APPLICATION OF L1 MINIMIZATION TECHNIQUE TO IMAGE SUPER-RESOLUTION AND SURFACE RECONSTRUCTION

A Thesis

by

HABIBALLAH TALAVATIFARD

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Jean-Luc Guermond
Co-Chair of Committee,	Nancy Amato
Committee Members,	Bojan Popov
	Wolfgang Bangerth
Head of Department,	Emil Straube

May 2013

Major Subject: Mathematics

ABSTRACT

A surface reconstruction and image enhancement non-linear finite element technique based on minimization of L^1 norm of the total variation of the gradient is introduced. Since minimization in the L^1 norm is computationally expensive, we seek to improve the performance of this algorithm in two fronts: first, local L^1-minimization, which allows parallel implementation; second, application of the Augmented Lagrangian method to solve the minimization problem. We show that local solution of the minimization problem is feasible. Furthermore, the Augmented Lagrangian method can successfully be used to solve the L^1 minimization problem. This result is expected to be useful for improving algorithms computing digital elevation maps for natural and urban terrain, fitting surfaces to point-cloud data, and image super-resolution.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Motivation

In geometric modeling and image reconstruction, one needs to extract a detailed and accurate surface model from a set of measurements. Namely, given a set of measured data point in R^d where $d = 1, 2$, one seeks to construct a piecewise smooth approximation u, that satisfies constraints or fits given data and is suitable in a certain sense. Intuitively, a suitable approximation appears pleasing to eye and preserves the shape of the surface. For example, one may want to reconstruct a convex body if the underlying data comes from a convex object, a flat surface if the data is locally flat, or preserve a particular structure of the level sets.

This type of problem is sometimes solved by minimizing an L^2 norm of the Hessian or the total variation of the gradient for an approximating spline [1]. It turns out that minimizing the total variation of the gradient of a smooth function amounts to minimizing the L^1-norm of its second derivatives. The key observation is that contrary to L^2 based methods, using L^1-norm in the minimization process produces oscillation free reconstructions [2]. This especially proves to be valuable when one deals with noisy data, or with sharp features in a man-made image. Further, for natural images L^1 offers a much sparser representation of the surface, and thus more suitable for compressing the data.

In this work, rather than using splines, we minimize the total variation of the gradient of a function constructed on a continuous finite element space satisfying interpolatory constraints following the approach in [4]. Next section introduces this approach in detail and discusses its up and down sides.

1.2 The Model Problem

Consider an open domain Ω in $\mathbb{R}^2$. We assume each surface u resides in X which is defined to be the subspace of $W^{1,1}(\Omega)$ composed of functions whose gradients have bounded variations:

$$X \coloneqq \{u \in W^{1,1}(\Omega) \mid \nabla u \in [BV(\Omega)]^2\}. \tag{1.1}$$

Measurements are given in form of a family of functionals $d \coloneqq \{d_i : i = 1 \dots I_h\}$ that evaluate $u \in X$. Two examples of d are the following: Q-measurements which evaluate u point-wise such that $d_i(u) = u(x_i)$ for some set of points x_i, and S-measurements which averages u such that $d_i(u) = \frac{1}{|T_i|}\int_{T_i} u(x)$ for some sub-domain T_i. We are given data $\varpi \coloneqq \{\varpi_i : i = 1 \dots I_h\}$, obtained from the surface at hand. We seek to find a $u \in X$ such that $d_i(u)$ are equal to ϖ_i.

1.2.1 The L^1 Minimization Problem

To solve above problem, we require the total variation of ∇u is minimum for $u \in X$. Note that functions in X are not necessarily smooth, and the total variation of ∇u provides an extension to X and the L^1-norm of the Hessian for smooth functions.

