Bregmanized Domain Decomposition for Image Restoration

Andreas Langer* Stanley Osher† Carola-Bibiane Schönlieb‡

January 31, 2012

Abstract

Computational problems of large-scale data are gaining attention recently due to better hardware and hence, higher dimensionality of images and data sets acquired in applications. In the last couple of years non-smooth minimization problems such as total variation minimization became increasingly important for the solution of these tasks. While being favourable due to the improved enhancement of images compared to smooth imaging approaches, non-smooth minimization problems typically scale badly with the dimension of the data. Hence, for large imaging problems solved by total variation minimization domain decomposition algorithms have been proposed, aiming to split one large problem into $N > 1$ smaller problems which can be solved on parallel CPUs. The N subproblems constitute constrained minimization problems, where the constraint enforces the support of the minimizer to be the respective subdomain.

In this paper we discuss a fast computational algorithm to solve domain decomposition for total variation minimization. In particular, we accelerate the computation of the subproblems by nested Bregman iterations. We propose a *Bregmanized Operator Splitting - Split Bregman* (BOS-SB) algorithm, which enforces the restriction onto the respective subdomain by a Bregman iteration that is subsequently solved by a Split Bregman strategy. The computational performance of this new approach is discussed for its application to image inpainting and image deblurring. It turns out that the proposed new solution technique is up to three times faster than the iterative algorithm currently used in domain decomposition methods for total variation minimization.

1 Introduction

Let $\Omega \subset \mathbb{R}^2$ be a bounded open set with Lipschitz boundary. We are interested in the minimization in $BV(\Omega)$ of the functional

$$\mathcal{J}(u) := \|Tu - g\|^2_{L_2(\Omega)} + 2\alpha \left|Du\right|(\Omega), \tag{1}$$

where $T : L_2(\Omega) \to L_2(\Omega)$ is a bounded linear operator, $g \in L_2(\Omega)$ is a datum, and $\alpha > 0$ is a fixed *regularization parameter* [13]. We recall, that for $u \in L_1(\Omega)$

$$V(u,\Omega) := \sup\left\{\int_\Omega u \operatorname{div}\varphi \, dx : \varphi \in \left[C_c^1(\Omega)\right]^2, \|\varphi\|_\infty \leq 1\right\}$$

*Johann Radon Institute for Computational and Applied Mathematics (RICAM), Austrian Academy of Sciences, Altenbergerstrasse 69, A-4040, Linz, Austria Email: andreas.langer@oeaw.ac.at

†Department of Mathematics, UCLA, 520 Portola Plaza, Los Angeles, California 90095, Email: sjo@math.ucla.edu

‡Department of Applied Mathematics and Theoretical Physics (DAMTP), University of Cambridge, Wilberforce Road, Cambridge CB3 0WA, United Kingdom Email: C.B.Schoenlieb@damtp.cam.ac.uk

is the variation of u. Further, $u \in BV(\Omega)$, the space of bounded variation functions [1, 14], if and only if $V(u,\Omega) < \infty$. In this case, we denote $|Du|(\Omega) = V(u,\Omega)$ the total variation of u. If $u \in W^{1,1}(\Omega)$ (the Sobolev space of L_1-functions with L_1-distributional derivatives), then $|Du|(\Omega) = \int_\Omega |\nabla u| \, d\, x$. In order to guarantee the existence of minimizers for (1) we assume that:

(C) $\mathcal{J}$ is coercive in $L_2(\Omega)$, i.e., there exists a constant $C > 0$ such that $\{u \in L_2(\Omega) : \mathcal{J}(u) \leq C\}$ is bounded in $L_2(\Omega)$.

It is well known that if $1 \notin \ker(T)$ then condition (C) is satisfied, see [30, Proposition 3.1].

In image restoration the minimization of the total variation (1), first proposed by Rudin, Osher and Fatemi [27], plays a fundamental role as a regularization technique, since it preserves edges and discontinuities in images. Since this pioneering work several numerical strategies to perform efficiently total variation minimization have been suggested in the literature, see for example [2, 4, 5, 6, 7, 8, 10, 12, 19, 25, 26, 30].

In this paper we are concerned with the numerical minimization of (1) for large scale imaging problems. Due to the continuous improvement of hardware, the dimensionality of images and measurements in general is increasing, resulting into large-scale data sets that want to be processed. A typical choice for image enhancement, e.g., image restoration, image denoising and image deblurring, is total variation minimization (1). While existing state-of-the-art numerical algorithms for the solution of (1) - as listed above - perform very efficiently for small- and medium-scale problems, none of them is able to address in real-time extremely large problems. In these situations subspace correction, domain decomposition and coordinate descent methods are fundamental allowing us to split the computational workload and solve a sequence of smaller problems rather than one large problem. Recently such methods have been successfully introduced for ℓ_1-norm and total variation minimization in [15, 16, 17, 28]. In particular, in [17] and [16] decomposition methods for the minimization of (1) have been proposed, splitting the spatial domain into non-overlapping and overlapping subdomains respectively. Note, that to the best of our knowledge, the latter two contributions are the first ones to propose a convergent domain decomposition strategy for total variation minimization. In both approaches the domain decomposition strategy amounts to minimize a convex functional under some linear constraints on each subdomain iteratively. These constraints are needed to ensure the correct treatment of the solution on the interfaces of the domain decomposition patches, i.e., to preserve crossing discontinuities. In particular, on each subdomain a constrained optimization problem of the general type

$$\min_{u \in H} \{F(u) := \|u - z\|^2_{L_2(\Omega)} + 2\alpha |Du|(\Omega) \text{ subject to } Au = f\} \tag{2}$$

has to be solved, where z and f are functions given on a Hilbert space H and A is a linear operator in H. For the overlapping domain decomposition method the linear constraint is simply a trace condition, while for the non-overlapping algorithm the constraint is the orthogonal projection onto a subspace. In [16, 17] these subminimization problems were solved by the *Iterative Oblique Thresholding*, which is based on an iterative proximity map algorithm and the computation of a Lagrange multiplier by a fixed point iteration, see [17, Section 4.2].

1.1 Our Approach

In this paper we are concerned with increasing the performance of the non-overlapping domain decomposition algorithms proposed in [17], by using a more efficient technique to solve the subminimization problems (2).

There already exist several numerical methods which solve (2) efficiently, such as the *Augmented Lagrange Method* [21] and its variations known under the name of Bregman iterations [26, 31], because of their relation to the Bregman distance. In this paper, instead of using the Iterative Oblique Thresholding technique proposed in [16, 17], we suggest to use the recently introduced *Bregmanized Operator Splitting* technique [32] combined with the *Split Bregman method* [19] for the solution of (2). This approach avoids the computation of a costly fixed point iteration in each domain decomposition step and consequently speeds up the overall computational time of the domain decomposition algorithms, cf. the numerical examples in Section 4.

Organization of the paper The rest of the paper is organized as follows. In Section 2 we propose a new algorithm for solving constrained minimization problems occurring in the subdomains of a domain decomposition. The non-overlapping domain decomposition approach is studied in Section 3, where also the integration of the new algorithm for such a approach is discussed. In Section 4 we describe the numerical implementation which we used for our numerical examples in Section 5.

2 Bregman Algorithms

Let us start this section with introducing some notations, which will be useful in the sequel. For a convex functional $F : H \to \bar{\mathbb{R}}$, we define the *subdifferential* of F at $v \in H$, as the set valued function

$$\partial F(v) := \begin{cases} \emptyset & \text{if } F(v) = \infty \\ \{v^* \in H : \langle v^*, u - v\rangle + F(v) \leq F(u) \quad \forall u \in H\} & \text{otherwise.} \end{cases}$$

It is clear from this definition that $0 \in \partial F(v)$ if and only if v is a minimizer of F. The *Bregman distance*, associated with a convex functional $F : H \to \bar{\mathbb{R}}$, of the vectors $u, v \in \mathrm{Dom}(F)$ is defined by

$$D_F^p(u, v) := F(u) - F(v) - \langle p, u - v\rangle,$$

for $p \in \partial F(v)$. Note that the Bregman distance is not a distance in the usual sense, since it is in general not symmetric and also the triangle inequality does not hold. However it satisfies $D_F^p(u, v) \geq 0$ and $D_F^p(u, v) = 0$ if $u = v$ [3].

In [26] the authors proposed the so-called Bregman Iteration to solve constrained optimization problems of the type (2):

Algorithm 1. *Bregman Iteration: Let $\lambda > 0$ and $u^{(0)} = 0$ then for $k = 0, 1, \ldots$ do*

$$\begin{aligned} p^{(k)} &\in \partial F(u^{(k)}) \\ u^{(k+1)} &= \arg\min_{u \in H} D_F^{p^{(k)}}(u, u^{(k)}) + \lambda \|Au - f\|^2_{L_2(\Omega)} \end{aligned} \tag{3}$$

In [26] the weak convergence of this algorithm to a solution of (2) is ensured and it is shown that the sequence of residuals $(\|Au^{(k)}-f\|)_k$ is monotonically decreasing to zero. Since the Bregman Iteration is equivalent to an augmented Lagrangian method its convergence is guaranteed by the results in [18]. Moreover, in [31] it has been shown that the Bregman Iteration is equivalent to the following simplified iterative scheme:

Algorithm 2. *Simplified Bregman Iteration: Let $\lambda > 0$. Initialize $u^{(0)} = 0$ and $f^{(0)} = f$ then for $k = 0, 1, \dots$ do*

$$\begin{aligned} u^{(k+1)} &= \arg\min_{u\in H} F(u) + \lambda\|Au - f^{(k)}\|^2_{L_2(\Omega)} \\ f^{(k+1)} &= f^{(k)} - Au^{(k+1)} + f. \end{aligned} \tag{4}$$

The direct computation of the update $u^{(k+1)}$ in (3) and (4) is sometimes not efficiently and exactly solvable, in particular if the constraint is ill-posed. In order to overcome this drawback we may suggest to solve the minimization problem in (4) via a forward-backward operator splitting, see [8] for more details. In particular, we are interested in the Bregmanized Operator Splitting algorithm [32], which is based on one forward-backward operator splitting iteration and a suitable update of the Lagrange multiplier:

Algorithm 3. *Bregmanized Operator Splitting (BOS): Let $\lambda, \delta > 0$. Initialize $u^{(0)} = 0$ and $f^{(0)} = f$ then for $k = 0, 1, \dots$ do*

$$\begin{aligned} u^{(k+1)} &= \arg\min_{u\in H} F(u) + \frac{\lambda}{\delta}\|u - (u^{(k)} - \delta A^*(Au^{(k)} - f^{(k)}))\|^2_{L_2(\Omega)} \\ f^{(k+1)} &= f^{(k)} - Au^{(k+1)} + f. \end{aligned} \tag{5}$$

This algorithm is ensured to converge to a minimal solution of (2) if $0 < \delta < \frac{1}{\|A^*A\|}$. Moreover, it is very stable in practice, and is usually easy to implement.

