Mathematics

http://www.aimspress.com/journal/Math

AIMS Mathematics, 9(1): 642–664.
DOI: 10.3934/math.2024034
Received: 18 August 2023
Revised: 16 October 2023
Accepted: 18 October 2023
Published: 01 December 2023

Research article

An optimal choice Dai-Liao conjugate gradient algorithm for unconstrained optimization and portfolio selection

Jamilu Sabi'u[1], Ibrahim Mohammed Sulaiman[2,3,∗], P. Kaelo[4], Maulana Malik[5] and Saadi Ahmad Kamaruddin[2,∗]

[1] Department of Mathematics, Yusuf Maitama Sule University Kano, Nigeria
[2] Institute of Strategic Industrial Decision Modelling, School of Quantitative Sciences, Universiti Utara Malaysia, Sintok 06010, Malaysia
[3] Faculty of Education and Arts, Sohar University, Sohar 311, Oman
[4] Department of Mathematics, University of Botswana, Private Bag UB00704, Gaborone, Botswana
[5] Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Indonesia, Depok 16424, Indonesia

* **Correspondence:** Email: i.mohammed.sulaiman@uum.edu.my, s.ahmad.kamaruddin@uum.edu.my.

Abstract: In this research, we propose an optimal choice for the non-negative constant in the Dai-Liao conjugate gradient formula based on the prominent Barzilai-Borwein approach by leveraging the nice features of the Frobenius matrix norm. The global convergence of the new modification is demonstrated using some basic assumptions. Numerical comparisons with similar algorithms show that the new approach is reliable in terms of the number of iterations, computing time, and function evaluations for unconstrained minimization, portfolio selection and image restoration problems.

Keywords: unconstrained optimization; BB approach; descent property; global convergence
Mathematics Subject Classification: 90C26, 90C30

1. Introduction

This paper is interested in unconstrained optimization model of the form:

$$\min_{x\in\mathbb{R}^n} f(x), \tag{1.1}$$

where f is a smooth nonlinear function, such that $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is continuously differentiable. This topic has several applications in finance, engineering, security, and scientific computing [1–7]. As a result, reliable and efficient numerical procedures for obtaining the solution of (1.1), such as Newton-type procedures, spectral gradient methods and conjugate gradient (CG) algorithms, have been widely investigated in the literature, see [8–13]. Of all these mentioned methods used for the solution of (1.1), the CG algorithms are the most extensively used because of their nice convergence properties in addition to less memory requirements [14]. Given the starting guess, $x_0 \in \mathbb{R}^n$, the CG algorithm recursively produces its iterative points via

$$x_{k+1} = x_k + \alpha_k d_k, \tag{1.2}$$

where α_k is the step size calculated based on either exact or inexact line search strategies. The vector d_k is the CG search direction with the formula

$$d_0 = -g_0, \quad d_{k+1} = -g_{k+1} + \beta_k d_k, \quad k = 0, 1, \cdots . \tag{1.3}$$

Here, β_k is the CG parameter and the gradient $g_k := \nabla f(x_k)$. This parameter measures the efficiency and reliability of various CG methods [8]. Hestenes and Stiefel [9] (HS) proposed one of the essential CG parameters, namely,

$$\beta_k^{HS} = \frac{g_{k+1}^T y_k}{d_k^T y_k},$$

in which $y_k = g_{k+1} - g_k$. The direction d_k of HS satisfies the conjugacy condition $d_{k+1}^T y_k = 0, \forall k \geq 0$, irrespective of the line search procedure employed. Dai and Liao [15] (DL) introduced a new CG parameter

$$\beta_k^{DL} = \frac{g_{k+1}^T y_k}{d_k^T y_k} - t\frac{g_{k+1}^T s_k}{d_k^T y_k}, \tag{1.4}$$

which is widely considered as an extension of HS, where t is defined as a nonnegative scalar parameter. The parameter (1.4) satisfies an extended conjugacy condition $d_{k+1}^T y_k = -t g_{k+1}^T s_k$, and it is easy to see that the parameter (1.4) reduces to β_k^{HS} for $t = 0$. Some efficient adaptive versions of (1.4) have also been presented by Hager and Zhang in [16] and Dai and Kou in [17]. Andrei [18] highlighted that the appropriate choice for t in DL method remains an open issue in this subject. This inspired Babaie-Kafaki and Ghanbari [19, 20] to use the beauty of the eigenvalue and singular values to offer some optimum alternatives for t as

$$t_{k1}^* = \frac{||y_k||}{||s_k||}, \quad \text{and} \quad t_{k2}^* = \frac{s_k^T y_k}{||y_k||^2} + \frac{||y_k||}{||s_k||}.$$

The authors also offered an additional optimal solution for t by minimizing the distance between the search direction of DL method and a three-term CG algorithm presented by Zhang et al. [21], as well as the search direction matrix's Frobenius condition number [22]. Zhang et al. [23] proposed an optimal value for t by clustering all singular values and minimizing the upper bound of the d_k matrix's spectral condition number, given as

$$t_{k3}^* = \frac{||y_k||^2}{s_k^T y_k} - \frac{s_k^T y_k}{4\,||s_k||^2}.$$

Furthermore, for information on other DL methods and their variants, we recommend the reader to [24–30] and the references therein.

In this article, we propose an optimal choice for t in the DL CG parameter based on the well-known Barzilai-Borwein (BB) approach [31]. In Section 2, we provide our choice for t based on the BB technique. Using the recommended choices for parameter t, we explore the global convergence of the DL method and is presented in Section 3. Section 4 presents numerical comparisons and Section 5 present the application of the new algorithm on portfolio selection and image restoration problems. Finally, in Section 6, we give the conclusions.

2. An optimal choice based on BB approach

This section presents an optimal choice for the DL CG method based on the prominent BB approach. Rewriting the search direction of the DL algorithm, we have

$$d_{k+1} = -H_{k+1}g_{k+1},$$

where

$$H_{k+1} = I - \frac{s_k y_k^T}{y_k^T s_k} + t_k \frac{s_k s_k^T}{y_k^T s_k}.$$

Among the excellent scaling parameters used in the spectral residual methods are ones proposed by Barzilai-Borwein [31] given by

$$\theta_k^1 = \frac{s_k^T s_k}{y_k^T s_k} \quad \text{and} \quad \theta_k^2 = \frac{s_k^T y_k}{y_k^T y_k}.$$

Now, our aim is to propose another non-negative optimal choice parameter of the DL algorithm by utilizing the prominent features of the Barzilai-Borwein [31] approach, that is, by considering the minimization problem

$$\min_t \|H_{k+1} - \theta_k I\|_F^2, \tag{2.1}$$

where $\|.\|_F^2$ denotes the norm of the Frobenius matrix, and

$$\theta_k = \max\left\{\theta_{\min}, \min\left\{\theta_k^1, \theta_k^2, \theta_{\max}\right\}\right\},$$

with $0 < \theta_{\min} < \theta_{\max} < \infty$.

If we let $D_{k+1} = H_{k+1} - \theta_k I$ and also by utilizing the beautiful properties of the Frobenius matrix norm, the minimization problem (2.1) is equivalent to minimizing trace$(D_{k+1}^T D_{k+1})$. Now, using

$$D_{k+1} = (1-\theta_k)I - \frac{s_k y_k^T}{y_k^T s_k} + t\frac{s_k s_k^T}{y_k^T s_k},$$

and simple algebra, by minimizing

$$\text{trace}(D_{k+1}^T D_{k+1}) = t^2 \frac{\|s_k\|^4}{(y_k^T s_k)^2} - 2t\theta_k \frac{\|s_k\|^2}{y_k^T s_k} + \psi,$$

where ψ represents terms independent of t, we derive another optimal choice parameter for the DL CG method as

$$t_k^* = \theta_k \frac{y_k^T s_k}{s_k^T s_k}. \tag{2.2}$$

Notice here that when $\theta_k = 1$, then we have the MDL3 method by Saman and Ghanbari [22]. Moreover, since the study considered the condition from strong Wolfe line search, then, $d_k^T y_k \geq -(1-\sigma)d_k^T g_k > 0$, and in this situation $t_k^* = \theta_k \frac{y_k^T s_k}{s_k^T s_k} > 0$.

The following algorithm demonstrates the computational process of the proposed method under strong Wolfe conditions.

Algorithm 1: Algorithm for DLBB method under strong Wolfe conditions.

Input : Initializing $x_0 \in \mathbb{R}^n$, and $0 < \epsilon < 1$ as tolerance.

