New Douglas-Rashford Splitting Algorithms for Generalized DC Programming with Applications in Machine Learning

Yonghong Yao *, Lateef O. Jolaoso†, Yekini Shehu‡, Jen-Chih Yao§

December 23, 2024

Abstract

In this work, we propose some new Douglas-Rashford splitting algorithms for solving a class of generalized DC (difference of convex functions) in real Hilbert spaces. The proposed methods leverage the proximal properties of the nonsmooth component and a fasten control parameter which improves the convergence rate of the algorithms. We prove the convergence of these methods to the critical points of nonconvex optimization under reasonable conditions. We evaluate the performance and effectiveness of our methods through experimentation with three practical examples in machine learning. Our findings demonstrated that our methods offer efficiency in problem-solving and outperform state-of-the-art techniques like the DCA (DC Algorithm) and ADMM.

Keywords: Douglas-Rachford splitting algorithm; DC programming; Nonconvex optimization; Machine learning.

2010 MSC classification: 65K05, 90C26, 90C30.

1 Introduction

Let us consider the following class of generalized DC programming in a real Hilbert space H:

$$\min_{x\in H}\{p(x) = f(x) + g(x) - h(x)\}, \tag{1}$$

where $f, g : H \to (-\infty, +\infty]$ are proper, convex, and lower semicontinuous (not necessarily smooth) functions, and $h : H \to (-\infty, +\infty]$ is a convex and smooth

*School of Mathematical Sciences, Tiangong University, Tianjin 300387, China; and Center for Advanced Information Technology, Kyung Hee University, Seoul 02447, South Korea; e-mail: yyhtgu@hotmail.com

†School of Mathematical Sciences, University of Southampton, SO17 1BJ, United Kingdom; e-mail: l.o.jolaoso@soton.ac.uk.

‡(Corresponding Author) School of Mathematical Sciences, Zhejiang Normal University, Jinhua 321004, People's Republic of China; e-mail: yekini.shehu@zjnu.edu.cn

§Center for General Education, China Medical University, Taichung 40402, Taiwan, Academy of Romanian Scientists, Bucharest, Romania; e-mail: yaojc@mail.cmu.edu.tw

function. The DC programming (1) was first presented by Tao et al. [41], which has received attention due to its applications in image processing [46], compressed sensing [31], statistics and machine learning [19,27,28,35], dimensionality reduction [16] and multiple-input-multiple-output (MIMO) [54]. For example, in the application of DC programming (1) in machine learning, the function f stands for a loss function that denotes the data fidelity and $g-h$ represents a regularizer that induces some expected structures in the solution [29,30].

The DCA (DC algorithm) is one of the most prominent methods for solving DC programming (1). The DCA linearizes the concave part of the DC programming (1) at the current iteration and obtains the next iteration via a convex subproblem. Further studies on the DCA have been given in [20,32,53]. In [11, Algorithm 1], Chuang et al. introduced the following unified Douglas-Rachdford splitting algorithm to solve DC programming (1):

$$\begin{cases} y_n = \underset{v\in H}{\operatorname{argmin}}\Big\{f(v)+\frac{1}{2\beta}\|v-x_n\|^2\Big\}, \\ z_n = \underset{v\in H}{\operatorname{argmin}}\Big\{g(v)+\frac{1}{2\beta}\|v-(2y_n-x_n+\beta\nabla h(y_n))\|^2\Big\}, \\ x_{n+1} = x_n+\kappa_n(z_n-y_n), \end{cases} \tag{2}$$

with $\beta>0$ and $\kappa_n\in(0,2)$, and obtained, under certain conditions, that $\lim\limits_{n\to\infty}\|y_n-x\|=0$, $\lim\limits_{n\to\infty}x_n=y$, where x is a stationary point of DC programming (1) and $\operatorname{prox}_{\beta f}(y)=x$. In [4], Bian and Zhang extended the three-operator splitting algorithm of Davis-Yin [15] from solving the optimization problem of the sum of three convex functions to solving nonconvex optimization problems. This three-operator in [15] could be seen as a slight extension of the Douglas-Rachford splitting algorithm [18] and the generalized forward-backward splitting algorithm in [3,38].

Motivated by the forward-backward splitting algorithm with deviations proposed in [39], Hu and Dong [24] proposed the following weakly convergent three-operator splitting algorithm with deviations for solving DC programming (1):

$$\begin{cases} s_n = x_n+u_n-\beta Lv_n, \\ y_n = \underset{v\in H}{\operatorname{argmin}}\Big\{f(v)+\frac{1}{2\beta}\|v-s_n\|^2\Big\}, \\ t_n = y_n-\frac{\beta L}{2}v_n, \\ q_n = y_n+v_n, \\ z_n = \underset{v\in H}{\operatorname{argmin}}\Big\{g(v)+\frac{1}{2\beta}\|v-(2t_n-s_n+\beta\nabla h(q_n))\|^2\Big\}, \\ x_{n+1} = x_n+\kappa_n(z_n-y_n), \end{cases} \tag{3}$$

where the vectors u_{n+1} and v_{n+1} are chosen such that

$$\frac{\kappa_{n+1}}{2-\kappa_{n+1}}\|u_{n+1}\|^2+\kappa_{n+1}\beta L\|v_{n+1}\|^2\le\zeta_n l_n^2 \tag{4}$$

with

$$l_n^2=\kappa_n(2-\kappa_n)\|z_n-y_n+\frac{1}{2-\kappa_n}u_n\|^2. \tag{5}$$

Consequently, both one-step and two-step inertial three-operator splitting algorithms are deduced in [24, Algorithm 2, Algorithm 3] from the deviation vector u_n. Numerical comparisons of [24, Algorithm 2, Algorithm 3] with unified Douglas-Rachdford splitting algorithm (2) using DC regularized sparse recovery problems showed that [24, Algorithm 2, Algorithm 3] outperformed unified Douglas-Rachdford splitting algorithm (2) in terms of cpu time and number of iterations.

Contribution. Our aim in this paper is to continue the approach of Chuang et al. [11] and Hu and Dong [24] by designing a new splitting algorithm to solve DC programming (1) with the following contributions:

- we introduce new fast Douglas-Rachford splitting algorithms with the aim of solving DC programming (1) in Hilbert spaces, which are also extensions of the unified Douglas-Rachford splitting algorithm proposed in (2);
- in our proposed algorithms, we relax the strict norm conditions (4) and (5) used in [24] in proving the convergence of the methods to critical point of the nonconvex optimization problem;
- we provide some numerical experiments and implement the proposed methods for solving real-life problems in machine learning. The results of the experiments indicate the accuracy and efficiency of the proposed methods over popular methods such as the DC algorithm and ADMM in the literature.

Organization. We structure the rest of the paper as follows: in Section 2, we put in place some basic concepts and lemmas while in Section 3, we introduce our proposed algorithms and weak convergence analysis. Section 4 deals with numerical experiments where we implement the proposed methods in three DC models in machine learning. Some final remarks and future considerations are given in Section 5.

2 Preliminaries

In the convergence analysis of this paper, we shall denote the weak convergence by the symbol "$\rightharpoonup$" and denote the strong convergence by the symbol "$\to$".

The following definitions and basic concepts can be found in [3].

Definition 2.1. *A mapping $T : H \to H$ is called*

(i) nonexpansive if $\|Tx - Ty\| \leq \|x - y\|$, for all $x, y \in H$;

(ii) L-Lipschitz continuous if there exists $L > 0$ such that $\|Tx - Ty\| \leq L\|x - y\|$ for all $x, y \in H$.

We define the domain of a function $f : H \to (-\infty, +\infty]$ to be $\mathrm{dom} f := \{x \in H : f(x) < +\infty\}$ and say that f is proper if $\mathrm{dom} f \neq \emptyset$.

Definition 2.2. *Suppose $f : H \to (-\infty, +\infty]$ is proper. The subdifferential of f at x is defined by*

$$\partial f(x) := \{u \in H : f(z) \geq f(x) + \langle z - x, u\rangle, \forall z \in H\}.$$

We say that f is subdifferentiable at x if $\partial f(x) \neq \emptyset$. In this case, the elements of $\partial f(x)$ are termed the subgradients of f at x.

Definition 2.3. *An operator $A : H \to 2^H$ is said to be monotone if for any $x, y \in H$,*

$$\langle x - y, u - v\rangle \geq 0, \forall u \in Ax, v \in Ay. \tag{6}$$

The Graph of A is defined by

$$Gr(A) := \{(x, u) \in H \times H : u \in Ax\}.$$

If $Gr(A)$ is not properly contained in the graph of any other monotone mapping, then A is maximal monotone operator. It is well-known that for each $x \in H$, and $\lambda > 0$, there is a unique $z \in H$ such that $x \in (I+\lambda A)z$. Furthermore, A is ρ-strongly monotone (with $\rho > 0$) if

$$\langle x - y, u - v\rangle \geq \rho\|u - v\|^2, \forall u \in Ax, v \in Ay.$$

Definition 2.4. *(i) A function f is called convex if $\mathrm{dom} f$ is a convex set and if $\forall x, y \in \mathrm{dom} f, \delta \in [0, 1]$, we have*

$$f(\delta x + (1 - \delta)y) \leq \delta f(x) + (1 - \delta)f(y).$$

(ii) A function f is said to be ρ-strongly convex with $\rho > 0$ if $f - \frac{\rho}{2}\|.\|^2$ is convex. Thus, $\forall x, y \in \mathrm{dom} f, \delta \in [0, 1]$,

$$f(\delta x + (1 - \delta)y) \leq \delta f(x) + (1 - \delta)f(y) - \frac{\rho}{2}\theta(1 - \theta)\|x - y\|^2.$$

Moreover, if f is convex, we have

$$f(x) \geq f(y) + \langle u, x - y\rangle, x, y \in \mathrm{dom} f,$$

with $u \in \partial f(y)$ is arbitrary. Also, if f is ρ-strongly convex with $\rho > 0$, we have

$$f(x) \geq f(y) + \langle u, x - y\rangle + \frac{\rho}{2}\|x - y\|^2, x, y \in \mathrm{dom} f,$$

with $u \in \partial f(y)$ is arbitrary.

Definition 2.5. *Given $\beta > 0$ and $f : H \to (-\infty, +\infty]$ a proper, lower semicontinuous and convex function. The proximal operator of f with β is defined by*

$$\mathrm{prox}_{\beta f}(x) := \underset{u \in H}{\mathrm{argmin}}\Big\{f(u) + \frac{1}{2\beta}\|u - x\|^2\Big\}$$

for each $x \in H$.

Using the first-order optimality condition of the minimization $\min_{u\in H}\Big\{f(u)+\frac{1}{2\beta}\|u-x\|^2\Big\}$, we have that $\text{prox}_{\beta f}(x)=(I+\beta\partial f)^{-1}(x)$, where I is the identity on H.

Lemma 2.6. *([3, Example 22.4]) Suppose $\rho>0$ and $f:H\to(-\infty,+\infty]$ is a proper, lower-semicontinuous and convex function. If f is ρ-strongly convex, then ∂f is ρ-strongly monotone.*

Lemma 2.7. *([3, Proposition 16.36]) Suppose $f:H\to(-\infty,+\infty]$ is a proper, lower-semicontinuous and convex function. If $\{v_n\}$ and $\{x_n\}$ are sequences in H with $(x_n,v_n)\in Gr(\partial f)$ for each $n\in\mathbb{N}$, $x_n\to x$ and $v_n\rightharpoonup v$, then $(x,v)\in Gr(\partial f)$.*

Lemma 2.8. *If $x,y\in H$, we have*

(i) $2\langle x,y\rangle=\|x\|^2+\|y\|^2-\|x-y\|^2=\|x+y\|^2-\|x\|^2-\|y\|^2$.