We note that the minimization problem in (5) is equivalent to the famous ROF-problem [27], i.e.,

$$\begin{aligned} &\arg\min_{u\in H}\|u - z\|^2_{L_2(\Omega)} + 2\alpha|Du|(\Omega) + \frac{\lambda}{\delta}\|u - (u^{(k)} - \delta A^*(Au^{(k)} - f^{(k)}))\|^2_{L_2(\Omega)} \\ &\qquad = \arg\min_{u\in H}\left\|u - \frac{1}{1+\frac{\lambda}{\delta}}\left(z + \frac{\lambda}{\delta}\left(u^{(k)} - \delta A^*(Au^{(k)} - f^{(k)})\right)\right)\right\|^2_{L_2(\Omega)} + 2\frac{\alpha}{1+\frac{\lambda}{\delta}}|Du|(\Omega). \end{aligned} \tag{6}$$

Hence there exist several numerical methods which solve this problem efficiently, see for example [4, 5, 11, 19, 20]. Among the fastest is the Split Bregman Method [19], whose main idea is to consider instead of (6) the following equivalent constrained problem

$$\arg\min_{u,d}\|u - z\|^2_{L_2(\Omega)} + 2\alpha|d|(\Omega) + \frac{\lambda}{\delta}\|u - (u^{(k)} - \delta A^*(Au^{(k)} - f^{(k)}))\|^2_{L_2(\Omega)} \quad \text{s.t.} \quad d = Du.$$

Solving this constrained minimization problem by the simplified Bregman Iteration we get

the Split Bregman Method:

$$\begin{aligned}(u^{(\ell+1)}, d^{(\ell+1)}) = \operatorname{argmin}_{u,d} \|u - z\|^2_{L_2(\Omega)} + 2\alpha|d|(\Omega) + \frac{\lambda}{\delta}\|u - (u^{(k)} - \delta A^*(Au^{(k)} - f^{(k)}))\|^2_{L_2(\Omega)} \\ + \mu\|d - Du - b^{(\ell)}\|^2_{L_2(\Omega)} \\ b^{(\ell+1)} = b^{(\ell)} + (Du^{(\ell+1)} - d^{(\ell+1)}),\end{aligned} \tag{7}$$

where $\mu > 0$. We propose to combine the Bregmanized Operator Splitting with the Split Bregman Iteration to solve (2), which results in an algorithm using two nested iterations:

Algorithm 4. *Bregmanized Operator Splitting - Split Bregman (BOS-SB): Let $\lambda, \delta, \mu > 0$ be regularization parameters. Initialize $u^{(0,L_0)} = 0$ and $f^{(0)} = f$ then for $k = 0, 1, \ldots$ do*

$$\begin{cases} u^{(k+1,0)} = u^{(k,L_k)}, d^{(k+1,0)} = b^{(k+1,0)} = 0 \\ \text{for } \ell = 0, \ldots, L_{k+1} \text{ do} \\ \quad \begin{cases} u^{(k+1,\ell+1)} = \arg\min_{u\in H} \|u - z\|^2_{L_2(\Omega)} + \frac{\lambda}{\delta}\|u - (u^{(k,L_k)} - \delta A^*(Au^{(k,L_k)} - f^{(k)}))\|^2_{L_2(\Omega)} \\ \qquad\qquad +\mu\|d^{(k+1,\ell)} - Du - b^{(k+1,\ell)}\|^2_{L_2(\Omega)} \\ d^{(k+1,\ell+1)} = \arg\min_d 2\alpha|d|(\Omega) + \mu\|d - Du^{(k+1,\ell+1)} - b^{(k+1,\ell)}\|^2_{L_2(\Omega)} \\ b^{(k+1,\ell+1)} = b^{(k+1,\ell)} + Du^{(k+1,\ell+1)} - d^{(k+1,\ell+1)} \end{cases} \\ f^{(k+1)} = f^{(k)} - Au^{(k+1,L_{k+1})} + f. \end{cases} \tag{8}$$

The number of inner iteration L_k is chosen such that $\|u^{(k,L_k)} - u^{(k,L_k-1)}\| \leq tol$.

3 Non-overlapping domain decomposition

In this section we discuss the minimization of functional (1) by using the non-overlapping domain decomposition approach suggested in [17] and propose to solve the corresponding subminimization problems with the help of Algorithm 4.

We decompose the spatial domain Ω into two disjoint subdomains Ω_1 and Ω_2 such that $\Omega = \Omega_1 \cup \Omega_2$ and $\Omega_1 = \Omega \setminus \Omega_2$. Note that in the discussion which follows we consider a splitting into two subdomains only. However, as also illustrated with our numerical examples in Section 4, everything works for multiple subdomains as well. Associated to this splitting we define $V_i = \{u \in L_2(\Omega) : \operatorname{supp}(u) \subset \Omega_i\}$ and orthogonal projections $\pi_{V_i} : L_2(\Omega) \to V_i$ for $i = 1, 2$. Since $L_2(\Omega) = V_1 \oplus V_2$ is a direct sum and $\pi_{V_i}(u) = u1_{\Omega_i}$, every $u \in L_2(\Omega)$ can be uniquely represented as $u = \pi_{V_1}(u) + \pi_{V_2}(u)$. In the following we denote $u_i = \pi_{V_i}(u)$, for $i = 1, 2$. With this splitting in [17] the following alternating algorithm is proposed to minimize $\mathcal{J}$: pick an initial $V_1 \oplus V_2 \ni u_1^{(0)} + u_2^{(0)} := u^{(0)} \in BV(\Omega)$, for example $u^{(0)} = 0$, and iterate

$$\begin{cases} u_1^{(n+1)} \approx \operatorname{argmin}_{v_1\in V_1} \mathcal{J}(v_1 + u_2^{(n)}) \\ u_2^{(n+1)} \approx \operatorname{argmin}_{v_2\in V_2} \mathcal{J}(u_1^{(n+1)} + v_2) \\ u^{(n+1)} := u_1^{(n+1)} + u_2^{(n+1)}. \end{cases} \tag{9}$$

Here we use "$\approx$" (the approximation symbol), since in practice we never perform the exact minimization.

3.1 Solution of the subspace minimization problems

In [17] an implementation of the individual subproblems of (9) is suggested by introducing so-called *surrogate functionals* of $\mathcal{J}$ [9, 10] on $V_1 \oplus V_2$ for $u_1 \in V_1$, $u_2 \in V_2$, and $a \in V_i$ by

$$\begin{aligned}\mathcal{J}_i(u_1, u_2; a) &:= \mathcal{J}(u_1 + u_2) + \|u_i - a\|^2_{L_2(\Omega)} - \|T(u_i - a)\|^2_{L_2(\Omega)} \\ &= \|u_i - (a + \pi_{V_i} T^*(g - Tu_{\hat{i}} - Ta))\|^2_{L_2(\Omega)} + 2\alpha \left|D(u_1 + u_2)\right|(\Omega) + \Phi(a, g, u_{\hat{i}}),\end{aligned} \tag{10}$$

for $i = 1, 2$ and $\hat{i} \in \{1, 2\} \setminus \{i\}$, where Φ is a function of $a, g, u_{\hat{i}}$ only. Assuming that $\|T\| < 1$, the subminimization iterations

$$u_i^{(m+1)} = \arg\min_{u_i \in V_i} \mathcal{J}_i(u_1, u_2; u_i^{(m)}) \quad m \geq 0 \tag{11}$$

converge to a minimizer of the corresponding subproblems of (9), i.e.,

$$\arg\min_{u_i \in V_i} \mathcal{J}(u_1 + u_2)$$

for $i = 1, 2$ [10]. We remark that the assumption $\|T\| < 1$ is not a restriction at all, since when the norm is exceeding 1, we just rescale the problem easily by multiplying the functional $\mathcal{J}$ by a positive constant $\gamma < \frac{1}{\|T\|^2}$, and we minimize the resulting functional

$$\mathcal{J}_\gamma(u) = \|\sqrt{\gamma} Tu - \sqrt{\gamma} g\|^2_{L_2(\Omega)} + 2\gamma\alpha |D(u)|(\Omega),$$

which has the same minimizers as $\mathcal{J}$.

Let us further decompose $\Omega_2 = \hat{\Omega}_2 \cup (\Omega_2 \setminus \hat{\Omega}_2)$ with $\partial\hat{\Omega}_2 \cap \partial\Omega_1 = \partial\Omega_2 \cap \partial\Omega_1$, where $\hat{\Omega}_2 \subset \Omega_2$ is a neighborhood stripe around the interface $\partial\Omega_2 \cap \partial\Omega_1$. Analogously we split $\Omega_1 = \hat{\Omega}_1 \cup (\Omega_1 \setminus \hat{\Omega}_1)$ with $\partial\hat{\Omega}_1 \cap \partial\Omega_2 = \partial\Omega_1 \cap \partial\Omega_2$. Associated to these decompositions we define $\hat{V}_i = \{u \in L_2(\Omega) : \mathrm{supp}(u) \subset \hat{\Omega}_i\}$. By the splitting of the total variation

$$\begin{aligned}|D(u_1 + u_2)|(\Omega) =& |D(u_1|_{\Omega_1 \cup \hat{\Omega}_2} + u_2|_{\Omega_1 \cup \hat{\Omega}_2})|(\Omega_1 \cup \hat{\Omega}_2) + |Du_2|_{\Omega_2 \setminus \hat{\Omega}_2}|(\Omega_2 \setminus \hat{\Omega}_2) \\ &+ \int_{\partial\hat{\Omega}_2 \cap \partial(\Omega_2 \setminus \hat{\Omega}_2)} |u_2^+ - u_2^-| d\mathcal{H}_{d-1}(x),\end{aligned} \tag{12}$$

where $\mathcal{H}_d$ is the Hausdorff measure of dimension d and $u_{|_{\Omega_1 \cup \hat{\Omega}_2}}$ is the restriction of u to $\Omega_1 \cup \hat{\Omega}_2$, we can restrict the minimization in (11) to the domain $\Omega_1 \cup \hat{\Omega}_2$ and $\Omega_2 \cup \hat{\Omega}_1$ respectively, i.e.,

$$\begin{aligned}u_1^{(m+1)} = \arg\min_{u_1 \in V_1} &\|u_1 - (u_1^{(m)} + \pi_{V_1} T^*(g - Tu_2 - Tu_1^{(m)}))\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ &+ 2\alpha \left|D(u_1|_{\Omega_1 \cup \hat{\Omega}_2} + u_2|_{\Omega_1 \cup \hat{\Omega}_2})\right|(\Omega_1 \cup \hat{\Omega}_2), \quad m \geq 0, \\ u_2^{(m+1)} = \arg\min_{u_2 \in V_2} &\|u_2 - (u_2^{(m)} + \pi_{V_2} T^*(g - Tu_1 - Tu_2^{(m)}))\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ &+ 2\alpha \left|D(u_1|_{\Omega_2 \cup \hat{\Omega}_1} + u_2|_{\Omega_2 \cup \hat{\Omega}_1})\right|(\Omega_2 \cup \hat{\Omega}_1), \quad m \geq 0.\end{aligned}$$

Eventually, these subminimization problems can be rewritten as problems on $V_i \oplus \hat{V}_{\hat{i}}$, $i \in \{1, 2\}$ and read

$$\arg\min_{u \in V_1 \oplus \hat{V}_2} \|u - z_1\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} + 2\alpha \left|D(u_{|_{\Omega_1 \cup \hat{\Omega}_2}} + u_2|_{\Omega_1 \cup \hat{\Omega}_2})\right|(\Omega_1 \cup \hat{\Omega}_2) \quad \text{s.t.} \quad \pi_{\hat{V}_2} u = 0, \tag{13}$$

$$\arg\min_{u\in V_2\oplus\hat{V}_1} \|u-z_2\|^2_{L_2(\Omega_1\cup\hat{\Omega}_2)} + 2\alpha\left|D(u_{1|_{\Omega_2\cup\hat{\Omega}_1}} + u_{|_{\Omega_2\cup\hat{\Omega}_1}})\right|(\Omega_2\cup\hat{\Omega}_1) \quad \text{s.t.} \quad \pi_{\hat{V}_1}u = 0, \quad (14)$$

where $z_1 = u_1^{(m)} + \pi_{V_1}T^*(g - Tu_2 - Tu_1^{(m)})$ and $z_2 = u_2^{(m)} + \pi_{V_2}T^*(g - Tu_1 - Tu_2^{(m)})$. Let us rewrite the constrained minimization problems (13) and (14) in another way. More precisely, we consider the problems

$$\arg\min_{\xi_1\in V_1\oplus\hat{V}_2} \|\xi_1 - u_2 - z_1\|^2_{L_2(\Omega_1\cup\hat{\Omega}_2)} + 2\alpha\left|D(\xi_{1|_{\Omega_1\cup\hat{\Omega}_2}})\right|(\Omega_1\cup\hat{\Omega}_2) \quad \text{s.t.} \quad \pi_{\hat{V}_2}\xi_1 = u_2 \quad (15)$$

$$\arg\min_{\xi_2\in V_2\oplus\hat{V}_1} \|\xi_2 - u_1 - z_2\|^2_{L_2(\Omega_1\cup\hat{\Omega}_2)} + 2\alpha\left|D(\xi_{2|_{\Omega_2\cup\hat{\Omega}_1}})\right|(\Omega_2\cup\hat{\Omega}_1) \quad \text{s.t.} \quad \pi_{\hat{V}_1}\xi_2 = u_1 \quad (16)$$

and note that for $i = 1, 2$ and $\hat{i} = \{1, 2\} \setminus \{i\}$ indeed ξ_i is optimal if and only if $u_i = \xi_i - u_{\hat{i}}$ is optimal. Moreover notice that the new problems (15) and (16) are now of the type (2).

3.1.1 Oblique Thresholding (OT)

In [17] the constrained minimization problems (13)-(14) are solved by *Oblique Thresholding*, which is based on an iterative proximity map algorithm and the computation of a Lagrange multiplier by a fixed point iteration. More precisely, iteration (11) for $i = 1$ is explicitly computed by

$$u_1^{(m+1)} = (I - P_{\alpha K})(z_1 + u_{2|_{\Omega_1\cup\hat{\Omega}_2}} - \eta_1^{(m)}) - u_{2|_{\Omega_1\cup\hat{\Omega}_2}}, \quad (17)$$

where $\eta_1^{(m)}$ is a solution of the fixed point iteration

$$\eta_1 = \pi_{\hat{V}_2}P_{\alpha K}(\eta_1 - z_1 - u_{2|_{\Omega_1\cup\hat{\Omega}_2}}),$$

with K being the closure of the set $\{\operatorname{div} p : p \in \left[C_c^1(\Omega)\right]^2, |p(x)| \le 1 \quad \forall x \in \Omega\}$, $|p(x)| = \sqrt{(p^1(x))^2 + (p^2(x))^2}$, and $P_K(u) = \arg\min_{v\in K}\|u - v\|_{L_2(\Omega)}$ is the orthogonal projection onto K. Indeed, the following proposition tells us that the oblique thresholding iteration (17) converges to a minimizer of $\mathcal{J}$ on the subspaces.

