

## Accelerating Data Envelopment Analysis Calculations with Big Data

**R. Hosseinzadeh**

Tabriz Branch, Islamic Azad University, Tabriz, Iran

**N. Azarmir Shotorbani**

Tabriz Branch, Islamic Azad University, Tabriz, Iran

**Y. Jafari\***

Shabestar Branch, Islamic Azad University, Shabestar, Iran

**J. Vakili**

Department of Mathematics, Statistics and Computer Science, University of  
Tabriz, Tabriz, Iran.

**Abstract.** The conventional approach in Data Envelopment Analysis (DEA) involves solving  $n$  linear programming (LP) problems to evaluate the efficiency of Decision-Making Units (DMUs), based on  $m$  inputs and  $s$  outputs, where  $n$  is the total number of DMUs. As the number of inputs, outputs, or DMUs increases, the computational complexity grows, leading to a steep rise in processing time for solving the standard models. This paper proposes an innovative method that significantly reduces computation time by leveraging parallel processing. The methodology consists of five distinct stages: (1) selecting a subset of DMUs using a specialized algorithm; (2) identifying the top-performing DMUs within the selected subset; (3) isolating non-essential DMUs located in the convex hull of the subset; (4) iteratively refining the selection to exclude additional inefficient units; and (5) determining the full set of efficient

---

Received: July 2024; Accepted:

\*Corresponding Author

DMUs. By systematically introducing and filtering subsets, the proposed approach reduces the problem's dimensionality, making it more computationally tractable. The effectiveness of the method is demonstrated through its application to a dataset and is compared against existing approaches.

**AMS Subject Classification:** 90C08; 68T09.

**Keywords and Phrases:** Data Envelopment Analysis (DEA), Big Data, Parallel processing.

## 1 Introduction

Data Envelopment Analysis (DEA), first introduced by [7] and later expanded by [2], is a non-parametric method used to assess the performance of a homogeneous set of Decision-Making Units (DMUs) with similar inputs and outputs. In the DEA framework,  $n$  linear programming (LP) models are solved to measure the efficiency of DMUs, where  $n$  represents the total number of DMUs. The number of DMUs, along with the number of inputs and outputs, directly influences the complexity of the model, determining the number of constraints and variables. However, in the context of large-scale datasets, the increased computational burden becomes a critical challenge. As the number of units grows, the run time required to solve standard DEA models escalates dramatically. A significant body of research has explored theoretical approaches to reducing the computational time required to solve DEA models. For instance, Ali [1] introduced an approach that first identifies non-dominated DMUs and subsequently reduces the number of decision variables by systematically eliminating those associated with inefficient DMUs. Building on this, Dula and Helgason [13] proposed a method to reduce the size of LPs for a subset of DMUs, thereby addressing computational challenges. Similarly, in [3], efforts were made to minimize the time required to evaluate the performance of all DMUs by leveraging multiple parallel processors. This approach integrated Ali's method [1] with a hierarchical division (HD) technique to partition DMUs into smaller, more manageable groups of approximately equal size. Further advancements include the work in [14], which introduced a novel algorithm specifically designed to solve smaller LPs, resulting in significantly reduced computational time. Extensive testing of this algorithm has

shown that it is computationally superior to standard approaches across a wide variety of problem types. Additionally, [16] proposed a distinct method that connects the problem of model size reduction with returns to scale, offering a new approach to improve DEA efficiency. In addressing returns to scale techniques, the method presented in [16] extracts the anti-decremental and anti-incremental returns to scale for efficient DMUs from the variable returns to scale (VRS) of these units. From this, a subset of efficient DMUs under constant returns to scale (CRS) can be identified. Similarly, two other notable methods that focus on both returns to scale and the reduction of computational complexity are discussed in [5] and [18]. The former proposes an approach to expedite the handling of problems under VRS when the dimensionality is relatively low, while the latter applies a multi-step method to isolate efficient DMUs and designs a simulation algorithm to estimate run times for large datasets, also considering VRS. For further insights, references [11, 17, 12, 15, 9, 21] provide valuable contributions. Specifically, [11] examines the influence of three parameters: the number of DMUs, the number of inputs and outputs (i.e., dimensionality), and the ratio of efficient units (density). Meanwhile, [17] proposes a hierarchical analysis method using lexicographic parametric programming to ease DEA computations, particularly for small-scale problems. Shortly thereafter, [12] introduced an effective method known as Build-Hull (BH) to significantly reduce the run time required to identify all efficient units. Dula and Lopes work [15] offers a procedural approach tailored to big data, presenting a theoretical and specific method designed for dynamic conditions where data are continually evolving. Furthermore, [9] proposes an algorithm that addresses the challenge of LP size constraints when dealing with large-scale datasets, allowing for the computation of efficiency in a reasonable number of iterations, without excessive time requirements. Big data often involves high-dimensional datasets, which can complicate analysis. Techniques such as Principal Component Analysis (PCA) and Multiple Correspondence Analysis (MCA) can be employed to reduce dimensionality before applying DEA, and interpretability [6]. In [8] A review article has been presented that highlights recent advancements in data envelopment analysis with big data. In [21], two algorithms are introduced to handle large-scale DMUs by splitting them into two modes:

one input and one output, as well as multiple inputs and multiple outputs. Moreover, [20] highlights the limitations of traditional DEA models when dealing with complex or large network structures, suggesting new opportunities for developing techniques to solve non-linear network DEA models. Finally, [10] presents a hybrid algorithm that utilizes a density-increasing mechanism alongside reference set selection. In this study, we focus on an input-oriented model under VRS technology, noting that the number of efficient DMUs identified under VRS is typically greater than those under CRS. Consequently, this increases the size and complexity of the VRS model. The model size under VRS is typically larger than under CRS. For our implementation, we utilize MATLAB 2018a on a system equipped with 16 cores, 32 GB of RAM, and a 64-bit Windows 2016 operating system. It is important to note that various programming approaches may yield different performance outcomes, as factors such as software selection, hardware configuration, system settings, and installed programs can significantly influence the run time for performance evaluation of a sample DMU. Our results demonstrate that the reduction in run time achieved by our method surpasses that of existing techniques. Using a computer with only 16 processors, our method shows considerable improvements in run time compared to the method proposed in [18], which introduced an effective approach for reducing computational complexity in large-scale DEA problems. In our approach, we begin by selecting a sub-sample of DMUs, identifying the non-essential DMUs within this subset. In subsequent iterations, we incrementally add more DMUs, progressively eliminating inefficient units. This paper is divided into four sections. The basic definitions of DEA and the interpretations are presented in Section 2. The method is developed in Sections 3. The run time of the proposed method is compared with the existing methods in Section 4. Finally, conclusions are given in Section 5.