(ii) Let $x,y\in H$ and $a\in\mathbb{R}$. Then

$$\|(1-a)x+ay\|^2=(1-a)\|x\|^2+a\|y\|^2-(1-a)a\|x-y\|^2.$$

Lemma 2.9. *([36]) Let K be a nonempty subset of a Hilbert space H and let $\{x_n\}$ be a bounded sequence in H. Assume the following two conditions are satisfied:*

- *$\lim_{n\to\infty}\|x_n-x\|$ exists for each $x\in K$,*
- *every weak cluster point of $\{x_n\}$ belongs to K.*

Then $\{x_n\}$ converges weakly to a point in K.

Definition 2.10. *We say that x is a stationary point of p in DC programming (1) if*

$$0\in\partial f(x)+\partial g(x)-\nabla h(x).$$

We shall denote by Ω, the set of all the stationary points of p in DC programming (1) and consequently assume that $\Omega\neq\emptyset$ throughout this paper.

In DC programming (1), assume for the rest of this paper that $\beta>0$ is given, f and g are ρ-strongly convex, and h is a smooth convex function having a Lipschitz continuous gradient with constant $L>0$. The following results have been established in [24, Proposition 3.1] regarding the set of stationary points of DC programming (1).

Lemma 2.11. *If $2\rho>L$ in DC programming (1), then the stationary point of p is unique.*

Lemma 2.12. *([11, Proposition 3.1]) Suppose $\beta>0$. Then $x\in\Omega$ if and only if there exists $y\in H$ such that*

$$\begin{cases} x &= \text{prox}_{\beta f}(y),\\ x &= \text{prox}_{\beta f}(2x-y+\beta\nabla h(x)).\end{cases}$$

In order to obtain our convergence analysis of the proposed Algorithm 1, we give the following conditions.

Assumption 2.13. Suppose the iterative parameters in Algorithm 1 fulfill these conditions:

(i) $2\rho > L$;

(ii) $0 < a \leq \kappa_n \leq b < 2$.

Going by Lemma 2.12, given $\beta > 0$, we set

$$\Gamma_\beta := \{y : \operatorname{prox}_{\beta f}(y) = x, \operatorname{prox}_{\beta f}(2x - y + \beta\nabla h(x)) = x, x \in \Omega\}.$$

3 Proposed Algorithm and Its Convergence

In this section, we introduce the proposed algorithms and obtain associated weak convergence results to the critical point of DC programming (1).

Algorithm 1 First Version of Douglas-Rachford Algorithm

1: **Initialization:** Choose $\beta > 0, \theta \geq 0$, and $x_0, v_0 \in H$ arbitrarily. Set $n = 0$.
2: **Iteration Step:** Given the iterates x_n, v_n, compute

$$\begin{cases} u_n = \frac{1}{1+\theta}x_n + \frac{\theta}{1+\theta}v_n, \\ y_n = \underset{v\in H}{\operatorname{argmin}}\Big\{f(v) + \frac{1}{2\beta}\|v - u_n\|^2\Big\}, \\ z_n = \underset{v\in H}{\operatorname{argmin}}\Big\{g(v) + \frac{1}{2\beta}\|v - (2y_n - u_n + \beta\nabla h(y_n))\|^2\Big\}, \\ x_{n+1} = u_n + \kappa_n(z_n - y_n), \\ v_{n+1} = \frac{1}{1+\theta}x_{n+1} + \frac{\theta}{1+\theta}v_n. \end{cases} \tag{7}$$

Set $n \leftarrow n + 1$, and go to **Iteration Step** if the stopping criterion is not met.

Remark 3.1. If $\theta = 0$ in Algorithm 1, then $u_n = x_n$ and $v_{n+1} = x_{n+1}$. Consequently, we obtain [11, Algorithm 1]. Therefore, [11, Algorithm 1] is recovered from Algorithm 1 when $\theta = 0$. It suffices to obtain the weak convergence of Algorithm 1 when $\theta > 0$ in Theorem 3.3 below.

We can easily obtain this lemma by following same arguments as in [11, Theorem 3.1]. For the sake of completeness and to make our results reader-friendly, we include the complete proof.

Lemma 3.2. *Suppose $\beta, \kappa > 0$, for each $u \in H$, let*

$$\begin{cases} y = \underset{v\in H}{\operatorname{argmin}}\Big\{f(v) + \frac{1}{2\beta}\|v - u\|^2\Big\}, \\ z = \underset{v\in H}{\operatorname{argmin}}\Big\{g(v) + \frac{1}{2\beta}\|v - (2y - u + \beta\nabla h(y))\|^2\Big\}, \\ x = u + \kappa(z - y). \end{cases} \tag{8}$$

Then for any $\bar{x} \in \Omega$, there exists $\bar{y} \in \Gamma_\beta$ such that

$$\begin{aligned} \|x - \bar{y}\|^2 &\leq \|u - \bar{y}\|^2 - \beta\kappa(2\rho - L)\Big(\|y - \bar{x}\|^2 + \|z - \bar{x}\|^2\Big) \\ &\quad -\kappa(2-\kappa)\|z - y\|^2. \end{aligned} \tag{9}$$

Proof. Using the first-order optimality condition on $y = \mathrm{prox}_{\beta f}(u)$ and $z = \mathrm{prox}_{\beta g}(2y - u + \beta\nabla h(y))$, we obtain

$$\frac{1}{\beta}(u - y) \in \partial f(y), \tag{10}$$

and

$$\nabla h(y) + \frac{1}{\beta}(2y - z - u) \in \partial g(z) \tag{11}$$

respectively. Now for any $\bar{x} \in \Omega$, there exists a $\bar{y} \in \Gamma_\beta$ such that

$$\frac{1}{\beta}(\bar{y} - \bar{x}) \in \partial f(\bar{x}) \tag{12}$$

and

$$\text{and} \quad \frac{1}{\beta}(\bar{x} - \bar{y}) + \nabla h(\bar{x}) \in \partial g(\bar{x}). \tag{13}$$

By observing that f is ρ-strongly convex, we have from Lemma 2.6 that ∂f is ρ-strongly monotone. Consequently,

$$\langle \zeta_y - \zeta_{\bar{x}}, y - \bar{x}\rangle \geq \rho\|y - \bar{x}\|^2, \quad \forall \zeta_y \in \partial f(y), \zeta_{\bar{x}} \in \partial f(\bar{x}). \tag{14}$$

If we apply (14) in (10) and (12), we obtain

$$\rho\|y - \bar{x}\|^2 \leq \frac{1}{\beta}\langle y - \bar{x}, u - y - \bar{y} + \bar{x}\rangle$$

which implies that

$$(2 + 2\beta\rho)\|y - \bar{x}\|^2 \leq 2\langle y - \bar{x}, u - \bar{y}\rangle. \tag{15}$$

Similarly, let us apply the property that g is ρ-strongly convex in (11) and (13). Then we obtain

$$\rho\|z - \bar{x}\|^2 \leq \langle z - \bar{x}, \nabla h(y) - \nabla h(\bar{x}) + \frac{1}{\beta}(2y - z - u) - \frac{1}{\beta}(\bar{x} - \bar{y})\rangle. \tag{16}$$

If we rearrange (16), noting the Cauchy-Schwarz inequality and Lipschitz continuity of ∇h, we get

$$\begin{aligned} 2\beta\rho\|z - \bar{x}\|^2 &\leq 2\beta\langle z - \bar{x}, \nabla h(y) - \nabla h(\bar{x}) + \frac{1}{\beta}(2y - z - u) - \frac{1}{\beta}(\bar{x} - \bar{y})\rangle \\ &= 2\beta\langle z - \bar{x}, \nabla h(y) - \nabla h(\bar{x})\rangle + 2\langle z - \bar{x}, 2y - z - u - \bar{x}\rangle \\ &\leq 2\beta L\|z - \bar{x}\|\|y - \bar{x}\| + 2\langle z - \bar{x}, 2y - z - u - \bar{x}\rangle \\ &\leq \beta L\|z - \bar{x}\|^2 + \beta L\|y - \bar{x}\|^2 + \|z - \bar{x}\|^2 \\ &\quad +\|2y - z - u + \bar{y} - \bar{x}\|^2 - \|2z - 2y + u - \bar{y}\|^2. \end{aligned} \tag{17}$$

Therefore, we have from (17) that

$$\begin{aligned}\|2z-2y+u-\bar{y}\|^2 &\leq (1+\beta L-2\beta\rho)\|z-\bar{x}\|^2+\beta L\|y-\bar{x}\|^2\\ &\quad +\|2y-z-u+\bar{y}-\bar{x}\|^2. \end{aligned} \tag{18}$$

Now, consider (noting (15))

$$\begin{aligned}\|2y-z-u+\bar{y}-\bar{x}\|^2 &= \|(y-z)+(y-\bar{x})-(u-\bar{y})\|^2\\ &= \|y-z\|^2+\|y-\bar{x}\|^2+\|u-\bar{y}\|^2+2\langle y-\bar{x},y-z\rangle\\ &\quad -2\langle u-y,y-z\rangle-2\langle u-\bar{y},y-\bar{x}\rangle\\ &= 2\|y-z\|^2+2\|y-\bar{x}\|^2+\|u-\bar{y}\|^2-\|z-\bar{x}\|^2\\ &\quad -2\langle u-\bar{y},y-z\rangle-2\langle u-\bar{y},y-\bar{x}\rangle\\ &\leq 2\|y-z\|^2+2\|y-\bar{x}\|^2+\|u-\bar{y}\|^2-\|z-\bar{x}\|^2\\ &\quad -2\langle u-\bar{y},y-z\rangle-(2+2\beta\rho)\|y-\bar{x}\|^2\\ &= 2\|y-z\|^2+\|u-\bar{y}\|^2-\|z-\bar{x}\|^2-2\beta\rho\|y-\bar{x}\|^2\\ &\quad -2\langle u-\bar{y},y-\bar{x}\rangle-2\langle u-\bar{y},\bar{x}-z\rangle\\ &\leq 2\|y-z\|^2+\|u-\bar{y}\|^2-\|z-\bar{x}\|^2\\ &\quad -(2+4\beta\rho)\|y-\bar{x}\|^2+2\langle u-\bar{y},z-\bar{x}\rangle. \end{aligned} \tag{19}$$

We next estimate the term $2\langle u-\bar{y},z-\bar{x}\rangle$ in (19). Now, if we look at (16) and we rearrange, we obtain

$$\begin{aligned}2\langle u-\bar{y},z-\bar{x}\rangle &\leq 2\beta\langle z-\bar{x},\nabla h(y)-\nabla h(\bar{x})\rangle+2\langle z-\bar{x},y-z\rangle\\ &\quad +2\langle z-\bar{x},y-\bar{x}\rangle-2\beta\rho\|z-\bar{x}\|^2\\ &\leq \beta L\|z-\bar{x}\|^2+\beta L\|y-\bar{x}\|^2+\|y-\bar{x}\|^2-\|z-y\|^2\\ &\quad -\|z-\bar{x}\|^2+\|z-\bar{x}\|^2+\|y-\bar{x}\|^2-\|z-y\|^2\\ &\quad -2\beta\rho\|z-\bar{x}\|^2\\ &= (\beta L-2\beta\rho)\|z-\bar{x}\|^2+(2+\beta L)\|y-\bar{x}\|^2\\ &\quad -2\|z-y\|^2. \end{aligned} \tag{20}$$

Plugging (20) into (19), we get

$$\begin{aligned}\|2y-z-u+\bar{y}-\bar{x}\|^2 &\leq 2\|y-z\|^2+\|u-\bar{y}\|^2-\|z-\bar{x}\|^2\\ &\quad -(2+4\beta\rho)\|y-\bar{x}\|^2+(\beta L-2\beta\rho)\|z-\bar{x}\|^2\\ &\quad +(2+\beta L)\|y-\bar{x}\|^2\\ &\quad -2\|z-y\|^2\\ &= \|u-\bar{y}\|^2+(\beta L-2\beta\rho-1)\|z-\bar{x}\|^2\\ &\quad +(\beta L-4\beta\rho)\|y-\bar{x}\|^2. \end{aligned} \tag{21}$$

