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

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
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\textsuperscript{a,b}; Jie Wu\textsuperscript{a}; Malin Song\textsuperscript{c*}

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

\textit{b Department of Business Administration, University of Illinois at Urbana-Champaign, Champaign, Illinois, 61822, USA}

\textit{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 repeatedly 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$_j$ ($j = 1, \ldots, n$) consumes amount $x_{ij}$ of input $i$ to produce amount $y_{rj}$ of output $r$. Define $x_j = (x_{i1}, x_{i2}, \ldots, x_{im})^T$ and $y_j = (y_{r1}, y_{r2}, \ldots, y_{rs})^T$ 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).

\[ E_{0}^{\text{Input}} = \max \frac{\sum_{r=1}^{s} u_{r} y_{r0} + u_{0}}{\sum_{i=1}^{m} v_{i} x_{i0}} \]
\[ \text{s.t.} \quad \frac{\sum_{r=1}^{s} u_{r} y_{rj} + u_{0}}{\sum_{i=1}^{m} v_{i} x_{ij}} \leq 1, \quad j = 1, \ldots, n; \]
\[ u_{r} \geq 0, \quad r = 1, \ldots, s; \]
\[ v_{i} \geq 0, \quad i = 1, \ldots, m; \]
\[ u_{0} \text{ free in sign.} \quad (1) \]

and

\[ E_{0}^{\text{Output}} = \max \frac{\sum_{i=1}^{m} v_{i} x_{i0} + u_{0}}{\sum_{r=1}^{s} u_{r} y_{r0}} \]
\[ \text{s.t.} \quad \frac{\sum_{i=1}^{m} v_{i} x_{ij} + u_{0}}{\sum_{r=1}^{s} u_{r} y_{rj}} \leq 1, \quad j = 1, \ldots, n; \]
\[ u_{r} \geq 0, \quad r = 1, \ldots, s; \]
\[ v_{i} \geq 0, \quad i = 1, \ldots, m; \]
\[ u_{0} \text{ free in sign.} \quad (2) \]

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$_0$ ($0 = 1, \ldots, n$) 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]).

$$E_{0}^{\text{input}} = \text{Max} \sum_{r=1}^{m} u_r y_{r0} + u_0$$

s.t. $$\sum_{r=1}^{m} u_r y_{r0} - \sum_{i=1}^{m} v_i x_{i0} + u_0 \leq 0, \quad j = 1, \ldots, n;$$

$$\sum_{i=1}^{m} v_i x_{i0} = 1,$$

$$u_r \geq 0, \quad r = 1, \ldots, s;$$

$$v_i \geq 0, \quad i = 1, \ldots, m;$$

$u_0$ free in sign.

and

$$E_{0}^{\text{output}} = \text{Max} \sum_{i=1}^{m} v_i x_{i0} + u_0$$

s.t. $$\sum_{i=1}^{m} v_i x_{i0} - \sum_{r=1}^{s} u_r y_{r0} + u_0 \leq 0, \quad j = 1, \ldots, n;$$

$$\sum_{r=1}^{s} u_r y_{r0} = 1,$$

$$u_r \geq 0, \quad r = 1, \ldots, s;$$

$$v_i \geq 0, \quad i = 1, \ldots, m;$$

$u_0$ free in sign.

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

$$E_{0}^{\text{input}} = \text{Max} \sum_{r=1}^{m} u_r y_{r0} + u_0$$

s.t. $$\sum_{r=1}^{m} u_r y_{r0} - \sum_{i=1}^{m} v_i x_{i0} + u_0 \leq 0, \quad j = 1, \ldots, n;$$

$$\sum_{i=1}^{m} v_i x_{i0} = 1,$$

$$u_r \geq 0, \quad r = 1, \ldots, s;$$

$$v_i \geq 0, \quad i = 1, \ldots, m;$$

$u_0$ free in sign.
\[ E_{\text{Input}}^0 = \text{Min } \theta_0 - \varepsilon \left( \sum_{i=1}^{m} s_i^- + \sum_{r=1}^{s} s_r^+ \right) \]