Step 1 : To control the process, check **if** $\|g_k\| = 0$, **then**
  terminate.
end
Step 2 : **if** $k = 0$, **then**
  set $d_k := -g_k$;
else
  Calculate the new d_k as:

$$d_k = -g_k + \beta_k d_{k-1}.$$

  where β_k is the CG coefficient defined by (1.4) with the new optimal parameter t calculated via (2.2).
end
Step 3 : Obtain α_k based on the following conditions of the strong Wolfe (SWP) strategies

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \delta \alpha_k g_k^T d_k, \tag{2.3}$$
$$|g(x_k + \alpha_k d_k)^T d_k| \leq \sigma |g_k^T d_k|, \tag{2.4}$$

where $0 < \delta < \sigma < 1$.
Step 4 : Calculate the next iterative point by (1.2).
Step 5 : Restart the process from Step 1 with $k := k + 1$.

3. Convergence analysis

This section will discuss the convergence analysis of the new choice parameter t for the DL method defined by (2.2).

To achieve the convergence of the proposed formula using the new choice parameter t, the assumptions defined below in addition to the Zoutendijk condition would be needed.

Assumption A.

(1) Given a starting point $x_0 \in \mathbb{R}^n$, the level set $\Omega = \{x \in \mathbb{R}^n | f(x) \leq f(x_0)\}$ of $f(x)$ is bounded.
(2) f is a smooth function in some neighborhood N of Ω and the gradient $g(x)$ is Lipschitz continuous on an open convex set N containing Ω, in such a way that a constant $L > 0$ exists and satisfies

$$\|g(x) - g(y)\| \leq L\|x - y\|, \ \forall x, y \in N.$$

For the function f, this assumption implies that there exists a constant $\gamma > 0$ satisfying

$$\|g(x)\| \leq \gamma \ \ \forall x \in N.$$

It is important to note here that the sufficient descent condition

$$d_k^T g_k \leq -c\|g_k\|^2, \quad c > 0, \tag{3.1}$$

for a DL method (1.4) with $t = t_k^*$ cannot always be guaranteed, in which case the steepest descent direction $d_k = -g_k$ is used. The result that follows is very important in the analysis of CG formulas and was presented by Zoutendijk [32].

Lemma 3.1. *Let Assumption A hold. Then, for any iteration scheme of the form* (1.2) *and* (1.3)*, where* d_k *is a descent direction and the step size is computed using the Wolfe strategies,*

$$\sum_{k=0}^{\infty} \frac{(g_k^T d_k)^2}{\|d_k\|^2} < +\infty.$$

Proof. From (2.4), if the curvature condition

$$g_{k+1}^T d_k \geq \sigma g_k^T d_k, \ \sigma < 1,$$

holds, then, using the Lipschitz condition, we obtain that

$$-(1-\sigma) g_k^T d_k \leq d_k^T (g_{k+1} - g_k) \leq L\alpha_k \|d_k\|^2.$$

This implies that

$$\alpha_k \geq \frac{(1-\sigma)|g_k^T d_k|}{L\|d_k\|^2}. \tag{3.2}$$

And from the Armijo condition (2.3), we get that

$$\frac{\delta(1-\sigma)}{L} \frac{(g_k^T d_k)^2}{\|d_k\|^2} \leq -\delta\alpha_k g_k^T d_k \leq f(x_k) - f(x_{k+1}),$$

which on summing over k, and using the fact that f is bounded from below, gives

$$\sum_{k=0}^{\infty} \frac{(g_k^T d_k)^2}{\|d_k\|^2} < +\infty.$$

□

Now, from the strong Wolfe conditions, (3.1) and (3.2), we conclude that there exists a constant $\bar{\alpha} > 0$ such that

$$\alpha_k \geq \bar{\alpha}, \ \forall k \geq 0.$$

For strong Wolfe line search conditions, we have the following important results for conjugate gradient methods.

Lemma 3.2. *[33] Let Assumption A hold. Then, for any iteration scheme of the form* (1.2) *and* (1.3)*, where d_k satisfies the descent condition* (3.1)*, with the step length computed using the SWP strategies* (2.3) *and* (2.4)*, either*

$$\sum_{k=0}^{\infty} \frac{||g_k||^4}{||d_k||^2} < +\infty \tag{3.3}$$

or

$$\liminf_{k\to\infty} ||g_k|| = 0.$$

Consequently, the following lemma follows.

Lemma 3.3. *Let Assumption A hold. Then, for any CG process in the form* (1.2) *and* (1.3)*, where the step size α_k is computed using SWP conditions* (2.3) *and* (2.4)*, and d_k satisfies the descent property* (3.1)*, if*

$$\sum_{k=1}^{\infty} \frac{1}{||d_k||^2} = \infty, \tag{3.4}$$

it follows that

$$\liminf_{k\to\infty} ||g_k|| = 0. \tag{3.5}$$

Proof. We prove this by contradiction. That is, we assume (3.5) is not true. Then there exists a constant $r > 0$ such that $||g_k|| \geq r$ for all $k \geq 0$. From Lemma 3.2, we have that (3.3) holds. Hence

$$\sum_{k=0}^{\infty} \frac{1}{||d_k||^2} \leq \frac{1}{r^4} \sum_{k=0}^{\infty} \frac{||g_k||^4}{||d_k||^2} < \infty,$$

which contradicts (3.4). Thus, (3.5) holds true. □

By Cauchy-Schwarz inequality, we have that

$$t_k^* \leq \theta_k \frac{||s_k||||y_k||}{||s_k||^2} \leq L\theta_{max},$$

and therefore, the norm of d_k generated by (1.4) can be proved to be bounded above for uniformly convex functions. Thus, we have the following theorem.

Theorem 3.1. *Suppose Assumption A holds. Let the CG method be defined by* (1.2) *and* (1.3)*, where β_k follows from* (1.4) *and $t = t_k^*$. If f is uniformly convex on N, that is, there exists a constant $\mu > 0$ such that*

$$(\nabla f(x) - \nabla f(y))^T (x - y) \geq \mu||x - y||^2, \;\; x, y \in N,$$

the descent condition (3.1) *holds and $\alpha_k > 0$ is computed using the SWP strategies* (2.3) *and* (2.4)*, then the new formula converges such that*

$$\lim_{k\to\infty} ||g_k|| = 0. \tag{3.6}$$

Proof. Suppose that $g_k \neq 0$ for all k. By the uniformly convex property of f, we have

$$d_{k-1}^T y_{k-1} \geq \mu\alpha_{k-1}||d_{k-1}||^2.$$

Using the triangle and Cauchy-Schwarz inequalities, it follows that

$$\begin{aligned} |\beta_k| &= \left| \frac{g_k^T y_{k-1}}{d_{k-1}^T y_{k-1}} - t^* \frac{g_k^T s_{k-1}}{d_{k-1}^T y_{k-1}} \right| \\ &\leq \left| \frac{g_k^T y_{k-1}}{d_{k-1}^T y_{k-1}} \right| + |t^*| \left| \frac{g_k^T s_{k-1}}{d_{k-1}^T y_{k-1}} \right| \\ &\leq \frac{||g_k|| ||y_{k-1}||}{\mu \alpha_{k-1} ||d_{k-1}||^2} + t^* \frac{||g_k||}{\mu ||d_{k-1}||} \\ &\leq \frac{L ||g_k|| ||d_{k-1}||}{\mu ||d_{k-1}||^2} + t^* \frac{||g_k||}{\mu ||d_{k-1}||} \\ &\leq \frac{L}{\mu}(1 + \theta_{max}) \frac{||g_k||}{||d_{k-1}||}. \end{aligned}$$

As a result, we obtain that

$$\begin{aligned} ||d_k|| &\leq ||g_k|| + |\beta_k| ||d_{k-1}|| \\ &\leq \frac{L}{\mu}(1 + \theta_{max}) ||g_k||| \\ &= \tilde{\gamma} ||g_k|| \leq \tilde{\gamma} \gamma, \end{aligned} \tag{3.7}$$

which implies (3.4) holds and hence (3.5) is true, which is equivalent to (3.6) for uniformly convex functions. □

Note that if the parameter β_k^{DLBB} is modified as

$$\beta_k^{DL} = \max \left\{ \frac{g_k^T y_{k-1}}{d_{k-1}^T y_{k-1}}, 0 \right\} - t^* \frac{g_k^T s_{k-1}}{d_{k-1}^T y_{k-1}},$$

with t_k^* defined by (2.2), and d_k satisfies the descent condition (3.1), then, Theorem 3.6 of Dai and Liao [15] ensures the global convergence of the algorithm for general nonlinear functions.

4. Numerical results

This section demonstrates the numerical efficiency of the proposed DLBB method by comparing its performance with other existing algorithms of the same class under the strong Wolfe conditions. The forty-nine (49) benchmark problems employed for this analysis (see Table 1) are taken from [34]. For each problem, different initial points are used and the dimensions considered range from 2 up to $100,000$. The efficiency of the methods are measured based on the following metrics: number of iterations (NOI), Number of function evaluations (NOF), and CPU time. For the comparison, we considered the following methods

- The classical Dai-Liao method in [15].
- The DLHZ method in [16].
- The MDL3 and MDL4 methods in [22].
- The EJHJ and MEJHJ methods in [6] with $\delta = 0.0001$ and $\sigma = 0.99$.

