiple outputs, in which some of the strongly efficient DMUs are reselected as a reduced-size sample set to precisely measure the efficiency of inefficient DMUs. Last, we test the proposed methods on simulated data in various scenarios.

Keywords: Data envelopment analysis; Decision making unit; Large-scale computation; Big data.

* Corresponding author. E-mail: Ma-Lin Song, songmartin@163.com

1. Introduction

Data envelopment analysis (DEA), developed by Charnes et al. [10], is a non-parametric mathematical method used to measure relative efficiency within a group of homogenous decision making units (DMUs), particularly a group with multiple inputs and multiple outputs (see, e.g., [6, 14, 26, 31]). As a nonparametric technique, DEA is not limited by any functional form, and does not require the numerous assumptions that arise from the use of statistical methods for function estimation and efficiency measurement, yet it can evaluate efficiency well (see, e.g., [27, 32, 3, 24]). To date, DEA has been extensively applied in the performance evaluation of hospitals (see, [23, 15]), universities (see, [25, 21]), banks (see, [29, 30]), supply chains (see, [5]), and in many other situations (see, e.g., [19, 28, 33, 35]).

DEA measures relative efficiency for a DMU against its peer *n-1* DMUs, supposing there are *n* DMUs in the evaluation system. Traditional DEA models require the solution of a linear programming problem with *n+1* variables and *m+s* constraints, where *m* and *s* are the numbers of inputs and outputs, respectively. The traditional DEA models can be solved by using standard linear programming techniques, thus, are theoretically considered computationally easy. However, in practice, the solution time increases significantly for large cases [12]. Emerging in 1980s, the concept of "big data" has become a hot issue in the computer industry and financial businesses. Wu et al. [34] indicated that big data has rapidly expanded in all science and engineering domains, including physical, biological and biomedical sciences. The emergence of the big data paradigm over the past few years, with its five major features (volume, velocity, variety, veracity, and valorization), has created a new set of problems and challenges [36, 8]. For example, Chen et al. [11] highlighted that we can obtain new science, discovery, and insights from the overwhelming amount of web-based, mobile, and sensor-generated data arriving at a terabyte and even exabyte scale. Michael and Miller [22] indicated that the development of big data could bring comprehensive analyses to support the development of improved policy regime-based systems. In the DEA field, big data has

brought many problems for researchers. For example, the larger scale number of DMUs in the big data context is the biggest issue since it may take an impractical amount of time to finish the evaluation of all DMU efficiencies. Therefore, methods for reducing solution time for DEA problems are practically beneficial, especially in the big data environment.

The literature on DEA computation for larger scale number of DMUs is mainly based on the idea of reducing the size of individual linear programming models [1, 2], to be precise, through reducing the number of variables or DMUs. Barr and Durchholz [7] proposed a new problem decomposition procedure that dramatically expedites the solution of these computationally intense problems and fully exploits parallel processing environments. Dulá and Thrall [16] introduced a new computational framework for DEA that reduces computation times and increases flexibility in application over multiple models and orientations. The process is based on the minimal subsets of the data needed to describe the models. Korhonen and Siitari [20] also proposed lexicographic parametric programming to decompose the dimensions to reduce the computational costs when identifying the efficient DMUs. Dulá and López [17] collect, organize, analyze, implement, test, and compare a comprehensive list of ideas for preprocessors for entity classification in DEA. The technique of preprocessing in DEA provides tools that will reduce the computational burden, especially in large scale applications. Dulá [18] presented an algorithm for DEA based on a two-phase procedure. The first phase identifies the efficient DMUs. Those efficient DMUs are then used as a sample set in a second phase to score the rest of the inefficient DMUs. Chen and Cho [12] proposed an accelerating procedure that properly identifies a few "similar" critical DMUs to compute DMUs' efficiency values. Chen and Lai [13] proposed a new algorithm for determining radial efficiency scores with a large data set by using small-size linear programs. Instead of trying to reduce the number of variables in individual linear programs, their proposed algorithm tries to repeated select some variables for controlling the variables of individual linear programs.

Surveying the existing studies on DEA based on larger data sets, we find that

they are either based on finding all of the efficient DMUs that are the possible benchmarks for scoring or based on selecting a sample as the virtual conference points for all DMUs. Firstly, they use the traditional method to identify all efficient DMUs and hence it may still take lots of time. In addition, the method of selecting a sample as the virtual conference can still be complex since it does not choose the sample from efficient DMUs. In this paper, we firstly propose an easy algorithm by dividing all DMUs into groups to identify all strongly efficient DMUs in order to further reduce the larger scale computational burdens of assessing all DMUs. Then considering the scale of strongly efficient DMUs (i.e. the size of the set of such DMUs), we propose two methods to measure the efficiency values of the remaining inefficient DMUs. More specifically, if the number of strongly efficient DMUs is not too large, then we can use those identified strongly efficient DMUs as the sample set for efficiency evaluation. However, if the number of strongly efficient DMUs is also impractically large, we consider how to select as few strongly efficient DMUs as possible while maintaining sufficient accuracy in the efficiency evaluation. Two cases are considered. One is the case with a single input and single output and the other is the case with multiple inputs and/or multiple outputs. In the single input and output case, we propose an easy method to find only two reference points to further accelerate the computational process for all inefficient DMUs. In the case of multiple inputs and/or multiple outputs, following Chen and Lai [13] we propose an algorithm to reselect a small sample from among the strongly efficient DMUs to evaluate the efficiency values. Last, the proposed methods are tested for effectiveness through simulated data in various scenarios.

The rest of this paper is organized as follows. In Section 2, we introduce the traditional DEA models. Section 3 then proposes an algorithm to identify the strongly efficient DMUs. In Section 4, we further propose methods to accelerate the computations in the situations of one-input-one-output and multiple inputs or multiple outputs. Section 5 demonstrates the effectiveness of the proposed methods. Finally, conclusions are given in Section 6.

2. Data envelopment analysis

Suppose that we have *n* DMUs and each of them consumes varying amounts of *m* different inputs to produce *s* different outputs. Specifically, DMU_j ($j=1,\cdots,n$) consumes amount x_{ij} of input *i* to produce amount y_{rj} of output *r*. Define $x_j=(x_{1j},x_{2j},\cdots,x_{mj})^T$ and $y_j=(y_{1j},y_{2j},\cdots,y_{sj})^T$ as the respective input and output vectors of DMU_j. A popular DEA model for evaluating the efficiency value is the input-oriented and output-oriented BCC (Banker-Charnes-Cooper) models proposed by Banker et al. [6], shown as the following models (1) and (2).

$$
\begin{aligned}
E_0^{Input} = Max\ & \frac{\sum_{r=1}^{s} u_r y_{r0} + u_0}{\sum_{i=1}^{m} v_i x_{i0}} \\
s.t.\ & \frac{\sum_{r=1}^{s} u_r y_{rj} + u_0}{\sum_{i=1}^{m} v_i^k x_{ij}} \le 1, \quad j=1,\cdots,n; \\
& u_r \ge 0, \qquad r=1,\cdots,s; \\
& v_i \ge 0, \qquad i=1,\cdots,m; \\
& u_0 \ \textit{free in sign.}
\end{aligned}
\tag{1}
$$

and

$$
\begin{aligned}
E_0^{Output} = Max\ & \frac{\sum_{i=1}^{m} v_i x_{i0} + u_0}{\sum_{r=1}^{s} u_r y_{r0}} \\
s.t.\ & \frac{\sum_{i=1}^{m} v_i x_{ij} + u_0}{\sum_{r=1}^{s} u_r y_{rj}} \le 1, \quad j=1,\cdots,n; \\
& u_r \ge 0, \qquad r=1,\cdots,s; \\
& v_i \ge 0, \qquad i=1,\cdots,m; \\
& u_0 \ \textit{free in sign.}
\end{aligned}
\tag{2}
$$

In models (1) and (2), the subscript 0 denotes the DMU under evaluation. Values u_r and v_i are the input and output multipliers/weights, respectively. In models (1) and (2), each DMU_0 ($0=1,\cdots,n$) chooses its own optimal weights u_r^* and v_i^* to maximize its BCC efficiency while maintaining all DMU efficiencies at no more than

1. The variable u_0 added in the numerator of models (1) and (2) reflects the variable returns to scale (VRS) assumption. If we set $u_0 = 0$, models (1) and (2) become the traditional CCR (Charnes-Cooper-Rhodes) model (see, [10]).