It is not clear how to find such a u in general, so we find an approximation. Let $\mathcal{T}_h$ be a mesh on Ω composed of open quadrilaterals (or cells). The set of vertices of $\mathcal{T}_h$ is denoted by $\mathcal{V}_h$, and the set of interior edges of $\mathcal{T}_h$ is denoted by $\mathcal{F}_h$. We define a discretization space X_h, a subspace of X, composed of continuous functions that are piecewise cubic on the mesh $\mathcal{T}_h$:

$$X_h = \{u \in \mathcal{C}^0(\overline{\Omega}) : u|_T \in \mathbb{Q}_3 \ \ \forall T \in \mathcal{T}_h\}, \tag{1.2}$$

Rather than minimizing the semi-norm $|\nabla u_h|_{BV}$ for $u_h \in X_h$, we can minimize an equivalent semi-norm:

$$J(u_h) = \sum_{T \in \mathcal{T}_h} \int_T \left(|\partial_{xx} u_h| + 2|\partial_{xy} u_h| + |\partial_{yy} u_h|\right) + \alpha \sum_{F \in \mathcal{F}_h} \int_F |[\![\partial_{\mathbf{n}} u_h]\!]_F|, \quad u_h \in X_h, \quad (1.3)$$

where $\alpha > 0$ is parameter that controls the importance of achieving a good approximation across the edges, and $[\![\partial_{\mathbf{n}} .]\!]$ is defined to be the jump operator on a face, i.e for face F between two neighboring cells T and T':

$$[\![\partial_{\mathbf{n}} u]\!]_F = (\nabla u|_T) \cdot \mathbf{n} - (\nabla u|_{T'}) \cdot \mathbf{n}. \quad (1.4)$$

Note that the smaller the above functional is, the better a given surface is approximated.

We are going to consider two cases to handle the measurements:

1. Interpolation. In this case we require to strictly satisfy the constraints imposed by measurements. We define the solution space Y_h to be the subspace $Y_h \subset X_h$ consisted of the functions that satisfy the measurements exactly

$$Y_h = \{u_h \in X_h : d_i(u_h) = \varpi_i, \ \forall i = 1 \dots I_h\}. \quad (1.5)$$

We seek to find $u_h \in Y_h$ approximating u such that the following functional is minimum:

$$J(u_h) = \sum_{T \in \mathcal{T}_h} \int_T \left(|\partial_{xx} u_h| + 2|\partial_{xy} u_h| + |\partial_{yy} u_h|\right) + \alpha \sum_{F \in \mathcal{F}_h} \int_F |[\![\partial_{\mathbf{n}} u_h]\!]_F|, \quad u_h \in Y_h, \quad (1.6)$$

2. Relaxation. We may relax the requirement to satisfy the constraints. In this

case we use the whole X_h as the solution space ($Y_h = X_h$) and minimize the following functional:

$$J(u_h) = \sum_{T \in \mathcal{T}_h} \int_T (|\partial_{xx} u_h| + 2|\partial_{xy} u_h| + |\partial_{yy} u_h|) + \alpha \sum_{F \in \mathcal{F}_h} \int_F |[\![\partial_{\mathbf{n}} u_h]\!]_F| + \beta \sum_i^{I_h} |d_i(u_h) - \varpi_i|, \quad u_h \in Y_h, \tag{1.7}$$

where β controls how strictly the interpolation conditions are enforced. In this thesis, $\beta = \infty$ means strict interpolation. We define $\Delta = 1/I_h \sum_i^{I_h} |d_i(u_h) - \varpi_i|$ to be the average constraints satisfaction error.