Table 1. List of Test Functions.

No	Function	No	Function
F1	Extended Penalty	F26	Diagonal 4
F2	Extended Maratos	F27	Diagonal 7
F3	Diagonal 5	F28	Diagonal 8
F4	Trecanni	F29	Diagonal 9
F5	Extended quadratic penalty QP1	F30	DENSCHNA
F6	Extended quadratic penalty QP2	F31	DENSCHNC
F7	Quadratic QF1	F32	Extended Block-Diagonal
F8	Quadratic QF2	F33	HIMMELBH
F9	POWER	F34	DQDRTIC
F10	Zettl	F35	QUARTICM
F11	Diagonal 2	F36	Linear Perturbed
F12	Test	F37	Tridiagonal White & Holst
F13	Sum Squares	F38	ENGVAL1
F14	Shallow	F39	ENGVAL8
F15	Quartic	F40	DENSCHNF
F16	Matyas	F41	ARWHEAD
F17	Diagonal 1	F42	Six hump
F18	Hager	F43	Price 4
F19	Zirilli or Aluffi-Pentini's	F44	Extended Himmelblau
F20	Raydan 1	F45	Rotated Ellipse
F21	Raydan 2	F46	El-Attar-Vidyasagar-Dutta
F22	FLETCHCR	F47	Extended Hiebert
F23	Diagonal 3	F48	Extended Tridiagonal 1
F24	Extended DENSCHNB	F49	Three hump
F25	Diagonal 6		

The codes used for this experiment are coded in MATLAB R2019b software and ran on a core i5 Windows 10 PC with 8GB RAM. The stopping criteria is set as $\|g_k\| \leq 10^{-6}$ or as number of maximum iterations, i.e., 2,000. If any of these conditions does not hold, we represent that point as failure and denote it by "*". The detailed presentation of the numerical results is presented in Tables 2–6.

To graphically interpret the performance of the obtained numerical results, we employed a performance profile tool introduced by [35]. Based on the performance profile, each algorithm is represented by a curve for comparison purpose. The algorithm with the best performance has its curve lying above all other algorithms implying that it solved more number of the test problems considered in Table 1. The experimental results for all the methods are graphed in Figure 1 (NOI), Figure 2 (NOF), and Figure 3 (CPU time) under strong Wolfe line search (2.3) and (2.4) with parameter values given as $\delta = 0.0001$ and $\sigma = 0.0002$.

Table 2. Numerical comparison of DLBB algorithm versus DL, DLHZ, MDL3, MDL4, EJHJ, and MEJHJ algorithms.

		DLBB			DL			DLHZ			MDL3			MDL4			EJHJ			MEJHJ		
Function	**DIM**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**
F1	100	3	4	0.002082	3	4	0.001972	4	5	0.021215	3	4	0.028434	3	4	0.002023	7	8	0.002013	6	7	0.047884
F1	10000	3	4	0.002613	22	23	0.002565	*	*	*	3	4	0.00296	3	4	0.00256	7	8	0.003086	*	*	*
F1	20000	14	15	0.00388	20	21	0.003333	*	*	*	4	5	0.003144	3	4	0.003638	*	*	*	*	*	*
F2	2	16	17	0.003618	16	17	0.001621	16	17	0.001623	29	30	0.002237	29	30	0.001559	34	35	0.14608	31	32	0.00174
F2	20	16	17	0.001804	16	17	0.00152	16	17	0.001533	*	*	*	*	*	*	42	43	0.004562	32	33	0.001812
F3	10	2	3	0.40622	2	3	0.007599	*	*	*	2	3	0.006907	2	3	0.027818	2	3	16.8813	2	3	2.7997
F3	50000	2	3	0.60107	2	3	0.45496	*	*	*	2	3	0.41073	2	3	0.26647	2	3	0.72594	2	3	19.8002
F3	100000	2	3	1.1023	4	5	2.1626	*	*	*	2	3	0.80044	2	3	0.58837	2	3	1.099	2	3	0.55087
F4	2	4	5	0.001863	4	5	0.001937	4	5	0.002585	4	5	0.002097	5	6	0.001784	8	9	0.021197	7	8	0.005167
F4	2	4	5	0.002285	4	5	0.001721	4	5	0.002246	5	6	0.00224	5	6	0.001938	7	8	0.012456	6	7	0.003696
F5	4	6	7	0.26128	6	7	0.011473	6	7	0.009658	6	7	0.010866	6	7	0.072683	10	11	5.996	7	8	0.75079
F5	100	4	5	0.007224	4	5	0.01019	4	5	0.008486	5	6	0.014262	5	6	0.01008	16	17	0.034523	8	9	0.010091
F5	10000	4	5	0.085455	4	5	0.079038	4	5	0.10045	*	*	*	*	*	*	*	*	*	10	11	0.56667
F6	500	19	20	0.43971	52	53	0.43716	*	*	*	*	*	*	*	*	*	5	6	8.4535	3	4	0.9851
F6	50000	31	32	11.7492	100	101	38.8894	*	*	*	*	*	*	*	*	*	5	6	0.73796	*	*	*
F6	100000	32	33	24.9882	104	105	83.7568	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
F7	10	5	6	0.057091	5	6	0.022205	5	6	0.021445	19	20	0.055992	21	22	0.077409	39	40	0.46801	41	42	0.62606
F7	50	5	6	0.012991	5	6	0.017794	5	6	0.01924	*	*	*	4	5	0.017261	450	451	5.0906	29	30	0.21465
F8	50	2	3	0.056291	2	3	0.011572	*	*	*	2	3	0.005114	2	3	0.011223	4	5	0.1411	2	3	5.9251
F8	1000	1	2	0.021703	1	2	0.006631	1	2	0.006328	1	2	0.00642	1	2	0.006966	4	5	0.036183	2	3	0.043514
F8	5000	1	2	0.010908	1	2	0.010132	1	2	0.009809	1	2	0.011596	1	2	0.00976	4	5	0.14749	2	3	0.65248
F9	2	2	3	0.004275	2	3	0.001598	2	3	0.002571	2	3	0.002307	2	3	0.002439	15	16	0.003946	6	7	0.00187
F9	2	2	3	0.00182	2	3	0.001466	2	3	0.001776	2	3	0.002199	2	3	0.002205	17	18	0.001761	6	7	0.002644
F10	2	9	10	0.001705	9	10	0.001563	9	10	0.001638	10	11	0.002207	10	11	0.002038	17	18	0.001817	12	13	0.001891
F10	2	10	11	0.001459	10	11	0.002217	10	11	0.001977	10	11	0.002033	10	11	0.001944	25	26	0.00147	11	12	0.001913

Table 3. Numerical comparison of DLBB algorithm versus DL, DLHZ, MDL3, MDL4, EJHJ, and MEJHJ algorithms.