s.t. \[ \sum_{j=1}^{n} \lambda_j x_{ij} + s_i^- = \theta_0 x_{io} \quad i = 1, \ldots, m; \]
\[ \sum_{j=1}^{n} \lambda_j y_{ij} - s_r^+ = y_{ro} \quad r = 1, \ldots, s; \] (5)
\[ \sum_{j=1}^{n} \lambda_j = 1 \]
\[ \lambda_j, s_i^-, s_r^+ \geq 0, \varepsilon > 0 \quad \forall \ j, i, r. \]

and

\[ E_{\text{Output}}^0 = \text{Max } \theta_0 + \varepsilon \left( \sum_{i=1}^{m} s_i^- + \sum_{r=1}^{s} s_r^+ \right) \]

s.t. \[ \sum_{j=1}^{n} \lambda_j x_{ij} + s_i^- = x_{io} \quad i = 1, \ldots, m; \]
\[ \sum_{j=1}^{n} \lambda_j y_{ij} - s_r^+ = \theta_0 y_{ro} \quad r = 1, \ldots, s; \] (6)
\[ \sum_{j=1}^{n} \lambda_j = 1 \]
\[ \lambda_j, s_i^-, s_r^+ \geq 0, \varepsilon > 0 \quad \forall \ j, i, r. \]

where \( \varepsilon \) 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 \( x_{io}, (i=1, \ldots, m) / y_{ro}, (r=1, \ldots, s) \) by \( \theta_0 \) 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 \( \left( \sum_{j=1}^{n} \lambda_j^* x_{ij} = x_{io} - s_i^-, \sum_{j=1}^{n} \lambda_j^* y_{ij} = \theta_0^* y_{ro} + s_r^+ \right) \) should be within the production possibility set (PPS) constructed by all DMUs, where the superscript *
denotes an optimal value of model (6). That is, \( \left( \sum_{j=1}^{n} \lambda_j^* x_{ij}, \sum_{j=1}^{n} \lambda_j^* y_{ij} \right) \) represents the “virtual DMU” composed of the peers for DMU\(_0\). DMU\(_j\) (\( \lambda_j^* \neq 0, j = 1, \ldots, n \)) is called the reference point, or simply the reference of DMU\(_0\). In addition, these DMUs with \( \lambda_j^* \neq 0, j = 1, \ldots, n \) are strongly efficient. The optimal solution \( \lambda_j^* = 0, j = 1, \ldots, n \) 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 \( s_i^- = s_i^+ = 0 \).

**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 \( s_i^+ \neq 0 \) 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 \( \psi \), 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 $J = \{j \mid j = 1, \ldots, n\}$. Then, equally divide the huge $n$ DMUs into $l+k$ groups. The set of DMUs in each group are denoted by $J_{11}, J_{12}, \ldots, J_{l+k-1}, J_{l+k}$. Note that the number of DMUs in each group cannot be bigger than the limit number $\psi$.

**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 $E^k_1, E^k_2, \ldots, E^k_{l+k-1}, E^k_{l+k}$. Let $J_k = E^k_1 \cup E^k_2 \cup \cdots \cup E^k_{l+k-1} \cup E^k_{l+k}$.