## 2 Data Envelopment Analysis

First, we provide some basic definitions to explain the concepts and interpretations used in this study. Consider a set of  $n$  observations,  $\Phi = \{\text{DMU}_1, \text{DMU}_2, \dots, \text{DMU}_n\}$  where each  $\text{DMU}_j$  produces  $s$  non-negative

outputs  $y_{1j}, y_{2j}, \dots, y_{sj}$  using  $m$  non-negative inputs  $x_{1j}, x_{2j}, \dots, x_{mj}$ . Also, let the input and output vectors of  $DMU_j$  be denoted by  $\mathbf{x}_j = (x_{1j}, \dots, x_{mj})^t \in \mathbb{R}_+^m$  and  $\mathbf{y}_j = (y_{1j}, \dots, y_{sj})^t \in \mathbb{R}_+^s$ , respectively, where the superscript  $t$  denotes the transpose operation. Moreover, assume that  $J = \{1, 2, \dots, n\}$ . The production possibility set (PPS) is represented by  $T$  and is defined as follows.

$$T = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}_+^m \times \mathbb{R}_+^s \mid \mathbf{y} \text{ can be produced by } \mathbf{x}\}.$$

The definition of the PPS depends on the production technology or the postulates selected by the manager. Banker et al. (1984) considering some postulates in [2] as the production technology based on postulates 1-4 below:

1. The observed activities  $(x_j, y_j)$ ,  $j = 1, \dots, n$ , belong to  $T$ .
2. If  $(x, y) \in T$ , any activity  $(\bar{x}, \bar{y})$  with  $\bar{x} \geq x$  and  $\bar{y} \leq y$  is included in  $T$ .
3. If  $(x, y) \in T$  and  $(\hat{x}, \hat{y}) \in T$ , then  $(\lambda x + (1 - \lambda)\hat{x}, \lambda y + (1 - \lambda)\hat{y}) \in T$  for all  $\lambda \in [0, 1]$ .
4.  $T$  is the intersections of all  $\hat{T}$  satisfying postulates 1, 2 and 3.

Thus, if postulates are satisfied, then  $T$  becomes the following PPS whose technology is the variable returns to scale (VRS).

$$T_v = \{(\mathbf{x}, \mathbf{y}) \mid \sum_{j \in J} \lambda_j \mathbf{x}_j \leq \mathbf{x}, \sum_{j \in J} \lambda_j \mathbf{y}_j \geq \mathbf{y}, \sum_{j \in J} \lambda_j = 1, \lambda_j \geq 0, j \in J\}.$$

Since in DEA, the goal is evaluating the performance of DMUs and comparing them to each other, therefore, we have the following definitions.

**Definition 2.1.** *It is said that  $DMU_A = (\mathbf{x}_A, \mathbf{y}_A)$  dominates  $DMU_B = (\mathbf{x}_B, \mathbf{y}_B)$  if*

$$(-\mathbf{x}_A, \mathbf{y}_A) \geq (-\mathbf{x}_B, \mathbf{y}_B) \quad \& \quad (\mathbf{x}_A, \mathbf{y}_A) \neq (\mathbf{x}_B, \mathbf{y}_B).$$

By assumption that  $DMU_A = (\mathbf{x}_A, \mathbf{y}_A) \in \Phi$  and  $\eta_A$  shows the set of all DMUs of  $\Phi$  which dominate  $DMU_A$ , the following definition is presented.

**Definition 2.2.**  *$DMU_A \in \Phi$  is called a non-dominated DMU if  $\eta_A = \emptyset$ .*

Now, let  $\eta$  show the set of all non-dominated observed DMUs in

$$\{\text{DMU}_1, \text{DMU}_2, \dots, \text{DMU}_n\}$$

**Definition 2.3.** *DMU<sub>A</sub> is called an efficient DMU if and only if it is not dominated by any other DMU in  $T_v$ . We denote the set of efficient DMUs with  $\zeta$ .*

It is clear that  $\eta \subseteq \Phi$ ,  $\zeta \subseteq \eta$  and it is possible  $\eta \not\subseteq \zeta$ .

We know that in the standard input-oriented BCC, the goal is to reduce the input level with ratio  $\theta$ , so that  $(\theta \mathbf{x}_l, \mathbf{y}_l) \in T_v$ . So, envelopment model for evaluation DMU<sub>l</sub> is as follows:

$$\begin{aligned} \text{Min} \quad & \theta_l \\ \text{s.t.} \quad & (\theta \mathbf{x}_l, \mathbf{y}_l) \in T_v \end{aligned}$$

According to the definition of  $T_v$ , we have:

$$\begin{aligned} \text{Min} \quad & \theta_l \\ \text{s.t.} \quad & \sum_{j=1}^n \lambda_j \mathbf{x}_j \leq \theta_l \mathbf{x}_l, \\ & \sum_{j=1}^n \lambda_j \mathbf{y}_j \geq \mathbf{y}_l, \\ & \sum_{j=1}^n \lambda_j = 1, \\ & \lambda_j \geq 0, \quad j = 1, 2, \dots, n. \end{aligned} \tag{1}$$

Now, consider  $\Phi^u \subseteq \Phi$  with  $|\Phi^u| = p$  where  $|\cdot|$  shows the cardinality of a set. In general,  $p$  can be any natural number less than or equal to  $n$ , i.e.,  $|\Phi^u| = p \leq n$ . It is obvious that the PPS corresponding to the DMUs in  $\Phi^u$ , is as follows.

$$T_v^u = \{(\mathbf{x}, \mathbf{y}) \mid \sum_{k=1}^p \lambda_k \mathbf{x}_k^u \leq \mathbf{x}, \sum_{k=1}^p \lambda_k \mathbf{y}_k^u \geq \mathbf{y}, \sum_{k=1}^p \lambda_k = 1, \lambda_k \geq 0, k = 1, \dots, p\}.$$