Now, let us plug (21) into (18) to obtain

$$\|2z-2y+u-\bar{y}\|^2 \leq (1+\beta L-2\beta\rho)\|z-\bar{x}\|^2+\beta L\|y-\bar{x}\|^2$$

$$
\begin{aligned}
&+\|u-\bar{y}\|^2+(\beta L-2\beta\rho-1)\|z-\bar{x}\|^2\\
&+(\beta L-4\beta\rho)\|y-\bar{x}\|^2\\
=\;&\|u-\bar{y}\|^2-2\beta(2\rho-L)\Big(\|y-\bar{x}\|^2+\|z-\bar{x}\|^2\Big).
\end{aligned}
$$

We then obtain from $x=u+\kappa(z-y)$ that (noting immediate last inequality)

$$
\begin{aligned}
\|x-\bar{y}\|^2 &= \|u+\kappa(z-y)-\bar{y}\|^2\\
&= \|\Big(1-\frac{\kappa}{2}\Big)(u-\bar{y})+\frac{\kappa}{2}(2z-2y+u-\bar{y})\|^2\\
&= \Big(1-\frac{\kappa}{2}\Big)\|u-\bar{y}\|^2+\frac{\kappa}{2}\|2z-2y+u-\bar{y}\|^2\\
&\quad -\frac{\kappa}{2}\Big(1-\frac{\kappa}{2}\Big)\|(2z-2y+u-\bar{y})-(u-\bar{y})\|^2\\
&= \Big(1-\frac{\kappa}{2}\Big)\|u-\bar{y}\|^2+\frac{\kappa}{2}\|2z-2y+u-\bar{y}\|^2\\
&\quad -\kappa(2-\kappa)\|z-y\|^2\\
&\leq \Big(1-\frac{\kappa}{2}\Big)\|u-\bar{y}\|^2+\frac{\kappa}{2}\|u-\bar{y}\|^2\\
&\quad -\beta\kappa(2\rho-L)\Big(\|y-\bar{x}\|^2+\|z-\bar{x}\|^2\Big)-\kappa(2-\kappa)\|z-y\|^2\\
&= \|u-\bar{y}\|^2-\beta\kappa(2\rho-L)\Big(\|y-\bar{x}\|^2+\|z-\bar{x}\|^2\Big)\\
&\quad -\kappa(2-\kappa)\|z-y\|^2.
\end{aligned}
$$

□

We now give the following weak convergence result of Algorithm 1 using Lemma 3.2.

Theorem 3.3. *The sequence $\{x_n\}$ generated by Algorithm 1 converges weakly to a point in Γ_β under Assumption 2.13.*

Proof. We shall only consider the case $\theta>0$ in our convergence analysis since the case $\theta=0$ has already been considered in [11, Algorithm 1] with convergence results given in [11, Theorem 3.1]. Now by Lemma 2.8 (ii), we have from Algorithm 1 that (putting $\bar{y}=y$ and $\bar{x}=x$ in Lemma 3.2)

$$
\begin{aligned}
\|u_n-y\|^2 &= \big\|\frac{1}{1+\theta}(x_n-y)+\frac{\theta}{1+\theta}(v_n-y)\big\|^2\\
&= \frac{1}{1+\theta}\|x_n-y\|^2+\frac{\theta}{1+\theta}\|v_n-y\|^2-\frac{\theta}{(1+\theta)^2}\|x_n-v_n\|^2 \qquad (22)
\end{aligned}
$$

and

$$
\|v_{n+1}-y\|^2 = \frac{\theta}{1+\theta}\|v_n-y\|^2+\frac{1}{1+\theta}\|x_{n+1}-y\|^2-\frac{\theta}{(1+\theta)^2}\|x_{n+1}-v_n\|^2. \qquad (23)
$$

If we plug (22) into (9) in Lemma 3.2, we have

$$
\|x_{n+1}-y\|^2 \leq \|u_n-y\|^2-\beta\kappa_n(2\rho-L)\Big(\|y_n-x\|^2+\|z_n-x\|^2\Big)
$$

$$
\begin{aligned}
& -\kappa_n(2-\kappa_n)\|z_n - y_n\|^2 \\
= \; & \frac{1}{1+\theta}\|x_n - y\|^2 + \frac{\theta}{1+\theta}\|v_n - y\|^2 \\
& -\frac{\theta}{(1+\theta)^2}\|x_n - v_n\|^2 - \beta\kappa_n(2\rho - L)\Big(\|y_n - x\|^2 + \|z_n - x\|^2\Big) \\
& -\kappa_n(2-\kappa_n)\|z_n - y_n\|^2. \qquad (24)
\end{aligned}
$$

We then obtain from (23) and (24) that

$$
\begin{aligned}
\|x_{n+1} - y\|^2 \; + \; & \theta\|v_{n+1} - y\|^2 \le \frac{1}{1+\theta}\|x_n - y\|^2 + \frac{\theta}{1+\theta}\|v_n - y\|^2 \\
& -\frac{\theta}{(1+\theta)^2}\|x_n - v_n\|^2 - \beta\kappa_n(2\rho - L)\Big(\|y_n - x\|^2 + \|z_n - x\|^2\Big) \\
& -\kappa_n(2-\kappa_n)\|z_n - y_n\|^2 + \frac{\theta}{1+\theta}\|x_{n+1} - y\|^2 + \frac{\theta^2}{1+\theta}\|v_n - y\|^2 \\
& -\frac{\theta^2}{(1+\theta)^2}\|x_{n+1} - v_n\|^2. \qquad (25)
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
& \frac{1}{1+\theta}\|x_{n+1} - y\|^2 + \theta\|v_{n+1} - y\|^2 + \frac{\theta^2}{(1+\theta)^2}\|x_{n+1} - v_n\|^2 \\
\le \; & \frac{1}{1+\theta}\|x_n - y\|^2 + \theta\|v_n - y\|^2 - \frac{\theta}{(1+\theta)^2}\|x_n - v_n\|^2 \\
& -\beta\kappa_n(2\rho - L)\Big(\|y_n - x\|^2 + \|z_n - x\|^2\Big) \\
& -\kappa_n(2-\kappa_n)\|z_n - y_n\|^2 + \frac{\theta^2}{(1+\theta)^2}\|x_n - v_{n-1}\|^2 \\
& -\frac{\theta^2}{(1+\theta)^2}\|x_n - v_{n-1}\|^2 \\
= \; & \frac{1}{1+\theta}\|x_n - y\|^2 + \theta\|v_n - y\|^2 + \frac{\theta^2}{(1+\theta)^2}\|x_n - v_{n-1}\|^2 \\
& -\frac{\theta^2}{(1+\theta)^2}\|x_n - v_{n-1}\|^2 - \beta\kappa_n(2\rho - L)\Big(\|y_n - x\|^2 + \|z_n - x\|^2\Big) \\
& -\kappa_n(2-\kappa_n)\|z_n - y_n\|^2 - \frac{\theta}{(1+\theta)^2}\|x_n - v_n\|^2. \qquad (26)
\end{aligned}
$$

Define

$$
\begin{aligned}
c_n : \; = \; & \frac{1}{1+\theta}\|x_n - y\|^2 + \theta\|v_n - y\|^2 \\
& + \frac{\theta^2}{(1+\theta)^2}\|x_n - v_{n-1}\|^2. \qquad (27)
\end{aligned}
$$

Then, (26) becomes

$$
c_{n+1} \; \le \; c_n - \frac{\theta^2}{(1+\theta)^2}\|x_n - v_{n-1}\|^2 - \beta\kappa_n(2\rho - L)\Big(\|y_n - x\|^2 + \|z_n - x\|^2\Big)
$$

$$-\kappa_n(2-\kappa_n)\|z_n-y_n\|^2-\frac{\theta}{(1+\theta)^2}\|x_n-v_n\|^2. \tag{28}$$

Thus, $\{c_n\}$ is monotone non-increasing and $\{c_n\}$ is convergent. Moreover, $\{c_n\}$ is bounded. Furthermore, we obtain from (28) that

$$\lim_{n\to\infty}\|x_n-v_{n-1}\|=0, \quad \lim_{n\to\infty}\|x_n-v_n\|=0.$$

Furthermore, we have from (28) that

$$\beta\kappa_n(2\rho-L)\Big(\|y_n-x\|^2+\|z_n-x\|^2\Big)\leq c_n-c_{n+1},$$

which implies that

$$\lim_{n\to\infty}\beta\kappa_n(2\rho-L)\Big(\|y_n-x\|^2+\|z_n-x\|^2\Big)=0.$$

Since $2\rho>L$ and $0<a\leq\kappa_n\leq b<2$, we get

$$\lim_{n\to\infty}\|y_n-x\|=0, \quad \text{and} \quad \lim_{n\to\infty}\|z_n-x\|=0. \tag{29}$$

Let us define

$$s_n:=\|v_n-x_n\|^2+2\langle v_n-x_n,x_n-y\rangle$$

and

$$t_n:=-\frac{\theta}{(1+\theta)^2}\|x_n-v_{n-1}\|^2.$$

Since $\{x_n\}$ is bounded and $\lim_{n\to\infty}\|x_n-v_n\|=0$, we have $\lim_{n\to\infty}s_n=0$ and $t_n\to 0$, as $n\to\infty$. Noting that

$$\begin{aligned}\|v_n-y\|^2 &= \|v_n-x_n\|^2+2\langle v_n-x_n,x_n-y\rangle\\ &\quad +\|x_n-y\|^2\end{aligned}$$

and

$$\begin{aligned}s_n &= \|v_n-x_n\|^2+2\langle v_n-x_n,x_n-y\rangle\\ &= \|v_n-y\|^2-\|x_n-y\|^2.\end{aligned}$$

We then obtain

$$c_n-\theta s_n+\theta t_n=\frac{\theta^2+\theta+1}{1+\theta}\|x_n-y\|^2.$$

By the existence of the limit $\lim_{n\to\infty}c_n$, we get $\lim_{n\to\infty}\|x_n-y\|$ exists. We conclude by Lemma 2.9 that $\{x_n\}$ converges weakly to a point in Γ_β. Consequently, $\{v_n\}$ also converges weakly to a point in Γ_β. This completes the proof.

□

Remark 3.4. One can obtain from (28) that

$$\frac{\theta}{(1+\theta)^2}\|x_n-v_n\|^2\leq c_n-c_{n+1}$$

and thus

$$\frac{\theta}{(1+\theta)^2}\sum_{j=0}^{n}\|x_j - v_j\|^2 \le \sum_{j=0}^{n}(c_n - c_{n+1}) \le c_0,$$

which implies that (if $x_0 = v_0$)

$$(n+1)\frac{\theta}{(1+\theta)^2}\min_{0\le j\le n}\|x_j - v_j\|^2 \le c_0 = \frac{1+\theta(1+\theta)}{1+\theta}\|x_0 - y\|^2.$$

Consequently, we have

$$\min_{0\le j\le n}\|x_j - v_j\| \le \mathcal{O}(\frac{1}{\sqrt{n+1}}).$$

In the same way, one can obtain

$$\min_{0\le j\le n}\|x_j - v_{j-1}\| \le \mathcal{O}(\frac{1}{\sqrt{n+1}}), \min_{0\le j\le n}\|z_j - y_j\| \le \mathcal{O}(\frac{1}{\sqrt{n+1}}).$$

We design another version of Douglas-Rachford splitting algorithm for DC programming (1) and obtain weak convergence results.

Algorithm 2 Second Version of Douglas-Rachford Algorithm

1: **Initialization:** Choose $\alpha_n \in [0,1)$, $\beta > 0$, and $x_0, v_0 \in H$ arbitrarily. Set $n = 0$.
2: **Iteration Step:** Given the iterates x_n, v_n, compute

$$\begin{cases} u_n = (1-\alpha_n)x_n + \alpha_n v_n, \\ y_n = \operatorname{argmin}_{v\in H}\Big\{f(v) + \frac{1}{2\beta}\|v - u_n\|^2\Big\}, \\ z_n = \operatorname{argmin}_{v\in H}\Big\{g(v) + \frac{1}{2\beta}\|v - (2y_n - u_n + \beta\nabla h(y_n))\|^2\Big\}, \\ x_{n+1} = u_n + \kappa_n(z_n - y_n), \\ v_{n+1} = (1-\alpha_n)v_n + \alpha_n x_n. \end{cases} \tag{30}$$

Set $n \leftarrow n+1$, and go to **Iteration Step** if the stopping criterion is not met.