		DLBB			DL			DLHZ			MDL3			MDL4			EJHJ			MEJHJ		
Function	**DIM**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**
F11	1000	156	157	0.37457	157	158	0.30382	159	160	0.31832	175	176	0.38061	163	164	0.36371	167	168	0.56639	170	171	0.63796
F11	10000	543	544	9.3228	522	523	8.6765	527	528	8.7013	531	532	9.3352	471	472	8.0788	519	520	7.0927	532	533	6.71
F11	50000	849	850	37.0818	944	945	41.8829	940	941	39.7494	867	868	37.194	851	852	33.9035	943	944	30.2434	864	865	28.3961
F11	100000	1136	1137	84.0919	1137	1138	84.9251	1102	1103	80.7658	1073	1074	81.954	1046	1047	76.436	1084	1085	57.0639	1140	1141	60.357
F12	3	7	8	0.29255	7	8	0.011095	7	8	0.018001	7	8	0.009686	15	16	0.012199	186	187	0.99943	32	33	0.74133
F12	3	14	15	0.01374	14	15	0.014908	14	15	0.012858	14	15	0.013591	13	14	0.014882	163	164	0.74828	23	24	0.01652
F13	1000	2	3	0.003455	2	3	0.002583	3	4	0.002433	3	4	0.001936	3	4	0.002753	13	14	0.002226	3	4	0.003722
F13	2000	2	3	0.001785	3	4	0.001774	3	4	0.002129	3	4	0.002387	3	4	0.002308	12	13	0.002049	12	13	0.002049
F13	5000	3	4	0.002086	4	5	0.001825	*	*	*	4	5	0.002595	4	5	0.002513	35	36	0.002027	35	36	0.00212
F14	1000	13	14	0.002143	13	14	0.001551	14	15	0.001538	14	15	0.002418	6	7	0.002228	36	37	0.047245	13	14	0.002278
F14	10000	10	11	0.002464	10	11	0.0024	10	11	0.002398	10	11	0.002467	12	13	0.003339	22	23	0.002798	12	13	0.002942
F15	100	2	3	0.25068	2	3	0.009905	*	*	*	2	3	0.007711	2	3	0.011959	3	4	6.5038	2	3	0.23952
F15	1000	2	3	0.038562	2	3	0.055482	*	*	*	2	3	0.022114	2	3	0.033326	3	4	0.026792	2	3	0.032356
F15	5000	2	3	0.098152	2	3	0.093159	*	*	*	2	3	0.053293	2	3	0.089636	3	4	0.19136	3	4	0.29064
F15	10000	2	3	0.16931	2	3	0.1498	*	*	*	2	3	0.087874	2	3	0.15005	3	4	0.29881	3	4	0.85874
F16	2	1	2	0.002498	1	2	0.002839	1	2	0.002672	1	2	0.001838	1	2	0.002479	1	2	0.004095	1	2	0.004131
F16	2	1	2	0.002051	1	2	0.00195	1	2	0.001895	1	2	0.002302	1	2	0.002194	1	2	0.00227	1	2	0.003331
F17	10	187	188	0.17229	202	203	0.14512	211	212	0.17735	31	32	0.026652	25	26	0.02554	188	189	0.42002	252	253	0.44903
F17	20	34	35	0.031242	34	35	0.029411	34	35	0.023777	*	*	*	33	34	0.027544	380	381	0.24289	524	525	1.6553
F18	10	12	13	0.003568	12	13	0.001906	12	13	0.001653	12	13	0.002073	13	14	0.002361	61	62	0.002506	100	101	0.003088
F18	50	21	22	0.002019	21	22	0.001668	21	22	0.002151	21	22	0.001571	21	22	0.002088	71	72	0.001667	188	189	0.00158
F18	100	25	26	0.002085	25	26	0.001425	25	26	0.001662	25	26	0.001399	25	26	0.001524	89	90	0.001687	245	246	0.002405
F19	2	4	5	0.002613	4	5	0.002506	3	4	0.001684	5	6	0.001794	5	6	0.00233	13	14	0.00362	7	8	0.001507
F19	2	4	5	0.001958	4	5	0.001889	4	5	0.001669	4	5	0.002055	4	5	0.002231	5	6	0.001606	5	6	0.001408
F20	50	47	48	0.08585	47	48	0.033502	47	48	0.038457	47	48	0.036282	47	48	0.042629	48	49	1.2978	48	49	2.2839

Table 4. Numerical comparison of DLBB algorithm versus DL, DLHZ, MDL3, MDL4, EJHJ, and MEJHJ algorithms.

		DLBB			DL			DLHZ			MDL3			MDL4			EJHJ			MEJHJ		
Function	**DIM**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**
F20	100	66	67	0.068643	67	68	0.068507	67	68	0.046007	66	67	0.062822	66	67	0.057604	70	71	0.073953	69	70	0.64725
F21	10	3	4	0.048289	2	3	0.009501	3	4	0.01164	3	4	0.008351	3	4	0.015991	4	5	0.45318	4	5	0.16934
F21	100	3	4	0.012193	3	4	0.01107	3	4	0.01186	3	4	0.008298	3	4	0.018361	4	5	0.006429	4	5	0.041784
F21	500	3	4	0.018414	3	4	0.013309	3	4	0.023495	3	4	0.01357	3	4	0.023755	4	5	0.079332	4	5	0.031636
F22	10	1	2	0.001553	1	2	0.001709	1	2	0.001836	1	2	0.002269	1	2	0.001762	2	3	0.001523	2	3	0.071092
F22	100	2	3	0.001811	2	3	0.002085	3	4	0.002022	2	3	0.002514	2	3	0.001658	3	4	0.040838	2	3	0.002063
F22	50000	2	3	0.008696	2	3	0.006917	3	4	0.00673	2	3	0.008065	2	3	0.006787	3	4	0.00859	2	3	0.012971
F23	2	6	7	0.11424	7	8	0.010821	6	7	0.008038	6	7	0.009378	4	5	0.015246	10	11	1.109	7	8	0.64014
F23	10	26	27	0.032245	26	27	0.031144	26	27	0.022413	27	28	0.02499	17	18	0.026918	26	27	0.043	28	29	0.058172
F24	100	5	6	0.003361	5	6	0.002423	5	6	0.002421	5	6	0.001869	6	7	0.002389	7	8	0.002766	5	6	0.001882
F24	5000	5	6	0.002653	5	6	0.002498	5	6	0.002428	5	6	0.002184	7	8	0.002676	9	10	0.002392	6	7	0.003724
F24	10000	5	6	0.003532	5	6	0.002512	5	6	0.002467	5	6	0.00262	7	8	0.003565	9	10	0.002443	6	7	0.004075
F25	1000	2	3	0.046142	2	3	0.031495	4	5	0.20147	2	3	0.023518	2	3	0.029136	2	3	1.4834	2	3	0.6707
F25	10000	2	3	0.042843	2	3	0.02976	4	5	0.189	2	3	0.02586	2	3	0.03183	2	3	0.019227	2	3	0.072281
F25	50000	2	3	0.16702	2	3	0.089995	4	5	0.82203	2	3	0.083256	2	3	0.097889	2	3	0.070062	2	3	0.38721
F26	1000	2	3	0.11593	2	3	0.007491	2	3	0.006241	2	3	0.007295	2	3	0.007518	2	3	0.93448	2	3	0.19509
F26	10000	2	3	0.0248	2	3	0.019749	2	3	0.013343	2	3	0.01258	2	3	0.016336	2	3	0.011617	2	3	0.037871
F26	100000	2	3	0.077709	2	3	0.072823	2	3	0.083059	2	3	0.091154	2	3	0.086951	2	3	0.10359	2	3	0.37038
F27	10	2	3	0.032315	2	3	0.008507	3	4	0.011977	2	3	0.008077	2	3	0.010768	3	4	0.21185	3	4	0.16647
F27	50	2	3	0.011376	2	3	0.008613	3	4	0.014438	2	3	0.007457	2	3	0.009927	3	4	0.1075	3	4	0.082605
F27	100	3	4	0.013874	2	3	0.006445	3	4	0.014155	2	3	0.009471	2	3	0.01267	3	4	0.007769	3	4	0.027418
F28	100	1	2	0.22493	1	2	0.006581	1	2	0.006465	1	2	0.007197	1	2	0.007419	3	4	0.38604	3	4	0.65344
F28	500	1	2	0.008501	1	2	0.037526	1	2	0.006499	1	2	0.008196	1	2	0.008411	3	4	0.033907	3	4	0.41581
F29	2	4	5	0.015505	4	5	0.00844	4	5	0.007445	5	6	0.011989	5	6	0.012143	*	*	*	8	9	0.79735
F29	4	789	790	0.31359	807	808	0.29147	799	800	0.2885	799	800	0.27431	825	826	0.29676	*	*	*	34	35	0.13938
F30	3000	8	9	0.002562	8	9	0.002317	11	12	0.002474	8	9	0.003166	8	9	0.00243	21	22	0.003976	12	13	0.007792
F30	15000	8	9	0.004427	8	9	0.005997	12	13	0.004529	8	9	0.004839	10	11	0.004595	23	24	0.005713	12	13	0.005194
F31	1000	2	3	0.002385	2	3	0.001798	2	3	0.001698	2	3	0.001677	2	3	0.00185	*	*	*	*	*	*

Table 5. Numerical comparison of DLBB algorithm versus DL, DLHZ, MDL3, MDL4, EJHJ, and MEJHJ algorithms.

