A Two-phased Gradient Technique for Budget Allocation: An Application to Data Collection Budget Allocation in Efficiency Measurement of DEA.

Wai Peng Wong
School of Management,
Universiti Sains Malaysia,
Penang, Malaysia
wongwp@usm.my

Loo Hay Lee
Department of Industrial &
Systems Engineering,
National University of Singapore
Singapore.
iseleelh@nus.edu.sg

Wikrom Jaruphongsa
Sasin Graduate Institute of
Business Administration
Chulalongkorn University
Thailand
wikromj@sasin.edu

***Abstract*—This paper presents a gradient based algorithm to solve the data collection budget allocation problem in efficiency measurement of DEA. A hard limit is put on how many total readings can be collected and we aim to determine how these should be allocated among the different attributes. A two-phased gradient technique is developed to solve this problem. The numerical results showed that the proposed gradient based algorithm performs well in solving the problem. The insight from the results reveal that, it is important to allocate the budget intelligently and the proposed algorithm serves as a potential method in this area.**

Keywords- component: DEA, budget allocation, data collection, gradient

I. INTRODUCTION

Budget allocation is a common task that has to be carried out in our daily activities, whether in work or personal life. The term budget can refer to money, workforce, time or any resources that contribute to the activities. Very often, we have hard time to determine how to allocate the budget in the best way; that is 'effectively' in order to get the best results that we desire and 'efficiently' in the shortest possible time. Simply allocate the budget naively may made us end up with getting the results which we do not want and waste of resources due to inefficiencies of management.

In this paper, we present a budget allocation problem for data collection. DEA (Data Envelopment Analysis) is chosen as the context of study. DEA is a data driven method, where the user needs to collect data for efficiency estimation. Therefore, addressing this budget allocation problem in the context of data collection in DEA is reasonable and pose an interesting problem to be tackled. Our research statement goes as follows: "Given that the budget for data collection (for the measurement of the means of the stochastic inputs/outputs attributes) in DEA are limited, how can the users effectively and efficiently allocate the budget among the different stochastic inputs/outputs in order to obtain a better estimate of the efficiency score?"

In the following section, the mathematical model for the budget allocation for data collection in DEA will be presented, followed by the explanation of the gradient based algorithm to solve the problem and lastly, results and discussion.

II. BUDGET ALLOCATION FOR DATA COLLECTION IN DEA

A basic DEA model, or more widely known as the CCR (Charnes-Cooper and Rhodes) model is given in (1). Let S be the set of inputs and R the set of outputs, where S and R are disjoint sets ($S \cap R = \varnothing$). Denote K as the set of combined inputs/outputs; i.e., $K = S \cup R$. J is the set of DMUs. Let $\mathbf{X_D} = (x_{kj})_{k \in K;\, j \in J}$, where x_{kj} represents k-th input/output for DMU j. If $k \in S$, then x_{kj} is an input; otherwise if $k \in R$, then x_{kj} is an output. Given that $\mathbf{X_D}$ is the matrix for the initial data of the inputs/outputs, the efficiency score for DMU j_0 denoted by $\theta(\mathbf{X_D})$ can be computed using (1).

$$
\begin{aligned}
\theta(\mathbf{X_D}) = \min \quad & \theta \\
\text{s.t.} \quad & \sum_{j \in J} \lambda_j x_{sj} \le \theta x_{sj_o} && s \in S \\
& \sum_{j \in J} \lambda_j x_{rj} \ge x_{rj_o} && r \in R \\
& \lambda_j \ge 0, && j \in J \qquad (1)
\end{aligned}
$$

The λ_j's are the weights of the inputs/outputs that optimize the efficiency score of each DMU j_0. The technical efficiency score will be equal to one if a DMU is efficient and less than one if it is inefficient. Note that in the CCR model, if the optimal solution $\theta^* = 1$ and has zero-slack, then the DMU is called CCR-efficient. Otherwise, it is CCR-inefficient. Both conditions i.e. $\theta^*=1$ and all slacks are zero must be satisfied if full efficiency is to be attained [1]. The efficiency value represents the proportion by which all inputs must be reduced in order to become efficient. For more detail explanation about DEA efficiency, please refer to Cooper et al. in [1].