Since our Algorithm 2 reduces to [11, Algorithm 1] when $\alpha_n = 0$ for each $n \in \mathbb{N}$ for which the convergence results have been obtained in [11]. It suffices to only consider the case $0 < a \le \alpha_n \le b < 1$ in our convergence result below for Algorithm 2.

Theorem 3.5. *The sequence $\{x_n\}$ generated by Algorithm 2 converges weakly to a point in Γ_β under Assumption 2.13 when either $0 < a \le \alpha_n \le b < 1$ or $\alpha_n = 0$ for each $n \in \mathbb{N}$.*

Proof. We can easily obtain from Algorithm 2 and Lemma 3.2 that (with $\bar{y} = y$)

$$\begin{aligned}\|x_{n+1} - y\|^2 \;\le\; & \|u_n - y\|^2 - \beta\kappa_n(2\rho - L)\Big(\|y_n - x\|^2 + \|z_n - x\|^2\Big) \\ & -\kappa_n(2-\kappa_n)\|z_n - y_n\|^2. \end{aligned} \tag{31}$$

By Lemma 2.8(ii), we have

$$\begin{aligned}\|u_n - y\|^2 &= \big\|(1-\alpha_n)(x_n - y) + \alpha_n(v_n - y)\big\|^2 \\ &= (1-\alpha_n)\|x_n - y\|^2 + \alpha_n\|v_n - y\|^2 - \alpha_n(1-\alpha_n)\|x_n - v_n\|^2 \end{aligned} \tag{32}$$

and

$$\begin{aligned}\|v_{n+1} - y\|^2 &= \|(1-\alpha_n)(v_n - y) + \alpha_n(x_n - y)\|^2 \\ &= (1-\alpha_n)\|v_n - y\|^2 + \alpha_n\|x_n - y\|^2 - \alpha_n(1-\alpha_n)\|x_n - v_n\|^2. \end{aligned} \tag{33}$$

Plugging (32) into (31), we obtain

$$\begin{aligned}\|x_{n+1} - y\|^2 \leq\ & (1-\alpha_n)\|x_n - y\|^2 + \alpha_n\|v_n - y\|^2 \\ & -\alpha_n(1-\alpha_n)\|x_n - v_n\|^2 - \beta\kappa_n(2\rho - L)\Big(\|y_n - x\|^2 + \|z_n - x\|^2\Big) \\ & -\kappa_n(2-\kappa_n)\|z_n - y_n\|^2. \end{aligned} \tag{34}$$

Denote

$$a_n := \|x_n - y\|^2 + \|v_n - y\|^2, \quad \forall n \in \mathbb{N}. \tag{35}$$

Then, summing (33) and (34) gives

$$\begin{aligned}a_{n+1} \leq\ & a_n - 2\alpha_n(1-\alpha_n)\|x_n - v_n\|^2 - \beta\kappa_n(2\rho - L)\Big(\|y_n - x\|^2 + \|z_n - x\|^2\Big) \\ & -\kappa_n(2-\kappa_n)\|z_n - y_n\|^2, \end{aligned} \tag{36}$$

which, by $\alpha_n \in [0,1)$, and $\kappa_n \in (0,2)$, implies that $\{a_n\}$ is a monotone non-increasing and $\{a_n\}$ is convergent. Moreover, $\{a_n\}$ is bounded. We then obtain from the definition of $\{a_n\}$ that both $\{x_n\}$ and $\{v_n\}$ are bounded. Furthermore, $\{u_n\}$, $\{y_n\}$ and $\{z_n\}$ are also bounded.

We also obtain from (36) that

$$\beta\kappa_n(2\rho - L)\Big(\|y_n - x\|^2 + \|z_n - x\|^2\Big) \leq a_n - a_{n+1},$$

which implies that

$$\lim_{n\to\infty} \beta\kappa_n(2\rho - L)\Big(\|y_n - x\|^2 + \|z_n - x\|^2\Big) = 0.$$

Since $2\rho > L$ and $0 < a \leq \kappa_n \leq b < 2$, we get

$$\lim_{n\to\infty} \|y_n - x\| = 0, \quad \text{and} \quad \lim_{n\to\infty} \|z_n - x\| = 0. \tag{37}$$

By the condition that $\liminf_{n\to\infty} \kappa_n(2-\kappa_n) > 0$, we also have from (36) that

$$\lim_{n\to\infty} \|z_n - y_n\| = 0.$$

Also from (36),

$$\lim_{n\to\infty} 2\alpha_n(1-\alpha_n)\|x_n - v_n\|^2 = 0.$$

Hence,

$$\lim_{n\to\infty}\|x_n - v_n\| = 0.$$

We have from $v_{n+1} = (1-\alpha_n)v_k + \alpha_n x_n$ that

$$v_{n+1} - v_n = \alpha_n(x_n - v_n) \to 0, n \to \infty.$$

So,

$$\|x_{n+1} - x_n\| \le \|x_{n+1} - v_{n+1}\| + \|v_{n+1} - v_n\| + \|x_n - v_n\| \to 0, n \to \infty.$$

Since $\{x_n\}$, $\{y_n\}$ are $\{z_n\}$ are bounded, let $v \in H$ be a weak cluster point of $\{x_n\}$. Then there exists $\{x_{n_j}\} \subset \{x_n\}$ such that $x_{n_j} \rightharpoonup v$, $j \to \infty$. Then $x_{n_j+1} \rightharpoonup v$, $j \to \infty$, $z_{n_j} \rightharpoonup v$, $j \to \infty$ and $u_{n_j} \rightharpoonup v$, $j \to \infty$. Replacing n with n_j in (10) and (11), we have from (37) and Lemma 2.7 that

$$\frac{1}{\beta}(v - x) \in \partial f(x) \quad \text{and} \quad \nabla h(x) + \frac{1}{\beta}(x - v) \in \partial g(x).$$

This implies that $\text{prox}_{\beta f}(v) = x$ and $\text{prox}_{\beta g}(2x - v + \beta\nabla h(x)) = x$. We then obtain that $v \in \Gamma_\beta$.

We now show that $\{x_n\}$ converges weakly to an element in Γ_β. Define

$$c_n := \|v_n - x_n\|^2 + 2\langle v_n - x_n, x_n - y\rangle.$$

Since $v_n - x_n \to 0$, as $n \to \infty$ and $\{x_n\}$ is bounded, we have that $c_n \to 0$, as $n \to \infty$. Observe also that

$$\|v_n - y\|^2 = \|v_n - x_n\|^2 + 2\langle v_n - x_n, x_n - y\rangle + \|x_n - y\|^2$$

which implies that

$$c_n = \|v_n - y\|^2 - \|x_n - y\|^2.$$

Combine it with the definition of a_n to have

$$\|x_n - y\|^2 = \frac{1}{2}\big(a_n - c_n\big).$$

Since $\lim_{n\to\infty} a_n$ exists, then $\lim_{k\to\infty}\|x_n - y\|$ exists. By Lemma 2.9, we conclude that $\{x_n\}$ converges weakly to a point in Γ_β. The proof is complete. □

Remark 3.6. The proposed Algorithm 2 can be viewed as an extension of an inertial-type algorithm. In particular, the following is a special case of (30) in Algorithm 2:

$$\begin{cases} u_{n-1} = v_n + \theta_{n-1}(v_n - v_{n-1}), \\ y_{n-1} = \underset{v\in H}{\text{argmin}}\Big\{f(v) + \frac{1}{2\beta}\|v - u_{n-1}\|^2\Big\}, \\ z_{n-1} = \underset{v\in H}{\text{argmin}}\Big\{g(v) + \frac{1}{2\beta}\|v - (2y_{n-1} - u_{n-1} + \beta\nabla h(y_{n-1}))\|^2\Big\}, \\ x_n = u_{n-1} + \kappa_{n-1}(z_{n-1} - y_{n-1}), \\ v_{n+1} = (1-\alpha_n)v_n + \alpha_n x_n \end{cases} \tag{38}$$

where $\theta_n := \frac{1-2\alpha_n}{\alpha_n} \in (-1, +\infty)$ and $0 < a \leq \alpha_n \leq b < 1$. To see this, suppose $\bar{u}_n := v_n - x_n$. Then the first update in (30) implies

$$u_n = v_{n+1} + \theta_n(v_{n+1} - v_n) = x_n + \bar{u}_n + (1 + \theta_n)(v_{n+1} - v_n). \tag{39}$$

We obtain from the last update in (30) that $v_{n+1} - v_n = -\alpha_n \bar{u}_n$. If we substitute this into (39) together with $\theta_n = \frac{1-2\alpha_n}{\alpha_n}$, we have

$$\begin{aligned} u_n &= x_n + (1 - \alpha_n - \theta_n\alpha_n)\bar{u}_n \\ &= x_n + (1 - \alpha_n - \theta_n\alpha_n)(v_n - x_n) \\ &= (1 - \alpha_n)x_n + \alpha_n v_n. \end{aligned}$$

Since $\theta_n = \frac{1-2\alpha_n}{\alpha_n} \in (-1, +\infty)$, we have $\alpha_n = \frac{1}{2+\theta_n} \in (0, 1) \subset [0, 1)$. Therefore, (38) is a special case of (30).

4 Numerical Experiments

The numerical experiments is devoted to the implementation of the proposed algorithms in solving real-life problems. Three examples are considered where we applied the algorithms to solve optimization models in machine learning and compare their performance with other methods in the literature such as [11, Algorithm 1] and [24, Algorithm 2] and the popular ADMM method. All codes are written on Python using Jupyter Notebook. Also, the datasets used are source from open source such as the UCI database.

4.1 Tuning of parameters

It is well known that control parameters are very important to the performance of the iterative algorithms. Hence, we start the numerical section by studying the behaviour of the proposed algorithms to change in the control parameters. For this purpose, we tune the values of the parameters θ and β in the Douglas-Rashford splitting Algorithm 1. We consider the following optimization model:

$$\min_{x \in \mathbb{R}^n} f(x) + g(x) - h(x) \tag{40}$$

where $f : \mathbb{R}^n \to [+\infty, -\infty)$ is the quadratic function defined by $f(x) = \frac{1}{2}x^\top Qx + c^\top x + d$, with $Q \in \mathbb{R}^{M\times N}$ is a positive definite matrix, $c \in \mathbb{R}^N$ is a constant vector and $d \in \mathbb{R}$ is a constant scalar; $g : \mathbb{R}^n \to \mathbb{R}$ is the ℓ_1 regularization term and $h : \mathbb{R}^n \to \mathbb{R}$ is the Huber loss function given by

$$h(x) = \begin{cases} \frac{1}{2}x^2, & \|x\|_2 \leq \delta, \\ \delta(|x| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

and δ being a constant scalar. This type of model is very common in machine learning. For example, in robust classification with sparsity regularization (see, e.g. [53]), problem (40) is used to learn a robust classifier that is also sparse. The Huber loss function is used in this case to handle noisy data, the quadratic term is

used to regularize the model complexity and the ℓ_1 regularization term is used to induce sparsity in the model parameters. Also problem (40) is used in to reconstruct a signal from noisy measurements while promoting sparsity in signal processing (see, e.g. [8]). The quadratic fidelity term is used to measure the fidelity of the reconstructed signal, the ℓ_1 regularization term is used to enforce sparsity and the Huber loss function is used to handle outliers or noise in the measurement.
The proximal operator of the quadratic term is calculated analytically, which is given by

$$\text{prox}_{\beta f}(x) = \left(Q + \frac{1}{\beta} I\right)^{-1} \left(\frac{1}{\beta} x - c\right),$$

where I is the identity operator (see Appendix for details). To handle the inverse of the matrix Q in the case where it is a non-square matrix, we used the pseudo-inverse Q^+ of Q which is calculated by $Q^+ = (Q^\top Q)^{-1} Q^\top$, where Q^+ is the transpose of Q. This allows us to handle non-square of Q properly. In the first experiment, we take $\kappa_n = \kappa = 0.009$, $\delta = 0.001$, max_iter $= 1000$. The algorithm is stopped if $Err = \|x_{n+1} - x_n\|^2 < 10^{-5}$ or max_iter is reached. In the first instance, we choose $\theta = 0.0005, 0.05, 0.5, 1, 5$ and vary the values of (M, N) as $(100, 25)$, $(200, 100)$, $(250, 150)$ and $(300, 200)$. The starting points x_0 and v_0 are generated randomly. The results of the experiments are shown in Table 1 and Figure 1.
Furthermore, taking $\theta = 0.0005$, we test the algorithm for $\beta = 0.0001, 0.001, 0.1, 1, 10$ and vary the value of (M, N) as $(100, 25)$, $(200, 100)$, $(300, 150)$, (400,200), (500,300) and $(600, 400)$. The results are shown in Table 2 and 2.