Note  $\Phi^u \cap \zeta$  can be an empty set. Here, without losing of generality, we can assume that

$$\text{DMU}_k^u = (\mathbf{x}_k^u, \mathbf{y}_k^u), k = 1, \dots, p$$

show the selected DMUs in the sub-sample  $\Phi^u$ . The radial VRS model based on the sub-sample  $\Phi^u$  for  $DMU_l, l \in J$  as a DMU under evaluation is as follows:

$$\begin{aligned}
\text{Min} \quad & \theta_l^u \\
\text{s.t.} \quad & \sum_{k=1}^p \lambda_k \mathbf{x}_k^u \leq \theta_l \mathbf{x}_l, \\
& \sum_{k=1}^p \lambda_k \mathbf{y}_k^u \geq \mathbf{y}_l, \\
& \sum_{k=1}^p \lambda_k = 1, \\
& \lambda_k \geq 0, \quad k = 1, 2, \dots, p.
\end{aligned} \tag{2}$$

**Definition 2.4.** *DMU<sub>l</sub> has the best performance in  $\Phi^u$  if  $\theta_l^{u*} \geq 1$  where  $\theta_l^{u*}$  denotes the optimal value of (2).*

The DMU with the best performance is not necessarily efficient in  $T_v$ . We consider the set of DMUs with the best performance at  $\Phi^u$  with  $\beta^u$ . In model (2), we have a decrease in the number of variables compared to model (1). Thus, using model (2), we encounter an LP problem with a much smaller number of decision variables, so the run time to solve the VRS model (2) is significantly reduced.

**Theorem 2.5.**  $T_v^u \subseteq T_v$ .

**Proof.**

$$\begin{aligned}
(\bar{X}, \bar{Y}) \in T_v^u \implies \exists \bar{\lambda}_j \ (j \in \Phi^u) \ni: \quad & \bar{X} \leq \sum_{j \in \Phi^u} \bar{\lambda}_j x_j, \\
& \bar{Y} \geq \sum_{j \in \Phi^u} \bar{\lambda}_j y_j, \\
& \sum_{j \in \Phi^u} \bar{\lambda}_j = 1, \\
& \bar{\lambda}_j \geq 0, \quad j \in \Phi^u
\end{aligned}$$

Suppose  $\bar{\lambda}_j = 0 \quad \forall j \in \Phi \setminus \Phi^u$ , then we have:

$$\begin{aligned} \exists \bar{\lambda}_j \quad (j \in \Phi) \ni: \quad & \bar{X} \leq \sum_{j \in \Phi} \bar{\lambda}_j x_j, \\ & \bar{Y} \geq \sum_{j \in \Phi} \bar{\lambda}_j y_j, \\ & \sum_{j \in \Phi} \bar{\lambda}_j = 1, \\ & \bar{\lambda}_j \geq 0, \quad j \in \Phi \end{aligned}$$

so  $(\bar{X}, \bar{Y}) \in T_v$ , therefore  $T_v^u \subseteq T_v$ .  $\square$

**Theorem 2.6.** *Inefficient DMUs in  $T_v^u$ , is inefficient in  $T_v$ .*

**Proof.** Assume that  $(x_l, y_l)$  is inefficient in  $T_v^u$ . So it is  $\theta_l^{u*} < 1$  in solving the following problem:

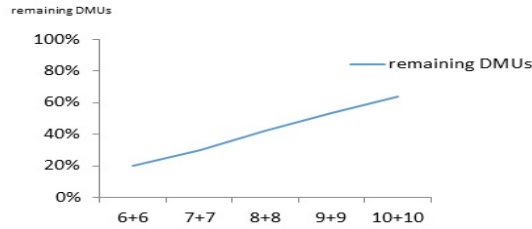
$$\begin{aligned} \text{Min} \quad & \theta_l^u \\ & (\theta_l^u x_l, y_l) \in T_v^u \end{aligned}$$

Because  $T_v^u \subseteq T_v$ , we have  $\theta_l^* \leq \theta_l^{u*}$ , we will have so  $\theta_l^* < 1$  and the theorem is proved.  $\square$



### 3 A way to reduce run time

Before presenting our method to speed up the evaluation method of DMUs, at first consider the method developed by [18]. In their method, by removing inefficient units and reducing the volume of the problem, they have significantly reduced the calculation time compared to the traditional method, we were also able to reduce the calculation time compared to their method with a different algorithm. Although the method proposed by [18] reduces the computational volume, interestingly, applying their method especially in large problems with large number of units and dimensions have some deficiencies. The basis of this method is that it removes inefficient units by using a sub-sample. If the number of removed units is not large enough, the computational volume will still be large and the problem will remain. An instance will clarify the aforementioned shortfall. Imagine a problem with 50000,  $10 + 10$  decision units, applying the method of [18], by removing the inefficient units, the number of remaining units are 32182 that is to say, 64% of units still remains. As another example with same number but different dimension, 50000 units with  $9 + 9$  dimensions, at the end of first stage, 26598 units remains, i.e. 53% of units still remains. The examples shed a light on an obvious shortfall, the time of solving the problems or the run time. In other words, the method [18] suffers from the high run time, especially when the number of units and dimensions of the problem are increasing. As figure 1 shows, for the problem with 50000 units, as the dimension of problem increases, the number of inefficient units slightly removes. Also, the time for problem solving is increasing as well.



**Figure 1:** Amount of units left after removing some inefficient units in the first stage using the method of [18] (Number of DMUs: 50000)

For the case of 50000 units with dimensions of  $9 + 9$  input and output, it takes 16 hours to solve. According to the calculated time, it is clear that the calculation time for the problem of 50000 units with dimensions of  $9 + 9$  is estimated twice compared with the time for problem of 50000 units with dimension of  $8 + 8$ . As a result, if the dimension increase, in the case of real big data, the computation time can increase also. In addition to calculation time, the more memory is needed to solve the problem. However, we emphasize that their method is very useful and effective compared to previous methods.