By using the Bayesian approach to determine the mean values of the inputs/outputs if the inputs/outputs are stochastic, data collection (for the inputs/outputs) has to be carried out in order to estimate the distribution of the efficiency score [2]. Note that the spread of the distribution of the efficiency score depends on the data collected for the inputs/outputs. Given that data collection is expensive and

often limited to certain budget, one has to address "how should he/she allocate the budget – as to how many data should be collected for each input/output in order to get a better estimate of the efficiency". Thus, this validates the subjectivity of budget allocation in DEA.

The model for data collection budget allocation in DEA, is given in (2). This model estimates the distribution of the efficiency score through Monte Carlo method.

$$\min F(\mathbf{n}) \approx \frac{1}{M}\sum_{i=1}^{M}(\theta(\hat{\mathbf{X}}_{[i]}) - \theta(\mathbf{X}))^2$$
$$s.t. \quad \sum_{k\in K} n_k = N \qquad (2)$$

Some notations: $\mathbf{X}$ is the matrix of sample averages based on initial data of inputs/outputs, N denotes the budget for additional samples. The allocation design is given by $\mathbf{n} = [n_k]_{k\in K}$, where n_k represents the number of additional data collected for input/output k. The objective function $F(\mathbf{n})$ is defined as the mean square error (MSE) of the efficiency score for allocation design $\mathbf{n}$ where $\hat{\mathbf{X}}_{[i]}$ is the realization of the inputs/outputs in the replication i of the Monte Carlo run for allocation design $\mathbf{n}$ and M is the cardinality of random data set. For further details, please refer to [2].

As there is no close-form formulation to compute MSE for a given allocation design $\mathbf{n}$ in model (2), to find an optimal solution (or allocation), a search-based method is used. The concept of the approach is first to generate some designs (i.e. different allocations $\mathbf{n}$); then estimate their MSE using Monte Carlo simulation; and repeat the process until the design with the best MSE is found. The search based method will be developed in the form of a two-phased gradient technique, which will be presented in the following section.

III. Two- Phased Gradient Technique

Given an allocation design, first we estimate the MSE using the Monte Carlo DEA approach; then we find the gradient of MSE, which provides a direction for finding a new allocation design that may have a lower MSE value. There are two phases in the approach. In the first phase, we find the gradient using Infinitesimal Perturbation Analysis (IPA) [3] and then implement the hill climbing algorithm. The second phase is a gradient-based improvement approach on fine tuning the solutions found in the first phase.

A. First phase

When we estimate the MSE using Monte Carlo method, at the same time, we can also estimate the gradient without rerunning simulation by using IPA. The idea of IPA is to consider how perturbation in a parameter affects the changes of the random variables generated and eventually how it changes the performance of the system. However, the IPA adopted here is a simplified version for a typical IPA method as it does not involve the dynamic of the system. Taking a derivative of (2) with respect to the number of additional data collected for input/output k (n_k), we obtain

$$\frac{\partial F(\mathbf{n})}{\partial n_k} \approx \left(\frac{1}{M}\sum_{i=1}^{M} 2(\theta(\hat{\mathbf{X}}_{[i]}) - \theta(\mathbf{X}))\right)\left(\frac{\partial\theta(\hat{\mathbf{X}}_{[i]})}{\partial\hat{x}_{k[i]}}\right)\left(\frac{\partial\hat{x}_{k[i]}}{\partial n_k}\right) \qquad (3)$$