Models (1) and (2) are non-linear programs which can be transformed into the following linear models (3) and (4) via the Charnes-Cooper transformation (see, [9]).

$$
\begin{aligned}
& E_0^{Input} = Max \ \sum_{r=1}^{s} u_r y_{r0} + u_0 \\
& s.t. \quad \sum_{r=1}^{s} u_r y_{rj} - \sum_{i=1}^{m} v_i x_{ij} + u_0 \le 0, \quad j = 1, \cdots, n; \\
& \qquad \sum_{i=1}^{m} v_i x_{i0} = 1, \\
& \qquad u_r \ge 0, \qquad\qquad r = 1, \cdots, s; \\
& \qquad v_i \ge 0, \qquad\qquad i = 1, \cdots, m; \\
& \qquad u_0 \ \textit{free in sign.}
\end{aligned}
\tag{3}
$$

and

$$
\begin{aligned}
& E_0^{Output} = Max \ \sum_{i=1}^{m} v_i x_{io} + u_0 \\
& s.t. \quad \sum_{i=1}^{m} v_i x_{ij} - \sum_{r=1}^{s} u_r y_{rj} + u_0 \le 0, \quad j = 1, \cdots, n; \\
& \qquad \sum_{r=1}^{s} u_r y_{r0} = 1, \\
& \qquad u_r \ge 0, \qquad\qquad r = 1, \cdots, s; \\
& \qquad v_i \ge 0, \qquad\qquad i = 1, \cdots, m; \\
& \qquad u_0 \ \textit{free in sign.}
\end{aligned}
\tag{4}
$$

The following models (5) and (6) considering slack variables are equivalent to the dual of models (3) and (4), respectively.

$$
\begin{aligned}
E_0^{Input} = Min \quad & \theta_0 - \varepsilon(\sum_{i=1}^{m} s_i^- + \sum_{r=1}^{s} s_r^+) \\
s.t. \quad & \sum_{j=1}^{n} \lambda_j x_{ij} + s_i^- = \theta_0 x_{io} \qquad i = 1,\cdots,m; \\
& \sum_{j=1}^{n} \lambda_j y_{rj} - s_r^+ = y_{ro} \qquad r = 1,\cdots,s; \\
& \sum_{j=1}^{n} \lambda_j = 1 \\
& \lambda_j, s_i^-, s_r^+ \geq 0,\ \varepsilon > 0 \qquad \forall\ j,i,r.
\end{aligned} \tag{5}
$$

and

$$
\begin{aligned}
E_0^{Output} = Max \quad & \theta_0 + \varepsilon(\sum_{i=1}^{m} s_i^- + \sum_{r=1}^{s} s_r^+) \\
s.t. \quad & \sum_{j=1}^{n} \lambda_j x_{ij} + s_i^- = x_{io} \qquad i = 1,\cdots,m; \\
& \sum_{j=1}^{n} \lambda_j y_{rj} - s_r^+ = \theta_0 y_{ro} \qquad r = 1,\cdots,s; \\
& \sum_{j=1}^{n} \lambda_j = 1 \\
& \lambda_j, s_i^-, s_r^+ \geq 0,\ \varepsilon > 0 \qquad \forall\ j,i,r.
\end{aligned} \tag{6}
$$

where ε is a so-called non-Archimedean element defined to be smaller than any positive real number. The s_i^- and s_r^+ are slack variables used to bring inefficient DMUs to efficiency. From a different point of view of models (1) and (2), model (5)/(6) attempts to proportionately minimize/maximize usage of $x_{i0}, (i = 1,\cdots,m)$ / $y_{r0}, (r = 1,\cdots,s)$ by θ_0 while maintaining at least the same output/input level. In the output-oriented BCC model (6), benchmarking information can be obtained for eliminating inefficiency, through equal and proportionate output expansion, while keeping the input fixed at the current level. An optimal benchmarking point $(\sum_{j=1}^{n} \lambda_j^* x_{ij} = x_{io} - s_i^{-*}, \sum_{j=1}^{n} \lambda_j^* y_{rj} = \theta_0^* y_{ro} + s_r^{+*})$ should be within the production possibility set (PPS) constructed by all DMUs, where the superscript $*$

denotes an optimal value of model (6). That is, $(\sum_{j=1}^{n}\lambda_j^* x_{ij}, \sum_{j=1}^{n}\lambda_j^* y_{rj})$ represents the "virtual DMU" composed of the peers for DMU_0. DMU_j ($\lambda_j^* \neq 0, j=1,\cdots,n$) is called the reference point, or simply the reference of DMU_0. In addition, these DMUs with $\lambda_j^* \neq 0, j=1,\cdots,n$ are strongly efficient. The optimal solution $\lambda_j^* = 0, j=1,\cdots,n$ indicates that DMU_j does not contribute to the virtual DMU and cannot affect the efficiency value for DMU_0.

Definition 1 (Strongly DEA efficient). The performance of DMU_0 is strongly/fully efficient if and only if both (1) $\theta_0^* = 1$ and (2) all slacks $s_i^{-*} = s_r^{+*} = 0$.

Definition 2 (Weakly DEA efficient). The performance of DMU_0 is weakly efficient if and only if both (1) $\theta_0^* = 1$ and (2) all slacks $s_i^{-*} \neq 0$ and/or $s_r^{+*} \neq 0$ for some *i* or *r* in some alternate optima.

3. Algorithms for accelerating the evaluation procedure

The concept of "big data", emerging in the 1980s, has become a hot topic in many industries and has seen quick development in recent years. As a major characteristic of big data, "volume" has become a major challenge for efficiency evaluation. For example, the huge number of DMUs has brought a big problem in calculating. That is, traditional software cannot easily calculate each DMU's efficiency; it may take too much time.

When calculating BCC efficiency values using models (5) and (6), we need to solve many different linear programming problems when the number of DMUs is extremely large. Actually, models (5) and (6) both have $n+s+m+1$ variables and $s+m+1$ constraints, typically with $n+s+m+1 \gg s+m+1$ in the big data environment. Having more variables must increase the burden of calculation. In addition, when there is a limit on the linear programming problems, e.g., the number of variables or constraints cannot be more than a fixed number, say ψ, the efficiency

values may be impossible to obtain. Therefore, we must reduce the number of variables or constraints to reduce the burdens or satisfy other requirements. We firstly propose the following Algorithm 1 to address these issues.

Algorithm 1.

Step I: Set $k=1$. Denote the set of all the DMUs as $J_1=\{j \mid j=1,\cdots,n\}$. Then, equally divide the huge *n* DMUs into *l*+*k* groups. The set of DMUs in each group are denoted by $J_{11}, J_{12},\cdots, J_{1l+k-1}, J_{1l+k}$. Note that the number of DMUs in each group cannot be bigger than the limit number ψ.

Step II: Set $k=k+1$. Calculate the "virtual" efficiency values for each group of DMUs. Denote the strongly efficient DMU sets in each group as $E_1^k, E_2^k,\cdots, E_{l+k-1}^k, E_{l+k}^k$. Let $J_k=E_1^k \bigcup E_2^k \cdots \bigcup E_{l+k-1}^k \bigcup E_{l+k}^k$.

Step III: If $J_{k-1}=J_k$, then the procedure stops, and we have found all the virtual strongly efficient DMUs, else go to Step I again.