In order to solve the mentioned deficiencies of method [18], a novel approach is presented. In this approach, some units can be selected and the inefficient units are removed. Repeating this approach, in the following iterations, other units are added to the sample. Consequently, some other inefficient units are removed and the approach is repeated again. Doing the approach contentiously, the eliminated number of inefficient units are increasing, thus, the volume of the problem is significantly reduced, especially, in problems with large numbers and dimensions.

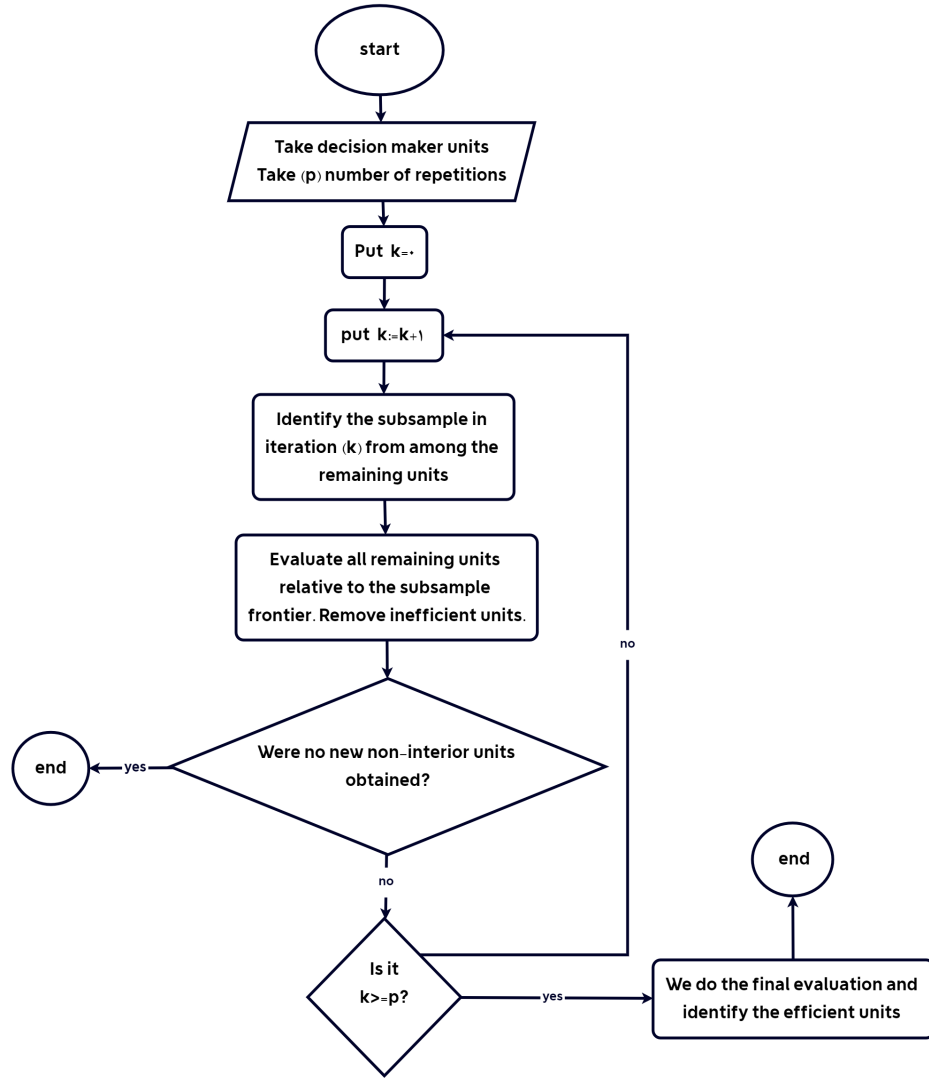
The figure 2 is flowchart of the proposed algorithm for our method.

To select the sub-sample, assumed that  $\lfloor \cdot \rfloor$  is the rounding function which rounds a number to the nearest integer. Theorem proposed by [1] to construct a sub-sample with cardinality  $p$ .  $DMU_l$  is located on the frontier of  $T_v$  based on one of the following equations,:

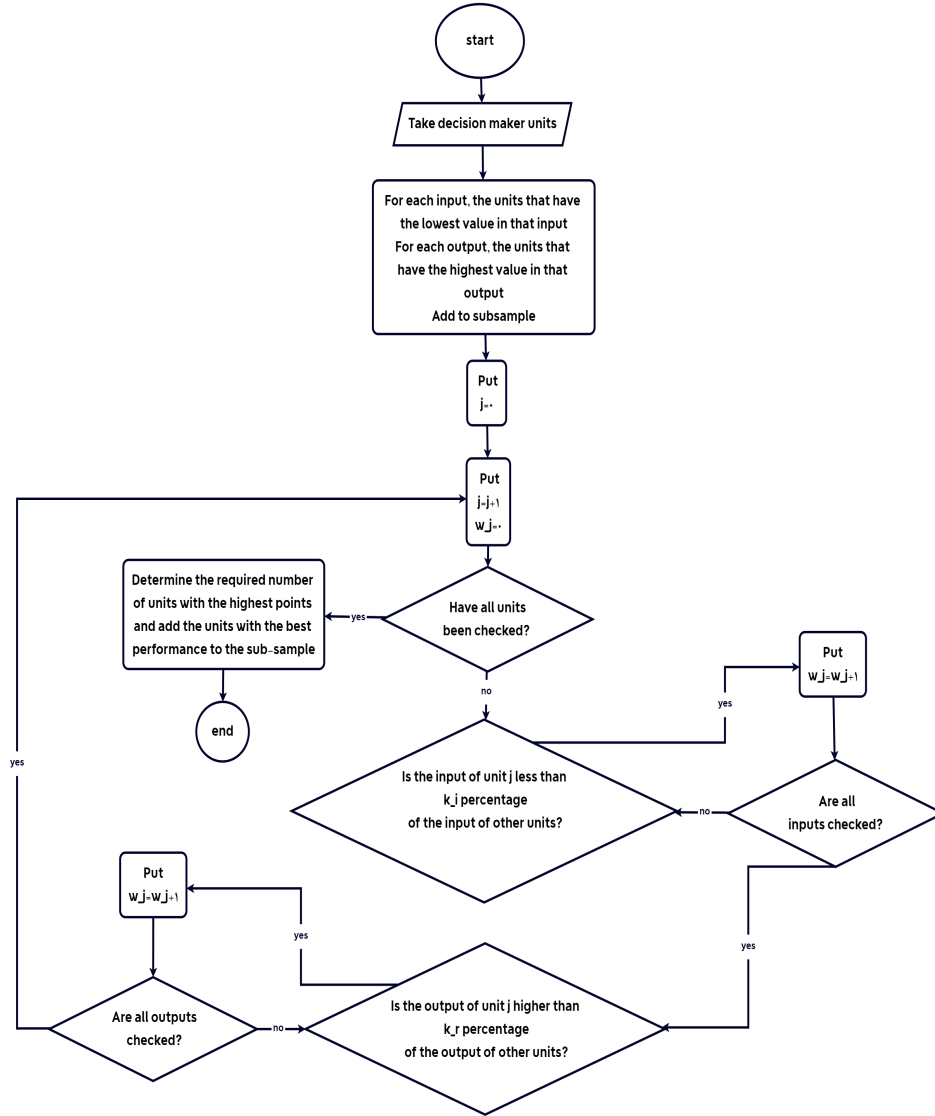
$$x_{il} = \min\{x_{ij} | j = 1, 2, \dots, n\}, \quad i = 1, 2, \dots, m,$$