Proposition 3.1. *[17, Theorem 4.9] Assume $u_2 \in V_2$ and $\|T\| < 1$. Then the iteration (17) converges weakly to a solution $u_1^* \in V_1$ of $\arg\min_{u_1\in V_1}\mathcal{J}(u_1 + u_2)$ for any initial choice of $u_1^{(0)} \in V_1$.*

In [17] the computation of the orthogonal projection P_K was implemented by using Chambolle's projection method [4]. For more details see [17, Section 4.2]. The oblique thresholding iteration can be very slow in general, cf. [24]. In the next section we shall see how we can accelerate this computation by replacing (17) and its solution via Chambolle's method by BOS-SB.

3.1.2 Bregmanized Operator Splitting - Split Bregman (BOS-SB)

In order to speed up the computation of algorithm (9) we suggest to solve each subproblem by using Algorithm 4. Actually by Algorithm 4 we can directly compute a solution of the constrained optimization problems (15) and (16). That is, for example, the minimizer for

(15) is computed by the following algorithm: Let $A = \pi_{\hat{V}_2}$ and $\lambda, \delta, \mu > 0$ be regularization parameters. Initialize $\xi_1^{(0,L_0)} = \xi_1^{(m)} = u_1^{(m)} + u_2^{(m)}$ and $f^{(0)} = u_2$ then for $k = 0, 1, \dots$ do

$$\begin{cases} \xi_1^{(k+1,0)} = \xi_1^{(k,L_k)}, d^{(k+1,0)} = b^{(k+1,0)} = 0 \\ \text{for } \ell = 0, \dots, L_{k+1} \text{ do} \\ \quad \begin{cases} \xi_1^{(k+1,\ell+1)} = \arg\min_{\xi_1 \in V_1 \oplus \hat{V}_2} \frac{1}{2\alpha} \|\xi_1 - u_2 - z_1\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ \qquad\qquad + \frac{\lambda}{\delta} \|\xi_1 - (\xi_1^{(k,L_k)} - \delta A^*(A\xi_1^{(k,L_k)} - f^{(k)}))\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ \qquad\qquad + \mu \|d^{(k+1,\ell)} - D(\xi_{1|_{\Omega_1 \cup \hat{\Omega}_2}}) - b^{(k+1,\ell)}\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ d^{(k+1,\ell+1)} = \arg\min_d |d|(\Omega_1 \cup \hat{\Omega}_2) + \mu \|d - D(\xi_{1|_{\Omega_1 \cup \hat{\Omega}_2}}^{(k+1,\ell+1)}) - b^{(k+1,\ell)}\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ b^{(k+1,\ell+1)} = b^{(k+1,\ell)} + D(\xi_{1|_{\Omega_1 \cup \hat{\Omega}_2}}^{(k+1,\ell+1)}) - d^{(k+1,\ell+1)} \end{cases} \\ f^{(k+1)} = f^{(k)} - A\xi_1^{(k+1,L_{k+1})} + u_2. \end{cases} \tag{18}$$

Then by setting $u_1^{(k+1,L_{k+1})} = \xi_1^{(k+1,L_{k+1})} - u_2$ for all $k = 0, 1, \dots$ we obtain a sequence $(u_1^{(k,L_k)})_k$, which is converging to a solution of (13). Instead of making a detour by computing the sequence $(\xi_1^{(k,L_k)})_k$ we would like to find a solution of (13), i.e., the update $u_1^{(m+1)}$, directly. Therefore we note that $\xi_1^{(k+1,\ell+1)}$ is a minimizer of

$$\begin{aligned} \arg\min_{\xi_1 \in V_1 \oplus \hat{V}_2} & \frac{1}{2\alpha} \|\xi_1 - u_2 - z_1\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} + \frac{\lambda}{\delta} \|\xi_1 - (\xi_1^{(k,L_k)} - \delta A^*(A\xi_1^{(k,L_k)} - f^{(k)}))\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ & + \mu \|d^{(k+1,\ell)} - D(\xi_{1|_{\Omega_1 \cup \hat{\Omega}_2}}) - b^{(k+1,\ell)}\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \end{aligned}$$

if and only if $u_1^{(k+1,\ell+1)} = \xi_1^{(k+1,\ell+1)} - u_2$ is a minimizer of

$$\begin{aligned} \arg\min_{u_1 \in V_1 \oplus \hat{V}_2} & \frac{1}{2\alpha} \|u_1 - z_1\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} + \frac{\lambda}{\delta} \|u_1 - (u_1^{(k,L_k)} - \delta A^*(Au_1^{(k,L_k)} + u_2 - f^{(k)}))\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ & + \mu \|d^{(k+1,\ell)} - D(u_{1|_{\Omega_1 \cup \hat{\Omega}_2}} + u_{2|_{\Omega_1 \cup \hat{\Omega}_2}}) - b^{(k+1,\ell)}\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)}, \end{aligned}$$

where $u_1^{(k,L_k)} = \xi_1^{(k,L_k)} - u_2$. By this observation, by the fact that $f^{(k)} = f^{(0)} + \sum_{i=1}^k u_2 - A\xi_1^{(i,L_i)} = f^{(0)} - \sum_{i=1}^k Au_1^{(i,L_i)}$, and because $f^{(0)} = u_2$ in the algorithm in (18) we can directly compute the update $u_1^{(m+1)}$ by the following algorithm: Let $A = \pi_{\hat{V}_2}$ and $\lambda, \delta, \mu > 0$ be regularization parameters. Initialize $u_1^{(0,L_0)} = u_1^{(m)}$ and $f^{(0)} = 0$ then for $k = 0, 1, \dots$ do

$$\begin{cases} u_1^{(k+1,0)} = u_1^{(k,L_k)}, d^{(k+1,0)} = b^{(k+1,0)} = 0 \\ \text{for } \ell = 0, \dots, L_{k+1} \text{ do} \\ \quad \begin{cases} u_1^{(k+1,\ell+1)} = \arg\min_{u_1 \in V_1 \oplus \hat{V}_2} \frac{1}{2\alpha} \|u_1 - z_1\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ \qquad\qquad + \frac{\lambda}{\delta} \|u_1 - (u_1^{(k,L_k)} - \delta A^*(Au_1^{(k,L_k)} - f^{(k)}))\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ \qquad\qquad + \mu \|d^{(k+1,\ell)} - D(u_{1|_{\Omega_1 \cup \hat{\Omega}_2}} + u_{2|_{\Omega_1 \cup \hat{\Omega}_2}}) - b^{(k+1,\ell)}\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ d^{(k+1,\ell+1)} = \arg\min_d |d|(\Omega_1 \cup \hat{\Omega}_2) + \mu \|d - D(u_{1|_{\Omega_1 \cup \hat{\Omega}_2}}^{(k+1,\ell+1)} + u_{2|_{\Omega_1 \cup \hat{\Omega}_2}}) - b^{(k+1,\ell)}\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} \\ b^{(k+1,\ell+1)} = b^{(k+1,\ell)} + D(u_{1|_{\Omega_1 \cup \hat{\Omega}_2}}^{(k+1,\ell+1)} + u_{2|_{\Omega_1 \cup \hat{\Omega}_2}}) - d^{(k+1,\ell+1)} \end{cases} \\ f^{(k+1)} = f^{(k)} - Au_1^{(k+1,L_{k+1})}. \end{cases} \tag{19}$$

Hence we can rewrite the algorithm in (19) for the domain Ω_i, for $i = 1, 2$, in the following compact form:

Algorithm 5. *Bregmanized Operator Splitting - Split Bregman in* Ω_i*:* BOS-SB$_i(u_1, u_2, z_i, A, \alpha)$

Input: $u_1, u_2 \in H$, $z_i \in H$, $A \in \mathcal{L}(H)$, $\alpha > 0$
Output: *approximate solution of* (13) *if* $i = 1$ *and of* (14) *if* $i = 2$
Initialize: $\lambda > 0$, $0 < \delta < \frac{1}{\|A^*A\|}$, $u_i^{(0)} = u_i$, $f^{(0)} = 0$
for $k = 0, 1, \ldots, \mathcal{K}$
$u_i^{(k+1)} = \mathrm{SB}_i(u_i^{(k)}, u_{\hat{i}}, z_i, f^{(k)}, A, \alpha, \lambda, \delta)$
$f^{(k+1)} = f^{(k)} - Au_i^{(k+1)}$
end.

where $\hat{i} \in \{1, 2\} \setminus \{i\}$, $\mathcal{K} \in \mathbb{N}$, and

Algorithm 6. *Split Bregman in* Ω_i*:* SB$_i(u_i^{(k)}, u_{\hat{i}}, z_i, f^{(k)}, A, \alpha, \lambda, \delta)$

Input: $u_i^{(k)}, u_{\hat{i}} \in H, z_i \in H, A \in \mathcal{L}(H), \alpha > 0, \lambda > 0, \delta > 0$
Output: *approximate solution of* (20)
Initialize: $\mu > 0$, $\tau > 0$, $d^{(0)} = 0$, $b^{(0)} = 0$
for $\ell = 0, 1, \ldots, L_{k+1}$

$$
\begin{aligned}
u_i^{(\ell+1)} &= \arg\min_{u_i \in V_1 \oplus \hat{V}_2} \tfrac{1}{2\alpha}\|u_i - z_i\|^2_{L_2(\Omega_i \cup \hat{\Omega}_{\hat{i}})} \\
&\quad + \tfrac{\lambda}{\delta}\|u_i - (u_i^{(k)} - \delta A^*(Au_i^{(k)} - f^{(k)}))\|^2_{L_2(\Omega_i \cup \hat{\Omega}_{\hat{i}})} \\
&\quad + \mu\|d^{(\ell)} - D(u_{i|_{\Omega_i \cup \hat{\Omega}_{\hat{i}}}} + u_{\hat{i}|_{\Omega_i \cup \hat{\Omega}_{\hat{i}}}}) - b^{(\ell)}\|^2_{L_2(\Omega_i \cup \hat{\Omega}_{\hat{i}})} \\
d^{(\ell+1)} &= \arg\min_d |d|(\Omega_i \cup \hat{\Omega}_{\hat{i}}) + \mu\|d - D(u^{(\ell+1)}_{i|_{\Omega_i \cup \hat{\Omega}_{\hat{i}}}} + u_{\hat{i}|_{\Omega_i \cup \hat{\Omega}_{\hat{i}}}}) - b^{(\ell)}\|^2_{L_2(\Omega_i \cup \hat{\Omega}_{\hat{i}})} \\
b^{(\ell+1)} &= b^{(\ell)} + D(u^{(\ell+1)}_{i|_{\Omega_i \cup \hat{\Omega}_{\hat{i}}}} + u_{\hat{i}|_{\Omega_i \cup \hat{\Omega}_{\hat{i}}}}) - d^{(\ell+1)}
\end{aligned}
$$

end.

In the finite dimensional setting, i.e., for $V_i = \mathbb{R}^{N \times M}$, where $N, M \gg 1$ is the dimension of the problem, Algorithm 5 enjoys the following convergence properties.

Proposition 3.2. *For* $i = 1, 2$ *and* $\hat{i} \in \{1, 2\} \setminus \{i\}$ *let* $u_i^{(0)} = u_i^{(k)} \in V_i$, $u_{\hat{i}} \in V_{\hat{i}}$, $d^{(0)} = 0$ *and* $(u_i^{(\ell)})_\ell$ *and* $(d^{(\ell)})_\ell$ *be sequences generated by Algorithm 6 (Split Bregman Iteration). Then* $u_i^{(\ell)} \to u_i^*$ *and* $d^{(\ell)} \to d^* = D(u_i^* + u_{\hat{i}})$ *for* $\ell \to \infty$*, where* u_i^* *solves the minimization problem given by one iteration of the Bregmanized Operator Splitting, i.e.,*

$$
\begin{aligned}
\arg\min_{u_i \in V_i} &\frac{1}{2\alpha}\|u_i - z_i\|^2_{L_2(\Omega_i \cup \hat{\Omega}_{\hat{i}})} + \frac{\lambda}{\delta}\|u_i - (u_i^{(k)} - \delta A^*(Au_i^{(k)} - f^{(k)}))\|^2_{L_2(\Omega_i \cup \hat{\Omega}_{\hat{i}})} \\
&+ \left| D(u_{i|_{\Omega_i \cup \hat{\Omega}_{\hat{i}}}} + u_{\hat{i}|_{\Omega_i \cup \hat{\Omega}_{\hat{i}}}}) \right| (\Omega_i \cup \hat{\Omega}_{\hat{i}}).
\end{aligned}
\tag{20}
$$

The proof is analogue to the one in [29], where the convergence of the Split Bregman Iteration is shown.