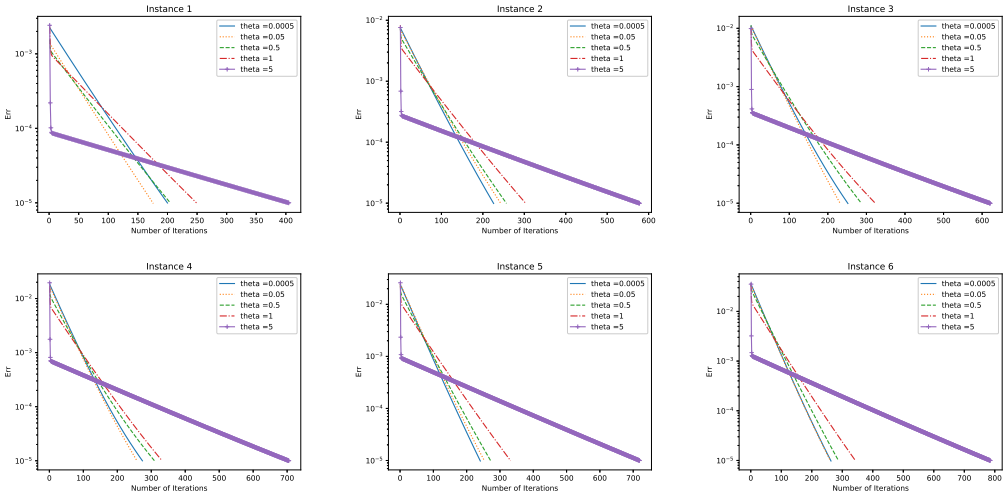

Figure 1: Comparing the performance of proposed Douglas-Rashford splitting algorithm for various values of θ in Example 4.1 using various dimension of (M, N). Instance 1: (100,25); Instance 2: (200,100), Instance 3: (300,150), Instance 4: (400, 200), Instance 5: (500,300) and Instance 6: (600,400).

The experimental results reveal several important trends regarding the performance of the Douglas-Rachford splitting algorithm. As θ increases beyond 1, the algorithm's performance deteriorates, particularly for larger problem dimensions, indicating that higher values of θ lead to instability and inefficiency. Conversely, smaller values of θ improve the algorithm's performance, with the optimal value

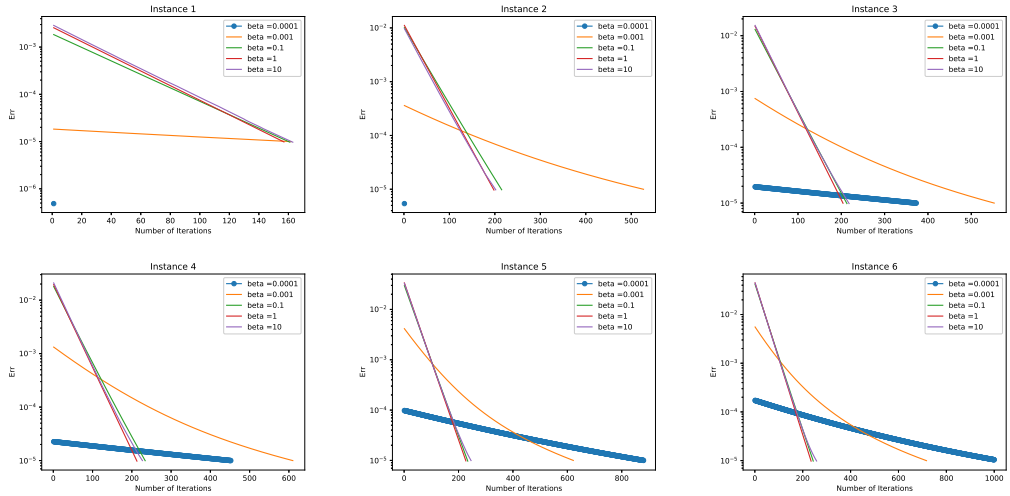

Figure 2: Comparing the performance of proposed Douglas-Rashford splitting algorithm for various values of β in Example 4.1 using various dimension of (M, N). Instance 1: (100,25); Instance 2: (200,100), Instance 3: (300,150), Instance 4: (400, 200), Instance 5: (500,300) and Instance 6: (600,400).

(M, N)	$\theta = 0.0005$ Iter/Time	$\theta = 0.05$ Iter/Time	$\theta = 0.5$ Iter/Time	$\theta = 1$ Iter/Time	$\theta = 5$ Iter/Time
(100,25)	201/ 0.0712	177/0.0532	206/0.0473	250/0.0626	406/0.1098
(200,100)	227/0.2670	244/0.3138	258/0.3298	304/0.4152	579/0.6904
(300,150)	253/0.276	233/0.2830	288/0.3451	324/0.3999	622/0.8319
(400,250)	275/0.5952	259/0.5378	311/0.6616	333/4.2368	706/1.5850
(600,300)	242/2.7268	254/2.5572	273/2.8729	333/4.2368	720/9.0847
(600,400)	264/6.1672	262/5.8376	289/6.8894	344/8.6013	786/26.6056

Table 1: Results of varying parameter θ in the proposed Douglas-Rashford splitting algorithm 1.

(M, N)	$\beta = 0.0001$ Iter/Time	$\beta = 0.001$ Iter/Time	$\beta = 0.1$ Iter/Time	$\beta = 1$ Iter/Time	$\beta = 10$ Iter/Time
(100,25)	2/0.0013	163/0.0584	162/0.0373	158/0.0313	164/0.0629
(200,100)	2/0.0020	528/0.3301	216/0.1566	199/0.1415	203/0.1254
(300,150)	374/0.8984	554/1.2979	214/0.5333	205/0.4861	219/0.6669
(400,250)	453/1.1677	611/1.7133	235/0.6431	214/0.5807	228/ 0.6119
(600,300)	881/9.2771	623/6.1512	235/2.3229	228/2.2761	247/2.9510
(600,400)	1000/17.0568	718/12.6837	245/4.8480	236/4.2068	259/5.3359

Table 2: Results of varying stepsize β in the proposed Douglas-Rashford splitting algorithm 1.

identified as $\theta = 0.05$, which consistently yields the best results across all tested dimensions. For the stepsize β, the algorithm performs well with smaller values (e.g., $\beta = 0.0001$) when the dimensions M and N are small; however, as M and N

increase, smaller β values result in significant performance degradation, suggesting an underestimation of the stepsize. The optimal performance across all dimensions is observed with $\beta = 1$, which balances convergence efficiency and stability. Regarding scalability, the algorithm shows reasonable robustness under optimal parameter settings but struggles with efficiency when suboptimal values of θ or β are used, particularly as M and N increase. These findings underscore the importance of tuning parameters to achieve maximum algorithmic efficiency and adaptability across varying problem sizes.

4.2 Regularized least squares problem with the logarithmic regularizer

Next, we shall consider the regularized least square (RLS) model with logarithmic regularizer given as follows:

$$\min_{w\in\mathbb{R}^N} \mathcal{J}(w) = \frac{1}{2}\|Aw - b\|^2 + \sum_{i=1}^{N}(\mu \log(|w_i| + \epsilon) - \mu \log \epsilon), \tag{41}$$

where $A \in \mathbb{R}^{m\times N}, b \in \mathbb{R}^m$, $\epsilon > 0$ is a constant and $\mu > 0$ is the regularization parameter. RLS with logarithmic regularizer is very common in machine learning, mainly for preventing overfitting in models, see, e.g. [5, 22, 33]. It is easy to see that the model (41) is a special case of DC programming (1) with $f(w) = \frac{1}{2}\|Aw - b\|^2$, $g(w) = \frac{\mu}{\epsilon}\|w\|_1$ and $h(w) = \sum_{i=1}^{N} \mu\Big(\frac{|w_i|}{\epsilon} - \log(|w_i| + \epsilon) + \log \epsilon\Big)$. The first-order optimization condition of the first subproblem in Algorithm 1 and Algorithm 2 give ([24])

$$A^*(Ay_n - b) + \frac{1}{\beta}(y_n - u_n) = 0,$$

which yields

$$(\beta A^*A + I)y_n = \beta A^*b + u_n,$$

and can be solved effectively by the LU factorization method, the conjugate gradient method and so on [6].

In our experiments, we test the performance of the proposed algorithms for finding the minimizer of (41) and compare the results with other methods in the literature. The matrices $A \in \mathbb{R}^{m\times N}$ and $b \in \mathbb{R}^m$ are generated randomly with i.i.d. standard Gaussian entries, and then normalized so that the columns of A have unit norms. Also the vector b is generated randomly such that b is non-zero vector. The proposed Algorithm 1 and 2 are compared with [37] (DCA) and [11, Algorithm 1] (GDCP). We initialize the algorithms by generating the starting points randomly, $\mu = 0.001, \epsilon = 0.5$, $max\_iter = 1000, \beta = 0.04$, $\kappa_n = \frac{n}{n+10}$, $\theta = 0.9$ for Alg 1, $\alpha_n = \frac{1}{n+1}$ for Alg 2 and $\alpha = 0.09$ for GDCP. We stopped the algorithms when

$$Err = \frac{\|x_n - x_{n-1}\|}{\max\{1, \|x_n\|\}} < 10^{-5}$$

and record the number of iteration and CPU time taken for each algorithm. The results of the experiments are recorded in Table 3 and Figure 3.

(m, N)	Alg 1 Iter/Time	Alg 2 Iter/Time	DCA Iter/Time	GDCP Iter/Time
(100,50)	156/0.0708	212/0.0398	331/0.0785	1000/0.1567
(200,128)	160/0.1762	217/0.1728	343/0.2659	1000/0.8564
(521,304)	168/1.0966	228/1.4425	366/1.8522	760/3.8603
(700,500)	169/6.3734	226/7.8930	365/12.1768	763/26.9791
(1000,700)	171/11.5587	231/19.5223	369/30.5246	760/63.9168
(1500,1000)	174/29.7912	236/47.7586	378/72.9514	759/145.9198

Table 3: Comparison of numerical results for Example 4.2.