$$y_{rl} = \max\{y_{rj} | j = 1, 2, \dots, n\}, \quad r = 1, 2, \dots, s.$$

In this way, the numbers of DMUs are selected (i.e.,  $m + s$  DMUs at maximum) to use the following command to select the rest of the DMUs to assign an initial value to the remaining DMUs  $\zeta$  in a short run time. For each unselected  $DMU_l$ , we consider the figure 3 that is flowchart for determining sub-sample.



**Figure 2:** Flowchart of our proposed algorithm



**Figure 3:** Flowchart of selecting sample

The flowchart of figure 3, the DMUs were sorted in descending order according to the assigned scores, and the remaining DMUs (to construct a sub-sample of size  $p$ ) were selected with the highest scores. In this method, DMUs that have higher values than the third quarter of the output values and lower values than the first quarter of the input values have a better chance of entering the sub-sample.

In this method, from a sample of 10,000 DMUs, only 100 DMUs are selected  $p = \sqrt{10000}$ . Of course, when the density is large (for example 10% or more), we know that a sub-sample with a size of 100 does not include more efficient DMUs. This method is powerful enough to determine more inefficient DMUs in a short time. In other words, this method can reduce the size of the original sample to a sub-sample with smaller size. Thus, the search for all efficient DMUs can continue on a sub-sample with high density but with a smaller size.

Therefore, in solving the models and comparing our method with their method, we have used the sample selection method for both methods in the same way.

In order to highlight the strength of the proposed method in comparison with the method of [18], a computer with following properties are used: 16 processors, 32 GB memory, a 64-bit operating system, Windows Service 2016 and Matlab 2018a. With parallel processing, MATLAB uses a par-for-loop to perform a series of parallel loop calculations to reduce run time. as far as we know, that to run iterations in parallel, all steps must be independent of each other. After finding a set of efficient DMUs, par-for-loop was used to measure the efficiency of inefficient DMUs. Data are randomly generated in the range  $[0, 1]$ . Over 100 executions are employed by the software in different numbers and dimensions. The difference between the times we obtained and the times in the article [18] is that we used more processors and a more advanced system. But what is important is that the conditions for our proposed method and their method in this study are the same. For example, selecting the sample, the computer system used, the number of processors, the number of samples selected, and ... .

**Theorem 3.1.** *Removing inefficient units from the problem has no effect on identifying the efficiency of units.*

**Proof.** In the evaluation of DMUs, the DMU is imaged to the efficiency

frontier. Hence  $(\theta_l^* x_l, y_l) \in T_v$ , so  $(\theta_l^* x_l, y_l)$  is a convex combination of units on the frontier and we know that inefficient units are an interior point, so in the evaluation of units,  $\lambda$  corresponding to inefficient units will be zero in the optimal solution, so removing them from the problem will not affect the final solution.  $\square$

**Corollary 3.2.** *If there are all efficient units in a sub-sample, after evaluation, the efficiency of the units is obtained.*

**Theorem 3.3.** *At the end of the algorithm, all efficient units are identified.*

**Proof.** In each iteration, the inefficient units are removed based on the sub-sample. From Theorem 2.6, we know that DMUs inefficient in  $T_v^u$ , is inefficient in  $T_v$ , and according to Theorem 3.1, removing inefficient units from the problem has no effect on identifying the efficiency of units, and according to corollary 3.2, at the end of the algorithm, which is evaluated for the remaining units, no efficiency unit has been removed and all efficient units are identified.  $\square$

**Theorem 3.4.** *The algorithm is convergent.*

**Proof.** The number of iterations is limited, and in each iteration, a finite number of linear programming problems are solved. After all iterations are completed, a finite number of problems are solved, and the algorithm ends.  $\square$

It is worth noting that the identification of efficient units for problems with a large amount of data is important in several ways. The efficiency and inefficiency of the decision-making units are obtained by comparing the units together. When we have efficient units, it is not necessary to solve a large-scale problem to determine the inefficiency of an inefficient unit, and it is only enough to build the set of production possibilities related to the efficient units and evaluate the inefficient unit, the reference set of It is determined between the efficient units and a pattern can be specified for the inefficient unit to reach the efficiency frontier. It should be noted that in calculating the inefficiency of the units, the dimensions of the problem have been reduced and the computational volume has been reduced. In our method and [18] method, all the efficient units are identified, but in both methods, the inefficiency

of the inefficient units is not calculated, our method is faster than their method. Having efficient units, we can calculate the inefficiency of the inefficient units with a problem with a smaller volume.