**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 $J_k$. 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 $J_j$. 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.

\[
E_{0}^{\text{Output}} = \text{Max} \quad \theta_{0} + \varepsilon \left( \sum_{i=1}^{m} s_{i}^{-} + \sum_{r=1}^{s} s_{r}^{+} \right) \\
\text{s.t.} \quad \sum_{j=1}^{n} \lambda_{j} x_{ij} + s_{i}^{-} = x_{io} \quad i = 1, \ldots, m; \\
\sum_{j=1}^{n} \lambda_{j} y_{ij} - s_{r}^{+} = \theta_{0} y_{io} \quad r = 1, \ldots, s; \\
\sum_{j=1}^{n} \lambda_{j} = 1 \\
\lambda_{j}, s_{i}^{-}, s_{r}^{+} \geq 0, \varepsilon > 0 \quad \forall \ j \in J, i, r.
\]

(7)

The “virtual DMU” composed of the peers for inefficient DMU\(_0\) is formed by those strongly efficient DMUs with \( \lambda_{j} \neq 0, j = 1, \ldots, n \) 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 \( \{ \lambda_j (j=1,\ldots,n) \} \) 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 efficient 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 DMU₀ in the direction of input x-axis. The horizontal distance between each strongly efficient DMU and inefficient DMU₀ can be calculated as:

\[ D_{j0} = x_{ij} - x_{i0}, \ (j \in J) \]  

(8)

Denote the strongly efficient DMUs that satisfy \( D_{j0} = x_{ij} - x_{i0} \leq 0, \ (j \in J) \) as the set \( D_1 \) and the strongly efficient DMUs that satisfy \( D_{j0} = x_{ij} - x_{i0} > 0, \ (j \in J) \) as the set \( D_2 \). In set \( D_1 \), choose one strongly efficient DMU denoted as DMU\(_{j1}\) that satisfies \( \{ j1 \mid \min D_{j0}, \ j \in D_1 \} \) when the set \( D_1 \neq \emptyset \). Similarly, in set \( D_2 \) select one strongly efficient DMU denoted as DMU\(_{j2}\) that satisfies \( \{ j2 \mid \max D_{j0}, \ j \in D_2 \} \) when the set \( D_2 \neq \emptyset \). Obviously, we have \( D_1 \cup D_2 \neq \emptyset \). 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 DMU₀. The above model (7) can be changed to:

\[
\begin{align*}
E_{\text{Output}}^{\text{Max}} &= \text{Max} \quad \theta_0 + \varepsilon (s_i^- + s_i^+) \\
\text{s.t.} &\quad \lambda_1 x_{i1j1} + \lambda_2 x_{i2j2} + s_i^- = x_{i0} \quad i = 1, \ldots, m; \\
&\quad \lambda_1 y_{i1j1} + \lambda_2 y_{i2j2} + s_i^+ = \theta_0 y_{i0} \quad r = 1, \ldots, s; \\
&\quad \lambda_1 + \lambda_2 = 1 \\
&\quad \lambda_1, \lambda_2, s_i^-, s_i^+ \geq 0, \ \varepsilon > 0 \quad \forall \ j \in J_k, i, r.
\end{align*}
\]

(9)

From the linear programming model (9), we know that when calculating the inefficient DMU₀ 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
Table 1 gives the detailed input-output data.

<table>
<thead>
<tr>
<th>DMU</th>
<th>Input x</th>
<th>Output y</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>F</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>G</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>H</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>K</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

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.

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$_k$ ($k \in J \setminus J$), define its associated linear programming problem $P(S)$ based on output-oriented BCC model as:

$$E^\text{Output}_k(k, S) = \text{Max} \theta_k + \varepsilon \left( \sum_{i=1}^{m} s^-_i + \sum_{r=1}^{s} s^+_r \right)$$

s.t. $$\sum_{j \in S \setminus k} \lambda_j x_{ij} + s^-_i = x_{ik}, \quad i = 1, \ldots, m;$$

$$\sum_{j \in S \setminus k} \lambda_j y_{ij} - s^+_r = \theta_0 y_{ik}, \quad r = 1, \ldots, s;$$