In the case when we solve the minimization problem in (20) exactly, for example via the Split Bregman Algorithm by setting formally $L_{k+1} = \infty$, then the following convergence property for Algorithm 5 holds:

Proposition 3.3. *Let* $0 < \delta < \frac{1}{\|A^*A\|}$ *and* $u_i^{(k+1)}$ *be the exact solution of the minimization problem in* (20)*. Then the sequences* $(u_i^{(k)})_k$ *generated by Algorithm 5 converges for* $k \to \infty$ *to a solution* u_1^* *of* (13) *if* $i = 1$ *and to a solution* u_2^* *of* (14) *if* $i = 2$*.*

The proof of this result can be found in [32].

3.2 Convergence Properties of the Sequential Domain Decomposition Algorithm

In this section we describe the convergence properties of the sequential non-overlapping domain decomposition algorithm (9), which can be expressed explicitly as follows: Pick an initial $V_1 \oplus V_2 \ni u_1^{(0,M_1)} + u_2^{(0,M_2)} := u^{(0)} \in BV(\Omega)$, for example $u^{(0)} = 0$, and iterate

$$\begin{cases} \begin{cases} u_1^{(n+1,0)} = u_1^{(n,M_1)} \\ u_1^{(n+1,m_1+1)} = \operatorname{argmin}_{u_1 \in V_1} \mathcal{J}_1(u_1, u_2^{(n,M_2)}; u_1^{(n+1,m_1)}) \quad m_1 = 0, \dots, M_1 - 1 \\ u_2^{(n+1,0)} = u_2^{(n,M_2)} \\ u_2^{(n+1,m_2+1)} = \operatorname{argmin}_{u_2 \in V_2} \mathcal{J}_2(u_1^{(n+1,M_1)}, u_2; u_2^{(n+1,m_2)}) \quad m_2 = 0, \dots, M_2 - 1 \end{cases} \\ u^{(n+1)} := u_1^{(n+1,M_1)} + u_2^{(n+1,M_2)}. \end{cases} \tag{21}$$

Note that we do prescribe a finite number M_1 and M_2 of inner iterations for each subspace respectively. Then we have the following properties:

Proposition 3.4. *[17, Theorem 5.1] The algorithm in (21) produces a sequence* $(u^{(n)})_{n\in\mathbb{N}}$ *in* $BV(\Omega)$ *with the following properties:*

(i) $\mathcal{J}(u^{(n)}) > \mathcal{J}(u^{(n+1)})$ *for all* $n \in \mathbb{N}$ *(unless* $u^{(n)} = u^{(n+1)}$*);*

(ii) $\lim_{n\to\infty} \|u^{(n+1)} - u^{(n)}\|_{L_2(\Omega)} = 0$*;*

(iii) the sequence $(u^{(n)})_{n\in\mathbb{N}}$ *has subsequences which converge weakly in* $L_2(\Omega)$ *and in* $BV(\Omega)$ *with the weak-*-topology.*

Remark 3.5. *Note that Proposition 3.4 does in general not guarantee the convergence of the sequence* $u^{(n)}$ *to a minimizer of* $\mathcal{J}$*. In [17] this convergence could be ensured only under some additional technical assumptions on the decomposition, cf. [17, Theorem 5.1 (iv)]. The numerical discussion in [17] as well as the numerical results presented in Section 5 in this paper, however suggest that the proposed method still works well in practice. Indeed when we are dealing with the discrete setting (i.e., all the considered function spaces are finite dimensional), which is the important case for numerical implementations, it is proven in [22] that the non-overlapping domain decomposition algorithm in* (9) *produces a sequence whose accumulation points are indeed minimizers of the original functional* $\mathcal{J}$*, see also [23]. Moreover in [16] the convergence picture is complemented for a modified version of algorithm* (21)*, where convergence to minimizers of* $\mathcal{J}$ *is proven in the discrete setting and for overlapping subdomains.*

Remark 3.6. *The BOS-SB approach can be also used for an overlapping domain decomposition, as it is considered in [16]. Then the operator* A *becomes a trace operator. Moreover, for this case it is possible to restrict the subproblems completely to* Ω_1 *and* Ω_2*, which is due to the induced trace condition.*

3.3 Parallel Versions

One initial motivation of introducing domain decomposition algorithms is parallelization. Simultaneously computing the solution of each subproblem on a multiple processor reduces the computational time of the algorithms significantly, as it was shown in [16] for total variation minimization. Therefore we introduce a parallel version of the previously discussed domain decomposition approaches.

The parallel non-overlapping domain decomposition algorithm reads as follows: Pick an initial $V_1 \oplus V_2 \ni u_1^{(0,M_1)} + u_2^{(0,M_2)} := u^{(0)} \in BV(\Omega)$, for example $u^{(0)} = 0$, and iterate

$$\begin{cases} \begin{cases} u_1^{(n+1,0)} = u_1^{(n,M_1)} \\ u_1^{(n+1,m_1+1)} = \operatorname{argmin}_{u_1 \in V_1} \mathcal{J}_1^s(u_1 + u_2^{(n,M_2)}, u_1^{(n+1,m_1)}) \quad m_1 = 0, \dots, M_1 - 1 \end{cases} \\ \begin{cases} u_2^{(n+1,0)} = u_2^{(n,M_2)} \\ u_2^{(n+1,m_2+1)} = \operatorname{argmin}_{u_2 \in V_2} \mathcal{J}_2^s(u_1^{(n,M_1)} + u_2, u_2^{(n+1,m_2)}) \quad m_2 = 0, \dots, M_2 - 1 \end{cases} \\ u^{(n+1)} := \frac{u_1^{(n+1,M_1)} + u_2^{(n+1,M_2)} + u^{(n)}}{2}. \end{cases} \tag{22}$$

Similar convergence properties as stated in Proposition 3.4 for the sequential algorithm (21) hold for the parallel algorithm (22), cf. [17].

4 Numerical Implementation

We want to implement algorithm (21) and (22) for the minimization of $\mathcal{J}$. To solve its subiterations (11) we consider two approaches: OT [17] and the proposed BOS-SB iteration. In the following we sketch the numerical implementation of both algorithms for the domain Ω_1 only, since the implementation is analogue for the other domain by just adjusting the notations accordingly. Hence we denote $u_2 = u_{2|_{\Omega_1 \cup \hat{\Omega}_2}}^{(n,M_2)}$, $u_1 = u_1^{(n+1,m_1+1)}$, and $z_1 = u_1^{(n+1,m_1)} + \pi_{V_1} T^*(g - Tu_2 - Tu_1^{(n+1,m_1)})$ and we would like to compute the minimizer

$$u_1 = \operatorname{argmin}_{u \in V_1} \|u - z_1\|^2_{L_2(\Omega_1 \cup \hat{\Omega}_2)} + 2\alpha |D(u + u_2)|(\Omega_1 \cup \hat{\Omega}_2). \tag{23}$$

4.1 Oblique Thresholding

By means of oblique thresholding the solution of (23) is computed by

$$u_1 = (I - P_{\alpha K})(z_1 + u_2 - \eta) - u_2,$$

where K is the closure of the set

$$\left\{ \operatorname{div} \xi : \xi \in \left[C_c^1(\Omega)\right]^2, |\xi(x)| \le 1 \quad \forall x \in \Omega \right\}.$$

and the element $\eta \in V_2$ is a limit of the corresponding fixed point iteration

$$\eta^{(0)} \in \hat{V}_2, \quad \eta^{(m+1)} = \pi_{\hat{V}_2} P_{\alpha K}(\eta^{(m)} - (z_1 + u_2)), \quad m \ge 0. \tag{24}$$

For the computation of the projection in the oblique thresholding we use an algorithm proposed by Chambolle in [4].

4.2 BOS-SB

Instead of using the above described oblique thresholding strategy we suggest to use Algorithm 5 to solve the minimization problem (23). Hence the minimizer u_1 is the limit of the sequence $(u_1^{(k)})_k$ generated by Algorithm 5.

Let us rewrite the non-overlapping domain decomposition algorithm (21) explicitly for this case as

$$\begin{cases} \begin{cases} u_1^{(n+1,0)} = u_1^{(n,M_1)} \\ \text{for } m_1 = 0, \dots, M_1 - 1 \\ \quad z_1^{(m_1)} = u_1^{(n+1,m_1)} + \pi_{V_1} T^*(g - Tu_2^{(n,M_2)} - Tu_1^{(n+1,m_1)}) \\ \quad u_1^{(n+1,m_1+1)} = \text{BOS-SB}_1(u_1, u_2^{(n,M_2)}, z_1^{(m_1)}, \pi_{\hat{V}_2}, \alpha) \\ \text{end} \end{cases} \\ \begin{cases} u_2^{(n+1,0)} = u_2^{(n,M_2)} \\ \text{for } m_2 = 0, \dots, M_2 - 1 \\ \quad z_2^{(m_2)} = u_2^{(n+1,m_2)} + \pi_{V_2} T^*(g - Tu_1^{(n+1,M_1)} - Tu_2^{(n+1,m_2)}) \\ \quad u_2^{(n+1,m_2+1)} = \text{BOS-SB}_2(u_1, u_2, z_2^{(m_2)}, \pi_{\hat{V}_1}, \alpha) \\ \text{end} \end{cases} \\ u^{(n+1)} := u_1^{(n+1,M_1)} + u_2^{(n+1,M_2)}. \end{cases} \tag{25}$$

The Split Bregman iteration (Algorithm 6), which is used in BOS-SB$_i$ (i=1,2), is implemented as suggested in [19], with the small adaptation that the distributional derivative of the sum of functions, i.e., $D(u_{i|_{\Omega_i \cup \hat{\Omega}_{\hat{i}}}} + u_{\hat{i}|_{\Omega_i \cup \hat{\Omega}_{\hat{i}}}})$, is considered on each subdomain.

4.3 Discretization

In order to guarantee the concrete computability and the correctness of these procedures, we need to discretize the problem and approximate it in finite dimensions. The continuous image domain $\Omega = [a, b] \times [c, d] \subset \mathbb{R}^2$ is approximated by a finite grid $\{a = x_1 < \ldots < x_N = b\} \times \{c = y_1 < \ldots < y_M = d\}$ with equidistant step-size $h = x_{i+1} - x_i = \frac{b-a}{N} = \frac{d-c}{M} = y_{j+1} - y_j$ equal to 1 (one pixel). The digital image u is an element in $H := \mathbb{R}^{N \times M}$. We denote $u(x_i, y_j) = u_{i,j}$ for $i = 1, \dots, N$ and $j = 1, \dots, M$. The gradient ∇u is a vector in $H \times H$ given by forward differences

$$(\nabla u)_{i,j} = ((\nabla_x u)_{i,j}, (\nabla_y u)_{i,j}),$$

with

$$(\nabla_x u)_{i,j} = \begin{cases} u_{i+1,j} - u_{i,j} & \text{if } i < N \\ 0 & \text{if } i = N, \end{cases}$$

$$(\nabla_y u)_{i,j} = \begin{cases} u_{i,j+1} - u_{i,j} & \text{if } j < M \\ 0 & \text{if } j = M, \end{cases}$$

for $i = 1, \dots, N$, $j = 1, \dots, M$. The discretized functional in two dimensions is given by

$$\mathcal{J}^\delta(u) := \sum_{1 \le i,j \le N} \left(((Tu)_{i,j} - g_{i,j})^2 + 2\alpha |(\nabla u)_{i,j}| \right),$$

with $|y| = \sqrt{y_1^2 + y_2^2}$ for every $y = (y_1, y_2) \in \mathbb{R}^2$. To give a meaning to $(Tu)_{i,j}$ we assume, for instance, that T is applied on the piecewise linear interpolant $\hat{u}$ of the matrix $(u_{i,j})_{i=1,j=1}^{N,M}$

We further introduce the *discrete divergence* $\nabla\cdot : H \times H \to H$ in two dimensions defined, by analogy with the continuous setting, by $\nabla\cdot = -\nabla^*$ (∇^* is the adjoint of the gradient ∇). That is, the discrete divergence operator is given by backward differences

$$(\nabla \cdot p)_{ij} = \begin{cases} (p^x)_{i,j} - (p^x)_{i-1,j} & \text{if } 1 < i < N \\ (p^x)_{i,j} & \text{if } i = 1 \\ -(p^x)_{i-1,j} & \text{if } i = N \end{cases} + \begin{cases} (p^y)_{i,j} - (p^y)_{i,j-1} & \text{if } 1 < j < M \\ (p^y)_{i,j} & \text{if } j = 1 \\ -(p^y)_{i,j-1} & \text{if } j = M, \end{cases}$$

for every $p = (p^x, p^y) \in H \times H$.