In solving linear programming problems by the simplex method, the number of constraints in the computational complexity is an important factor, because the basis of the simplex method is the use of base  $B$  corresponding to the columns of basic variables.  $B$  is a square matrix whose number of rows and columns is equal to the number of constraints, so the more the number of constraints increases, the larger the matrix  $B$  will be, and since the various factors in the simplex table are determined according to the matrix  $B$ , as the matrix  $B$  increases Calculations will also increase. On the other hand, we know that in the problem with  $n$  units and  $m$  inputs and  $s$  outputs, the BCC envelopment model has  $m + s + 1$  constraints and  $n + 1$  variables, and in the multiplier form, the number of constraints is  $n + 1$  and the number of variables is  $m + s + 1$ . Because  $n + 1 \gg m + s + 1$  is and as we said, the number of constraints is decisive in increasing the calculation, we prefer to use the envelopment form instead of the multiplier form. It should also be noted that the maximum number of feasible base for a problem also depends on the number of variables. The maximum number of feasible base is  $\binom{n + m + s + 1}{m + s + 1}$ , and the larger  $n$  is, the larger this number will be. In our proposed algorithm, the number of sub-sample units is much less compared to the total units, and therefore we are faced with smaller problems with much less computational volume. Also, in each iteration, a large number of inefficient units are removed, and therefore, fewer problems are solved in subsequent iterations. By reducing the dimensions of the problem and by removing inefficient units, we finally reduced the time complexity.

We know that in the traditional DEA method (with  $n$  DMUs,  $m$  inputs and  $s$  outputs),  $n$  problems are solved with  $m + s + 1$  constraints and  $n + 1$  variables. In this case, the maximum operation for evaluating all DMUs is equal to:

$$T_1 = n \binom{n + m + s + 1}{m + s + 1} (n + 1)(m + s + 2)$$

In the proposed method, if the number of iterations is equal to  $C$  and the number of inefficient DMUs removed in each iteration is  $l_i$  ( $i = 1, \dots, c$ ). As we said, we assumed that the number of members of the sub-sample in the  $i$ th iteration is equal to  $p_i = \sqrt{n_i - l_i}$ , which  $n_i$  is, the number of remaining units in each iteration. In this case, we will have  $\sum_{i=1}^c (n_i - l_i)$  problems with  $m + s + 1$  constraints and  $p_i + 1$  variables, and at the end of the algorithm we will have  $n_c - l_c$  problems with  $m + s + 1$  constraints and  $p_i + 1$  variables. In such a case, the maximum number of operations to solve all problems is equal to:

$$T_2 = \sum_{i=1}^{c-1} (n_i - l_i) \binom{p_i + m + s + 1}{m + s + 1} (p_i + 1)(m + s + 2) +$$

$$(n_c - l_c) \binom{n_c - l_c + m + s + 1}{m + s + 1} (n_c - l_c + 1)(m + s + 2)$$

Note that  $p_i$  is very small compared to  $n$ , it is clearly that  $T_1 \gg T_2$ .

## 4 Empirical Testing

We first examined the proposed method for data with a small number of DMUs. The number of iterations in the following tables present the units move into the sample in several steps as described in the previous section. As the results depict the differences in the run time of the proposed method and method [18] in presence of small dimensions and different data set, is insignificant. As stated before, the difference in run time are significant in presence of large numbers and dimensions. Suppose that  $T_{kh}$  indicates the run time of their method (in seconds) [18] and  $T_i$  indicates run time of the proposed method with  $i$  iteration (in seconds). An example for 5000 units with different dimensions are run and the results are compared to elicit the strength of the proposed method.



**Table 1:** Comparison of run time of method of [18] and our proposed method, with 2 and 3 iteration for problems with different dimensions.

Number of units	Dimensions	$T_{kh}$	$T_2$	$T_3$
5000	1+1	7.85	7.56	7.95
5000	2+2	8.14	7.57	8.52
5000	3+3	8.58	8.13	9.87
5000	4+4	10.14	9.96	11.52
5000	5+5	27.90	14.46	16.70

As Table 1 present, for 5000 DMUs with 1 + 1 dimensions, the difference is almost imperceptible, but for 5000 units with 5 + 5 dimensions, the run time with two iteration in the first stage is almost halved. With more iteration, it is naturally possible that the time will gradually become longer than the previous iterations, because the same number of deleted units may have been done enough in the same previous iteration and the iteration will only increase the time unnecessarily. Therefore the number of iterations may be different for different problems. We have considered the number of iterations experimentally, therefore the run time mentioned is not necessarily the best time for our method, and it may be possible to achieve a better time by changing the number of iterations. Thus the times mentioned for the proposed method, is the upper bound for the best possible time. Suppose  $PT$  indicates Percentage reduction in run time whit optimal iteration (percent).

We now run the problem for 50000 units with different dimensions:

**Table 2:** Comparing the runt time of the method of [18] and our proposed method, with different iterations, for different dimensions.

N. of units	Dimensions	$T_{kh}$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$PT$
50000	1+1	90.16	78.47	-	-	-	-	12.66%
50000	2+2	123.81	80.61	-	-	-	-	34.89%
50000	3+3	133.43	86.71	147.6	-	-	-	35.01%
50000	4+4	213.41	158.92	140.69	152.25	-	-	33.95%
50000	5+5	380.86	-	-	344.92	250.90	290.30	34.12%

As we can see, for a large number of units with small dimensions, the difference in run time, considering the randomness of the data, is about

12 to 35% and as the size of the problem increases, this difference will naturally increase. Given that we have used parallel processing with 16 cores for run our method, and at the same time we run the [18], method with the same 16 cores, the calculation time of the method of [18], which used 8 cores in parallel processing, has been reduced. For example, they calculated the run time for a problem with 50000 DMU with  $5+5$  dimensions 1039 seconds. In our method, by using 16 parallel processors their time is reduced to approximately 381 seconds, i.e. 63% reduction in time only due to the increase in the number of processors in parallel processing. On the other hand, by applying the proposed method, we have been able to reduce the 34% in the same 381 seconds and reduce it to approximately 251 seconds.

**Table 3:** Comparison of the run time of the method of [18] and our proposed method, with different iterations, (dimensions:  $6+6$ ).