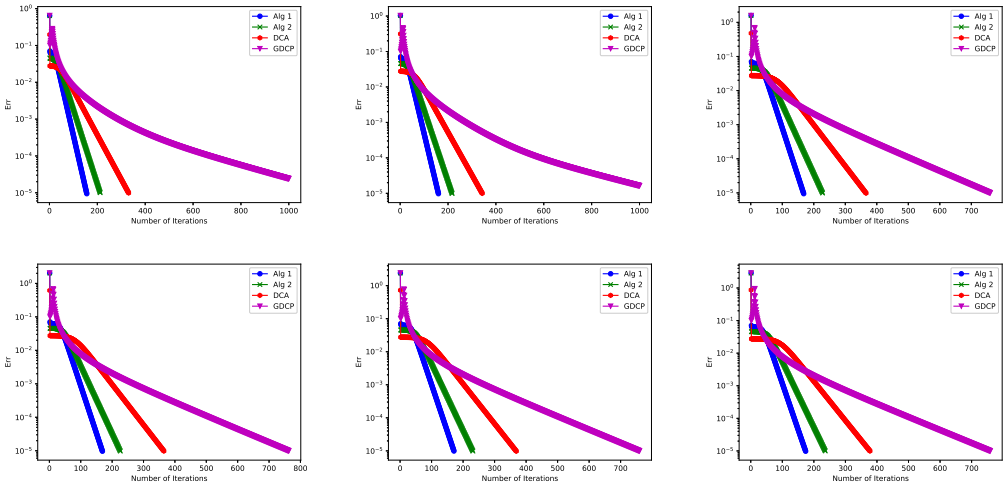

Figure 3: Graphical illustration of values of Err against number of iterations by each method in Example 4.2. Instance 1: (100,50); Instance 2: (250,128), Instance 3: (521,304), Instance 4: (700, 500), Instance 5: (1000,700) and Instance 6: (1500,1000).

The numerical results show that efficiency and accuracy of the proposed algorithms. Both proposed methods consistently have the best performance than the comparing methods in terms of number of iterations and time of execution irrespective of the size of the data size. This is a desirable result which give credence to the importance of the proposed methods.

4.3 Support vector machines (SVM) with L1 regularization

Given a dataset $\{(x_i, y_i)\}_{i=1}^n$ where x_i represents the number of the features and y_i is the corresponding label ($+1$ or -1) of the i-th sample, the primal form of the SVM optimization model can be written as

$$\underset{w,b}{\text{minimize}} \ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i(w^\top x_i + b)\right) + \lambda\|w\|_1 \tag{42}$$

where w is the weight of vector, b is the bias term, λ is the regularization parameter for the L1 penalty, C is the regularization parameter for the hinge loss function.

The first term of the model represents a convex function which deals with sparsity in the model, the second term is the hinge loss function penalizing misclassification and the third term is the L1 regularization term dealing with large weights. The objective function is a good example of a generalized DC programming problem written as

$$\min_{w,b} \; f(w,b) + g(w) - h(w),$$

where $f(w,b) = \|w\|_2^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(w^\top x_i + b))$, $g(w) = \lambda\|w\|_1$ and $h(w) = \frac{1}{2}\|w\|_2^2$.
The SVM was introduced by Vapnik [12, 48] as a kernel-based model used for classification and regression tasks in machine learning. Its exceptional generalization ability, optimal solution, and discriminative power have garnered significant attention from the data mining, pattern recognition, and machine learning communities in recent times. SVM has been used as a powerful tool for addressing real-world binary classification problems. Researches have demonstrated the superiority of SVMs over alternative supervised learning techniques [25, 47, 49]. Due to their robust theoretical foundations and remarkable generalization capabilities, SVMs have risen to prominence as one of the most widely adopted classification methods in recent times. A detailed theoretical explanation and formulation of the SVM can be found in, for instance, [5, 22, 33, 48].

Related works

Considerable research works have been dedicated to exploring the applications of DC programming across various domains including operational research, machine learning, and economics. An intriguing application of DC programming was presented in the 2006 paper of Argiou et al [1] titled "A DC programming algorithm for kernel selection". The author discussed a greedy algorithm aimed at learning kernels from a convex hull comprising basic kernels. While this methodology had been previously introduced, it was confined to a finite set of basic kernels. Additionally, the authors noted that the applicability of their approach was constrained by the non-convex nature of a critical maximization step involved in the learning process. However, they discovered that the optimization problem could be reformulated as a DC program without solving the problem. Another noteworthy paper discussed in this direction is the paper by Thi et al [42]. The authors introduced an innovative application of DC programming within the context of SVM algorithms. Their focus lay in the optimal selection of representative features from data, a pivotal task in enhancing the efficiency and interpretability of machine learning models. The authors equate this problem to minimizing a zero norm function over step-k feature vectors. The zero-norm function counts the number of non-zero elements in the feature vector, essentially quantifying the sparsity of the selected features. This innovative approach opens up new avenues for enhancing the interpretability and performance of machine learning models in various applications. In the paper [43], the authors leveraged DC decomposition techniques to address a challenging optimization problem within the context of machine learning. Specifically, they employed a DC decomposition-based algorithm known as DCA (DC Algorithm) to efficiently find local minima of the objective function. This algorithm iteratively updates the solution by solving convex

subproblems and performing a DC decomposition-based update step, converging towards a local minimum of the objective function. The authors applied this approach to ten datasets, some of which were particularly sparse. In the thesis [34], the author conducted a comprehensive survey focusing on the unified DC programming method for feature selection within the framework of semi-supervised SVM and multiclass SVM. The primary algorithm emphasized in the paper is based on the subgradient optimality principle. The unified DC programming method discussed in the thesis aims to identify the most relevant subset of features from high-dimensional data while optimizing the performance of the SVM classifier, and demonstrating its applicability and efficacy across different problem domains. In the more recent thesis by Ho [52], the author delved into the application of DC programming for learning a kernel tailored to a specific supervised learning task. Notably, what sets this approach apart from others in the literature is its capability to handle an infinite set of basic kernels. Other related results can be found in [44, 45].

Experimental setup

We begin by describing the datasets used in the experiments. We consider the following two datasets which are source from the UCI machine learning database and Kaggle:

- Dataset1: Bank-Note-Authentication: https://archive.ics.uci.edu/dataset/267/banknote+authentication
- Dataset2: Creditcard: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Dataset1 contains 1,372 rows with 5 columns consisting of 4 features and 1 target attribute/class classifying whether a given banknote is authentic or not. The data were extracted from images that were taken from genuine and forged back note specimen and has been used in many research works in machine learning. The target class contains two variables; namely, 0 which represents genuine note and 1 which represents fake note. More so, the data contains a balance ratio of both classes which is 55:45 (genuine:fake) and has no missing value. Hence, there is no need for a data cleaning purpose, which means that, we implemented the data directly in our experiment. Details of the dataset can be found in Table 4.
Dataset2 contains transaction made by credit cards in September 2013 by European cardholders within two days. It contains 234,807 transaction out of which 492 are fraud which indicates a highly imbalanced dataset. Hence, we carried out a feature engineering task to avoid wrong classification of the model due to the imbalance in the dataset. We used the undersampling technique by reducing the number of samples in the majority class to match the number of samples in the minority class. This helps to balance the dataset and makes it less skewed towards the majority class. Then we used the 'resample' function from scikit-learn to randomly selects a subset of samples from the majority class without replacement, ensuring that each sample is selected only once. The number of samples selected is equal to the number of samples in the minority class, ensuring a balance class distribution. This is then used to form a new dataset which is used for our experiment. Details of the original imbalanced dataset and the balance dataset after undersampling can be seen in Figure 4 and 5.

Table 4: Description of dataset1.

	variance	skewness	curtosis	entropy	class
count	1372	1372	1372	1372	1372
mean	0.433735	1.92235	1.39763	-1.19166	0.444606
std	2.84276	5.86905	4.31003	2.10101	0.497103
min	-7.0421	-13.7731	-5.2861	-8.5482	0
25%	-1.773	-1.7082	-1.57498	-2.41345	0
50%	0.49618	2.31965	0.61663	-0.58665	0
75%	2.82147	6.81462	3.17925	0.39481	1
max	6.8248	12.9516	17.9274	2.4495	1

In this experiments, we compare the performance of Algorithm 1 and 2 with the ADMM [40], DCA [37], GDCP [11], DYSA [4] and pDCAe [51]. In the model, we choose the following general parameters for implementing the algorithm: $C = 1$, $\lambda = 0.001$, for Algorithm 1, 2, ADMM, DCA and GDCP, we choose $\theta = 0.01, \beta = 0.001, \kappa_n = 0.3, \alpha_n = \frac{1}{10(n+1)}$ and the declared parameters used in the original papers of the comparing methods. For DYSA, we take $\gamma = 2.3$ and for pDCAe, we take $\beta_t = 0.5$. The maximum allowed iteration is 2000 and we also set the algorithms to stop if $\|x_{k+1} - x_k\| < 10^{-4}$.

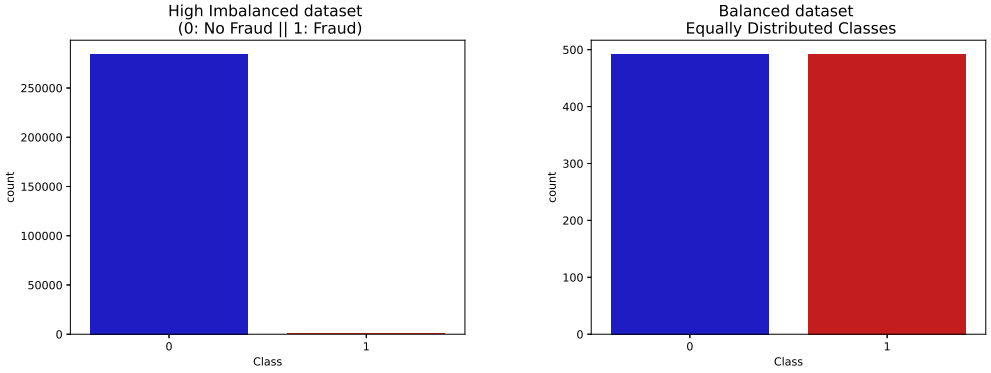

Figure 4: Highly imbalanced dataset2 (Left) and Balanced dataset2 after undersampling (Right)

Experimental results

We divided the training data into four subsets with varying ratios of testing to training data: 10% testing and 90% training, 20% testing and 80% training, 30% testing and 70% training, and 40% testing and 60% training using the `train_test_split` function from `sklearn` in python. For dataset1, we carried out preprocssing technique to ensure that the data is standardized and suitable for training the model. For dataset2, we carried out undersampling feature selection to ensure that the model does not wrongly classify the testing data. We train each of the algorithm using the designated training data preference specified above. Consistent starting points and parameters are applied to ensure fair comparison among algorithms. The performance metrics are recorded for each algorithm to assess their effectiveness. These

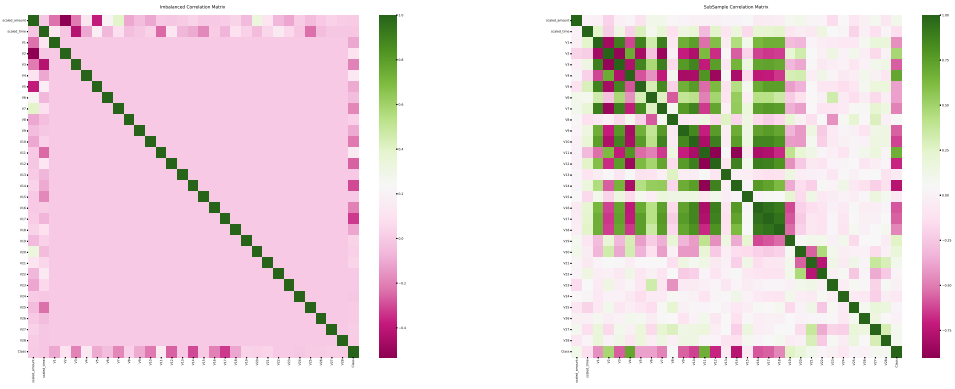

Figure 5: Correlation matrix of features of imbalanced dataset2 (Left) and correlation matrix of features of balanced dataset2 after undersampling (Right).

include accuracy, execution time, precision, MAE, MSE and RMSE scores which are defined below:

(i) Accuracy: This measures the proportion of correctly classified instances out of the total number of instances. It is the ratio of the number of correct predictions to the total number of predictions.

(ii) Precision: This measures the proportion of true positive predictions among all positive predictions made by the model. It is calculated as the ratio of true positive to sum of true positive and false positive. A desirable high precision score indicates that the model makes fewer false predictions.

(iii) Mean absolute error (MAE): This is use to measures the average absolute difference between the predicted values and the actual values, with the formula given by

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where y_i is the actual value and $\hat{y}_i$ is the predicted value.