4.4 Domain decompositions

4.4.1 Sequential algorithm

For the sequential algorithm we split the domain Ω into horizontal stripes, i.e., the domain $\Omega = [a, b] \times [c, d]$ is split with respect to its rows. In particular we have $\Omega_1 = [a, x_{\lceil \frac{N}{2} \rceil}] \times [c, d]$ and $\Omega_2 = [x_{\lceil \frac{N}{2} \rceil + 1}, b] \times [c, d]$, compare Figure 1. The splitting in more than two domains is done similarly, cf. also [17].

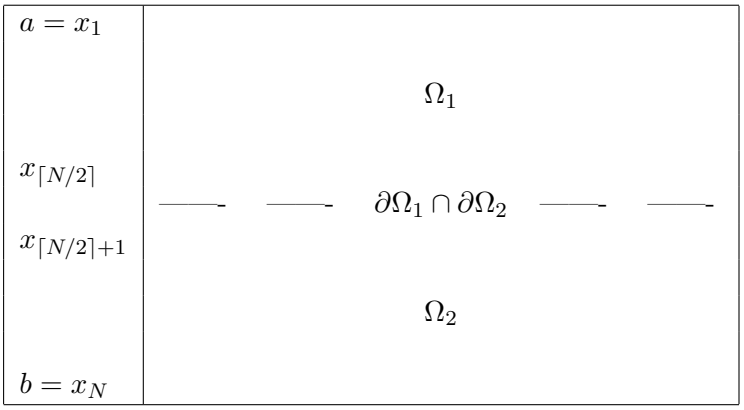


Figure 1: *Decomposition of the discrete image in two domains for the Ω_1 and Ω_2 with interface $\partial\Omega_1 \cap \partial\Omega_2$*

In both subminimization strategies we additionally have to introduce stripes $\hat{\Omega}_1$ and $\hat{\Omega}_2$ on the interfaces of the domain patches. These stripes arise naturally from the splitting of the total variation (12). In case of oblique thresholding, the stripe $\hat{\Omega}_1 \cup \hat{\Omega}_2$ defines the domain in which the η-computation (24) takes place. This is motivated by the observation that η is only supported on Ω_2 and that the due to the restriction to this stripe produced errors are in practice negligible, cf. [17] for more details. In the other case when we use Algorithm 4 we just expand Ω_1 by the domain $\hat{\Omega}_1$, in which an additional constraint is constituted, see (13) and (14).

We note that the size of the neighborhood stripe around the interface is not important from a theoretical point of view and hence in practice one is naturally interested to keep it as small as possible, in order to keep the size of the subproblems as small as possible.

4.4.2 Parallel algorithm

The splitting in the parallel version of the algorithm when applied to image deblurring is done as described above, cf. Figure 1. For image inpainting we choose a different strategy in the parallel case. While the choice of a different splitting for the parallel version has no theoretical motivation (in fact all the theory holds true for arbitrary domain splittings), it shows that the domain decomposition algorithm and its implementation are quite flexible with respect to the type of domain splitting.

For the parallel algorithm applied to inpainting the domain $\Omega = [a, b] \times [c, d]$ is split with respect to its rows and its columns. More precisely, we split Ω into powers of 4 rectangles, i.e., into $4, 16, 64, \ldots$ rectangles. This is done as shown in Figure 2. Note, that it is necessary to expand each of the subdomains by stripes around the interfaces (as in the previous section).

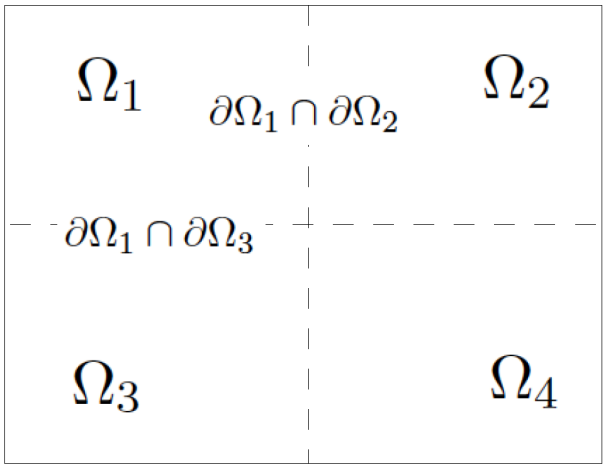


Figure 2: *Domain decomposition for the parallel algorithm in four subdomains* Ω_i, $i = 1, 2, 3, 4$. *The stripes for the subminimization on* Ω_1 *are located around the interfaces* $\partial\Omega_1 \cap \partial\Omega_2$ *and* $\partial\Omega_1 \cap \partial\Omega_3$.

5 Numerical Evaluation

We conclude this paper with a numerical evaluation of the performance of the newly proposed subminimization strategy in Algorithm 4, i.e, *Bregmanized Operator Splitting - Split Bregman* (BOS-SB). To do so, we compare both the sequential- and parallel version of the non-overlapping domain decomposition algorithm, where the subminimization problems are solved with iterative oblique thresholding (OT) (17) and with BOS-SB (19) as a subminimization solver respectively. Here, we focus on its application to image inpainting. In this case the operator T is given by the multiplier $T = 1_{\Omega \setminus D}$, where D is a hole in the given image.

Let us first discuss the choice of the different parameters. For the choice of the regularization parameter see the respective examples discussed below. In the domain decomposition algorithm, we consider domain splittings into $D = 2, 3, 4, 5$ for the sequential version, and $D = 4, 16$ and $D = 2, 4, 6, 8$ for the parallel version, cf. also section 4.4 for more details on the splitting strategy. The domain decomposition algorithm (9) is iterated until the error $e^{(n)} = \|u_{org} - u^{(n)}\|_2 / \|u^{(n)}\|_2$ is smaller than a certain tolerance tol, where u_{org} is the original image. While this stopping criterion is rather unrealistic for practical applications, it serves

us as a good basis for comparing the computational behaviour of the subminimization solvers. Note however, that in our experience the choice of this tolerance has a slightly different impact on the domain decomposition algorithm when we solve the subminimization problems with OT rather than BOS-SB. We shall discuss this in more detail when presenting the numerical examples.

The number of subminimization iterations M_1 and M_2 has been chosen to optimize the computation time for both OT and BOS-SB. In each subdomain we choose the same number of subiteration, i.e., $M_1 = M_2 =: sub$. For OT it turned out that $sub = 1$ is optimal (also see [17]). For BOS-SB $sub = 1$ or $sub = 4$ subminimization iterations give similar computational results in the sequential algorithm (see Table 3), while $sub = 4$ performs noticeably better in the parallel version of the algorithm. The reasons seem to be that each BOS-SB computation is much cheaper in terms of computational time than the OT solution (cf. Table 1), but also that the BOS-SB computation makes more progress in terms of decreasing the error $e^{(n)}$ in each subspace iteration than OT does (cf. Figure 8).

Next, we discuss the parameter choice taken for the subminimization algorithms. We start with oblique thresholding.

Parameter choice for oblique thresholding (OT) The choice of parameters in the oblique thresholding algorithm and their reasoning is discussed in much detail in [17]. In particular, in [17] this choice has been optimized with respect to the computational efficiency of the domain decomposition algorithm, for both its sequential (21) and its parallel (22) version. Therefore, we borrow the parameter values from there and only report them here. The width of the stripe in which η is computed is taken equal to 6 (this can be decreased or increased depending on the size of the regularization parameter α; again see [17] for a discussion on this). The fixed point algorithm for η either terminates when the normalized L_2 distance between two subsequent iterates is smaller than $tol_\eta = 10^{-6}$ or after a maximal number of 10 iterations. The fixed point algorithm of Chambolle [4] for the computation of the projection $P_{\alpha K}(\cdot)$ terminates when the normalized L_2 distance between two subsequent iterates is smaller than $tol_p = 10^{-3}$.

Parameter choice for Bregmanized operator splitting - split Bregman (BOS-SB) Algorithm 4, i.e., BOS-SB, consists of two nested iterations. The outer iteration is the Bregmanized operator splitting (BOS) iteration Algorithm 3, in which the corresponding minimization problem in each iteration is solved via the Split Bregman (SB) algorithm (7) that is again solved iteratively (inner iteration), cf. also Algorithm 5 and Algorithm 6.

The number of BOS-iterations has been chosen equal to 1. In fact, our numerical tests confirm that there is absolutely no gain in terms of computational performance when iterating more. In particular, the number of domain decomposition iterations undertaken to reach a certain accuracy $e^{(n)} < tol$ is exactly the same when iterating BOS once or iterating twice or more. The reason for this is that we chose the parameter λ/δ in front of the Bregman fidelity term $\|u-(u^{(k,L_k)}-\delta A^*(Au^{(k,L_k)}-f^{(k)}))\|_2^2$ very small, i.e., $\lambda/\delta = 10^{-8}/2$. Although this means that we ensure the constraint $Au = 0$ very loosely only, this is adjusted by the reconsideration of this constraint in every domain decomposition iteration. If the value λ/δ is increased also the number of subiterations has to be increased in order to preserve computational accuracy. Additionally our numerical tests showed that in this case also the overall convergence is slower and more outer iterations n are needed. However, note that the BOS-iterations are proven to

converge for every choice of the parameters $\lambda > 0$, $\frac{1}{\|A^*A\|} > \delta > 0$, cf. Proposition 3.3.

In our experiments it turned out that the parameter μ in the Split Bregman algorithm (7) should be reasonable large in order to keep the computational cost low. Hence we choose $\mu = 10$, and iterated the Split Bregman algorithm until the normalized L_2 distance of two subsequent iterates $u^{(l)}$ and $u^{(l+1)}$ is smaller than 10^{-3}. This choice of the stopping criterion is comparable to the tolerance for the Chambolle algorithm in the previous paragraph.

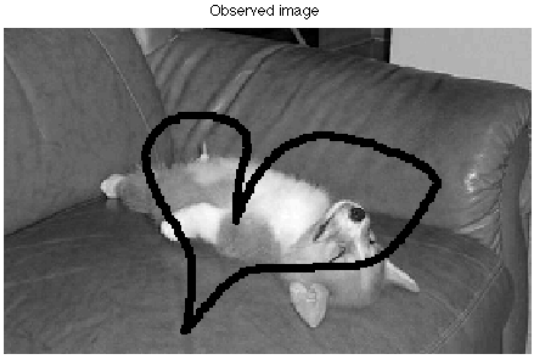
Numerical results - sequential algorithm We present numerical results for the sequential version of the domain decomposition algorithm (9) first. We compare the performance of OT and BOS-SB in terms of quality of the inpainting results and the computational time needed to achieve them. The numerical examples presented here have been computed on a 2×3.2 GHz-Quad-Core MacPro. In Figure 3(a) we start our numerical discussion with an inpainting task for an image of size 270×167. The decompositions of the image domain into D subdomains are done differently for the sequential- and the parallel version of the algorithm, cf. Section 4.4. For further reference, we plotted the decompositions for the image in Figure 3(a) in $D = 4$ domains for the sequential- and parallel case in Figure 3(b) and 3(c) respectively.

In Figure 4 we apply the sequential domain decomposition algorithm (21) with $D = 5$ subdomains for inpainting the image in 3(a). The inpainting results computed with OT and with BOS-SB are presented in Figure 4(a) and 4(b) respectively.

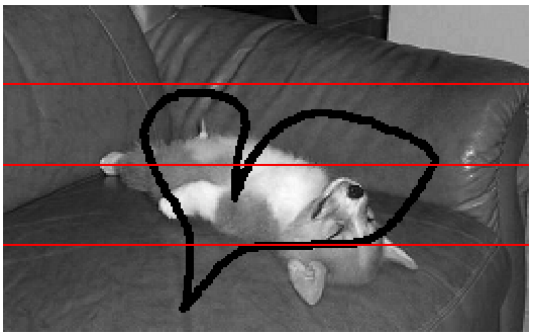
For a first comparison between the two algorithms, we report their computational speed for solving one subspace minimization averaged over the first 100 domain decomposition iterations in Table 1. For the OT algorithm, we have to differ between the subdomains, which are at the border of the image domain, i.e., the first and the last stripe in Figure 3(b), and the ones which are in the inner part of the image domain. The subdomains on the borders share only one interface with the neighbouring subdomain, resulting in only one η-iteration (24), while the other one require the solution of two η-iterations on the lower and upper interfaces. Consequently, the solution of the subspace minimization problem with OT for the border elements is a bit faster than its solution for the inner elements. In either case, BOS-SB by far outperforms the OT algorithm in terms of computational speed. BOS-SB is about three times faster than OT in the computation of one subminimization problem.