N. of units	Dimensions	$T_{kh}$	$T_6$	$T_7$	$T_8$	$PT$
50000	6+6	1690.79	935.39	933.76	1023.90	44.77%

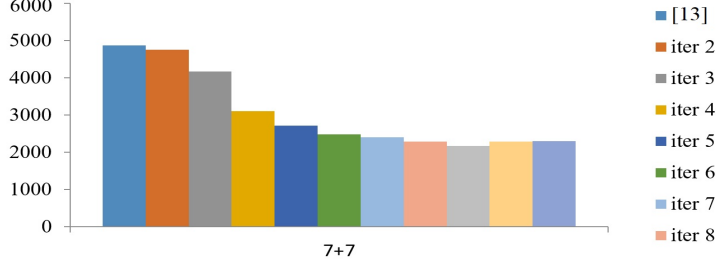
For 50000 units with dimensions of  $6+6$ , the run time of our method is reduced by 44% compared to the method of [18].

To show the effect of increasing the iteration, we have run the problem with 50000 units and dimensions of  $7+7$  in different iterations, which can be seen in the table 4:

**Table 4:** Comparison of the run time of the method of [18] and our proposed method, with different iterations, (dimensions:  $7+7$ ).

N. of units	Dimensions	$T_{kh}$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$
50000	7+7	4870.88	4749.27	4161.19	3101.22	2718.09	2477.71
		$T_7$	$T_8$	$T_9$	$T_{10}$	$T_{11}$	$PT$
		2400.58	2287.98	2176.07	2288.67	2298.78	44.67%

For 50000 units with dimensions of  $7+7$  with 9 iterations, the run time of our proposed method is reduced by 44% compared to the method of [18], which shows the need for iteration in the first stage. The chart below shows the time reduction for  $7+7$  in different iterations.



**Figure 4:** Comparison diagram of the run time of the method of [18] and our proposed method, with different iterations, (dimensions:  $7 + 7$ ).

**Table 5:** Comparison of run time of method of [18] and our proposed method, with different iterations, for larger problems.

N. of units	Dimensions	$T_{kh}$	$T_9$	$T_{11}$	$T_{13}$	$T_{14}$	$PT$
50000	8+8	16014.73	7498.97	6919.01	7436.23	-	56.79%
50000	9+9	39496.67	-	-	-	16631.12	57.89%
50000	10+10	95891.46	-	-	-	41066.81	57.17%

In their article, [18] calculated the run time of 50000 units with dimensions of  $9 + 9$  with 8 cores, 59994.77 seconds, i.e. approximately 16 hours and a half. We calculated their method with 16 cores, 39496.67 seconds which is approximately 11 hours. By applying our proposed method, we reduced the time to 16631.12 seconds, which is approximately 4 hours and a half. This means that we have a 57% reduction in run time compared to the method of [18]. This difference in time is more noticeable in  $10 + 10$  and the time difference is about 15 hours and a half. This demonstrates the efficiency of our proposed method in problems with a large number of units and large dimensions compared to their method [18]. Note that the number of iterations is considered experimentally and that they are not necessarily optimal.

**The real data used in the article:**

E-GRID is a comprehensive source of data on the environmental characteristics of all electric power generated in the United States. These environmental characteristics include air emissions for nitrogen oxides, sulfur dioxide, carbon dioxide, methane, and nitrous oxide. This infor-

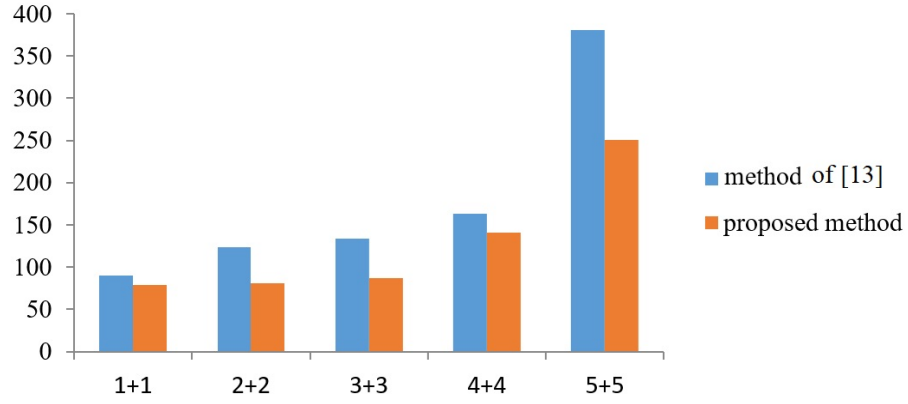
mation can be used in DEA analysis of a plants efficiency and its impact on its carbon foot print. GRID2016 was released in February 2016, and includes data from 1996 through 2023. The data for some years and some particular DMUs were not available. Overall, we could select 22444 power plants (DMUs) and the following DEA inputs and outputs. The two inputs are (1) plant annual total heat used to generate the electricity, measured Millions of British Thermal Units (MMBtu), and (2) net generator (nameplate) capacity measured in Megawatt (MW) units. The outputs include (1) good output: the net electricity generated in MW hours (MWh), and (2-4) three bad outputs: annual NOx, SO2 and CO2 emissions, all measured in tons emitted, respectively. Each bad output is treated by subtracting the maximum value of the corresponding year with the recorded output value so that they can be used as DEA outputs [19].

**Table 6:** Comparison of run time of method of [18] and our proposed method, for E-GRID data.

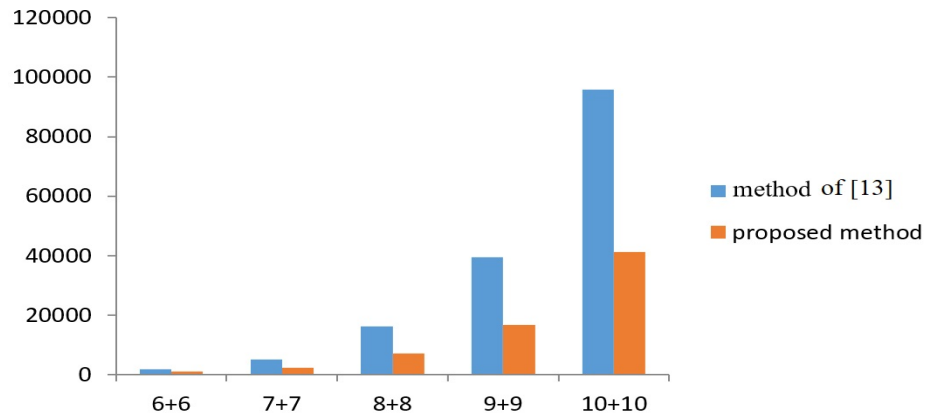
N. of units	Dimensions	$T_{kh}$	$T_4$	$PT$
22444	2+4	567.52	36.17	93.63%

In our method, we successfully eliminated 21,718 ineffective units with 4 iterations, while the [18] removed 13,512 ineffective units. As a result of eliminating a larger number of ineffective units, we faced a smaller problem compared to their method, leading to A decrease of 93 percent in run time compared to their approach.