Note that $\frac{\partial\theta(\hat{\mathbf{X}}_{[i]})}{\partial\hat{x}_{k[i]}}$ can be found using the sensitivity analysis for LP and $\frac{\partial\hat{x}_{k[i]}}{\partial n_k}$ is the perturbation in samples when the parameter is perturbed. For details of the derivation of each term, please refer to [4]. Eventually, using chain rules, the gradient of the performance (MSE) with respect to $\mathbf{n}$, i.e., $\nabla\mathbf{f}_{\mathbf{n}} = \frac{\partial F(\mathbf{n})}{\partial\mathbf{n}}$ is derived as:

$$\nabla f_{n_k} \approx \frac{2}{M}\sum_{i=1}^{M}\left[\left(\theta(\hat{\mathbf{X}}_{[i]}) - \theta(\mathbf{X})\right)\cdot\left(\theta'(\hat{\mathbf{X}}_{[i]}) - \lambda'_{[i]}\right)\right.$$
$$\left.\pi_{k[i]}\cdot\left(-\frac{Z\sigma_k}{2(n_{ok}+n_k)^{3/2}}\right)\right]$$

if $k \in S$,

otherwise

$$\nabla f_{n_k} \approx \frac{2}{M}\sum_{i=1}^{M}\left[\left(\theta(\hat{\mathbf{X}}_{[i]}) - \theta(\mathbf{X})\right)\cdot(1-\lambda'_{[i]})\pi_{k[i]}\cdot\left(-\frac{Z\sigma_k}{2(n_{ok}+n_k)^{3/2}}\right)\right]$$

if $k \in R$.

After finding the gradient for a given starting/current design, we then find the move direction and decide how far to move so that a lower value of MSE is obtained. After moving to the new design, we set it as the current design and repeat the process until the termination condition, i.e., $\sum_{k\in K} n_k = N$ is reached. Let us denote $\mathbf{n}_{(t)}$ as the starting allocation design for iteration t and $\mathbf{d}_{(t)} = [d_{k(t)}]_{k\in K}$ as the move direction used during iteration t.

For our problem, the move direction has to be controlled so that the total number of allocation increases and eventually reaches the budget N. To achieve this, we have to address the followings: (i) the move direction must be determined from negative gradient as we are minimizing the objective values, (ii) the allocations must be rounded off to maintain integrality and budget requirements, and (iii) determine an appropriate step size. The hill climbing algorithm is shown as below.

Algorithm 1: First phase (hill-climbing)

Step 0: Initialization: Set $\mathbf{n}_{(1)} = \mathbf{0}$ and $t = 1$.

Step 1: Gradient: Compute $\nabla f_{n_{k(t)}}$ using Lemmas 1-3.

Step 2: Direction: Determine the direction $\mathbf{d}_{(t)}$.

Step 3: Step Size: Set the appropriate step size, δ.

Step 4: New Design: Set $\mathbf{n}_{(t+1)} = \mathbf{n}_{(t)} - \delta\mathbf{d}_{(t)}$. Round off $\mathbf{n}_{(t+1)}$ to obtain the new design.

Step 5: Termination: If $\sum_{k\in K} n_{k(t+1)} < N$, set $t = t + 1$ and return to Step 1; otherwise, stop.

It is possible that the summation of gradient is positive, i.e., $\sum_{k\in K} \nabla f_{n_{k(t)}} > 0$. We need to modify some of its direction so that $\sum_{k\in K} \nabla f_{n_{k(t)}} < 0$ by selecting a value of multiplier β such that $\sum_{k\in K} d_{k(t)} > 0$, as shown in (4) below.

$$d_{k(t)} = \begin{cases} -\nabla f_{n_{k(t)}} & \text{if } \nabla f_{n_{k(t)}} < 0 \\ -\beta \nabla f_{n_{k(t)}} & \text{otherwise} \end{cases} \quad (4)$$

As the step size δ cannot be too small (it may not move at all) nor too large (it may move too far away), we determine the appropriate value by choosing the greater between the increment number of allocation by a fraction, e.g., ϕ of N or increment of n_k for at least one $k \in K$, i.e.,

$$\delta = \max\left\{ \frac{\phi N}{\left\{-\sum_{k\in K} \nabla f_{n_{k(t)}}\right\}}, \frac{0.5}{\max\{-\nabla f_{n_{k(t)}}\}} \right\} \quad (5)$$

When the total number of allocation exceeds the budget, i.e., $\sum_{k\in K} n_{k(t+1)} > N$ after rounding off, we will adjust some allocations downward so that the total number of budget is equal to N.