		DLBB			DL			DLHZ			MDL3			MDL4			EJHJ			MEJHJ		
Function	**DIM**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**
F31	10000	2	3	0.003632	2	3	0.003086	2	3	0.003001	2	3	0.003832	2	3	0.003383	*	*	*	*	*	*
F32	5000	24	25	0.002402	24	25	0.002241	*	*	*	24	25	0.002387	26	27	0.002556	*	*	*	*	*	*
F32	10000	24	25	0.00303	24	25	0.002894	*	*	*	25	26	0.003042	26	27	0.003124	*	*	*	*	*	*
F32	100000	24	25	0.01429	24	25	0.014204	*	*	*	26	27	0.014435	26	27	0.015171	*	*	*	*	*	*
F33	1000	5	6	0.00196	5	6	0.002274	5	6	0.001888	5	6	0.001835	5	6	0.003088	12	13	0.001721	7	8	0.003875
F33	10000	5	6	0.003956	5	6	0.004327	5	6	0.003827	5	6	0.002893	*	*	*	*	*	*	*	*	*
F34	100	48	49	0.001669	52	53	0.001499	53	54	0.001825	55	56	0.001861	53	54	0.001837	146	147	0.0017	146	147	0.002156
F34	100	38	39	0.002672	52	53	0.002667	46	47	0.002722	49	50	0.002783	49	50	0.002956	151	152	0.003422	151	152	0.003006
F34	1000	50	51	0.007253	*	*	*	149	150	0.008002	155	156	0.007963	135	136	0.007901	203	204	0.008946	203	204	0.008931
F34	10000	52	53	0.013424	*	*	*	*	*	*	151	152	0.014574	153	154	0.014971	235	236	0.017104	235	236	0.016468
F35	1000	2	3	0.003461	2	3	0.002088	*	*	*	2	3	0.003086	2	3	0.002744	4	5	0.004868	2	3	0.00575
F35	10000	2	3	0.004953	2	3	0.003822	*	*	*	2	3	0.003883	2	3	0.004316	4	5	0.00414	2	3	0.010784
F36	5000	2	3	0.003767	2	3	0.002376	2	3	0.003648	3	4	0.00268	3	4	0.00283	2	3	0.005402	2	3	0.004779
F36	10000	2	3	0.003017	2	3	0.008692	*	*	*	3	4	0.002533	3	4	0.003363	3	4	0.002746	3	4	0.004451
F36	20000	2	3	0.003975	2	3	0.004488	*	*	*	3	4	0.00305	3	4	0.003265	3	4	0.004136	3	4	0.009098
F37	2	20	21	0.001562	20	21	0.00149	20	21	0.001496	27	28	0.001449	27	28	0.00146	74	75	0.002809	34	35	0.002226
F37	2	13	14	0.001544	13	14	0.001977	13	14	0.00299	14	15	0.001398	13	14	0.002218	36	37	0.003506	82	83	0.001485
F38	50	21	22	0.001422	21	22	0.001443	20	21	0.001494	19	20	0.001545	18	19	0.001491	24	25	0.029472	31	32	0.021539
F38	100	20	21	0.001669	20	21	0.002106	20	21	0.001564	18	19	0.001507	19	20	0.001519	25	26	0.004604	30	31	0.002157
F39	4	18	19	0.001423	13	14	0.001382	15	16	0.001684	17	18	0.001476	17	18	0.001476	30	31	0.001735	34	35	0.003387
F39	4	22	23	0.001545	20	21	0.001991	21	22	0.001611	21	22	0.001971	21	22	0.001514	32	33	0.16558	32	33	0.003419
F40	1000	7	8	0.001932	7	8	0.001765	7	8	0.001901	7	8	0.001911	7	8	0.002212	17	18	0.029741	13	14	0.001656
F40	10000	8	9	0.004509	8	9	0.004515	8	9	0.003447	8	9	0.003617	9	10	0.003618	17	18	0.004225	13	14	0.003624
F40	50000	8	9	0.011006	8	9	0.010437	8	9	0.013075	8	9	0.013968	9	10	0.011325	17	18	0.012575	13	14	0.011708

Table 6. Numerical comparison of DLBB algorithm versus DL, DLHZ, MDL3, MDL4, EJHJ, and MEJHJ algorithms.

		DLBB			DL			DLHZ			MDL3			MDL4			EJHJ			MEJHJ		
Function	**DIM**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**	**NOI**	**NOF**	**CPU**
F41	10	8	9	0.001436	9	10	0.001396	3	4	0.001869	5	6	0.00219	5	6	0.001886	*	*	*	*	*	*
F41	100	8	9	0.001502	8	9	0.002027	3	4	0.002411	5	6	0.00154	5	6	0.001541	*	*	*	*	*	*
F41	500	8	9	0.001661	7	8	0.00143	3	4	0.0022	5	6	0.001631	5	6	0.001638	*	*	*	*	*	*
F42	2	8	9	0.002203	8	9	0.001591	8	9	0.002335	9	10	0.00209	8	9	0.002082	16	17	0.001976	22	23	0.003377
F42	2	7	8	0.001689	7	8	0.001638	8	9	0.001893	8	9	0.002071	8	9	0.002475	13	14	0.001409	13	14	0.003248
F43	2	1	2	0.00155	1	2	0.001783	1	2	0.001823	1	2	0.002433	1	2	0.001781	2	3	0.003039	2	3	0.003729
F43	2	1	2	0.0022	1	2	0.002418	1	2	0.004698	1	2	0.002684	1	2	0.001875	2	3	0.00159	2	3	0.003413
F44	500	10	11	0.091278	25	26	0.26935	3	4	0.04364	10	11	0.1027	12	13	0.12931	67	68	0.92586	67	68	0.20985
F44	1000	13	14	0.12685	*	*	*	5	6	0.055475	*	*	*	33	34	0.35887	56	57	0.1184	60	61	0.24304
F44	2000	31	32	0.6616	*	*	*	*	*	*	*	*	*	*	*	*	57	58	0.3918	61	62	0.21142
F45	2	17	18	0.001599	17	18	0.001592	*	*	*	17	18	0.001454	17	18	0.002174	*	*	*	*	*	*
F45	2	17	18	0.001667	16	17	0.001678	*	*	*	17	18	0.001429	17	18	0.001588	*	*	*	*	*	*
F46	2	6	7	0.001748	6	7	0.002564	6	7	0.001737	6	7	0.002211	7	8	0.002229	7	8	0.019176	13	14	0.003363
F46	2	10	11	0.002122	10	11	0.002052	11	12	0.001615	10	11	0.002234	10	11	0.002088	17	18	0.002347	18	19	0.003298
F47	2	25	26	0.001749	25	26	0.001467	27	28	0.002065	20	21	0.001554	20	21	0.001611	80	81	0.001574	*	*	*
F47	2	18	19	0.001741	19	20	0.001509	18	19	0.001462	26	27	0.001716	17	18	0.001558	70	71	0.00154	*	*	*
F48	100	4	5	0.001749	4	5	0.002168	4	5	0.00174	9	10	0.002174	12	13	0.001538	528	529	0.001867	8	9	0.004559
F48	500	4	5	0.001972	4	5	0.002258	5	6	0.001659	9	10	0.002005	12	13	0.001834	724	725	0.001968	8	9	0.002194
F49	2	7	8	0.001624	10	11	0.001504	7	8	0.002199	12	13	0.001677	13	14	0.002154	14	15	0.003917	39	40	0.003072
F49	2	15	16	0.002023	15	16	0.001712	15	16	0.001546	8	9	0.001998	4	5	0.00235	11	12	0.001744	29	30	0.001963

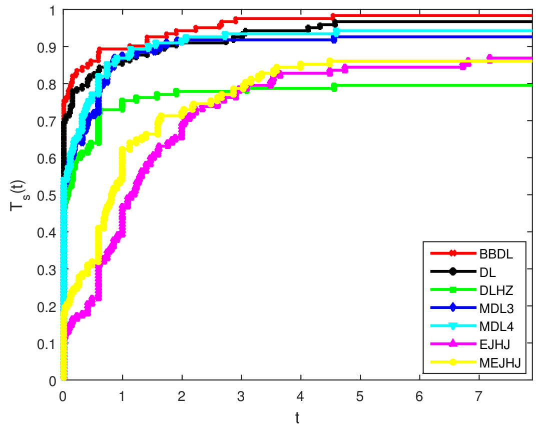

Figure 1. Performance profile according to the NOI.

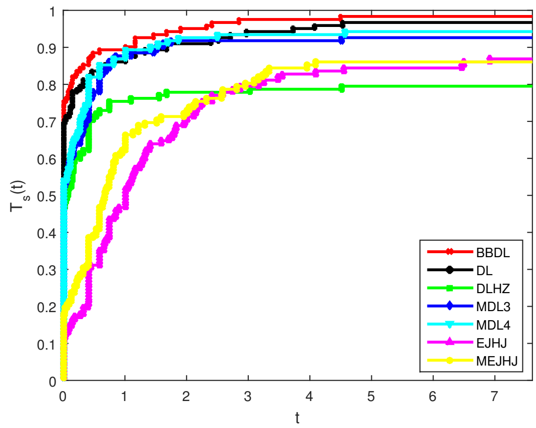

Figure 2. Performance profile according to the NOF.

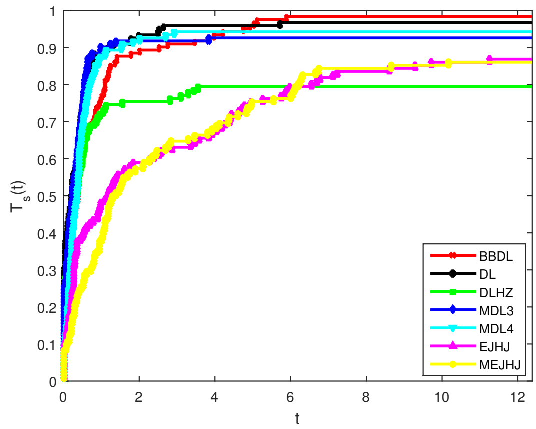

Figure 3. Performance profile according to the CPU time.