In any case, one can express the problem as

$$u_h = \text{argmin}_{v \in Y_h} J(v). \tag{1.8}$$

1.3 Global L^1 Minimization Algorithm

We evaluate 4.10 and 1.7 functionals by replacing the terms of integrals with quadratures. The approximate functionals will read

$$J_h(u_h) = \sum_{T \in \mathcal{T}_h} \sum_{\mathcal{L} \in \{\partial_{xx}, 2\partial_{xy}, \partial_{yy}\}} \sum_{(p,\omega) \in I(T,\mathcal{L})} \omega |\mathcal{L}(u_h)(p)| + \alpha \sum_{F \in \mathcal{F}_h} \sum_{(p,\omega) \in I(F, [\![\partial_{\mathbf{n}}]\!])} \omega |[\![\partial_{\mathbf{n}} u_h]\!]_F(p)| \tag{1.9}$$

for the interpolation case, and

$$
\begin{aligned}
J_h(u_h) = & \sum_{T\in\mathcal{T}_h} \sum_{\mathcal{L}\in\{\partial_{xx},2\partial_{xy},\partial_{yy}\}} \sum_{(p,\omega)\in I(T,\mathcal{L})} \omega|\mathcal{L}(u_h)(p)| \\
& + \alpha \sum_{F\in\mathcal{F}_h} \sum_{(p,\omega)\in I(F,[\![\partial_{\mathbf{n}}]\!])} \omega|[\![\partial_{\mathbf{n}} u_h]\!]_F(p)| \\
& + \beta \sum_{i}^{I_h} |d_i(u_h) - \varpi_i|
\end{aligned} \tag{1.10}
$$

for the relaxation case. Here $\mathcal{L}$ is one of the linear operators $\{\partial_{xx}, 2\partial_{xy}, \partial_{yy}\}$ or $[\![\partial_{\mathbf{n}}]\!]$. $I(.,\mathcal{L})$ denotes the indices for the set of quadrature points used for a given cell or face, and is composed of pairs (p,ω) of points $p \in \mathbb{R}^2$ and weights $\omega > 0$.

To solve problem numerically, one first needs a matrix formulation of minimization problem. Let $\{\phi_i\}_{i=1}^n$ be a basis for X_h and let $\{(p_i,\omega_i)\}_{i=1}^m$ be an enumeration of all the quadrature points (and weights) used in the discretization of the functionals in each case, and let $\{\mathcal{L}_i\}_{i=1}^m$ be the collection of linear operators corresponding to the quadrature rules. Define $I(\mathcal{T}_h)$ to be indices for the set of all quadratures with $\mathcal{L}_i \in \{\partial_{xx}, 2\partial_{xy}, \partial_{yy}\}$ and $I(\mathcal{F}_h)$ be the set of all quadrature points with $\mathcal{L}_i = [\![\partial_{\mathbf{n}}]\!]$. Again we consider two cases:

1. Interpolation. Define:

$$
\widehat{A}_{ij} = \begin{cases} \omega_i \mathcal{L}_i(\phi_j)(p_i) & i \in I(\mathcal{T}_h) \\ \alpha\omega_i \mathcal{L}_i(\phi_j)(p_i) & i \in I(\mathcal{F}_h), \end{cases} \qquad j = 1,\ldots,n. \tag{1.11}
$$

Then the functional J_h can be re-written as:

$$
J_h(u_h) = |\widehat{A}x|_1 \qquad \text{where} \qquad x \in \mathbb{R}^n : u_h = \sum_{j=1}^n x_j \phi_j, \tag{1.12}
$$

We define

$$B_{ij} = d_i(\phi_j), \quad i = 1 \dots I_h, \; j = 1 \dots n \tag{1.13}$$

Then the measurements may be expressed as I_h linear constraints in the form

$$B\,x = \varpi \tag{1.14}$$

where B is a $I_h \times n$ matrix. The minimization is carried out in Y_h, hence the minimization becomes

$$x = \text{argmin}_{y \in \mathbb{R}^n} \; |\widehat{A}y|_1, \quad \text{subject to } By = \varpi \tag{1.15}$$

To ensure the constraints are always satisfied, we do the following. Let $i = 1 \dots I_h$. For each i we can always find a nodal basis (with repetition not allowed) with index k such that $B_{ik} \neq 0$. Denote the mapping that associates the ith measurement with index k with a, so $k = a(i)$. For all $i = 1 \dots I_h$ we rewrite 1.14 as [1]:

$$x_{a(i)} = \frac{1}{B_{i\,a(i)}} \Big(\varpi - \sum_{j=1,\, j \neq a(i)}^{n} B_{ij} x_j \Big) \tag{1.16}$$