The first phase only guides the search to reach a solution at the boundary; which is a feasible solution, but may not be a good solution. In order to explore the other solutions on the boundary which can give better performance, an improvement stage is needed.

B. Second phase (Gradient improvement stage)

The Gradient Improvement Stage (GIS) aims to perform some neighbourhood search around the design so as to further improve the solution quality. Neighbourhood here is defined as the set of feasible solutions which are near to the current design/solution. The overall concept is: given a current design, first we identify the feasible neighbourhood; then, we select which design from the neighbourhood that we should move to and update it as current design; and the entire process (identifying neighbourhood and selection) is repeated until the best design, i.e., the design with the lowest MSE is found.

Before defining the neighbourhood, first we need to find the direction that we can move from the current solution. Note that this direction, d_k should have a good potential to improve the current objective value, i.e., it should minimize $\sum_{k\in K} \nabla f_{n_k} d_k$. To maintain integrality of the solution, we set d_k = -1, 0 or 1. As we only want to explore solutions that are at the boundary, it is required that $\sum_{k\in K} d_k = 0$; in addition, $n_k + d_k \geq 0$ for all $k \in K$. Note that with these requirements, the number of d_k's having the value of +1 must be equal to those having the value of -1. For our desired direction, we also want to control the number of d_k's that have a nonzero value. With this reason, we impose a constraint $\sum_{k\in K} |d_k| \leq 2L$, where L is the maximum number of pairs of +1 and -1 direction. Hence, the mathematical model for finding a direction for the second phase is as follows:

$$\begin{aligned} \min \quad & \sum_{k\in K} \nabla f_{n_k} d_k \\ \text{s.t.} \quad & \sum_{k\in K} d_k = 0 \\ & \sum_{k\in K} |d_k| \leq 2L \\ & n_k + d_k \geq 0, \quad k \in K \\ & d_k \in \{-1, 0, 1\} \quad k \in K \end{aligned} \quad (6)$$

Let $\mathbf{d}^*$ denote the optimal direction obtained from (8). We use the solution from the first phase as the starting point/design $n_{(1)}$. In iteration t, after an improving direction $\mathbf{d}_{(t)}$ is found, we define the neighbourhood as the set of solutions $\mathbf{n}_{(t)} + \gamma \mathbf{d}_{(t)}$, γ= 1, 2, ..., and to maintain feasibility, i.e., $\mathbf{n}_{(t)} + \gamma \mathbf{d}_{(t)} \geq 0$), it is required that $\gamma \leq \min_{k(t)}\{n_{k(t)} : d_{k(t)} = -1\}$. Let $A_{(t)}$ be the set of feasible neighbourhood for design $\mathbf{n}_{(t)}$ and is given by

$$A_{(t)} = \left\{ \mathbf{n}_{(t)} + \gamma \mathbf{d}_{(t)} : \gamma = 1, 2, ..., \min_{k(t)}\{n_{k(t)} : d_{k(t)} = -1\} \right\} \quad (7)$$