In Algorithm 1, we firstly divide all DMUs into disjoint groups. Then we calculate the virtual efficiency values for each group's DMUs. Through repeatedly calculating each group of DMU's virtual efficiency values we can obtain all "virtual" efficient DMUs in set J_1. Obviously, all actual strongly efficient DMUs are all also in set J_{k-1} or J_k and the DMUs in set J_k must be reduced largely compared to J_1. Last, we can calculate the small sample DMUs' efficiency values in set J_k to identify those actual strongly efficient DMUs. Denote the strongly efficient DMUs as the set J.

Through dividing all DMUs into different groups for calculation, the number of variables was largely decreased which accelerates the calculation process. Note that, when calculating the DMUs' virtual efficiency values in each of the *l*+*k* groups, we can use *l*+*k* computers in parallel to further save time. For example, suppose the numbers of DMUs is 20000 and suppose there are three inputs and three outputs, e.g., $n=20000$, $m=3$, and $s=3$. Finding the strongly efficient DMUs takes about forty

minutes using the traditional BCC model solved by Matlab R2014a, a commercial mathematical software package sold by MathWorks, on an Intel Core i3 CPU with 8GB memory and a Windows 10 operating system. However, if the 20000 DMUs are divided into 20 groups, then each group with 1000 DMUs will take about 1 minute and therefore about twenty minutes for the total 20 groups. If the 20 group DMUs are calculated in parallel on 20 computers, then only 1 minute is needed. Although it may take several rounds to stop the algorithm, this still could save significant time. Therefore, our proposed Algorithm 1 is likely to needs little time to identify all efficient DMUs in a large set of DMUs compared to the traditional BCC model.

Although, our Algorithm 1 can effectively obtain all efficient DMUs in a short time, we also need to calculate the efficiency values for the remaining inefficient DMUs. One possible solution is to use all strongly efficient DMUs in set J as a sample to calculate the efficiency values of inefficient DMUs. Taking the output-oriented BCC model as an example, we show the proposed model as follows.

$$
\begin{aligned}
E_0^{Output} = Max \quad & \theta_0 + \varepsilon(\sum_{i=1}^{m} s_i^- + \sum_{r=1}^{s} s_r^+) \\
s.t. \quad & \sum_{j\in J} \lambda_j x_{ij} + s_i^- = x_{io} \qquad i = 1,\cdots,m; \\
& \sum_{j\in J}^{n} \lambda_j y_{rj} - s_r^+ = \theta_0 y_{ro} \qquad r = 1,\cdots,s; \\
& \sum_{j\in J}^{n} \lambda_j = 1 \\
& \lambda_j, s_i^-, s_r^+ \geq 0,\ \varepsilon > 0 \qquad \forall\ j \in J, i, r.
\end{aligned}
\tag{7}
$$

The "virtual DMU" composed of the peers for inefficient DMU_0 is formed by those strongly efficient DMUs with $\lambda_j^* \neq 0, j = 1,\cdots,n$ in model (6). Therefore, in model (7) we just use the strongly efficient DMUs identified in Algorithm 1 as the sample DMUs to calculate the inefficient DMUs' efficiency values. Suppose that we identify N efficient DMUs in the larger set of n DMUs. Therefore, model (7) has $N+s+m+1$ variables and $s+m+1$ constraints. Obviously, we have $N \ll n$, hence model (7) can accelerate the calculation process for those inefficient DMUs.

4. Extensions

In most real-world big data cases, where intuition tells us that few DMUs compared to the larger *n* DMUs are strongly efficient, our proposed Algorithm 1 can quickly obtain the strongly efficient DMUs and model (7) can further accelerate the calculation process for inefficient DMUs. However, if the number of strongly efficient DMUs N is also large enough, that is, the number of variables $N+s+m+1$ in model (7) is relatively big, too much time may still be required to calculate the inefficient DMUs' efficiency values.

Actually, the optimal solution of the corresponding linear programming problem for the inefficient DMUs has at most $s+m$ members of $\{\lambda_j (j=1,\cdots,n)\}$ in model (7) that are possibly non-zero and the remaining $N+1$ members must be zero [12]. This fact reveals that for a simple standard DEA computation, regardless of the number of variables, at most $s+m$ DMUs are related to the efficiency values, and only the corresponding DMUs are needed for linear programming problem solving. Therefore, we want to select as few strongly DMUs as possible to construct the corresponding linear programming problem when calculating the efficiency values for inefficient DMUs. In the next two subsections, we will show two cases of how to select as few strongly efficient DMUs as possible. One is the case with only a single input and single output and the other is the case with multiple inputs or multiple outputs.

4.1 Single input and single output case

If there are just a single input and single output, at most two strongly efficient DMUs are needed to solve the linear programming problem for any inefficient DMUs. In addition, it is clear from Fig. 1, which shows a typical numerical example, that these two strongly efficient DMUs must be the nearest neighbors of the inefficient DMU_0 from a certain direction, namely the horizontal direction for the output-oriented model or vertical direction for the input-oriented model. Therefore, we just need to select two strongly efficient DMUs for any inefficient DMUs in the case of single input and single output to accelerate the calculation process.

As mentioned above, the selected two strongly efficient DMUs are the nearest neighbors that bracket the inefficient DMU from a certain direction, the horizontal direction for the output-oriented BCC model and the vertical direction for the input-oriented BCC model. Taking the output-oriented BCC model (7) for example, we should firstly find the two strongly efficient DMUs that are nearest DMU_0 in the direction of input *x*-axis. The horizontal distance between each strongly efficient DMU and inefficient DMU_0 can be calculated as:

$$D_{j0} = x_{1j} - x_{10}, (j \in J) \tag{8}$$

Denote the strongly efficient DMUs that satisfy $D_{j0} = x_{1j} - x_{10} \geq 0, (j \in J)$ as the set D_1 and the strongly efficient DMUs that satisfy $D_{j0} = x_{1j} - x_{10} < 0, (j \in J)$ as the set D_2. In set D_1, choose one strongly efficient DMU denoted as DMU_{j1} that satisfies $\left\{ j1 \,|\, \min D_{j0},\ j \in D_1 \right\}$ when the set $D_1 \neq \phi$. Similarly, in set D_2 select one strongly efficient DMU denoted as DMU_{j2} that satisfies $\left\{ j2 \,|\, \max D_{j0},\ j \in D_2 \right\}$ when the set $D_2 \neq \phi$. Obviously, we have $D_1 \bigcup D_2 \neq \phi$. Therefore, these must be at least one strongly efficient DMU or at most two strongly efficient DMUs as the reference points to measure the efficiency value for inefficient DMU_0. The above model (7) can be changed to:

$$\begin{aligned} E_0^{Output} = Max \quad & \theta_0 + \varepsilon(s_1^- + s_1^+) \\ s.t. \quad & \lambda_1 x_{1j1} + \lambda_2 x_{1j2} + s_1^- = x_{10} \qquad i = 1, \cdots, m; \\ & \lambda_1 y_{1j1} + \lambda_2 y_{1j2} + s_1^+ = \theta_0 y_{10} \qquad r = 1, \cdots, s; \\ & \lambda_1 + \lambda_2 = 1 \\ & \lambda_1, \lambda_2, s_1^-, s_1^+ \geq 0,\ \varepsilon > 0 \qquad \forall\ j \in J_k, i, r. \end{aligned} \tag{9}$$

From the linear programming model (9), we know that when calculating the inefficient DMU_0 efficiency value we need to consider only 3 constraints and at most 5 variables, which can obviously accelerate the calculation process compared to using the standard BCC method with *n* variables.

Consider a simple one-input one-output example that was shown in Chen and Lai

[13]. Table 1 gives the detailed input-output data.

Table 1. Input-output data

DMU	Input x	Output y
A	1	1
B	3	1
C	2	3
D	4	4
E	5	4
F	7	7
G	10	7
H	9	10
K	7	5

Using the proposed Algorithm 1, we can firstly identify DMUA, DMUC, and DMUH as the three strongly efficient DMUs and the remaining six DMUs as inefficient DMUs. The bold solid line of Fig. 1 corresponds to the strongly efficient frontier under the VRS assumption, and it consists of two segments, AC and CH. The weakly efficient frontier corresponds to the union between the strongly efficient frontier and the red dashed line.