$$\sum_{j \in S \setminus k} \lambda_j = 1$$

$$\lambda_j, s^-_i, s^+_r \geq 0, \quad \varepsilon > 0 \quad \forall \ j \in S \cup k, i, r.$$
Note that in order to assure the problem $P(S)$ is feasible for any $k \in J_1 \setminus J$ we keep $j \in S \cup k$ in model (10). We have $E_{k}\text{Output}(k,S) \leq E_{k}\text{Output}^*$, where $E_{k}\text{Output}^*$ 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 $E_{k}\text{Output}(k,S) = E_{k}\text{Output}^*$. 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 $C \not\subseteq S$ do
      S ← ReSample;
    until $C \subseteq S$
  return $\theta_k$;
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 y_k proportionately while keeping x_k the same means that (x_k, y'_k) must be on the improvement path without considering the feasibility of the linear programming problem, where y'_k = max_j y_j (j ∈ J). Therefore, (x_k, y'_k) - (x_k, y_k) = (0, y'_k - y_k) is the moving direction of DMU_k. For any DMUp (p ∈ J_k), (x_p, y_p) - (x_k, y_k) is the vector from DMU_k to DMUp. Formally, for any strongly efficient DMUp (p ∈ J), define its similarity with respect to the inefficient DMU_k (k ∈ J \ J_k) as follows:

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

where $\langle \cdot, \cdot \rangle$ denotes the inner product of the two vectors, in this case (0, y'_k - y_k) and
\[(x_p - x_k, y_p - y_k). \| \| \text{ is the 2-norm of a vector } g. \ \rho(p, k) (\rho(p, k) \geq 0) \text{ measures the angle between the two vectors } (0, y'_k - y_k) \text{ and } (x_p - x_k, y_p - y_k). \text{ The smaller the } \rho(p, k), \text{ the more similarity exists between DMUp and DMU}_k. \text{ Therefore, we could select the top several strongly DMUs most similar to DMU}_k \text{ as the initial sample set } S. \text{ Note that Chen and Lai’s } [13] \text{ 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 \( \{\theta^*_k, s_i^{**}, s_r^{**}, \lambda_j^* \mid j \in S \cup k\} \) and the optimal solution of its dual by \( \{u^*_i, v_i^*, u_0^*\} \). Suppose \( \{\theta'_i, s_i^{**}, s_r^{**}, \lambda_j^* \mid j \in S \cup k, \lambda_j^* = 0 \mid j \in J \setminus S\} \) and \( \{u^*_i, v_i^-, u_0^*\} \) 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:**

\[
\sum_{j \in J \cup S} \lambda_j^* x_j + \sum_{j \in S \cup k} \lambda_j^* x_j + s_i^{**} = x_{ik} \quad i = 1, \ldots, m; \tag{12.1}
\]

\[
\sum_{j \in J \cup S} \lambda_j^* y_j + \sum_{j \in S \cup k} \lambda_j^* y_j - s_r^{**} = \theta_k^* y_{r0} \quad r = 1, \ldots, s; \tag{12.2}
\]

\[
\sum_{j \in J \cup S} \lambda_j^* + \sum_{j \in S \cup k} \lambda_j^* = 1 \tag{12.3}
\]

\( s_i^{**}, s_r^{**} \geq 0 \forall \ i, r; \lambda_j^* \geq 0, \forall \ j \in J \setminus S \text{ and } \lambda_j^* \geq 0, \forall \ j \in S \tag{12.4} \)
Dual feasibility:

\[
\sum_{i=1}^{m} v_i^j x_{y} - \sum_{r=1}^{s} u_r^* y_{y} + u_0^* \leq 0, \quad j \in S \cup k; \tag{12.5}
\]

\[
\sum_{i=1}^{m} v_i^j x_{y} - \sum_{r=1}^{s} u_r^* y_{y} + u_0^* \leq 0, \quad j \in J \setminus S; \tag{12.6}
\]

\[
\sum_{r=1}^{s} u_r^* y_{r} = 1; \tag{12.7}
\]

\[
u_r \geq 0, \; v_j \geq 0, \; \forall \; r, i. \tag{12.8}
\]