(iv) Mean square error (MSE): This is the average squared difference between the predicted values and the actual values. It penalizes larger errors more heavily that smaller errors due to squaring and it is sensitive to outliers. It's formula is given by

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

(v) Root mean square error(RMSE): This represents the average magnitude of the errors in the same units as the target variable. It provides a measure of spread of the errors. It's formula is given by

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}.$$

Table 5: Experimental results of algorithms across splits for Dataset1.

Metric	Split	ADMM	Alg1	Alg2	DCA	GDCP	DYSA	pDCAe
Accuracy	10%	0.9855	0.9348	0.9348	0.9348	0.8768	0.9058	0.9058
	20%	0.9818	0.9345	0.9345	0.9345	0.8981	0.9090	0.9091
	30%	0.9836	0.9399	0.9399	0.9399	0.9290	0.9345	0.9345
	40%	0.9878	0.9466	0.9466	0.9466	0.9150	0.9346	0.9345
Precision	10%	0.9841	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	20%	0.9747	0.9902	0.9902	0.9902	0.9592	0.9266	0.9266
	30%	0.9881	0.9874	0.9874	0.9874	0.9863	0.9596	0.9596
	40%	0.9752	0.9768	0.9768	0.9768	0.9808	0.9870	0.9870
Time	10%	0.2941	0.2497	0.2508	0.2826	0.2540	0.2370	0.1853
	20%	0.2889	0.2599	0.2625	0.2858	0.2772	0.2097	0.1516
	30%	0.2695	0.2156	0.2208	0.2635	0.2641	0.2682	0.4832
	40%	0.5294	0.3004	0.3068	0.4760	0.3936	0.4684	0.3681
MAE	10%	0.0290	0.1304	0.1304	0.1304	0.2464	0.1884	0.1883
	20%	0.0364	0.1309	0.1309	0.2464	0.2036	0.1311	0.1311
	30%	0.0328	0.1202	0.1202	0.1202	0.1421	0.1311	0.1311
	40%	0.0583	0.1068	0.1608	0.2036	0.1699	0.1311	0.1311
MSE	10%	0.0580	0.2609	0.2609	0.2609	0.4928	0.3768	0.3768
	20%	0.0727	0.2618	0.2618	0.4928	0.4073	0.2621	0.2621
	30%	0.0656	0.2404	0.2404	0.2404	0.2815	0.2621	0.2621
	40%	0.1165	0.2136	0.2136	0.4073	0.3399	0.2621	0.2621
RMSE	10%	0.2408	0.5108	0.5108	0.5108	0.7020	0.6138	0.6138
	20%	0.2697	0.5117	0.5117	0.7020	0.6382	0.5119	0.5119
	30%	0.2561	0.4903	0.4903	0.4903	0.5331	0.5119	0.5119
	40%	0.3413	0.4621	0.4621	0.6382	0.5829	0.5120	0.5120

The experimental results for Dataset1 and Dataset2 are presented in Table 5 and Table 6. Additionally, we compare the MAE, MSE, and RMSE scores for each algorithm across the split datasets to evaluate their performance in predicting test data values. The average performance results for each method are illustrated in Figure 6. Similarly, the experimental results for Dataset2 are provided in Table 7 through Table 8, as well as in Figure 7.

Discussion of results

The experimental results across Dataset1 highlight the competitive performance of Algorithm 1 and Algorithm 2 when compared to other methods. Both algorithms achieve the same average accuracy of 93.90%, which is on par with DCA and surpasses GDCP, DYSA, and pDCAe. Moreover, in terms of precision, Algorithm 1 and Algorithm 2 stand out with the highest value of 98.86%, outperforming all other methods, including ADMM, which achieves 98.05%. When analyzing computational time, Algorithm 1 demonstrates an advantage, with an average time of 0.2564, which is lower than Algorithm 2 (0.2602) and significantly better than DCA (0.3260) and other competing methods. This efficiency indicates that Algorithm 1 is particularly well-suited for scenarios where time complexity is critical. For error metrics, Algo-

Table 6: Average experimental results across splits for Dataset1.

Metric	ADMM	Alg1	Alg2	DCA	GDCP	DYSA	pDCAe
Average Accuracy	0.9847	0.9390	0.9390	0.9390	0.9047	0.9210	0.9210
Average Precision	0.9805	0.9886	0.9886	0.9886	0.9816	0.9683	0.9683
Average Time	0.3455	0.2564	0.2602	0.3260	0.2972	0.2958	0.2971
Average MAE	0.0391	0.1221	0.1356	0.1752	0.1905	0.1454	0.1454
Average MSE	0.0782	0.2442	0.2442	0.3528	0.3804	0.2908	0.2908
Average RMSE	0.2770	0.4937	0.4937	0.5858	0.6140	0.5374	0.5374

Table 7: Experimental results of algorithms for different dataset splits.

Metric	Split	ADMM	Alg1	Alg2	DCA	GDCP	DYSA	pDCAe
Accuracy	10%	0.9368	0.8316	0.9050	0.9052	0.7473	0.8421	0.8421
	20%	0.9316	0.9211	0.8684	0.8474	0.8632	0.8263	0.8263
	30%	0.9087	0.9018	0.9298	0.8596	0.7895	0.8245	0.8245
	40%	0.9076	0.8179	0.8285	0.8258	0.8073	0.8500	0.8500
Precision	10%	0.9512	0.7454	0.8696	0.8696	0.6727	0.7833	0.7833
	20%	0.9468	0.9271	0.8158	0.7931	0.8142	0.7838	0.7838
	30%	0.8868	0.9116	0.9640	0.8471	0.9298	0.7374	0.7374
	40%	0.9508	0.7963	0.8082	0.8131	0.7873	0.8144	0.8144
Time	10%	5.7984	2.5741	2.2906	4.6751	5.7383	5.0278	5.4613
	20%	5.8183	2.5693	2.3673	4.4144	5.7594	5.6522	5.2570
	30%	5.8389	2.0428	2.3072	5.3861	5.5579	5.8134	5.6380
	40%	5.9516	2.5901	2.5424	4.6523	5.9498	5.2308	5.4268
MAE	10%	0.1263	0.3368	0.1895	0.1895	0.5052	0.3158	0.3158
	20%	0.1368	0.1579	0.2632	0.3053	0.2737	0.3474	0.3474
	30%	0.1824	0.1965	0.1404	0.2807	0.4211	0.3508	0.3509
	40%	0.1847	0.3641	0.3430	0.3482	0.3852	0.3000	0.3000
MSE	10%	0.2526	0.6737	0.3789	0.3789	1.0105	0.6316	0.6316
	20%	0.2737	0.3158	0.5263	0.6105	0.5474	0.6947	0.6947
	30%	0.3649	0.3930	0.2807	0.5614	0.8421	0.7017	0.7018
	40%	0.3694	0.7282	0.6861	0.6964	0.7704	0.6000	0.6000
RMSE	10%	0.5026	0.8206	0.6156	0.6156	1.0052	0.7947	0.7947
	20%	0.5231	0.5620	0.7255	0.7813	0.7398	0.8335	0.8335
	30%	0.6040	0.6269	0.5298	0.7493	0.9177	0.8377	0.8377
	40%	0.6078	0.8533	0.8282	0.8346	0.88775	0.7746	0.7746

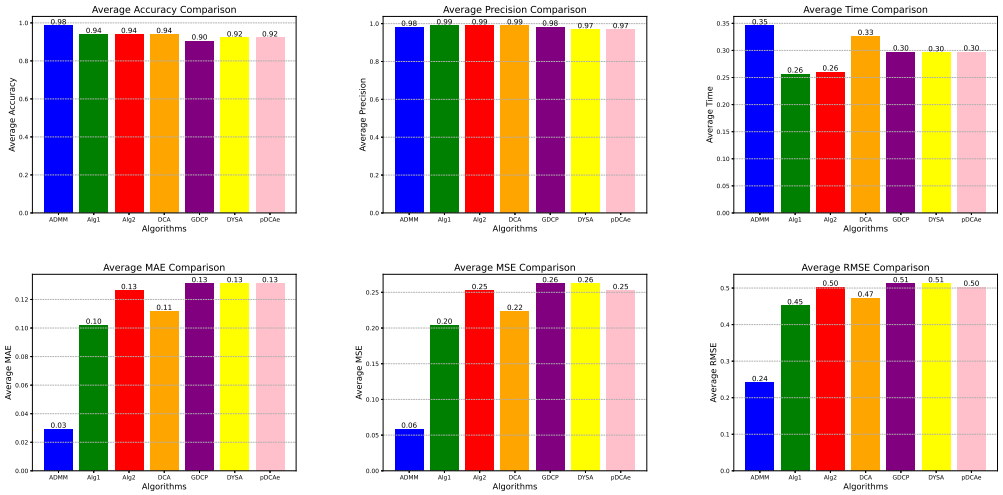

Figure 6: Average Accuracy, Precision, Time, MAE, MSE and RMSE performance for different algorithms for dataset1

.

Table 8: Summary of experimental results across dataset splits.

Metric	ADMM	Alg1	Alg2	DCA	GDCP	DYSA	pDCAe
Average Accuracy	0.9212	0.8681	0.8829	0.8595	0.8018	0.8357	0.8357
Average Precision	0.9339	0.8451	0.8644	0.8307	0.8009	0.7797	0.7797
Average Time	5.8518	2.4441	2.3769	4.5320	5.7514	5.4310	5.4458
Average MAE	0.1576	0.2638	0.2340	0.2814	0.3963	0.3285	0.3285
Average MSE	0.3151	0.5277	0.4688	0.5618	0.7926	0.6570	0.6570
Average RMSE	0.5594	0.7157	0.6747	0.7452	0.8873	0.8101	0.8101

rithm 1 consistently performs better in terms of MAE (0.1221) and MSE (0.2442) compared to Algorithm 2 (0.1356), with both outperforming DCA and GDCP. In terms of RMSE, Algorithm 1 and Algorithm 2 achieve similar results (0.4937), which are superior to DCA (0.5858) and GDCP (0.6140). The experimental results for Dataset2 indicate that Algorithm 2 and Algorithm 1 demonstrate competitive performance across various metrics compared to other methods. In terms of average accuracy, Algorithm 1 achieves 88.29%, which is higher than Algorithm 2 (86.81%) and better than DCA (85.95%), GDCP (80.18%), and DYSA (83.57%). Both algorithms consistently outperform GDCP, DYSA, and pDCAe in this regard. For precision, Algorithm 1 achieves 86.44%, slightly higher than Algorithm 2 (84.51%) and surpassing DCA (83.07%) and GDCP (80.09%). These results highlight the strong predictive capabilities of both algorithms compared to their counterparts. In terms of computational time, Algorithm 1 achieves the best performance with an average time of 2.3769, slightly outperforming Algorithm 2 (2.4441), and significantly faster than ADMM (5.8518), DCA (4.5320), GDCP (5.7514), DYSA (5.4310), and pDCAe (5.4458). This efficiency demonstrates the practical advantage of both algorithms, especially for large-scale problems. Regarding error metrics, Algorithm 1

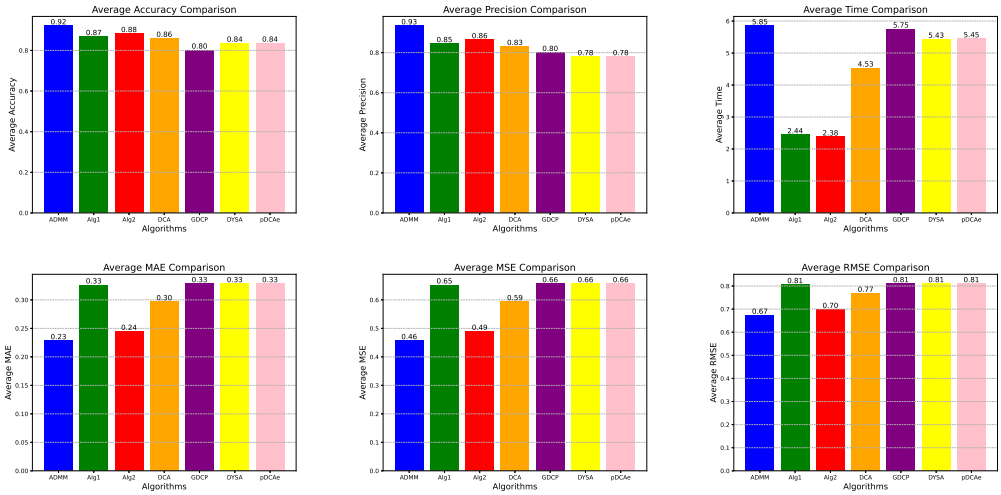

Figure 7: Average Accuracy, Precision, Time, MAE, MSE and RMSE performance for different algorithms for dataset2

.

exhibits better results than Algorithm 2 across all metrics, achieving a lower MAE (0.2340), MSE (0.4688), and RMSE (0.6747). Both algorithms consistently outperform DCA, GDCP, DYSA, and pDCAe in error metrics, further emphasizing their robustness and accuracy.