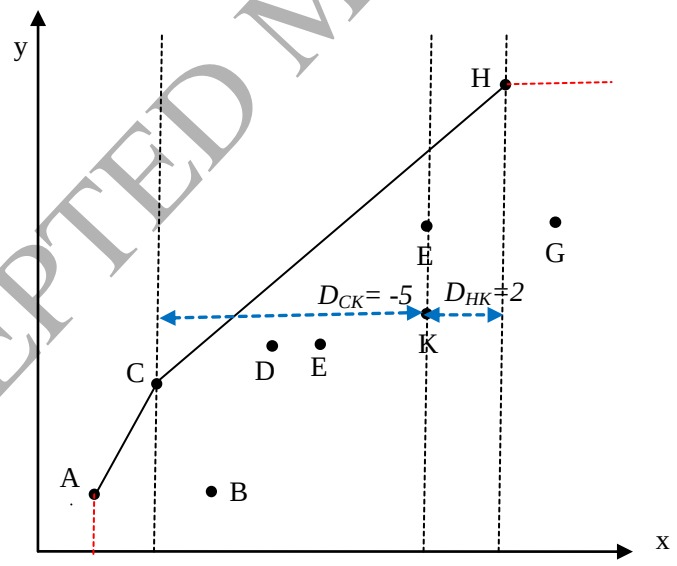

Fig. 1. Visualization of measuring the efficiency of inefficient DMUs

In Fig. 1, we see that the two reference points for any inefficient DMU are must be its nearest neighbors in a certain orientation (x-axis direction or y-axis direction). Suppose, for example, that DMUK is under evaluation and we are using the output-

oriented BCC model. We firstly calculate $D_{AK} = x_A - x_K = -6$, $D_{CK} = x_C - x_K = -5$ and $D_{HK} = x_H - x_K = 2$. Then, we have $D_1 = \{DMUH\}$ and $D_2 = \{DMUA, DMUC\}$. Last, we identify $DMU_{j1} = DMUH$ and $DMU_{j2} = DMUC$. Therefore, strongly efficient DMUC and DMUH are chosen as the two reference points to measure the efficiency of DMUK. Our proposed approach avoids choosing any inefficient DMUs as reference points and can quickly find the reference point among the strongly efficient DMUs, while Chen and Lai's [13] method requires a more complicated 4-step process to find the right sample among all the nine DMUs.

4.2 Multiple inputs or multiple outputs case

In the case of multiple inputs or multiple outputs, at most $m+s$ strongly efficient DMUs are needed to solve the linear programming problem for any inefficient DMU. However, these $m+s$ strongly efficient DMUs are no longer the nearest neighbors of the inefficient DMU_0. Therefore, when the set of strongly efficient DMUs identified in Algorithm 1 is still quite large, we could follow Chen and Lai [13] and Chen and Cho [12] to set a small sample set S such that $S \subseteq J$ and $|S| \geq m+s$. That is, S is a sample set drawn from the strongly efficient DMU set J with a size no less than $m+s$. For an inefficient DMU_k ($k \in J_1 \setminus J$), define its associated linear programming problem $P(S)$ based on output-oriented BCC model as:

$$
\begin{aligned}
& E_k^{Output}(k,S) = Max \quad \theta_k + \varepsilon\Big(\sum_{i=1}^{m} s_i^- + \sum_{r=1}^{s} s_r^+\Big) \\
& s.t. \qquad \sum_{j \in S \cup k} \lambda_j x_{ij} + s_i^- = x_{ik} \qquad i = 1, \cdots, m; \\
& \qquad \sum_{j \in S \cup k}^{n} \lambda_j y_{rj} - s_r^+ = \theta_0 y_{rk} \qquad r = 1, \cdots, s; \qquad (10) \\
& \qquad \sum_{j \in S \cup k}^{n} \lambda_j = 1 \\
& \qquad \lambda_j, s_i^-, s_r^+ \geq 0, \ \varepsilon > 0 \qquad \forall \ j \in S \cup k, i, r.
\end{aligned}
$$

Note that in order to assure the problem $P(S)$ is feasible for any $k \in J_1 \setminus J$ we keep $j \in S \bigcup k$ in model (10). We have $E_k^{Output}(k,S) \le E_k^{Output*}$, where $E_k^{Output*}$ is the optimal value based on model (7) and also the actual efficiency value of DMU_0, since $S \subseteq J$. If and only if $C \subseteq S$, where C is the reference point set of DMU_0, will we have $E_k^{Output}(k,S) = E_k^{Output*}$. Identifying all reference points that could be in the sample S may be costly, hence we prefer a "trial and error" approach (see, [13]) to address this problem. The "trial and error" approach reselects the strongly efficient DMUs to find another sample S when $C \not\subseteq S$ and continues reselecting until $C \subseteq S$.

```
Procedure(k, P, s)
Begin
    S ← InitialSample(k, P, s);
    repeat
        Solve P(S);
        if C ⊄ S do
            S ← ReSample;
    until C ⊆ S

    return θ_k;
End
```

Fig. 2. Algorithm 2: Pseudocode to identify a good sample set

Fig. 2 shows the pseudocode of the proposed Algorithm 2 that will produce a solution for model (10). Four major steps are included in the algorithm. (i) Choose an initial DMU sample; (ii) solve model (10) based on this sample; (iii) check optimality; and (iv) if necessary, redefine the sample and repeat.

(i). Choosing an initial DMU sample S

The selection of initial sample S is an important process which can directly influence Algorithm 2's performance. In other words, successfully selecting S containing reference points will terminate the algorithm and reduce the computational effort significantly. Therefore, we must choose a proper sample S to place as many good reference points in set S as possible.

Chen and Cho [12] proposed placing into the initial sample set only DMUs that are "similar" to DMU_k, believing that these are more likely to be the actual reference points needed. The similarity is defined based on the input-output values. Specifically, they firstly transform the input value vectors to a polar coordinate system, and then utilize the angular coordinate to define the similarity. In this paper, we suggest to use a more straightforward similarity measure proposed by Chen and Lai [13]. The new similarity measure is based on the improvement or moving direction of the inefficient DMU_k under evaluation. In the work of Chen and Lai [13], their similarity measure is based on the input-oriented BCC model. Here, we extend it to the situation of output-oriented BCC model.

Maximizing $\mathbf{y_k}$ proportionately while keeping $\mathbf{x_k}$ the same means that $(\mathbf{x_k}, \mathbf{y'_k})$ must be on the improvement path without considering the feasibility of the linear programming problem, where $\mathbf{y'_k} = \max \mathbf{y_j}\ (j \in J)$. Therefore, $(\mathbf{x_k}, \mathbf{y'_k}) - (\mathbf{x_k}, \mathbf{y_k}) = (\mathbf{0}, \mathbf{y'_k} - \mathbf{y_k})$ is the moving direction of DMU_k. For any DMUp ($p \in J_k$), $(\mathbf{x_p}, \mathbf{y_p}) - (\mathbf{x_k}, \mathbf{y_k})$ is the vector from DMU_k to DMUp. Formally, for any strongly efficient DMUp ($p \in J$), define its similarity with respect to the inefficient DMU_k ($k \in J \setminus J_k$) as follows:

$$\rho(p,k) = ar\cos\left(\frac{\left\langle (\mathbf{0}, \mathbf{y'_k} - \mathbf{y_k}), (\mathbf{x_p} - \mathbf{x_k}, \mathbf{y_p} - \mathbf{y_k}) \right\rangle}{\left\| (\mathbf{0}, \mathbf{y'_k} - \mathbf{y_k}) \right\| \times \left\| (\mathbf{x_p} - \mathbf{x_k}, \mathbf{y_p} - \mathbf{y_k}) \right\|} \right), \tag{11}$$

where $\langle \bullet, \bullet \rangle$ denotes the inner product of the two vectors, in this case $(\mathbf{0}, \mathbf{y'_k} - \mathbf{y_k})$ and

$(\mathbf{x_p - x_k, y_p - y_k})$. $\|\bullet\|$ is the 2-norm of a vector g. $\rho(p,k)$ ($\rho(p,k) \geq 0$) measures the angle between the two vectors $(\mathbf{0, y'_k - y_k})$ and $(\mathbf{x_p - x_k, y_p - y_k})$. The smaller the $\rho(p,k)$, the more similarity exists between DMUp and DMU_k. Therefore, we could select the top several strongly DMUs most similar to DMU_k as the initial sample set S. Note that Chen and Lai's [13] similarity measure selects from the whole DMU set, including efficient and inefficient DMUs, while our new similarity measure just chooses DMUs from among the strongly efficient DMUs, which can obviously save much time if there are relatively few strongly efficient DMUs.