Complementary slackness:

\[
v_i^* \left( \sum_{j \in S} \lambda_j^* x_{y} + \sum_{j \in S \setminus k} \lambda_j^* x_{y} + s_i^* - x_i^* \right) = 0 \quad i = 1, \ldots, H; \tag{12.9}
\]

\[
u_r^* \left( \sum_{j \in S} \lambda_j^* y_{y} + \sum_{j \in S \setminus k} \lambda_j^* y_{y} - y_r^* - \theta_r^* y_{r} \right) = 0 \quad r = 1, \ldots, s; \tag{12.10}
\]

\[
\lambda_j^* \left( \sum_{i=1}^{m} v_i^j x_{y} - \sum_{r=1}^{s} u_r^* y_{y} + u_0^* \right) = 0 \quad j \in S \cup k; \tag{12.11}
\]

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

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 \( \{ \theta_k^*, s_i^*^*, s_r^*^*, \lambda_j^* \; j \in S \cup k, \lambda_j^* = 0 \; j \in J \setminus S \} \) is optimal with model (7) for DMU \( k \) and we have \( E_{k_{Out}}^* (k, S) = E_{k_{Out}}^* \).

(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 \( \Psi \) denote the DMUs to be added to the sample set \( S \) and suppose \( |\Psi| = c \). Let set \( \Omega \) 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,

$$\Psi \subseteq \left\{ j \in J \setminus S: \sum_{i=1}^{m} v_{ij}^{*} x_{ij} - \sum_{r=1}^{s} u_{rj}^{*} y_{rj} + u_{0j}^{*} > 0 \right\}.$$ 

The larger positive value of

$$\sum_{i=1}^{m} v_{ij}^{*} x_{ij} - \sum_{r=1}^{s} u_{rj}^{*} y_{rj} + u_{0j}^{*}$$

indicates “more infeasible” with respect model (7) (see, e.g., [12, 13]). Therefore, calculate

$$\sum_{i=1}^{m} v_{ij}^{*} x_{ij} - \sum_{r=1}^{s} u_{rj}^{*} y_{rj} + u_{0j}^{*}$$

for $j \in J \setminus S$, and the most infeasible $c$ strongly efficient DMUs will be selected for inclusion into $\Psi$. The potential DMUs that are need to drop from the sample $S$ are those satisfying constraint (12.6), specifically,

$$\Omega \subseteq \left\{ j \in S: \sum_{i=1}^{m} v_{ij}^{*} x_{ij} - \sum_{r=1}^{s} u_{rj}^{*} y_{rj} + u_{0j}^{*} < 0 \right\},$$

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 $\Psi$ 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], Dula and Lopez [17], Dula [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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input 1</td>
<td>Input 2</td>
</tr>
<tr>
<td>Max</td>
<td>998</td>
<td>2000</td>
</tr>
<tr>
<td>Min</td>
<td>102</td>
<td>501</td>
</tr>
<tr>
<td>Average</td>
<td>512.4</td>
<td>1376.8</td>
</tr>
<tr>
<td>S.D.</td>
<td>321.6</td>
<td>1246.2</td>
</tr>
<tr>
<td>Median</td>
<td>661</td>
<td>1386</td>
</tr>
</tbody>
</table>

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 \( n=2000, 5000, 10000, 15000, 20000, 30000, 50000 \), 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