The following figures shows a comparison of our method with the method of [18].



**Figure 5:** Comparison diagram of the run time of the proposed method with the method of [18].



**Figure 6:** Comparison diagram of the run time of the proposed method with the method of [18].

## 5 Conclusion

Reducing runtime in the presence of big data has garnered significant attention among DEA researchers. In this study, we developed an effective method to minimize the computational time required to solve large-scale DEA problems. By injecting sub-samples in multiple stages during the initial phase of our method, we were able to achieve substantial reductions in runtime for problems involving large numbers of DMUs and high-dimensional datasets. The need for such methods becomes increasingly critical as the size and complexity of DEA models grow. Moreover, our approach can be adapted to other DEA models beyond the one presented here. Future research could explore determining the optimal number of iterations for high-dimensional problems and investigating the potential benefits of combining our method with the Build-Hull (BH) approach. This could further enhance the efficiency of solving large-scale DEA problems.

## References

- [1] A.I. Ali, Streamlined computation for data envelopment analysis, *European Journal of Operational Research*, 64(1) (1993), 61-67.
- [2] R.D. Banker, A. Charnes and W.W. Cooper, Some models for estimating technical and scale inefficiencies in data envelopment analysis, *Management Science*, 30(9) (1984), 1078-1092.
- [3] R.S. Barr, M.L. Durchholz, Parallel and hierarchical decomposition approaches for solving large-scale data envelopment analysis models, *Annals of Operations Research*, 73 (1997), 339-372.
- [4] M. S., Bazaraa, J. J., Jarvis, H. D., Sherali *Linear programming and network flows*, Sep28, John Wiley & Sons, USA (2011).
- [5] W.C. Chen, W.J. Cho, A procedure for large-scale DEA computations, *Computers & Operations Research*, 36(6) (2009), 1813-1824.
- [6] N. Castellano, R.D. Cobbo and L. Leto, Using Big Data to enhance data envelopment analysis of retail store productivity, *International*

*Journal of Productivity and Performance Management*, 73 (2023), 213-242.

- [7] A. Charnes, W.W. Cooper and E. Rhodes, Measuring the inefficiency of decision making units, *European Journal of Operational Research*, 2(6) (1978), 429-444.
- [8] v. Charles, T. Gherman and J. Zhu, Data Envelopment Analysis and Big Data: A Systematic Literature Review with Bibliometric Analysis, *International Series in Operations Research & Management Science* , 312 (2021), 1-29.
- [9] W.C. Chen, S.Y. Lai, Determining radial efficiency with a large data set by solving small-size linear programs, *Annals of Operations Research*, 250(1) (2017), 147-166.
- [10] J. Chu, Y. Rui, D. Khezrimotlagh and J. Zhu, A general computational framework and a hybrid algorithm for large-scale data envelopment analysis, *European Journal of Operational Research*, 316(2) (2024), 639-650.
- [11] J.H. Dula, A computational study of DEA with massive data sets, *Computers & Operations Research*, 35(4) (2008), 1191-1203.
- [12] J.H. Dula, An algorithm for data envelopment analysis, *INFORMS Journal on Computing*, 23(2) (2011), 284-296.
- [13] J.H. Dula, R.V. Helgason, A new procedure for identifying the frame of the convex hull of a finite collection of points in multidimensional space, *European Journal of Operational Research*, 92(2) (1996), 352-367.
- [14] J.H. Dula, R.V. Helgason and N. Venugopal, An Algorithm for Identifying the Frame of a Pointed Finite Conical Hull, *INFORMS Journal on Computing*, 10(3) (1998), 323-330.
- [15] J.H. Dula, F. J. Lopez, DEA with streaming data, *Omega*, 41(1) (2013), 41-47.
- [16] J. H. Dula, R. M. A. Thrall, Computational framework for accelerating DEA, *Journal of Productivity Analysis*, 16(1) (2001), 63-78.

- [17] P.J. Korhonen, P. A. Siitari, A dimensional decomposition approach to identifying efficient units in large-scale DEA models, *Computers & Operations Research*, 36(1) (2009), 234-244.
- [18] D. Khezrimotlagh, J. Zhu and W. Cook, and M. Toloo, Data envelopment analysis and big data, *European Journal of Operational Research*, 274(3) (2018), 1047-1054.
- [19] L.M. Seford, J. Zhu, A response to comments on modeling undesirable factors in efficiency evaluation, *European Journal of Operational Research*, 161(2) (2005), 579-581.
- [20] J. Zhu, DEA under big data: data enabled analytics and network data envelopment analysis, *Annals of Operation Research*, 309(2) (2022), 761-783.
- [21] Q. Zhu, J. Wu and M. Song, Efficiency evaluation based on data envelopment analysis in the big data context, *Computers & Operations Research*, 98 (2017), 291-300.

**Roya Hosseinzadeh**

Department of Mathematics  
 PhD student in mathematics  
 Tabriz Branch, Islamic Azad University  
 Tabriz, Iran  
 E-mail: hosseinzadeh13@gmail.com

**Nima Azarmir Shotorbani**

Department of Mathematics  
 Assistant Professor of Mathematics  
 Tabriz Branch, Islamic Azad University  
 Tabriz, Iran  
 E-mail: azarmir\_nim@yahoo.com

**Yasser Jafari**

Department of Mathematics  
 Assistant Professor of Mathematics  
 Shabestar Branch, Islamic Azad University  
 Shabestar, Iran



E-mail: yassermath2006@gmail.com

**Javad Vakili**

Department of Mathematics, Statistics and Computer Science

Associate Professor of Mathematics

University of Tabriz

Tabriz, Iran

E-mail: j.vakili@tabrizu.ac.ir