(iii). Checking optimality

The proposed Algorithm 2 terminates if and only if $C \subseteq S$. Therefore, we need an optimality checking mechanism, so we devise the following based on Chen and Lai [13] and Chen and Cho [12].

Denote the optimal solution for inefficient DMU_k in model (10) by $\left\{\theta_k^*, s_i^{-*}, s_r^{+*}, \lambda_j^* \ j \in S \bigcup k\right\}$ and the optimal solution of its dual by $\left\{u_r^*, v_i^*, u_0^*\right\}$. Suppose $\left\{\theta_k^*, s_i^{-*}, s_r^{+*}, \lambda_j^* \ j \in S \bigcup k, \lambda_j^* = 0 \ j \in J \setminus S\right\}$ and $\left\{u_r^*, v_i^{-*}, u_0^*\right\}$ are the optimal solutions of model (7) and its dual, respectively, for DMU_k. Next, we need to check the Karush-Kuhn-Tucker (KKT) optimality condition associated with model (7) that holds for following conditions:

Primal feasibility:

$$\sum_{j \in J \setminus S} \lambda_j^* x_{ij} + \sum_{j \in S \bigcup k} \lambda_j^* x_{ij} + s_i^{-*} = x_{ik} \qquad i = 1, \cdots, m; \qquad (12.1)$$

$$\sum_{j \in J \setminus S}^{n} \lambda_j^* y_{rj} + \sum_{j \in S \bigcup k}^{n} \lambda_j^* y_{rj} - s_r^{+*} = \theta_k^* y_{ro} \qquad r = 1, \cdots, s; \qquad (12.2)$$

$$\sum_{j \in J \setminus S}^{n} \lambda_j^* + \sum_{j \in S \bigcup k}^{n} \lambda_j^* = 1 \qquad (12.3)$$

$$s_i^{-*}, s_r^{+*} \geq 0 \forall \ i,r; \lambda_j^* \geq 0, \forall \ j \in J \setminus S \ and \ \lambda_j^* \geq 0, \forall \ j \in S \qquad (12.4)$$

Dual feasibility:

$$\sum_{i=1}^{m} v_i^* x_{ij} - \sum_{r=1}^{s} u_r^* y_{rj} + u_0^* \le 0, \quad j \in S \bigcup k; \tag{12.5}$$

$$\sum_{i=1}^{m} v_i^* x_{ij} - \sum_{r=1}^{s} u_r^* y_{rj} + u_0^* \le 0, \quad j \in J \setminus S; \tag{12.6}$$

$$\sum_{r=1}^{s} u_r y_{r0} = 1; \tag{12.7}$$

$$u_r \ge 0,\ v_i \ge 0,\ \forall\ r,i. \tag{12.8}$$

Complementary slackness:

$$v_i^* \left(\sum_{j \in J \setminus S} \lambda_j^* x_{ij} + \sum_{j \in S \bigcup k} \lambda_j^* x_{ij} + s_i^{-*} - x_{ik} \right) = 0 \qquad i = 1, \cdots, m; \tag{12.9}$$

$$u_r^* \left(\sum_{j \in J \setminus S}^{n} \lambda_j^* y_{rj} + \sum_{j \in S \bigcup k}^{n} \lambda_j^* y_{rj} - s_r^{+*} - \theta_k^* y_{ro} \right) = 0 \quad r = 1, \cdots, s; \tag{12.10}$$

$$\lambda_j^* \left(\sum_{i=1}^{m} v_i^* x_{ij} - \sum_{r=1}^{s} u_r^* y_{rj} + u_0^* \right) = 0 \qquad j \in S \bigcup k; \tag{12.11}$$

$$\lambda_j^* \left(\sum_{i=1}^{m} v_i^* x_{ij} - \sum_{r=1}^{s} u_r^* y_{rj} + u_0^* \right) = 0 \qquad j \in J \setminus S; \tag{12.12}$$

Constraints (12.1) to (12.5) and (12.7) to (12.12) are all satisfied since $\lambda_j^* = 0\ j \in J \setminus S$. Hence, we only need to check constraint (12.6). If constraint (12.6) holds, the proposed solution set $\left\{ \theta_k^*, s_i^{-*}, s_r^{+*}, \lambda_j^*\ j \in S \bigcup k, \lambda_j^* = 0\ j \in J \setminus S \right\}$ is optimal with model (7) for DMU_k and we have $E_k^{Output}(k, S) = E_k^{Output*}$.

(iv). Redefining the sample S

When constraint (12.6) is not satisfied, the sample S needs to be redefined in Algorithm 2. That is, we need to reselect and place some other strongly efficient DMUs into the sample S, which requires dropping some existing strongly efficient DMUs from S to satisfy the sample set size constraint.

Let set Ψ denote the DMUs to be added to the sample set S and suppose $|\Psi| = c$. Let set Ω be the DMUs which will be dropped from S, so we must have $|\Omega| = c$. The potential strongly efficient DMUs which will be selected for inclusion in

Ψ (i.e. for adding to S) are those not satisfying the constraint (12.6), specifically, $\Psi \subseteq \left\{ j \in J \setminus S : \sum_{i=1}^{m} v_i^* x_{ij} - \sum_{r=1}^{s} u_r^* y_{rj} + u_0^* > 0 \right\}$. The larger positive value of $\sum_{i=1}^{m} v_i^* x_{ij} - \sum_{r=1}^{s} u_r^* y_{rj} + u_0^*$ indicates "more infeasible" with respect model (7) (see, e.g., [12, 13]). Therefore, calculate $\sum_{i=1}^{m} v_i^* x_{ij} - \sum_{r=1}^{s} u_r^* y_{rj} + u_0^*$ for $j \in J_k \setminus S$, and the most infeasible c strongly efficient DMUs will be selected for inclusion into Ψ. The potential DMUs that are need to drop from the sample S are those satisfying constraint (12.6), specifically, $\Omega \subseteq \left\{ j \in S : \sum_{i=1}^{m} v_i^* x_{ij} - \sum_{r=1}^{s} u_r^* y_{rj} + u_0^* < 0 \right\}$, so $\Omega \subseteq \left\{ j \in S : \lambda_j^* = 0 \right\}$ from the complementary slackness. However, $\lambda_j^* = 0, j \in S$ does not mean $j \notin c$, therefore following Chen and Lai [13], we can also drop the candidate from the first position of the list and to choose one from Ψ to the last position of the list after storing the elements in sample S in an ordered list.

5. Case study

5.1 Data set

In this section, we will demonstrate the effectiveness and computational efficiency of our proposed approach by using simulated cases following Chen and Cho [12], Dulá and López [17], Dulá [18], and Chen and Lai [13]. We test the simulated data sets with sizes n = 2000, 5000, 10000, 15000, 20000, 30000, and 50000. The data sets include five dimensions: simple one-input-one-output, two-inputs-two-outputs, three-inputs-three-outputs, four-inputs-four-outputs, and five-inputs-five-outputs, denoted as $(m,s) = (1,1), (2,2), (3,3), (4,4)$, and $(5,5)$. Combining the seven cardinalities and five dimensions, we have 35 scenarios. Table 2 below shows the descriptive statistical analysis for one example scenario of $n = 15000$ and $(m,s) = (3,3)$.