<table>
<thead>
<tr>
<th>DMUs</th>
<th>TM</th>
<th>Proposed method</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Case 1</td>
<td>Method 1</td>
<td>Method 2</td>
<td></td>
<td></td>
<td>Method 1</td>
<td>Method 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Algorithm 1</td>
<td>Model (7)</td>
<td>Model (9)</td>
<td></td>
<td></td>
<td>Model (7)</td>
<td>Model (9)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Algorit 1</td>
<td>Total time</td>
<td>Total time</td>
<td></td>
<td></td>
<td>Total time</td>
<td>Total time</td>
</tr>
<tr>
<td>n=2000</td>
<td></td>
<td>108.2</td>
<td>61.4</td>
<td>38.1</td>
<td>99.5</td>
<td>30.6</td>
<td>62.0</td>
<td>34.6</td>
</tr>
<tr>
<td>n=5000</td>
<td></td>
<td>297.5</td>
<td>198.6</td>
<td>69.2</td>
<td>267.8</td>
<td>51.3</td>
<td>249.9</td>
<td>48.3</td>
</tr>
<tr>
<td>n=10000</td>
<td></td>
<td>684.4</td>
<td>485.7</td>
<td>116.6</td>
<td>602.3</td>
<td>82.4</td>
<td>568.1</td>
<td>57.4</td>
</tr>
<tr>
<td>n=15000</td>
<td></td>
<td>1231.8</td>
<td>774.5</td>
<td>284.8</td>
<td>1059.3</td>
<td>235.6</td>
<td>1010.1</td>
<td>78.6</td>
</tr>
<tr>
<td>n=20000</td>
<td></td>
<td>2053.1</td>
<td>1410.2</td>
<td>314.4</td>
<td>1724.6</td>
<td>252.8</td>
<td>1663.0</td>
<td>99.8</td>
</tr>
<tr>
<td>n=30000</td>
<td></td>
<td>3941.9</td>
<td>2468.4</td>
<td>764.0</td>
<td>3252.4</td>
<td>685.1</td>
<td>3153.5</td>
<td>136.3</td>
</tr>
<tr>
<td>n=50000</td>
<td></td>
<td>8540.8</td>
<td>4911.6</td>
<td>1921.0</td>
<td>6832.6</td>
<td>1750.2</td>
<td>6661.8</td>
<td>277.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1921.0</td>
<td>2198.6</td>
<td>1750.2</td>
<td>2027.8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 $|\mathcal{S}|=200$. The following Table 4 shows the corresponding calculation time for the remaining four scenarios.

<table>
<thead>
<tr>
<th>DMUs</th>
<th>$(m, s) = (2, 2)$</th>
<th>$(m, s) = (3, 3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TM</td>
<td>Algorithm 1</td>
</tr>
<tr>
<td>$n=2000$</td>
<td>116.8</td>
<td>45.9</td>
</tr>
<tr>
<td>$n=5000$</td>
<td>330.4</td>
<td>71.5</td>
</tr>
<tr>
<td>$n=10000$</td>
<td>731.9</td>
<td>89.0</td>
</tr>
<tr>
<td>$n=15000$</td>
<td>1327.6</td>
<td>131.3</td>
</tr>
<tr>
<td>$n=20000$</td>
<td>2162.4</td>
<td>161.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DMUs</th>
<th>$(m, s) = (4, 4)$</th>
<th>$(m, s) = (5, 5)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TM</td>
<td>Algorithm 1</td>
</tr>
<tr>
<td>$n=2000$</td>
<td>155.4</td>
<td>75.7</td>
</tr>
<tr>
<td>$n=5000$</td>
<td>432.7</td>
<td>126.8</td>
</tr>
<tr>
<td>$n=10000$</td>
<td>869.8</td>
<td>167.8</td>
</tr>
<tr>
<td>$n=15000$</td>
<td>1525.8</td>
<td>228.6</td>
</tr>
<tr>
<td>$n=20000$</td>
<td>2361.3</td>
<td>288.4</td>
</tr>
<tr>
<td>$n=30000$</td>
<td>4346.1</td>
<td>365.9</td>
</tr>
<tr>
<td>$n=50000$</td>
<td>8913.4</td>
<td>534.3</td>
</tr>
</tbody>
</table>

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 $(m, s) = (2, 2), (3, 3), (4, 4), \text{ and } (5, 5)$. 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.

![Graph](image_url)

**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
References


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


5-20.