Next we compare OT and BOS-SB in their ability to solve the domain decomposition problem accurately and fast. To do so, we first have to find a reasonable basis for comparison. In particular, we have to find the right stopping criterion. One standard choice for stopping an algorithm is to check the distance between two subsequent iterates, i.e., the value of $\|u^{(n+1)} - u^{(n)}\|_2$. If this value is smaller than a prescribed tolerance, i.e., if we are close to a fixed point of the algorithm, the iteration is stopped and the current iterate is accepted as a good approximation to the minimizer. While this is a good criterion for stopping an algorithm, it is not a good one for comparing two algorithms with each other. The iterative behaviour of two algorithms can be very different and being close to a fixed point does not necessarily mean that the algorithm is close to the desired solution. The next generic choice then is the value of the energy evaluated in the iterates. But again, the energy value does not seem to be applicable either because the energy decrease and the energy values seem to be different for the two algorithm, cf. Figure 5.

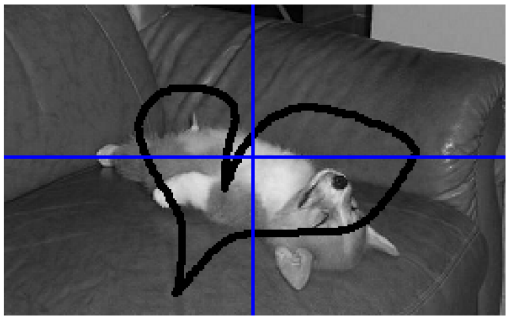
For image inpainting one quality measure is, how close the inpainted image is to the original image. While the original image is usually unknown in practice, for the comparative



(a) *Vandalized image*



(b) *Splitting for the sequential algorithm*

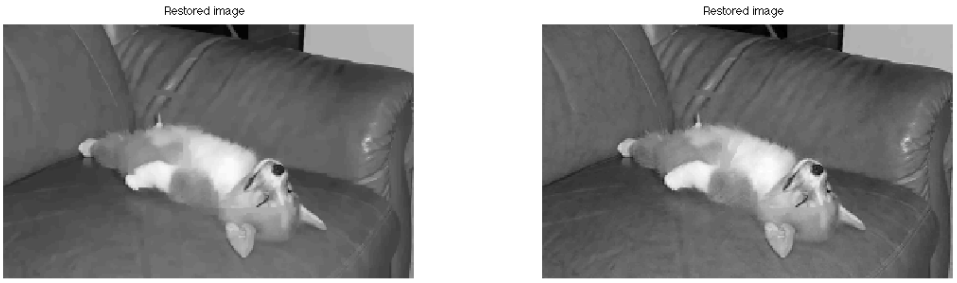


(c) *Splitting for the parallel algorithm*

Figure 3: *The vandalized image and its domain splitting for the sequential and the parallel version of the algorithm for $D = 4$ domains.*

	OT	BOS-SB
$D = 2$: ø CPU time / iteration	0.6 s	0.2 s
$D = 3$: ø CPU time / iteration	0.52 s (border), 0.74 s (inner)	0.17 s
$D = 4$: ø CPU time / iteration	0.46 s (border), 0.72 s (inner)	0.14 s
$D = 5$: ø CPU time / iteration	0.46 s (border), 0.72 s (inner)	0.14 s

Table 1: *Computational performance of the* subminimization solvers *in the sequential version of the domain decomposition algorithm* (21) *with D subdomains for inpainting of Figure 3(a) with $\alpha = 0.005$: CPU times are compared for the OT strategy [17] and the proposed Bregman Operator Splitting - Split Bregman strategy (BOS-SB) Algorithm 4. The reported CPU time is the time in seconds needed for one subspace minimization, averaged over the subdomains and over the first* 100 *domain decomposition iterations. For the evaluation of the OT algorithm one has to defer between a subminimization problem on the border of the image domain (only one interface and hence only one η-iteration) and a subminimization problem in the inner of the image domain (two interfaces and hence two η-iterations).*



(a) *Image inpainted with OT* (b) *Image inpainted with BOS-SB*

Figure 4: *Inpainting with the* sequential version *of domain decomposition* (21) *in* $D = 5$ *subdomains and with* $\alpha = 0.005$*: (a) inpainted image with OT used to solve the subminimization problems; (b) inpainted image with BOS-SB used to solve the subminimization problems.*

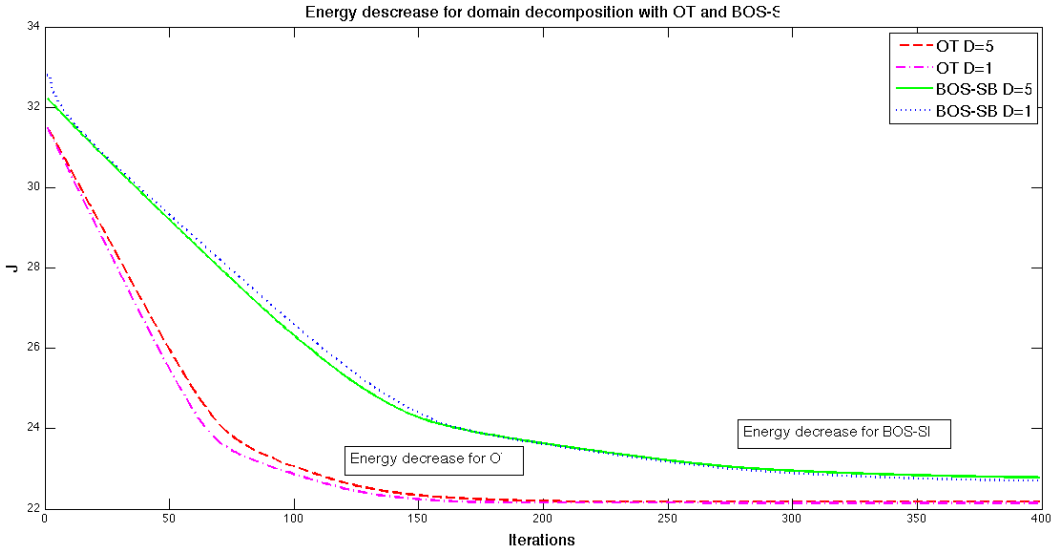


Figure 5: *Decrease of the energy* $\mathcal{J}$ (1) *for the OT algorithm and BOS-SB iteration for the inpainting example in Figure 4, shown for the iterative minimization of* $\mathcal{J}$ *on the whole domain and the domain decomposition iteration with* $D = 5$ *subdomains.*

# domains	OT	BOS-SB (sub= 1)
$D = 2$	223 iterations / 263.32 CPU s	399 iterations / 192.02 CPU s
$D = 3$	290 iterations / 510.45 CPU s	405 iterations / 210.38 CPU s
$D = 4$	843 iterations / 2004.82 CPU s	372 iterations / 210 CPU s
$D = 5$	224 iterations / 636.65 CPU s	394 iterations / 242.39 CPU s
$D = 6$	245 iterations / 817.45 CPU s	375 iterations / 251.62 CPU s

Table 2: *Inpainting for Figure 3(a) with $\alpha = 0.005$: Comparison of computational performance for the* sequential version *of the domain decomposition algorithm* (21) *for using iterative thresholding versus BOS-SB to solve the subminimization problems. The domain decomposition algorithm with OT has been run until $e^{(n)} < tol_1 = 0.02662$, the domain decomposition algorithm with BOS-SB terminated when $e^{(n)} < tol_2 = 0.0225$. When increasing the number of subdomains D, the CPU time seem to increase tremendously for OT, while the computational time for BOS-SB only slightly increases.*

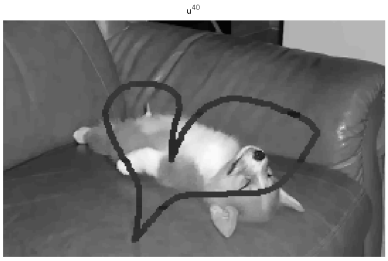
tests we are running, it seems to be a good measure for comparing the computational time of the two algorithms to reach a certain qualitative result. More precisely, let $e^{(n)} = \|u_{org} - u^{(n)}\|_2/\|u^{(n)}\|_2$, $n = 1, 2, \ldots$ be the error between the current iterate $u^{(n)}$ and the original image u_{org}. The algorithms are stopped when $e^{(n)}$ falls below a certain tolerance *tol* the first time. We stick to this choice for a quality measure and a stopping criterion for the two algorithms, although we again have to adapt the aspired tolerance to the two algorithms, cf. Figure 6.

Moreover, in Figure 7 we check the respective inpainting results in detail for the chosen tolerances $tol_1 = 0.02662$ for OT domain decomposition and $tol_2 = 0.0225$ for BOS-SB domain decomposition. The inpainting results seem to be comparable for these choices of stopping criteria.

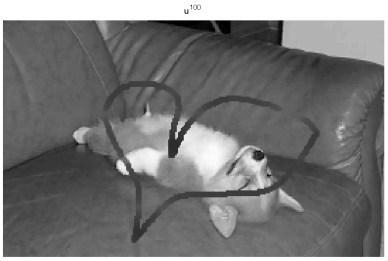
From our numerical discussion up to now we have seen the BOS-SB is three times faster than OT in each subminimization problem, cf. Table 1, but that BOS-SB needs a larger number of iterations to achieve the same quality in the inpainting result as OT. What is the effect of these two subminimization strategies onto the performance of the domain decomposition algorithm as a whole? Looking at Figure 8 and Table 2 one immediately sees that the domain decomposition algorithm with BOS-SB is faster than OT, where the computational advantage of BOS-SB increases with the number of subdomains. In particular, a surprising result for us was, that the CPU time for the sequential version of the algorithm computed with BOS-SB is - in contrast to the same computation with OT - only slightly increasing. Note, that we have not parallelized our computations yet.

Numerical results - parallel algorithm Finally, we also compare the parallel performance of the domain decomposition algorithm when computed with OT and BOS-SB. The computations are made in Matlab on a PC with 8 kernels. The multithreading-option in Matlab is activated such that all algorithms (including the restoration algorithms without domain decomposition) are able to take advantage of the parallel infrastructure offered by the hardware. Also, kindly remember that the number of kernels is 8 when inspecting the computational times for a splitting into 16 subdomains.

Note, that for the sequential version it turned out that solving the subminimization prob-



(a) *OT iterate with* $e^{(40)} = 0.0579$



(b) *BOS-SB iterate with* $e^{(100)} = 0.0456$

(c) *OT iterate with* $e^{(100)} = 0.0333$

(d) *BOS-SB iterate with* $e^{(200)} = 0.0290$

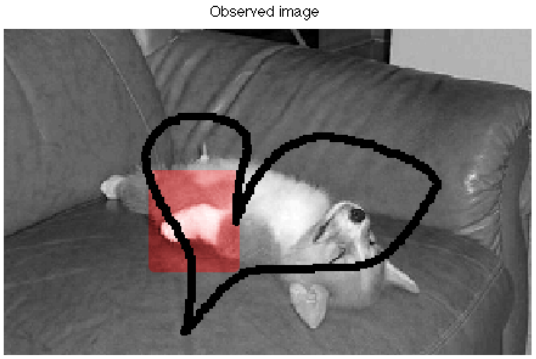
(e) *OT iterate with* $e^{(160)} = 0.0277$

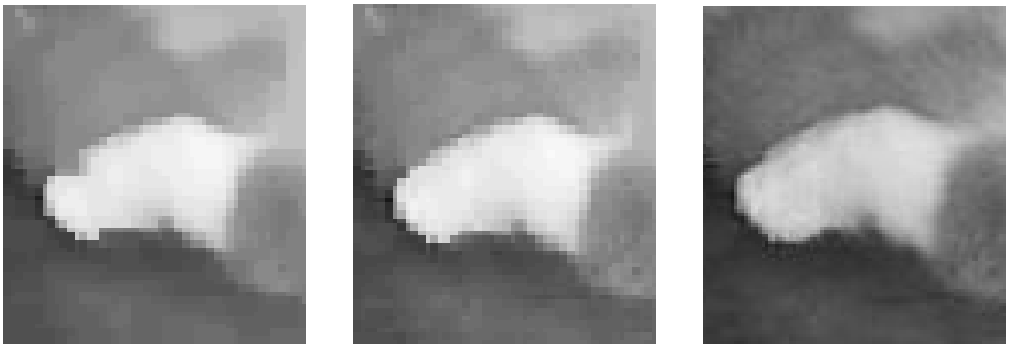
(f) *BOS-SB iterate with* $e^{(300)} = 0.0241$

(g) *OT iterate with* $e^{(220)} = 0.02663$

(h) *BOS-SB iterate with* $e^{(400)} = 0.0224$

Figure 6: *Intermediate results of the inpainting result in Figure 4. The error* $e^{(n)}$ *in the* sequential domain decomposition algorithm (21) *evolves differently when computed with OT and when computed with BOS-SB.*