Table 2. Descriptive statistical analysis of the set of 15000 DMUs with three inputs and three outputs

Variable	Inputs			Outputs		
	Input 1	Input 2	Input 3	Output 1	Output 2	Output 3
Max	998	2000	2902	9716	4889	1499
Min	102	501	1532	5112	3014	500
Average	512.4	1376.8	2187.9	8014.8	4215.9	1024.8
S.D.	321.6	1246.2	2654.5	6678.4	4412.4	921.6
Median	661	1386	2098	7688	4214	966

5.2 Effectiveness

The proposed algorithms were implemented and the efficiency values calculated by using Matlab R2014a, performing the computation on an Intel Core i3 CPU with 8GB memory and a Windows 10 operating system. In the application, we calculate the efficiency values based on the output-oriented BCC model. To avoid possible inaccuracy in only one-time calculation, each calculation process was done five times. The average time is recorded as the final calculating time, barring the time for reading data file.

We firstly consider the seven scenarios all with simple one-input and one-output, that is, n=2000, 5000, 10000, 15000, 20000, 30000, 50000 and $(m,s)=(1,1)$. Firstly, we use the traditional method (TM) to calculate the efficiency value for each DMU and record the time in each scenario. The corresponding results are shown in column 2 of the Table 3 below. Then, the proposed method was used to measure the efficiency value for each DMU and record the corresponding time in each scenario. Using our proposed Algorithm 1, in step 1 we divide the DMUs into 4, 10, 20, 30, 40, 60, and 100 groups for the scenarios of n=2000, 5000, 10000, 15000, 20000, 30000, and 50000, respectively in order to have no more than 500 DMUs in each group. We record the times for two cases. Case 1 is that all groups of DMUs are calculated by using only one computer; Case 2 is that all groups DMUs are calculated in parallel by using ten computers for further saving of time. Through Algorithm 1,

we can identify all strongly efficient DMUs in each scenario and record the corresponding calculation time, which are shown in columns 3 and 8 of Table 3. After identifying all strongly efficient DMUs, we also need to calculate the efficiency values for the inefficient DMUs. Two methods are considered. Method 1 shown in model (7) is that we use all of the strongly efficient DMUs as the sample to measure the inefficient DMUs' efficiency values. Method (2) shown in model (9) is that we just use two strongly efficient DMUs as the sample to evaluate the inefficient DMUs' efficiencies. The corresponding calculation times are shown in columns 4 and 9 of Table 3. Columns 5 and 10 show the total time by using the proposed method.

Table 3. Calculation time of different methods

DMUs	TM	Proposed method									
		Case 1					Case 2				
		Algorithm 1	Method 1		Method 2		Algorithm 1	Method 1		Method 2	
			Model (7)	Total time	Model (9)	Total time		Model (7)	Total time	Model (9)	Total time
n=2000	108.2	61.4	38.1	99.5	30.6	92.0	34.6	38.1	72.7	30.6	65.2
n=5000	297.5	198.6	69.2	267.8	51.3	249.9	48.3	69.2	117.5	51.3	99.6
n=10000	684.4	485.7	116.6	602.3	82.4	568.1	57.4	116.6	174.0	82.4	139.8
n=15000	1231.8	774.5	284.8	1059.3	235.6	1010.1	78.6	284.8	363.4	235.6	314.2
n=20000	2053.1	1410.2	314.4	1724.6	252.8	1663.0	99.8	314.4	414.2	252.8	352.6
n=30000	3941.9	2468.4	764.0	3232.4	685.1	3153.5	136.3	764.0	900.3	685.1	821.4
n=50000	8540.8	4911.6	1921.0	6832.6	1750.2	6661.8	277.6	1921.0	2198.6	1750.2	2027.8

From Table 3, we have the following conclusions. (i) It may take much time to calculate all DMU efficiency values by using traditional calculation method. As the number of DMUs increases, the time may increase more quickly than linearly. For example, when n=2000 the average time is 108.2 seconds, while the n=50000 scenario (factor of increase = 25) requires 8540.8 seconds (factor of increase ~79) to finish the calculation process. (ii) In case 1, where only one computer is used, it also takes much time to identify all strongly efficient DMUs, while in case 2, where ten computers are used, much time was saved in finding the strongly efficient DMUs. For example, when n=30000 2468.4 seconds are needed when using one computer, while it only takes 136.3 seconds using ten computers. (iii) After identifying all strongly efficient DMUs, we use just one computer to calculate the remaining inefficient

DMUs' efficiency values. Therefore, this step takes the same time in the two cases as reflected in columns 4 and 9 containing the same calculation time, and columns 6 and 11 also. (iv) Method 1 uses all strongly efficient DMUs as the sample to evaluate the inefficient DMUs' efficiency values, while method 2 uses just two reference points as the sample. Therefore, method 2 saves time, as reflected in columns 4 and 6 or columns 9 and 11. For example, when n=50000 it takes an average time of 1921 seconds to finish the calculation for inefficient DMUs by using method 1, while using method 2 needs only 1750.2 seconds including the time required to identify the nearest neighbors. (v) To clearly compare the total time of the different methods, we graph the total time results in Fig. 2, including the total time of traditional method (TM), total time of method 1 (M1) in case 1 (C1), total time of method 2 (M2) in C1, total time of M1 in case 2 (C2), and total time of M2 in C2. Combining Table 3 and Fig. 2, we know that traditional method is the most costly, while the method 2 in case 2 is the least costly. In all cases, our proposed methods saves much time, especially when the number of DMUs is extremely large.

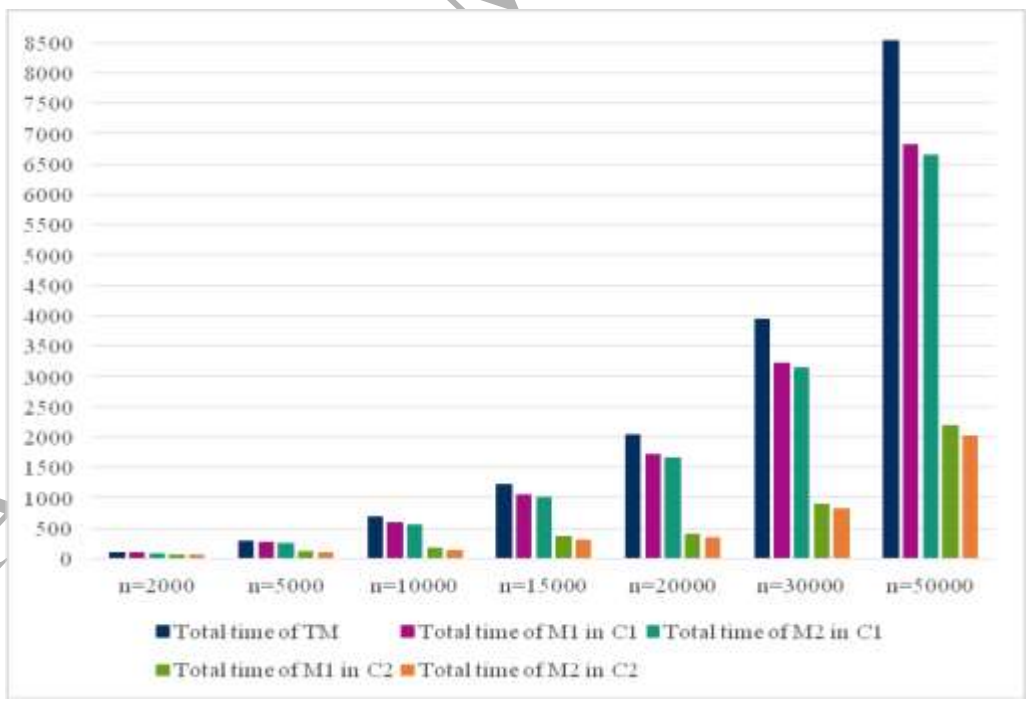

Fig. 2. Total calculation time of different methods

Above we have demonstrated the effectiveness of the proposed approach for the

single input and single output scenario. Next, we will show the results for other scenarios with multiple inputs or multiple outputs. Note that we also need our Algorithm 1 to identify strongly efficient DMUs first and we use ten computers to accelerate the computation. Then we select some candidates from the strongly efficient DMUs as the sample to form model (10) for inefficient DMUs. Here, we set the sample size $|S|=200$. The following Table 4 shows the corresponding calculation time for the remaining four scenarios.