In order to select a solution in $A_{(t)}$ that we should move to, we evaluate each of them in terms of two metrics, i.e., the current performance (MSE) and the potential (good future) of the solution, which can be defined as follows: $V(\mathbf{n}) = M(\mathbf{n}) + \alpha \nabla \mathbf{f}_{\mathbf{n}} \mathbf{d}$ (8)

where $M(\mathbf{n})$ denotes the MSE of design $\mathbf{n}$ and α is a given constant which relates a linear relationship between the potential, i.e., $V(\mathbf{n})$ and the improvement in performance, i.e., $\nabla \mathbf{f}_{\mathbf{n}} \mathbf{d}$. The greater the potential of the design, the more improvement in performance (reduction in MSE) may be expected from the design for the future move. For those designs/solutions which are good at both, i.e., with the smallest $M(\mathbf{n})$ and smallest $V(\mathbf{n})$), we will move to those solutions. Alternatively, if only $V(\mathbf{n})$ is good, we will also move to that solution but later we may have to explore the other solution which have the best $M(\mathbf{n})$ if we have not explored it yet. Following this concept, we will now illustrate the details of our GIS algorithm.

GIS algorithm

Some additional notations are: $\mathbf{n}_{\text{best}}$ = best solution/design that we have found (the solution with lowest MSE), $\mathbf{n}_{\text{bv}} = \arg\min_{\mathbf{n}\in A_{(t)}} V(\mathbf{n})$ = solution with best potential in $A_{(t)}$ and $\mathbf{n}_{\text{bm}} = \arg\min_{\mathbf{n}\in A_{(t)}} M(\mathbf{n})$ = solution with best MSE in $A_{(t)}$. In our approach, we will always explore the most potential solution first and keep the best solution to be explored later. The flag 'unexplore' indicates whether the best solution kept has already been explored or not. Every time, when we found a new best solution, we set unexplore = 1. The unexplore flag will be set to 0 if the best solution will be explored in the

next iteration. Suppose that we are currently in iteration *t*. We will now discuss how to select the solution to be explored in the next iteration, i.e., $\mathbf{n}_{(t+1)}$. There are four cases to be considered.

Case 1: $V(\mathbf{n}_{bv}) < V(n_{best})$ and $M(\mathbf{n}_{bm}) < M(\mathbf{n}_{best})$

In this case, we have found the new best solution $\mathbf{n}_{bm}$; that is, we set $\mathbf{n}_{best} = \mathbf{n}_{bm}$ and unexplore = 1. The best potential solution is n_{bv} which will be explored next; that is, we set $\mathbf{n}_{(t+1)} = \mathbf{n}_{bv}$.

Case 2: $V(\mathbf{n}_{bv}) \geq V(\mathbf{n}_{best})$ and $M(\mathbf{n}_{bm}) < M(\mathbf{n}_{best})$

In this case, we also have found the new best solution $\mathbf{n}_{bm}$ while the current best solution is the most potential. If the current best solution has not been explored, i.e., unexplore = 1, it will be the next solution to consider ($\mathbf{n}_{(t+1)} = \mathbf{n}_{best}$). Otherwise, if the current best solution has been explored, i.e., unexplore = 0, then the next most potential solution is $\mathbf{n}_{bv}$; thus, we set $\mathbf{n}_{(t+1)} = \mathbf{n}_{bv}$. After that, we update $\mathbf{n}_{best} = \mathbf{n}_{bm}$ and set unexplore = 1.

Case 3: $V(\mathbf{n}_{bv}) < V(\mathbf{n}_{best})$ and $M(\mathbf{n}_{bm}) \geq M(\mathbf{n}_{best})$

In this case, the best potential solution is $\mathbf{n}_{bv}$, therefore we set $\mathbf{n}_{(t+1)} = \mathbf{n}_{bv}$. The best solution, $\mathbf{n}_{best}$, remains unchanged.

Case 4: $V(\mathbf{n}_{bv}) \geq V(\mathbf{n}_{best})$ and $M(\mathbf{n}_{bm}) \geq M(\mathbf{n}_{best})$

In this case, the best solution remains unchanged and it also indicates that the current neighbourhood is not good at all. If the best solution has not been explored, i.e., unexplore = 1, we set $\mathbf{n}_{(t+1)} = \mathbf{n}_{best}$; otherwise, i.e., unexplore = 0, we stop.