(a) OT with $tol_1 = 0.0225$ (b) BOS-SB with $tol_2 = 0.0225$ (c) Original image

Figure 7: *Detail of the inpainting result in Figure 4 and the original image. The* sequential domain decomposition algorithm (21) *with OT stops after* 620 *iterations with an error* $e^{(620)} = 0.02662$*, while the same algorithm with BOS-SB used for solving the subminimization problems terminates after* 406 *iterations with an error of* $e^{(406)} = 0.0225$*. Again the error* $e^{(n)} = \|u_{org} - u^{(n)}\|_2 / \|u^{(n)}\|_2$*.*

# domains	BOS-SB (sub= 1)	BOS-SB (sub=4)
$D = 2$	399 iterations / 192.02 CPU s	102×4 iterations / 191.18 CPU s
$D = 3$	405 iterations / 210.38 CPU s	103×4 iterations / 206.8 CPU s
$D = 4$	372 iterations / 210 CPU s	95×4 iterations / 208.69 CPU s
$D = 5$	394 iterations / 242.39 CPU s	101×4 iterations / 245.54 CPU s
$D = 6$	375 iterations / 251.62 CPU s	95×4 iterations / 247.86 CPU s

Table 3: *Inpainting for Figure 3(a) with* $\alpha = 0.005$*: Comparison of computational performance for the* sequential version *of the domain decomposition algorithm* (21) *solved with BOS-SB with sub* $= 1$ *subminimization iteration* (11) *and with sub* $= 4$ *subminimization iterations.*

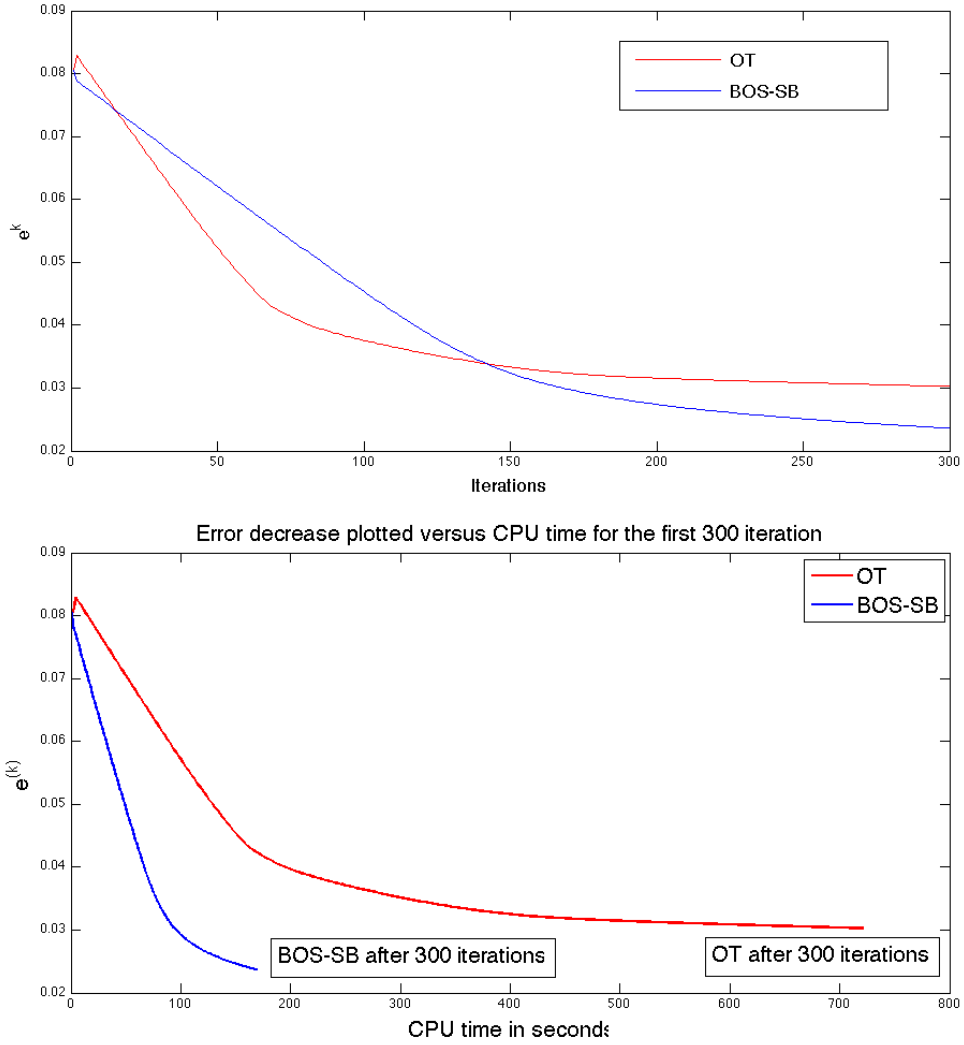


Figure 8: *Error decrease for inpainting of Figure 3(a) with* $\alpha = 0.005$ *with the* sequential version *of the domain decomposition* (21) *in* $D = 4$ *subdomains: In each domain decomposition iteration we measure the error between the original- and inpainted image, i.e.,* $e^{(n)} = \|u_{org} - u^{(n)}\|_2 / \|u^{(n)}\|_2$ *for iterations* $n = 1, 2, \ldots 372$. *While the OT error decreases much faster at the beginning than the error in BOS-SB, it slows down as iterations progress and BOS-SB catches up.*

# domains	OT	BOS-SB
$D = 4$	161 iterations / 135.87 CPU s *(reached accuracy $e^{(k)} = 0.029$)*	92×4 iterations / 65.95 CPU s *(reached accuracy $e^{(k)} = 0.025$)*

Table 4: *Inpainting for Figure 3(a) with $\alpha = 0.005$: Comparison of computational performance for the* parallel version *of the domain decomposition algorithm* (22) *for using OT versus BOS-SB to solve the subminimization problems.*

# domains	Figure 10 ($tol = e^{(k)} = 0.0127$)	Figure 11 ($tol = e^{(k)} = 0.0046$)
$D = 1$	200 iterations / $\approx$ 115 CPU min	50 iterations / $\approx$ 295 CPU min
$D = 4$	47×4 iterations / $\approx$ 47 CPU min	21×4 iterations / $\approx$ 139 CPU min
$D = 16$	50×4 iterations / $\approx$ 43 CPU min	21×4 iterations / $\approx$ 113 CPU min

Table 5: *Inpainting for the images in Figure 10 and 11 with $\alpha = 0.05$: Computational performance for the parallel version of the domain decomposition algorithm* (22) *when using BOS-SB to solve the subminimization problems.*

lems (11) with $sub = 1$ or $sub = 4$ BOS-SB computations does not make a significant difference in terms of computational time needed to solve the inpainting task with the domain decomposition algorithm, cf. Table 3. We take advantage of this fact for the parallel computations. Here, choosing $sub = 4$ and making more progress in each domain decomposition iteration with approximately the same computational effort, reduces the computational time as a whole because we reduce the number of domain decomposition iterations and hence, the amount of communication we have to do between the processes. As already discussed in [17], this strategy cannot be applied for the domain decomposition algorithm solved with OT. For the parallel computations we therefore choose $sub = 1$ for the algorithm with OT and $sub = 4$ for the algorithm solved with BOS-SB. See Table 4 and Figure 9 for the computational results for inpainting of the image in Figure 3(a).

We also test the parallel BOS-SB domain decomposition algorithm for vandalized images of larger scale. In Figure 10 we consider an image of size 1768×2656 pixels where we decomposed it into four non-overlapping domains in order to restore it on multiple processors. As before, the algorithms are compared with respect to the computational time they need to reach a certain accuracy tol, cf. Table 5 left column. A similar example is shown in Figure 11. For getting a different viewpoint on the computational performances, for this example the CPU times are compared for a fixed number of 50 iterations (which seem to be sufficient to inpaint the image satisfactorily), cf. Table 5 right column. The number of subiterations in this case is set to $sub = 1$. We see that by increasing the number of domains, the number of iterations and the CPU time decrease.

Up to now we have only presented numerical examples for the task of inpainting an image. In this case, the operator $T = \chi_{\Omega \setminus D}$ is a local operator and can be easily decomposed with respect to a domain splitting. The same holds true for image denoising, i.e., $T = Id$, and the numerical discussion for this case would look approximately the same. However, we shall also discuss the applicability of the proposed domain decomposition approach for operators

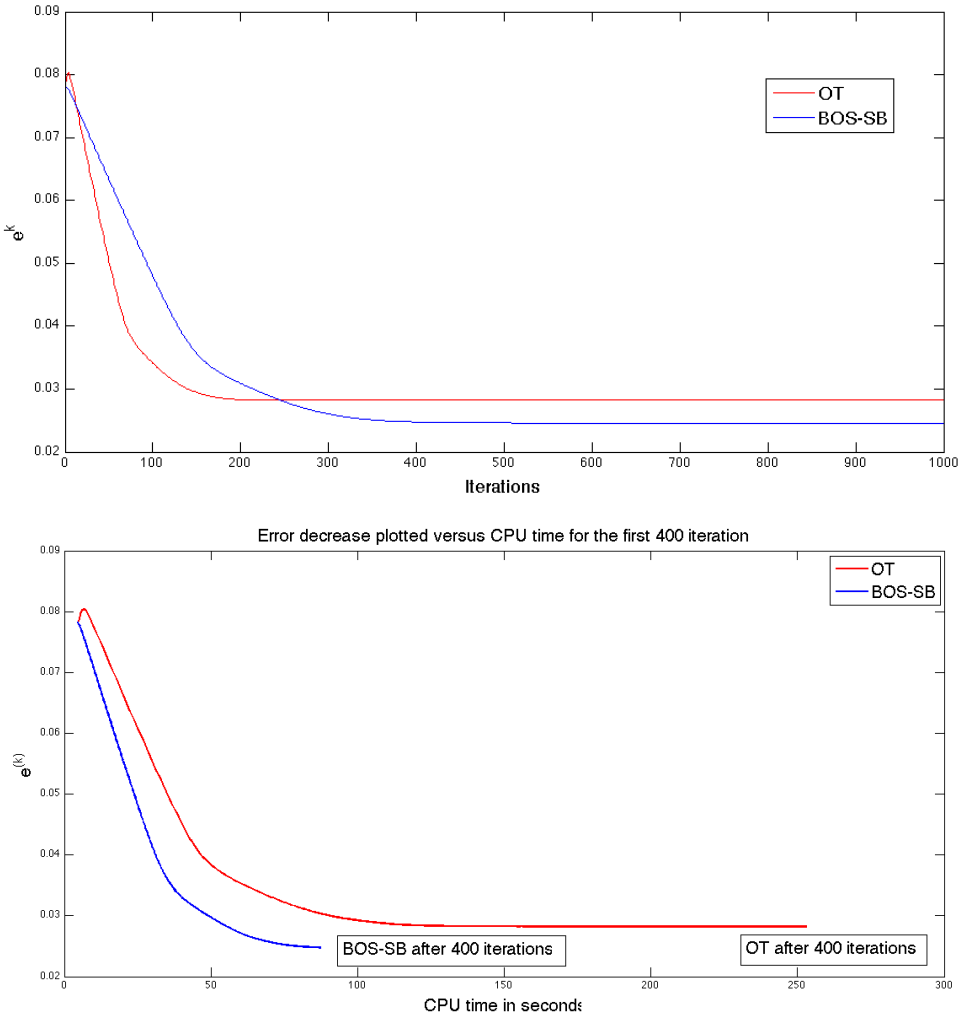


Figure 9: *Error decrease for inpainting of Figure 3(a) with the parallel version of domain decomposition* (22) *in four subdomains and with* $\alpha = 0.005$*: In each domain decomposition iteration we measure the error between the original- and inpainted image, i.e.,* $e^{(n)} = \|u_{org} - u^{(n)}\|_2 / \|u^{(n)}\|_2$ *for iterations* $n = 1, 2, \ldots 1000$*. The final error of OT is* $e^{(1000)} = 0.028$*, while the error in BOS-SB in the final iteration is* $e^{(1000)} = 0.0246$*.*

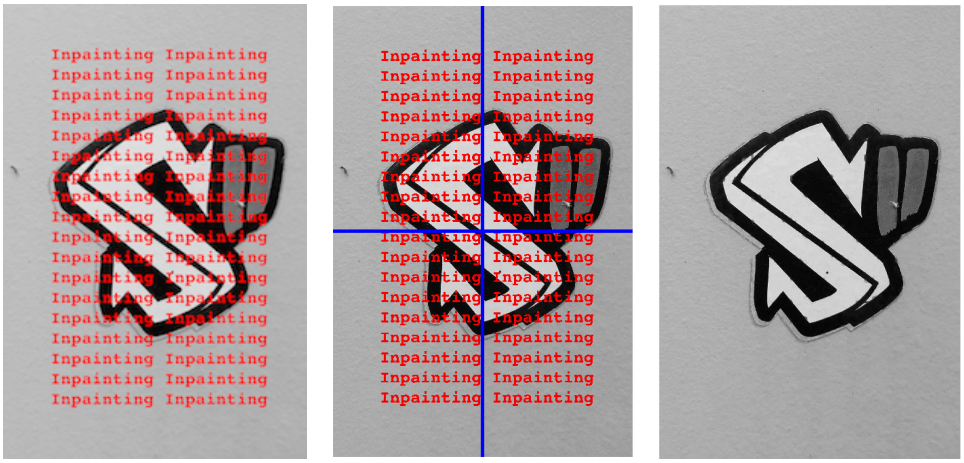


Figure 10: *Inpainting with the parallel domain decomposition strategy* (22) *with* $\alpha = 0.05$*: (l.) the vandalized image of size* 1768×2656 *pixels; (m.) its decomposition into four domains; (r.) the restored image computed with* (22)