The curve from performance results depicted in Figures 1 and 2 show that the DLBB method obviously performed better than the DL, DLHZ, MDL3, MDL4, EJHJ and MEJHJ algorithms based on NOI and NOF. However, in terms of CPU time (see Figure 3), there was a close competition from DL, MDL3 and MDL4 methods, yet, the proposed DLBB method slightly outperforms these methods. On the other hand, it can also be seen that the performance of EJHJ and MEJHJ algorithms moves in a similar pattern. This can be attested to the fact that MEJHJ is an improvement of EJHJ. However, EJHJ, MEJHJ, and DLHZ have the highest number of failure and thus, have the curves of their algorithms lying below other curves. Based on the performance analysis, it is obvious to conclude that the new DLBB method presents the best performance since the DLBB curves are always the top performer for a large number of problems and it generated a higher number of the efficient search directions compare to DL, DLHZ, MDL3, MDL4, EJHJ and MEJHJ algorithms.

5. Application

This section investigate the performance of the new formula on portfolio selection and image restoration problems.

5.1. Portfolio selection

The analytical process of choosing and distributing a collection of investment assets is called portfolio selection. One of the well-known models for portfolio selection is the Markowitz model. Markowitz [36] proposed the mean-variance model in 1952, which calculates the expected return and risk of the generated portfolio using historical asset prices. However, in this work, we consider only the minimum variance with the simple model

$$\begin{cases} \text{minimize} : \sigma^2 = \sum_{i=0}^{N} \sum_{j=0}^{N} w_i w_j C_{ij}, \\ \text{subject to} : \sum_{i=1}^{N} w_i = 1, \end{cases} \tag{5.1}$$

where w_i is the weight of each asset, C_{ij} is the covariance of return between asset i and j and N is the total number of assets.

Stock investment is one of the investment products available to support the development of financial strength [3, 7]. Stock can be seen as a person or party's capital participation sign in a limited liability company or company. The party has a claim on the income of the company as well as a claim on the company's assets by including the said capital. Therefore, we will consider investment stocks in our portfolio selection [37].

In this application, we use 5 stocks in LQ45 index, namely, Barito Pacific Tbk. (BRPT), Semen Indonesia (Persero) Tbk. (SMGR), Charoen Pokphand Indonesia Tbk. (CPIN), Waskita Karya (Persero) Tbk. (WSKT) and Unilever Indonesia Tbk. (UNVR). The data used as a reference is the closing price from June 1, 2020, to May 31, 2022 which is taken from https://finance.yahoo.com. For the data, we assume that the data follow a normal distribution, so to calculate the mean, variance and covariance using the formulas in the normal distribution.

Tables 7 and 8 present summary statistics of close prices for the five stocks. Table 7 provides the expected return and variance of the five stocks, while Table 8 provides the values of covariance among the stocks. By letting $w_5 = 1 - w_1 - w_2 - w_3 - w_4$, where w_1, w_2, w_3, w_4 and w_5 are proportional to CPIN, WSKT, BRPT, SMGR, and UNVR stocks, respectively, and using the data from Table 8, we obtained the following unconstrained minimization type portfolio selection model as

$$\begin{aligned}\min_{(w_1,w_2,w_3,w_4)\in\mathbb{R}^4} & (w_1+w_2+w_3+w_4-1)((41w_1)/10^5+w_2/3125+(29w_3)/10^5 \\ & +(41w_4)/10^5-51/10^5)+w_4(w_2/6250-(3w_1)/10^5+(3w_3)/25000 \\ & +(27w_4)/25000+1/10^4)+w_1((29w_1)/10^5+w_2/50000-w_3/50000 \\ & -(3w_4)/10^5+1/10^4)+w_2(w_2/2500-(7w_1)/10^5+w_3/25000 \\ & +(7w_4)/10^5+19/10^5)+w_3(w_2/10^5-(7w_1)/50000+(37w_3)/50000+11/50000).\end{aligned}$$

Table 7. Expected return (μ) and variance (σ^2).

Stock	μ	σ^2
CPIN	0.00029	0.00051
BRPT	0.00145	0.00101
SMGR	0.00093	0.00059
WSKT	0.00090	0.00118
UNVR	0.00135	0.00039

Table 8. Covariance among the considered stocks.

Stock	CPIN	WSKT	BRPT	SMGR	UNVR
CPIN	0.00051	0.00010	0.00022	0.00019	0.00010
WSKT	0.00010	0.00118	0.00022	0.00026	0.00007
BRPT	0.00022	0.00022	0.00096	0.00023	0.00008
SMGR	0.00019	0.00026	0.00023	0.00059	0.00012
UNVR	0.00010	0.00007	0.00008	0.00012	0.00039

Now, we test the performance of all methods in solving unconstrained optimization problem defined above. By taking some initial points: P1: (0.1,0.2,0.3,0.4), P2: (0.4,0.3,0.2,0.1), P3: (0.1,0.1,0.1,0.1), P4: (0.5,0.1,0.2,0.2), P5 (0.5,0.5,0.5,0.5), P6: (1,1,1,1), P7: (1.5,1.5,1.5,1.5), P8: (0.1,0.5,0.5,0.1), P9: (0.8,0.5,0.3,0.1) and P10: (0.1,0.3,0.5,0.8), we have the numerical outcome as in Table 9.

Based on Table 9, it is obvious the proposed DLBB method presents the best performance with regards to NOI, NOF and CPU time when compare to DL, DLHZ, MDL3 and MDL4 methods for the above defined problem. By using all methods, we get the values $w_1 = 0.4334, w_2 = 0.1362, w_3 = 0.0856, w_4 = 0.0972$ and $w_5 = 0.2476$. Of the total allocated funds from the formed portfolio, UNVR accounted for about 43.34% as indicated by the values of $w_1, ..., w_5$, while the proportion of SMGR is 13.62%, the BRPT is 8.56%, the WSKT is 9.72% and the CPIN is 24.76%. Furthermore, we have the value of portfolio risk is 2.2397e-04 and the expected return is 0.000824524.

Table 9. Test result of DL, DLBB and DLHZ for portfolio selection model.

Points	DL	DLBB	DLHZ	MDL3	MDL4
	NOI/NOF/CPU	NOI/NOF/CPU	NOI/NOF/CPU	NOI/NOF/CPU	NOI/NOF/CPU
P1	7/75/0.0031	4/52/0.0006436	49/379/0.0126	49/435/0.0065	49/432/0.0075
P2	6/63/0.0011	3/38/0.0007091	41/315/0.0054	39/346/0.0067	39/343/0.0063
P3	12/120/0.0026	4/52/0.001	44/344/0.0055	44/395/0.0035	45/402/0.0054
P4	7/73/0.0009912	4/50/0.0003859	44/322/0.0028	42/359/0.0036	41/349/0.0033
P5	7/80/0.0005273	4/50/0.0004141	53/387/0.0042	50/421/0.0038	51/428/0.0043
P6	5/57/0.0007518	4/51/0.0007264	57/411/0.0034	57/472/0.0063	57/469/0.0048
P7	7/74/0.000945	4/51/0.0003664	63/455/0.0041	62/511/0.0067	63/519/0.0052
P8	9/100/0.0011	4/51/0.0006875	43/345/0.0033	42/387/0.0035	43/394/0.0038
P9	12/124/0.0011	4/51/0.0009069	51/377/0.0034	50/424/0.0028	50/424/0.0038
P10	8/83/0.0015	4/50/0.0008931	56/424/0.0045	56/484/0.0041	57/491/0.0051

5.2. Image restoration

The Conjugate Gradient (CG) method has recently been extended to solve real-life application problems of image restoration that involve the restoring a degraded image to its original form [1,2,38]. These types of problems often require solving ill-posed inverse problems, where the image has been corrupted by noise and other factors. The CG algorithm is employed to recover the underlying clean image from these corrupted observations with best precision. In this study, the proposed DLBB CG algorithm is be extended to solve the image restoration model

$$\min \mathcal{H}(u),$$

and

$$\mathcal{H}(u) = \sum_{(i,j)\in G} \left\{ \sum_{(m,n)\in T_{i,j}/G} \phi_\alpha(u_{i,j} - \xi_{m,n}) + \frac{1}{2} \sum_{(m,n)\in T_{i,j}\cap G} \phi_\alpha(u_{i,j} - u_{m,n}) \right\},$$

where x is the original image with $M \times N$ pixel whose index set G is given as

$$G = \{(i, j) \in Q | \bar{\xi}_{ij} \neq \xi_{ij},\ \xi_{ij} = s_{\min} \text{ or } s_{\max}\}. \tag{5.2}$$

From (5.2), we have ξ denoting the observed noisy image whose adaptive median filter is given as $\bar{\xi}$. The maximum and minimum of a noisy pixel are defined by $s_{\max}$ and $s_{\min}$. Also, $i, j \in Q = \{1, 2, \cdot, M\}\times\{1, 2, \cdot, N\}$, with its neighborhood computed as $T_{i\ j} = \{(i,\ j-1), (i, j+1), (i-1, j), (i+1, j)\}$.