In general, when n(t+1)= nbest , i.e., the local optimal solution is nearby, we reduce α proportionately so that we can keep exploring. The algorithm will terminate by itself once it has found the optimal solution. To avoid the same solutions being explored again, we set

$$A_{(t)} = \left\{ \mathbf{n}_{(t)} + \gamma \mathbf{d}_{(t)} : \gamma = 1,2,..., \min_{k(t)} \left\{ n_{k(t)} : d_{k(t)} = -1 \right\} \text{ and } \mathbf{n}_{(t)} + \gamma \mathbf{d}_{(t)} \notin S \right\} \quad (9)$$

where S is the updated set of all the neighbourhood solutions of $A_{(t)}$; initially, $S = \{\mathbf{n}_{(1)}\}$; as iteration proceeds, $S \leftarrow S \cup A_{(t)}$. For the detailed pseudo-code for the GIS algorithm, readers may refer to [4].

IV. Numerical Results

All models and algorithms are coded in Matlab (version 6.5) and tested on an Intel Pentium IV 2.6 GHz CPU with 512 MB RAM under the Microsoft Windows XP Operating System. The results obtained from the gradient method are compared against the 'uniform' allocation method where the data collection budget is equally allocated. Experiments are performed using the data sets and the supply chain model from Wong et al. [5]. Note the *D* represents the total number of stochastic inputs/outputs.

TABLE I. Comparison of N and savings

D	MSE	Gradient	Uniform	Savings
5	0.25288	30	95	3.17
5	0.19508	60	375	6.25
5	0.15173	90	580	6.44
10	0.01055	30	1590	53.00
10	6.94E-03	60	3375	56.25
10	3.69E-03	90	4850	53.89

The results showed that the savings can be very significant, which is as high as 50 times. Moreover, the saving increases when the size of the problem increases.

In terms of computational time (refer Table 2), the two-phased gradient technique are compared against two other heuristics technique, i.e., greedy and batch. Average CPU time taken by gradient is much lesser compared to the other two heuristics techniques.

TABLE II. Average CPU time

	Gradient	Greedy	Batch
Average(hrs)	0.3	2.7	3.0

V. Conclusion

This paper developed a two-phased gradient technique to solve the budget allocation problem. The context of the problem is focused on the data collection budget allocation in DEA. The results showed that the gradient technique performs well and yield satisfactory results. In addition, this paper provides a significant insight that it is very important to use a sophisticated technique to allocate the budget effectively and efficiently. The proposed two-phased technique serves as a potential tool in this area. For future research, we can look into ways of improving the proposed algorithm. This will include developing other neighbourhood structures to solve the model more efficiently and yield better solutions.

References

[1] Cooper, W.W., Seiford, L.M., Tone, K. Introduction to Data Envelopment Analysis and its Uses: with DEA-Solver Software and Reference, Springer, New York. 2006.

[2] Lee, L.H., Wong, W.P., Jaruphongsa, W., "Data Collection Budget Allocation (DCBA) for Stochastic Data Envelopment Analysis," Proceedings of the 2009 INFORMS Simulation Society Research Workshop, by L.H. Lee, M.E. Kuhl, J.W. Fowler and S.Robinson, eds

[3] Y.C. Ho, X.R. Cao, Discrete Event Dynamic Systems and Perturbation Analysis, Kluwer Academic Publishers, Boston, UK, 1991.

[4] Wong, W.P., Lee, L.H., Jaruphongsa, W. "Budget Allocation for effective data collection in prediction of an accurate efficiency score", working paper.

[5] Wong,W.P., Jaruphongsa, W., Lee, L.H., "Supply Chain Measurement System – A Monte Carlo DEA based approach," International Journal of Industrial and Systems Engineering, vol. 3(2), 2008, pp. 162-188.