Table 4. Calculation time for different scenarios

DMUs	$(m,s)=(2,2)$				$(m,s)=(3,3)$			
	TM	Algorithm 1	Model (10)	Total time	TM	Algorithm 1	Model (10)	Total time
n=2000	116.8	45.9	41.1	87.0	132.4	60.1	61.5	121.6
n=5000	330.4	71.5	72.7	144.2	377.3	101.3	116.3	217.6
n=10000	731.9	89.0	122.7	211.7	792.6	131.6	202.8	334.4
n=15000	1327.6	131.3	325.0	456.3	1436.4	185.4	452.7	638.1
n=20000	2162.4	161.1	363.1	524.2	2271.2	232.7	547.4	780.1
n=30000	4127.1	214.8	847.3	1062.1	4244.6	297.2	1072.1	1369.3
n=50000	8676.9	366.8	1930.8	2297.6	8781.3	454.9	2211.0	2665.9
DMUs	$(m,s)=(4,4)$				$(m,s)=(5,5)$			
	TM	Algorithm 1	Model (10)	Total time	TM	Algorithm 1	Model (10)	Total time
n=2000	155.4	75.7	88.4	164.1	186.5	94.5	118.6	213.1
n=5000	432.7	126.8	159.9	286.7	493.8	153.1	211.2	364.3
n=10000	869.8	167.8	268.5	436.3	944.9	204.6	342.6	547.2
n=15000	1525.8	228.6	549.4	778.0	1617.4	271.8	675.1	946.9
n=20000	2361.3	288.4	672.2	960.6	2462.6	343.7	848.1	1191.8
n=30000	4346.1	365.9	1250.8	1616.7	4477.2	432.4	1462.2	1894.6
n=50000	8913.4	534.3	2436.5	2970.8	9064.9	613.2	2721.8	3335.0

From Table 4, we know that with the same cardinalities, that is, n=2000, 5000, 10000, 15000, 20000, 30000, or 50000, as the number of inputs or outputs increases, the total calculation time of TM and all varieties of the proposed method all increase. In order to clearly reflect the difference of total time in the different methods, we give the following Fig. 3, which includes four small pictures. Each picture is used to compare the total time of TM with the total time of M2 (with ten computers in identifying the strongly efficient DMUs in Algorithm 1) in each scenario of $(m,s)=(2,2)$, $(3,3)$, $(4,4)$, and $(5,5)$. Obviously, our proposed methods can save

more calculation time in different scenarios. Therefore, it is much more suitable for application in situations with massive DMU sets.

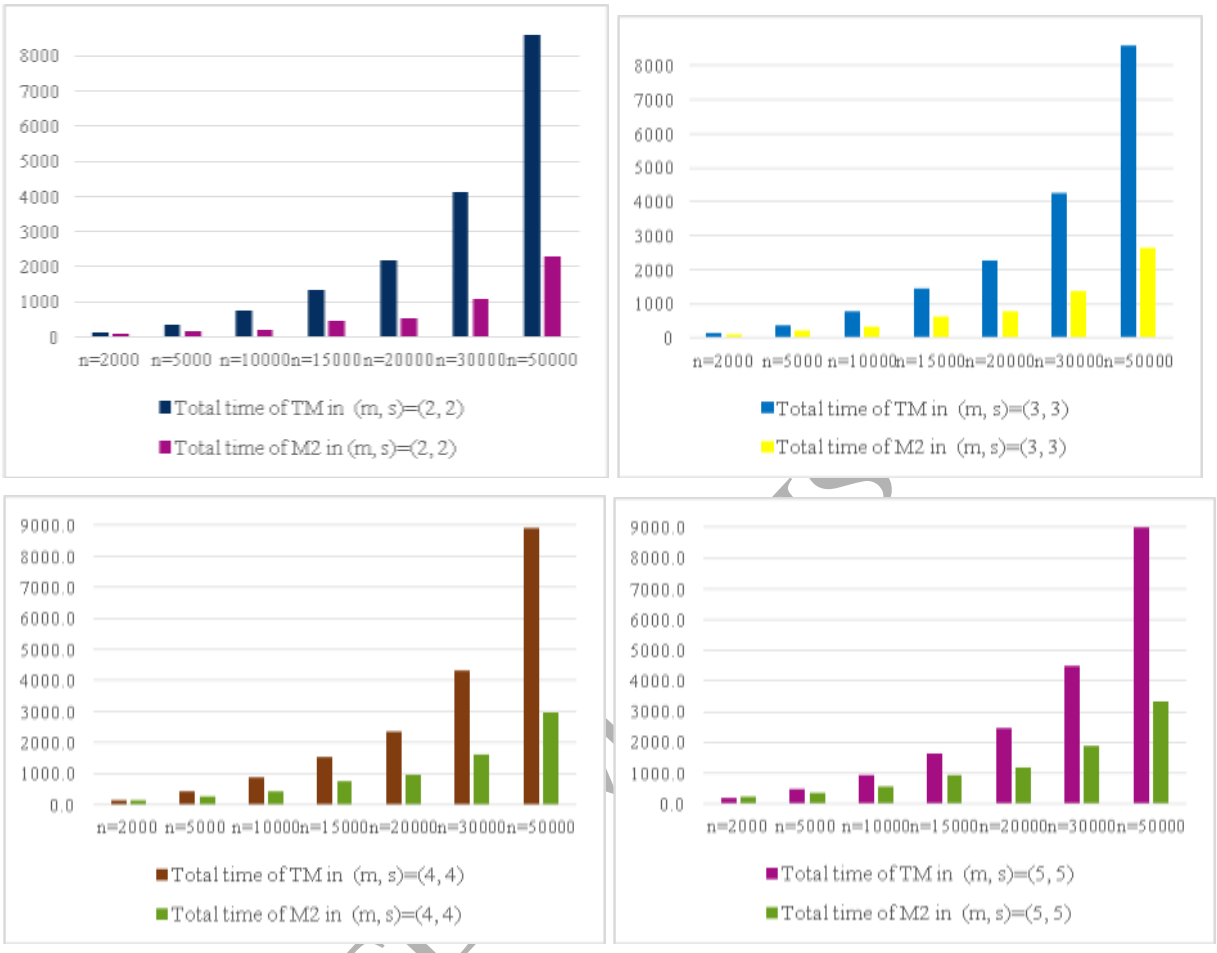

Fig. 3. Total calculation time in different scenarios

6. Conclusions

DEA is an effective tool which has been widely applied in evaluating the efficiency of DMUs. Using a self-evaluation mode, it measures relative efficiency of a DMU by comparing it against a peer group. The traditional DEA models can be solved by using standard linear programming techniques, thus, are theoretically considered computationally easy. However, with big data envelopment, a great number of DMUs need to be evaluated. The large DMU set significantly increases the computation time in a nonlinear fashion and yields challenges for many applications.

To overcome these disadvantages of DEA in the big data environment, in this paper we present novel algorithms to accelerate the computation process. Firstly, Algorithm 1 is proposed to divide the large DMU set into groups with a small number of DMUs to reduce the computational burden and identify all strongly efficient DMUs quickly. Using only the strongly efficient DMUs as the sample for evaluating inefficient DMU efficiency can accelerate the computation and thereby save time. Furthermore, if the strongly efficient DMUs also form a large set, further saving of time can be obtained with the proposed algorithms. Two situations are considered: one-input-one-output and multiple-input-multiple-output. In the one-input-one-output situation, we quickly identify two reference points to evaluate the efficiency values for inefficient DMUs. In the situation of multiple inputs or multiple outputs, we use Algorithm 2 to reselect some strongly efficient DMUs as the sample for inefficient DMUs. Last, the proposed methods were tested for effectiveness using simulated data in various scenarios.