Figure 11: *Inpainting with the parallel domain decomposition strategy* (22) *with* $\alpha = 0.05$*: (l.) the vandalized image of size* 5000×3333 *pixels; (r.) the restored image computed with* (22) *with four subdomains.*

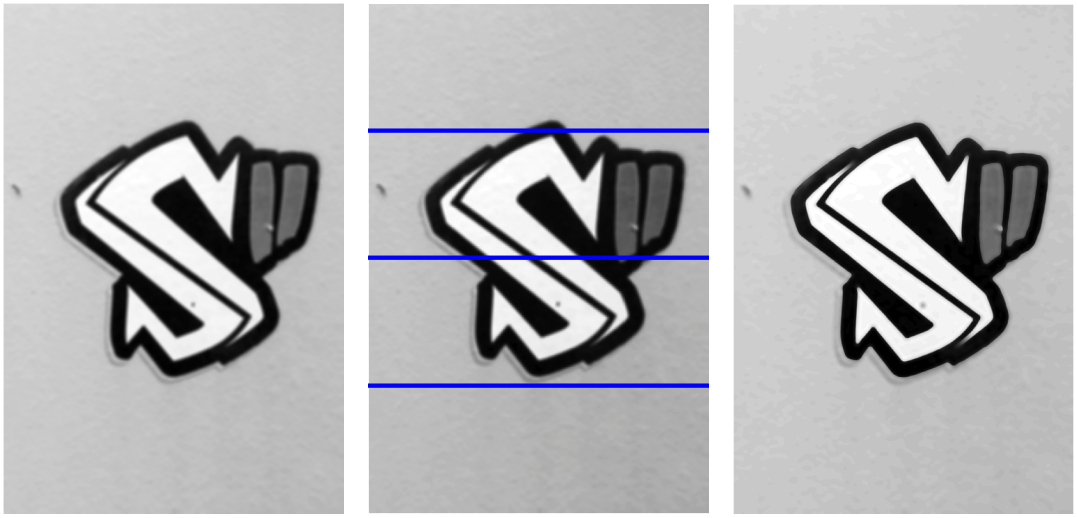


Figure 12: *Deblurring with the parallel domain decomposition strategy* (22) *with* $\alpha = 0.05$*: (l.) the image of size* 1768×2656 *pixels blurred with a Gaussian kernel of size* 30×30 *and standard deviation* $\sigma = 10$*; (m.) its decomposition into four domains; (r.) the deblurred image computed with* (22)

that are not splittable with respect to a domain decomposition. In particular, in what follows we consider the application of our algorithm to image deblurring where the operator T is a convolution with a Gaussian kernel G_σ of standard deviation $\sigma = 10$. For the example in Figure 12 we set the mask size of the Gaussian kernel to be equal to 30×30 pixels, which results in a rather small blurring effect, whereas in Figure 13 an example with stronger blurring is considered with a Gaussian kernel of mask size 40×40 pixels. Note, that the size of the Gaussian kernel should be considered relative to the respective image size.

In Table 6 the computational times for different splittings are reported. While for the inpainting examples we choose a threshold on the error $e^{(k)}$ as the stopping criterion for the algorithms, in the case of image deblurring such a choice does not make much sense, especially if the blurring kernel is large. Hence, for simplicity we let each algorithm run for a fixed number of iterations and compare the CPU times needed. Again, the domain decomposition strategy takes advantage of the parallel structure of the computer. The deblurring example also nicely shows the limitations of this parallel procedure. The deblurring operator T is a global linear operator, i.e., is not splittable with respect to a domain decomposition. Hence, in each domain decomposition iteration the whole solution u has to be communicated to all parallel processes to be able to compute Tu. This results in a trade-off between the reduction of computational time needed for the solution of the subminimization problems and increase of communication time when increasing the number of subdomains, cf. CPU times for $D = 6$ and $D = 8$ in Table 6.

Acknowledgment

Andreas Langer acknowledges the financial support provided by the FWF project Y 432-N15 START-Preis *Sparse Approximation and Optimization in High Dimensions* and the project WWTF Five senses-Call 2006, *Mathematical Methods for Image Analysis and Processing in*

Figure 13: *Deblurring with the parallel domain decomposition strategy* (22) *with* $\alpha = 0.001$*: (l.) the image of size* 1948×1780 *pixels blurred with a Gaussian kernel of size* 40×40 *and standard deviation* $\sigma = 10$*; (r.) the deblurred image computed with* (22) *with a splitting into two domains.*

# domains	Figure 12 (fixed nr. of iterations)	Figure 13 (fixed nr. of iterations)
$D = 1$	50 iterations / $\approx$ 26 CPU min	100 iterations / $\approx$ 19 CPU min
$D = 2$	50 iterations / $\approx$ 20 CPU min	25×4 iterations / $\approx$ 15 CPU min
$D = 4$	50 iterations / $\approx$ 17 CPU min	25×4 iterations / $\approx$ 13 CPU min
$D = 6$	50 iterations / $\approx$ 16 CPU min	25×4 iterations / $\approx$ 12 CPU min
$D = 8$	50 iterations / $\approx$ 17.5 CPU min	25×4 iterations / $\approx$ 14 CPU min

Table 6: *Deblurring for the images in Figure 12 and 13: Computational performance for the parallel version of the domain decomposition algorithm* (22) *with* $\alpha = 0.05$ *for the example in Figure 12 and* $\alpha = 0.001$ *for the example in Figure 13 when using BOS-SB to solve the subminimization problems.*

the Visual Arts. Stanley Osher acknowledges the NSF grants DMS0835863 and DMS0914561, the ONR grant N000140910360, and ARO Muri subs from Rice University and the University of South Carolina. Carola-Bibiane Schönlieb acknowledges the financial support provided by the Cambridge Centre for Analysis (CCA), the DFG Graduiertenkolleg 1023 *Identification in Mathematical Models: Synergy of Stochastic and Numerical Methods* and the Royal Society International Exchanges Award IE110314. Further, this publication is based on work supported by Award No. KUK-I1-007-43 , made by King Abdullah University of Science and Technology (KAUST).

References

[1] L. Ambrosio, N. Fusco, and D. Pallara, *Functions of Bounded Variation and Free Discontinuity Problems.*, Oxford Mathematical Monographs. Oxford: Clarendon Press. xviii, 2000.

[2] G. Aubert and P. Kornprobst, *Mathematical Problems in Image Processing. Partial Differential Equations and the Calculus of Variation*, Springer, 2002.

[3] L. Bregman, *The relaxation method of finding the common points of convex sets and its application to the solution of problems in convex optimization*, USSR Computational Mathematics and Mathematical Physics, **7** (1967), 200–217.

[4] A. Chambolle, *An algorithm for total variation minimization and applications.* J. Math. Imaging Vision **20** (2004), no. 1-2, 89–97.

[5] A. Chambolle, J. Darbon, *On total variation minimization and surface evolution using parametric maximum flows*, Int. J. Comput. Vis. **84** (2009), 288–307

[6] A. Chambolle and P.-L. Lions, *Image recovery via total variation minimization and related problems.*, Numer. Math. **76** (1997), no. 2, 167–188.

[7] T. F. Chan, G. H. Golub, and P. Mulet, *A nonlinear primal-dual method for total variation-based image restoration*, SIAM J. Sci. Comput. **20** (1999), no. 6, 1964–1977.

[8] P. L. Combettes and V. R. Wajs, *Signal recovery by proximal forward-backward splitting*, Multiscale Model. Simul., **4** (2005), no. 4, 1168–1200.

[9] I. Daubechies, M. Defrise, and C. DeMol, *An iterative thresholding algorithm for linear inverse problems*, Comm. Pure Appl. Math. **57** (2004), no. 11, 1413–1457.

[10] I. Daubechies, G. Teschke, and L. Vese, *Iteratively solving linear inverse problems under general convex constraints*, Inverse Probl. Imaging **1** (2007), no. 1, 29–46.

[11] J. Darbon and M. Sigelle, *A fast and exact algorithm for total variation minimization*, IbPRIA 2005, 3522(1), pp. 351-359, 2005.

[12] D. Dobson and C. R. Vogel, *Convergence of an iterative method for total variation denoising*, SIAM J. Numer. Anal. **34** (1997), no. 5, 1779–1791.

[13] H.W. Engl, M. Hanke, and A. Neubauer, *Regularization of inverse problems.*, Mathematics and its Applications (Dordrecht). 375. Dordrecht: Kluwer Academic Publishers., 1996.

[14] L. C. Evans and R. F. Gariepy, *Measure Theory and Fine Properties of Functions.*, CRC Press, 1992.

[15] M. Fornasier, *Domain decomposition methods for linear inverse problems with sparsity constraints*, Inverse Problems **23** (2007), no. 6, 2505–2526.

[16] M. Fornasier, A. Langer, and C.-B. Schönlieb, *A convergent overlapping domain decomposition method for total variation minimization*, Numerische Mathematik, Vol. 116, Nr. 4, pp. 645 - 685 (2010).

[17] M. Fornasier and C.-B. Schönlieb, *Subspace correction methods for total variation and ℓ_1-minimization*, SIAM J. Numer. Anal., **47** (2009), no. 4, 3397–3428.

[18] K. Frick and O. Scherzer, *Regularization of ill-posed linear equations by the non-stationary augmented Lagrangian method*, J. Integral Euqations Appl. **22** no.2 (2010), 217–257

[19] T. Goldstein and S. Osher, *The split bregman method for ℓ_1 regularized problems*, SIAM J. Imaging Sci. **2** no. 2 (2009), 323–343

[20] M. Hintermüller and G. Stadler, *An Infeasible Primal-Dual Algorithm for Total Bounded Variation-based Inf-Convolution-Type Image Restoration*, SIAM J. Sci. Comput. **28**, no. 1, 1–23.

[21] K. Ito and K. Kunisch, *Lagrange Multiplier Approach to Variational Problems and Applications*, Series: Advances in Design and Control (No. 15) SIAM, 2008

[22] A. Langer, *Convergence Analysis of a Non-overlapping Domain Decomposition Algorithm for Total Variation Minimization*, 2009, preprint.

[23] A. Langer, *Subspace Correction and Domain Decomposition Methods for Total Variation Minimization*, PhD Thesis, Johannes Kepler University Linz, 2011

[24] I. Loris, *On the performance of algorithms for the minimization of* 1*-penalized functionals*, Inverse Problems **25** (2009).

[25] Y. Nesterov, *Smooth minimization of non-smooth functions.* Mathematical Programming., Ser. A, **103** (2005), 127–152.

[26] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin, *An iterative regularization method for total variation-based image restoration*, Multiscale Model. Simul., **4** no. 2 (2005), 460–489

[27] L. I. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms.*, Physica D **60** no. 1-4 (1992), 259–268.

[28] C.-B. Schönlieb, *Total variation minimization with an H^{-1} constraint*, CRM Series 9, Singularities in Nonlinear Evolution Phenomena and Applications proceedings, Scuola Normale Superiore Pisa 2009, 201-232.

[29] S. Setzer, *Split Bregman Algorithm, Douglas-Rachford Splitting and Frame Shrinkage.* In X.-C. Tai et al. (Ed..): Proceedings of the Second International Conference on Scale Space Methods and Proceedings of the 2nd International Conference on Scale Space and Variational Methods in Computer Vision, LNCS, vol. 5567, pp. 464-476, Springer, Berlin, (2009).

[30] L. Vese, *A study in the BV space of a denoising-deblurring variational problem.*, Appl. Math. Optim. **44** (2001), 131–161.

[31] W. Yin, S. Osher, D. Goldfarb, and J. Darbon, *Bregman Iterative Algorithms for ℓ_1-Minimization with Applications to Compressed Sensing*, SIAM J. Imaging Sci., **1** (2008), no. 1 , 143–168.

[32] X. Zhang, M. Burger, X. Bresson, and S. Osher, *Bregmanized Nonlocal Regularization for Deconvolution and Sparse Reconstruction*, SIAM J. Imaging Sci. **3** no. 3 (2010), 253–276.