From the above image model (5.2), the potential edge-preserving function ϕ_α in $\mathcal{H}(u)$ is defined as

$$\phi_\alpha(t) = \sqrt{t^2 + \alpha}, \tag{5.3}$$

where α is a constant whose value is chosen as 1.

To evaluate the relative accuracy of the proposed DLBB algorithm in restoring the corrupted images, the study compared its performance with some state-of-the-art algorithms including ALG 4.1 by Dai and Kou [17], CG-Descent by Hager and Zhang [16], EJHJ and MEJHJ by [6]. All comparisons are based on three metrics, being CPU time (CPUT), relative error (RelErr) and peak signal-to-noise ratio (PSNR). The corrupted images considered for restoration include Forest (512×512) and

Building (512 × 512). The performance of each algorithm in restoring the images is presented in Tables 10–12, and Figure 4.

Table 10. Image restoration outputs for DLBB, EJHJ, MEJHJ, CG DESCENT, and ALG 4.1, based on CPUT.

METHOD		DLBB	EJHJ	MEJHJ	CG DESCENT	ALG 4.1
IMAGE	NOISE	CPUT	CPUT	CPUT	CPUT	CPUT
FOREST	40%	87.6860	85.0866	85.4401	87.6971	92.4160
	80%	196.5231	166.8942	***	***	***
BUILDING	40%	86.0780	86.0877	86.4588	86.0191	88.0290
	80%	274.0619	247.9383	***	***	***

Table 11. Image restoration outputs for DLBB, EJHJ, MEJHJ, CG DESCENT, and ALG 4.1, based on RelErr.

METHOD		DLBB	EJHJ	MEJHJ	CG DESCENT	ALG 4.1
IMAGE	NOISE	RelErr	RelErr	RelErr	RelErr	RelErr
FOREST	40%	1.4388	1.3921	1.3940	1.3430	1.3868
	80%	2.7396	2.5712	***	***	***
BUILDING	40%	1.9220	2.0385	1.9976	1.9284	1.9320
	80%	5.0971	4.9126	***	***	***

Table 12. Image restoration outputs for DLBB, EJHJ, MEJHJ, CG DESCENT, and ALG 4.1, based on PSNR.

METHOD		DLBB	EJHJ	MEJHJ	CG DESCENT	ALG 4.1
IMAGE	NOISE	PSNR	PSNR	PSNR	PSNR	PSNR
FOREST	40%	26.7360	26.7547	26.7799	26.8930	26.7732
	80%	21.7918	22.1444	***	***	***
BUILDING	40%	28.1270	28.0919	27.9754	28.0040	28.0608
	80%	22.3129	22.4682	***	***	***

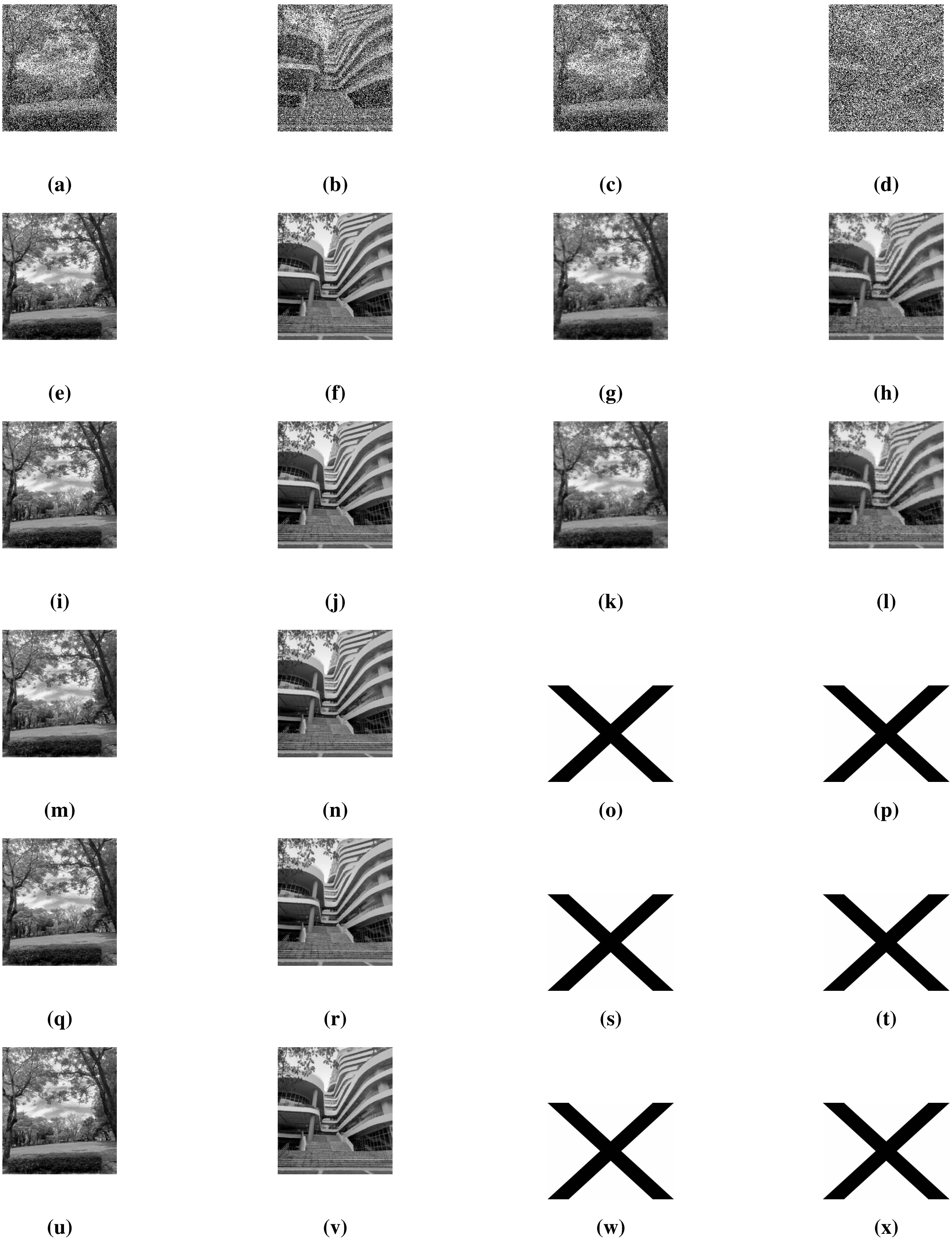

Figure 4. Forest and building images corrupted by 40% salt-and-pepper noise (a), (b), and 80% noise degree (c) and (d), the restored images using DLBB: (e,f,g,h), EJHJ: (i,j,k,l), MEJHJ: (m,n,o,p), CG-Descent (q,r,s,t), ALG 4.1 (u,v,w,x).

Based on the results presented in Tables 10–12, it is obvious to see that only DLBB and EJHJ algorithms were able to generate descent directions by solving all the problems. This is because the other algorithms including MEJHJ, CG-Descent and ALG 4.1 where unable to generate descent directions when the noise degree of corrupted forest image was increased to 80%. The point of failure for each metric including CPUT, RelErr and PSNR is denoted as $***$. These results have shown that the proposed DLBB algorithm has been able to improve the correlation in signals and further decorrelates the salt and pepper grey noise with better accuracy compared to the other algorithms used in the comparison which has further demonstrated the efficiency and robustness of our method.

6. Conclusions

In this paper, we investigated the performance of a novel modification of DL algorithm for solving unconstrained optimization and portfolio selection problems. The success of the proposed algorithm is attributed to the new optimal choice parameter for the modified DL CG method that derived based on the promising Barzilai-Borwein approach. Under some suitable assumptions, we discussed the convergence analysis of the proposed method. Results from computational experiments are discussed to highlight the robustness and efficiency of the new algorithm for unconstrained optimization, portfolio selection, and image restoration problems.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This research was supported by Ministry of Higher Education (MoHE) of Malaysia through Fundamental Research Grant Scheme (FRGS/1/2021/STG06/UUM/02/4) with S/O code 20123.

Conflict of interest

The authors declare that they have no competing interests with regards to this study.