Overall, Algorithm 1 exhibits a balance of high accuracy, strong precision, computational efficiency, and lower error metrics, making it the most robust and reliable method for the tested scenarios. Algorithm 2, while slightly trailing Algorithm 1, also demonstrates strong performance, particularly in terms of time efficiency and precision. These findings confirm the effectiveness and practicality of both algorithms in addressing optimization problems compared to other state-of-the-art methods.

5 Final Remarks

We have proved in this paper that new two variants of the unified Douglas-Rachford splitting algorithm converge weakly to a critical point of a generalized DC programming in Hilbert spaces. We recovered the unified Douglas-Rachford splitting algorithm from our proposed algorithms. Numerical illustrations from machine learning showed that our algorithms have practical implementations, are efficient, and outperform some other related algorithms for solving this class of generalized DC programming. As part of the future project, we study the accelerated versions of our proposed algorithms.

References

[1] Argyriou, A., Hauser, R., Micchelli, C. and M. Pontil A.: DC-programming algorithm for kernel selection. Proceedings of the 23rd International Conference

on Machine Learning (ICML '06), 2006.

[2] Banert, S., Rudzusika, J., Oktem, O., Adler, J.: Accelerated forward-backward optimization using deep learning, https://arxiv.org/abs/2105.05210.

[3] Bauschke, H.H., Combettes, P.L.: Convex Analysis and Monotone Operator Theory in Hilbert Spaces. 2nd ed. Springer, New York, 2017.

[4] Bian, F., Zhang, X.: A three-operator splitting algorithm for nonconvex sparsity regularization, Siam J. Sci. Comput. 43, (2021).

[5] Bishop CM.: Pattern Recognition and Machine Learning. New York : Springer, 2006

[6] Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. 3(1), 1-122 (2011).

[7] Ceng, L.C., Ansari, Q.H., Schaible, S.: Hybrid extragradient-like methods for generalized mixed equilibrium problems, systems of generalized equilibrium problems and optimization problems. J. Glob. Optim. 53, 69-96 (2012).

[8] Candès, E.J. and Wakin, M.B.: An Introduction To Compressive Sampling, IEEE Signal Processing Magazine, 25, 21-30 (208).

[9] Ceng, L.C., Coroian, I., Qin, X., Yao, J.C.: A general viscosity implicit iterative algorithm for split variational inclusions with hierarchical variational inequality constraints. Fixed Point Theory 20, 469-482 (2019).

[10] Ceng, L.C., Li, X., Qin, X.: Parallel proximal point methods for systems of vector optimization problems on Hadamard manifolds without convexity. Optimization 69, 357-383 (2020).

[11] Chuang, C.S., He, H., Zhang, Z.: A unified Douglas-Rachford algorithm for generalized DC programming. J. Glob. Optim. 82, 331-349 (2022).

[12] Cortes C and Vapnik, VN.: Support vector networks, Machine Learning, 20, 273–297, (1995).

[13] Cui, F., Tang, Y., Yang, Y.: An inertial three-operator splitting algorithm with applications to image inpainting. Appl. Set-Valued Anal. Optim. 1, 113-134 (2019).

[14] Cui, H.H., Ceng, L.C.: Convergence of over-relaxed contraction-proximal point algorithm in Hilbert spaces. Optimization 66, 793-809 (2017).

[15] Davis, D., Yin, W.: A three-operator splitting scheme and its optimization applications, Set-Valued Variat. Anal. 25, 829-858 (2017).

[16] Ding, M., Song, X. Yu, B.: An Inexact Proximal DC Algorithm with Sieving Strategy for Rank Constrained Least Squares Semidefinite Programming. J. Sci. Comput. 91, 75 (2022).

[17] Dong, Q.L., Huang, J.Z., Li, X.H., Cho, Y.J., Rassias, Th.M.: MiKM: multi-step inertial Krasnosel'skii–Mann algorithm and its applications. J. Glob. Optim. 73, 801-824 (2019).

[18] Douglas, J., Rachford, H.H.: On the numerical solution of heat conduction problems in two or three space variables. Trans. Am. Math. Soc. 82, 421-439 (1956).

[19] Gaudioso, M., Gorgone, E., Hiriart-Urruty, J.B.: Feature selection in SVM via polyhedral k-norm. Optim. Lett. 14, 19-36 (2020).

[20] Gotoh, JY, Takeda, A., Tono, K.: DC formulations and algorithms for sparse optimization problems. Math. Program. 169, 141-176 (2018).

[21] Guan, J.L., Ceng, L.C., Hu, B.: Strong convergence theorem for split monotone variational inclusion with constraints of variational inequalities and fixed point problems. J. Inequal. Appl. 311, (2018).

[22] Hastie, T. and Tibshirani, R. and Friedman, J.H.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed. Springer Series in Statistics, 2001.

[23] He, L., Cui, Y.L., Ceng L.C., Zhao, T.Y., Wang, D.Q., Hu, H.Y.: Strong convergence for monotone bilevel equilibria with constraints of variational inequalities and fixed points using subgradient extragradient implicit rule. J. Inequal. Appl. 146, (2021).

[24] Hu, Z., Huang, Dong, Q.L.: A three-operator splitting algorithm with deviations for generalized DC programming. Appl. Numer. Math. 191, 62-74 (2023).

[25] Huang, G.B., Mao, K.Z., Siew, C.K., Huang D.S.: Fast Modular Network Implementation for Support Vector Machines. IEEE Trans Neural Netw., 16(6), Issue 6, p1651 (2005).

[26] Latif, A., Ceng, L.C., Al-Mezel, S.A.: Some iterative methods for convergence with hierarchical optimization and variational inclusions. J. Nonlinear Convex Anal. 17, 735-755 (2016).

[27] Le Thi, H.A., Pham Dinh, T., Le, H.M., Vo, X.T.: DC approximation approaches for sparse optimization. European J. Oper. Res. 244(1), 26-46 (2015).

[28] Li, G., Yang, L., Wu, Z., Wu, G.: D.C. programming for sparse proximal support vector machines. Inform. Sciences 547, 187-201 (2021).

[29] Liu, T., Pong, T.K., Takeda, A.: A refined convergence analysis of pDCAe with applications to simultaneous sparse recovery and outlier detection. Comput. Optim. Appl. 73, 69-100 (2019).

[30] Lou, Y., Yan, M.: Fast L1-L2 Minimization via a Proximal Operator. SIAM J. Sci. Comput. 74, 767-785 (2018).

[31] Lou, Y., Yin, P., He, Q., Xin, J.: Minimization of ℓ_{1-2} for compressed sensing. SIAM J. Sci. Comput. 37, A536-A563 (2015).

[32] Lu, Z., Zhou, Z.: Nonmonotone enhanced proximal DC algorithms for a class of structured nonsmooth DC programming. SIAM J. Optim. 29(4), 2725-2752 (2019).

[33] Murphy, KP.: Machine Learning: A Probabilistic Perspective. MIT Press, 2012.

[34] Nguyen, M.C.: DC programming and DCA for some classes of problems in machine learning and data mining. Other [cs.OH]. Université de Lorraine, 2014. English. NNT: 2014LORR0080.tel-01750803.

[35] Nouiehed, M., Pang, J.S., Razaviyayn, M.: On the pervasiveness of difference-convexity in optimization and statistics. Math. Program. 174, 195-222 (2019).

[36] Opial. Z.: Weak convergence of the sequence of successive approximations for nonexpansive mappings. Bull. Am. Math. Soc. 73, 591-597 (1967).

[37] Pham D.T., Souad, E.B.: Algorithms for solving a class of nonconvex optimization problems. Methods of subgradients. In: Hiriart-Urruty, J.B. (ed.) Fermat Days 85: Mathematics for Optimization. NorthHolland Mathematics Studies, vol. 129, pp. 249–271. North-Holland, Amsterdam (1986) North-Holland Mathematics Studies, vol. 129, pp. 249–271. North-Holland, Amsterdam (1986)

[38] Raguet, H., Fadili, J., Peyre, G.: Generalized forward-backward splitting. SIAM J. Imaging Sci. 6(3), 1199-1226 (2011).

[39] Sadeghi, H., Banert, S., Giselsson, P.: Forward-backward splitting with deviations for monotone inclusions. https://arxiv.org/abs/2112.00776.

[40] Sun, T., Yin, P., Cheng, L., Jiang, H.: Alternating direction method of multipliers with difference of convex functions. Adv. Comput. Math. 44, 723–744 (2018).

[41] Tao, P.D., Souad, E.B.: Algorithms for solving a class of nonconvex optimization problems: methods of subgradients. Math. Optim. 129, 249-271 (1986).

[42] Thi, H.A.L., Le, H.M., Nguyen, V.V., Dinh, T.P.: A DC programming approach for feature selection in support vector machines learning. Adv Data Anal Classif 2, 259–278 (2008).

[43] Thi H.A.L., Nguyen, M.C., Dinh, T.P.: A DC programming approach for finding communities in networks, Neural Comput. 26(12), 2827-54 (2014).

[44] Thi, H.A.L., Le, H.M., Phan, D.N., Tran, B.: Novel DCA based algorithms for a special class of nonconvex problems with application in machine learning, Appl. Mathem. Comp., 409 (15), 125904 (2021).

[45] Yao, C., Jiang, X.: A globally convergent difference-of-convex algorithmic framework and application to log-determinant optimization problems. arXiv:2306.02001.

[46] Tu, K., Zhang, H., Gao, H. et al. A hybrid Bregman alternating direction method of multipliers for the linearly constrained difference-of-convex problems. J. Glob. Optim. 76, 665-693 (2020).

[47] Ukey, K.P., Alvi, A.S.: Text classification using support vector machine. J. Softw Eng. Appl., 5(12), 55–58 (2012).

[48] Vapnik, VN.: The Nature of Statistical Learning Theory. Springer-Verlag, New York, 1995.

[49] Wang, J., Zhao, C., Huo, Z., Qiao, Y., Sima, H.: High quality proposal feature generation for crowded pedestrian detection. Pattern Recognition, 128(5), 108605 (2022).

[50] Wen, J., Fang, X., Cui, J., Fei, L., Yan, K., Chen, Y., Xu, Y.: Robust Sparse Linear Discriminant Analysis, IEEE Trans. Circ. Syst Video Tech, 29, 390-403 (2019).

[51] Wen, B., Chen, X.J., and Pong, T.K.: A proximal differennce-of-convex algorithm with extrapolation. Comput. Optim. Appl., 69, 297-324, (2018).

[52] Ho, V.T.: Advanced machine learning techniques based on DC programming and DCA. Machine Learning [cs.LG]. Université de Lorraine, 2017. English. NNT : 2017LORR0289.tel-01810274

[53] Wen, B., Chen, X., Pong, T.K.: A proximal difference-of-convex algorithm with extrapolation. Comput. Optim. Appl. 69, 297-324 (2018).

[54] Wu, P., Cheng, J.: Nonconvex regularized gradient projection sparse reconstruction for massive MIMO channel estimation. IEEE T. Commun. 69(11), 7722-7735 (2021).