Some further research directions can be drawn from our study. Firstly, our proposed method just considers the traditional input-oriented and output-oriented BCC models, and hence it can extend to other DEA models, e.g. SMB model. Secondly, the thoughts of divide the huge DMUs into groups in the proposed algorithms can also be used to solve other problems (e.g., resource allocation, environmental problems, etc.) in the context of big data. Finally, DEA can also be extended as a data mining tool to identify and excavate more meaningful information in the big data environment.

Acknowledgments

The research is supported by the National Natural Science Funds of China (Nos. 71222106, 71110107024, 71171001, 71471001, and 71501139), Research Fund for the Doctoral Program of Higher Education of China (No. 20133402110028), Foundation for the Authors of National Excellent Doctoral Dissertation of P. R. China (No. 201279), Top-Notch Young Talents Program of China, and Internet of Things

Industry Development Research Base Biding Project, Nanjing University of Posts and Telecommunications (No. JDS215005). Qingyuan Zhu thanks the support of the State Scholarship Fund by the Office of China Scholarship Council (No. 201606340054).

References

[1] Ali AI. Streamlined computation for data envelopment analysis. Eur J Oper Res 1993; 64(1): 61-67.

[2] Ali, AI. Computational aspects of DEA. In A. Charnes, W. W. Cooper, A. Lewin, & L. M. Seiford (Eds.), Data envelopment analysis. Methodology and applications: Theory (pp. 63–88). Netherlands: Springer; 1994.

[3] Ang S, Chen CM. Pitfalls of decomposition weights in the additive multi-stage DEA model. Omega 2016; 58: 139-153.

[4] Azadeh A, Nazari T, Charkhand H. Optimisation of facility layout design problem with safety and environmental factors by stochastic DEA and simulation approach. Int J Prod Res 2015; 53(11): 3370-3389.

[5] Azadi M, Jafarian M, Saen RF, Mirhedayatian SM. A new fuzzy DEA model for evaluation of efficiency and effectiveness of suppliers in sustainable supply chain management context. Comput Oper Res 2015; 54: 274-285.

[6] Banker RD, Charnes A, Cooper WW. Some models for estimating technical and

scale inefficiencies in data envelopment analysis. Manag Sci 1984; 30(9): 1078-1092.

[7] Barr RS, Durchholz ML. Parallel and hierarchical decomposition approaches for solving large-scale data envelopment analysis models. Ann Oper Res 1997; 73: 339-372.

[8] Baru C, Bhandarkar M, Nambiar R, Poess M, Rabl T. Benchmarking big data systems and the big data top100 list. Big Data 2013; 1(1): 60-64.

[9] Charnes A, Cooper WW. Programming with linear fractional functionals. Nav Res Logist Q 1962;9(3‐4): 181-186.

[10] Charnes A, Cooper WW, Rhodes E. Measuring the efficiency of decision making units. Eur J Oper Res 1978; 2(6): 429-444.

[11] Chen H, Chiang RHL, Storey VC. Business Intelligence and Analytics: From Big Data to Big Impact. MIS Quart 2012; 36(4): 1165-1188.

[12] Chen WC, Cho WJ. A procedure for large-scale DEA computations. Comput Oper Res 2009; 36(6): 1813-1824.

[13] Chen WC, Lai SY. Determining radial efficiency with a large data set by solving small-size linear programs. Ann Oper Res 2015. doi:10.1007/s10479-015-1968-4.

[14] Cook WD, Seiford LM. Data envelopment analysis (DEA)–Thirty years on. Eur J Oper Res 2009; 192(1): 1-17.

[15] Du J, Wang J, Chen Y, Chou SY, Zhu J. Incorporating health outcomes in Pennsylvania hospital efficiency: an additive super-efficiency DEA approach. Ann Oper Res 2014; 221(1): 161-172.

[16] Dulá JH, Thrall RM. A computational framework for accelerating DEA. J Prod Anal 2001; 16(1): 63-78.

[17] Dulá JH, López FJ. Preprocessing DEA. Comput Oper Res 2009; 36(4): 1204-1220.

[18] Dulá JH. An algorithm for data envelopment analysis. INFORMS J Comput 2011; 23(2): 284-296.

[19] Fang L, Hecheng L. Duality and efficiency computations in the cost efficiency model with price uncertainty. Comput Oper Res 2013; 40(2): 594-602.

[20] Korhonen PJ, Siitari PA. A dimensional decomposition approach to identifying efficient units in large-scale DEA models. Comput Oper Res 2009; 36(1): 234-244.

[21] López-Torres L, Prior D. Centralized allocation of human resources. An application to public schools. Comput Oper Res 2016; 73: 104-114.

[22] Michael K, Miller KW. Big data: New opportunities and new challenges [guest editors' introduction]. Computer 2013; 46(6): 22-24.

[23] Mitropoulos P, Mitropoulos I, Giannikos I. Combining DEA with location analysis for the effective consolidation of services in the health sector. Comput Oper Res 2013; 40(9): 2241-2250.

[24] Piran FAS, Lacerda DP, Camargo LFR, Viero CF, Dresch A, Cauchick-Miguel PA. Product modularization and effects on efficiency: An analysis of a bus manufacturer using data envelopment analysis (DEA). Int J Prod Econ 2016; 182: 1-13.

[25] Rahimian M, Soltanifar M. An application of DEA based Malmquist productivity index in university performance analysis. Manag Sci Lett 2013; 3(1): 337-344.

[26] Shen WF, Zhang DQ, Liu WB, Yang GL. Increasing discrimination of DEA evaluation by utilizing distances to anti-efficient DEA frontiers. Comput Oper Res 2016; 75: 163-173.

[27] Song M, An Q, Zhang W, Wang Z, Wu J. Environmental efficiency evaluation based on data envelopment analysis: A review. Renew Sust Energ Rev 2012; 16(7): 4465-4469.

[28] Song M, Tao J, Wang S. FDI, technology spillovers and green innovation in China: analysis based on Data Envelopment Analysis. Ann Oper Res 2015; 228(1): 47-64.

[29] Tortosa-Ausina E, Armero C, Conesa D, Grifell-Tatjé E. Bootstrapping profit change: An application to Spanish banks. Comput Oper Res 2012; 39(8): 1857-1871.

[30] Wang K, Huang W, Wu J, Liu YN. Efficiency measures of the Chinese commercial banking system using an additive two-stage DEA. Omega 2014; 44:

5-20.

[31] Wanke P, Barros CP, Emrouznejad A. Assessing productive efficiency of banks using integrated Fuzzy-DEA and bootstrapping: A case of Mozambican banks. Eur J Oper Res 2016; 249(1): 378-389.

[32] Wu J, Zhu Q, Chu J, An Q, Liang L. A DEA-based approach for allocation of emission reduction tasks. Int J Prod Res 2016; 54(18): 5618-5633.

[33] Wu J, Zhu Q, Ji X, Chu J, Liang L. Two-stage network processes with shared resources and resources recovered from undesirable outputs. Eur J Oper Res 2016; 251(1): 182-197.

[34] Wu X, Zhu X, Wu GQ, Ding W. Data mining with big data. IEEE T Knowl Data En 2014; 26(1): 97-107.

[35] Zha Y, Zhao L, Bian Y. Measuring regional efficiency of energy and carbon dioxide emissions in China: A chance constrained DEA approach. Comput Oper Res 2016; 66: 351-361.

[36] Zhu Q, Wu J, Li X, Xiong B. China's regional natural resource allocation and utilization: a DEA-based approach in a big data environment. J Clean Prod 2016; 142: 809-818.