References

1. X. Z. Jiang, H. H. Yang, J. B. Jian, X. D. Wu, Two families of hybrid conjugate gradient methods with restart procedures and their applications, *Optim. Method. Softw.*, **38** (2023), 947–974. https://doi.org/10.1080/10556788.2023.2189718
2. X. Z. Jiang, X. M. Ye, Z. F. Huang, M. X. Liu, A family of hybrid conjugate gradient method with restart procedure for unconstrained optimizations and image restorations, *Comput. Oper. Res.*, **159** (2023), 106341. https://doi.org/10.1016/j.cor.2023.106341
3. A. M. Awwal, I. M. Sulaiman, M. Malik, M. Mamat, P. Kumam, K. Sitthithakerngkiet, A spectral RMIL+ conjugate gradient method for unconstrained optimization with applications in portfolio selection and motion control, *IEEE Access*, **9** (2021), 75398–75414. https://doi.org/10.1109/ACCESS.2021.3081570

4. I. M. Sulaiman, N. A. Bakar, M. Mamat, B. A. Hassan, M. Malik, A. M. Alomari, A new hybrid conjugate gradient algorithm for optimization models and its application to regression analysis, *Indones. J. Electr. Eng. Comput. Sci.*, **23** (2021), 1100–1109. https://doi.org/10.11591/ijeecs.v23.i2.pp1100-1109
5. I. M. Sulaiman, M. Mamat, A new conjugate gradient method with descent properties and its application to regression analysis, *J. Numer. Anal. Ind. Appl. Math.*, **14** (2020), 25–39.
6. Z. Aminifard, S. Babaie-Kafaki, Dai-Liao extensions of a descent hybrid nonlinear conjugate gradient method with application in signal processing, *Numer. Algorithm*, **89** (2022), 1369–1387. https://doi.org/10.1007/s11075-021-01157-y
7. M. Malik, I. M. Sulaiman, A. B. Abubakar, G. Ardaneswari, Sukono, A new family of hybrid three-term conjugate gradient method for unconstrained optimization with application to image restoration and portfolio selection, *AIMS Mathematics*, **8** (2023), 1–28, https://doi.org/10.3934/math.2023001
8. W. W. Hager, H. Zhang, A survey of nonlinear conjugate gradient methods, *Pac. J. Optim.*, **2** (2006), 35–58.
9. M. R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bureau Stand.*, **49** (1952), 409–436.
10. I. M. Sulaiman, M. Malik, A. M. Awwal, P. Kumam, M. Mamat, S. Al-Ahmad, On three-term conjugate gradient method for optimization problems with applications on COVID-19 model and robotic motion control, *Adv. Contin. Discret. Models*, **2022** (2022), 1. https://doi.org/10.1186/s13662-021-03638-9
11. N. Aini, M. Mamat, M. Rivaie, I. S. Ibrahim, A hybrid of quasi-Newton method with CG method for unconstrained optimization, *J. Phys. Conf. Ser.*, **1366** (2019), 012079. https://doi.org/10.1088/1742-6596/1366/1/012079 .
12. M. Maulana, M. Mamat, S. S. Abas, I. M. Sulaiman, F. Sukono, Performance analysis of new spectral and hybrid conjugate gradient methods for solving unconstrained optimization problems, *IAENG Int. J. Comput. Sci.*, **48** (2021), 66–79.
13. P. Kaelo, P. Mtagulwa, M. V. Thuto, A globally convergent hybrid conjugate gradient method with strong Wolfe conditions for unconstrained optimization, *Math. Sci.* **14** (2020), 1–9.
14. N. Salihu, P. Kumam, A. M. Awwal, I. M. Sulaiman, T. Seangwattana, The global convergence of spectral RMIL conjugate gradient method for unconstrained optimization with applications to robotic model and image recovery, *PLOS One*, **18** (2023), e0281250. https://doi.org/10.1371/journal.pone.0281250
15. Y. H. Dai, L. Z. Liao, New conjugacy conditions and related nonlinear conjugate gradient methods, *Appl. Math. Optim.*, **43** (2001), 87–101, https://doi.org/10.1007/s002450010019
16. W. W. Hager, H. Zhang, A new conjugate gradient method with guaranteed descent and an efficient line search, *SIAM J. Optim.*, **16** (2005), 170–192. https://doi.org/10.1137/030601880
17. Y. H. Dai, C. X. Kou, A nonlinear conjugate gradient algorithm with an optimal property and an improved Wolfe line search, *SIAM J. Optim.*, **23** (2013), 296–320. https://doi.org/10.1137/100813026
18. N. Andrei, Open problems in nonlinear conjugate gradient algorithms for unconstrained optimization, *Bull. Malays. Math. Sci. Soc. (2)*, **34** (2011), 319–330.

19. S. Babaie-Kafaki, R. Ghanbari, The Dai-Liao nonlinear conjugate gradient method with optimal parameter choices, *Eur. J. Oper. Res.*, **234** (2014), 625–630. https://doi.org/10.1016/j.ejor.2013.11.012

20. S. Babaie-Kafaki, R. Ghanbari, A descent family of Dai-Liao conjugate gradient methods, *Optim. Method. Softw.*, **29** (2014), 583–591. https://doi.org/10.1080/10556788.2013.833199

21. L. Zhang, W. J. Zhou, D. H. Li, Some descent three-term conjugate gradient methods and their global convergence, *Optim. Method. Softw.*, **22** (2007), 697–711. https://doi.org/10.1080/10556780701223293

22. S. Babaie-Kafaki, R. Ghanbari, Two optimal Dai-Liao conjugate gradient methods, *Optimization*, **64** (2015), 2277–2287. https://doi.org/10.1080/02331934.2014.938072

23. K. Zhang, H. Liu, Z. Liu, A new Dai-Liao conjugate gradient method with optimal parameter choice, *Numer. Funct. Anal. Optim.*, **40** (2019), 194–215. https://doi.org/10.1080/01630563.2018.1535506

24. U. A. Yakubu, M. Mamat, M. A. Mohamad, M. Rivaie, J. Sabi'u, A recent modification on Dai-Liao conjugate gradient method for solving symmetric nonlinear equations, *Far East J. Math. Sci. (FJMS)*, **103** (2018), 1961–1974. http://doi.org/10.17654/MS103121961

25. M. Y. Waziri, K. Ahmed, J. Sabi'u, A. S. Halilu, Enhanced Dai-Liao conjugate gradient methods for systems of monotone nonlinear equations, *SeMA J.*, **78** (2021), 15–51. https://doi.org/10.1007/s40324-020-00228-9

26. J. Sabi'u, A. Shah, M. Y. Waziri, A modified Hager-Zhang conjugate gradient method with optimal choices for solving monotone nonlinear equations, *Int. J. Comput. Math.*, **99** (2021), 332–354. https://doi.org/10.1080/00207160.2021.1910814

27. J. Sabi'u, A. Shah, M. Y. Waziri, K. Ahmed, Modified Hager-Zhang conjugate gradient methods via singular value analysis for solving monotone nonlinear equations with convex constraint, *Int. J. Comput. Method.*, **18** (2020), 2050043, https://doi.org/10.1142/S0219876220500437

28. J. Sabi'u, A. Shah, M. Y. Waziri, Two optimal Hager-Zhang conjugate gradient methods for solving monotone nonlinear equations, *Appl. Numer. Math.*, **153** (2020), 217–233. https://doi.org/10.1016/j.apnum.2020.02.017

29. M. Y. Waziri, K. Ahmed, J. Sabi'u, A Dai-Liao conjugate gradient method via modified secant equation for system of nonlinear equations, *Arab. J. Math.*, **9** (2020), 443–457. https://doi.org/10.1007/s40065-019-0264-6

30. M. Y. Waziri, K. A. Hungu, J. Sabi'u, Descent Perry conjugate gradient methods for systems of monotone nonlinear equations, *Numer. Algorithms*, **85** (2020), 763–785. https://doi.org/10.1007/s11075-019-00836-1

31. J. Barzilai, J. M. Borwein, Two-point step size gradient methods, *IMA J. Numer. Anal.*, **8** (1988), 141–148. https://doi.org/10.1093/imanum/8.1.141

32. G. Zoutendijk, Nonlinear programming computational methods, In: *Integer and nonlinear programming*, Amsterdam: North-Holland, 1970, 37–86.

33. Y. Dai, J. Han, D. Sun, H. Yin, Y. X. Yuan, Convergence properties of nonlinear conjugate gradient methods, *SIAM J. Optim.*, **10** (2000), 345–358. https://doi.org/10.1137/S10562349426844.

34. N. Andrei, An unconstrained optimization test functions collection, *Adv. Model. Optim.*, **10** (2008), 147–161.

35. E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles, *Math. Program.*, **91** (2002), 201–213, https://doi.org/10.1007/s101070100263

36. H. Markowitz, Portfolio selection, *J. Finance*, **7** (1952), 77–91. https://doi.org/10.2307/2975974

37. H. Mayo, *Investments: An introduction*, 12 Eds., Cengage Learning EMEA, 2016.

38. X. Wu, H. Shao, P. Liu, Y. Zhang, Y. Zhuo, An efficient conjugate gradient-based algorithm for unconstrained optimization and its projection extension to large-scale constrained nonlinear equations with applications in signal recovery and image denoising problems, *J. Comput. Appl. Math.*, **422** (2023), 114879. https://doi.org/10.1016/j.cam.2022.114879

AIMS Press