The set K, defined as $K = \{k \,|\, k = a(i), i = 1 \dots I_h\}$, represents the set of all constrained nodal basis that their corresponding value is not directly given by the minimization procedure. Rather, the values of bases indexed by K are computed by 1.16 after the corresponding value of all other nodal bases are determined. This is done by making the corresponding column to each $k \in K$ zero in the linear system $\widehat{A}x$. We

[1] We may also need to resolve chains of constraints. For example, one might have $x_{13} = x_3/2 + x_7/2$ while $x_7 = x_2/2 + x_4/2$. Then, the resolution will be $x_{13} = x_3/2 + x_2/4 + x_4/4$.

form a modified linear system given by matrix A and right hand side b for $j = 1 \dots n$:

$$A_j = \begin{cases} \widehat{A}_j + \sum_{i=1}^{I_h} \frac{B_{ij}}{B_{i\,a(i)}} \widehat{A}_{a(i)} & j \notin K \\ 0 & j \in K \end{cases} \qquad j = 1 \dots n \tag{1.17}$$

$$b = -\sum_{i=1}^{I_h} \frac{\varpi}{B_{i\,a(i)}} A_{a(i)} \tag{1.18}$$

where $\widehat{A}_j$ and A_j denote the jth columns of $\widehat{A}$ and A respectively. The minimization problem finally reads:

$$x = \operatorname{argmin}_{y \in \mathbb{R}^n} |Ay - b|_1 \tag{1.19}$$

Once x is computed[2] x_k values for $k \in K$ are discarded, and recomputed by 1.16. In the very special case of point Q-measurements, matrix B has exactly one 1 in each row. Thus to update A_j for $j \notin K$ in 1.17, one only needs to make the corresponding columns in $\widehat{A}$ zero and form the right hand side b.

2. Relaxation. Define

$$A_{ij} = \begin{cases} \omega_i \mathcal{L}_i(\phi_j)(p_i), & i \in I(\mathcal{T}_h), \\ \alpha\omega_i \mathcal{L}_i(\phi_j)(p_i), & i \in I(\mathcal{F}_h^i), \\ \beta d_{i-m}(\phi_j), & i = m+1, \dots, m+I_h, \end{cases} \qquad j = 1, \dots, n. \tag{1.20}$$

and

$$b_i = \begin{cases} 0, & i = 0, \dots, m, \\ \beta\varpi_{i-m}, & i = m+1, \dots, m+I_h. \end{cases} \tag{1.21}$$

[2]Note that all the minimization methods used in this thesis involve solution of equations of the form $A^T A\, w = z$ and $A^T DA\, w = z$. It is easy to see that the elements on the main diagonal of $A^T A$ and $A^T DA$ corresponding to $k \in K$ are zero. To prevent numerical breakdown one has to reset all these diagonal elements from zero to one. This ensures the value computed for w_k for $k \in K$ will be zero rather than undetermined.

Then the functional J_h can be re-written as:

$$J_h(u_h) = |Ax - b|_1 \qquad \text{where} \qquad x \in \mathbb{R}^n : u_h = \sum_{j=1}^{n} x_j \phi_j, \tag{1.22}$$

The minimization problem reads:

$$x = \text{argmin}_{y \in \mathbb{R}^n} |Ay - b|_1 \tag{1.23}$$

In both cases, A is a $m \times n$ matrix and $m > n$. Thus $Ay - b$ for $y \in \mathbb{R}^n$ is an over-determined system with no solution. The minimization problem 1.23 does not have a unique minimizer in general either.

In [4] the above algorithm convergence was established and an Interior Point method was used to solve the associated linear programming problem 1.23. It was shown there that the L^1-minimization algorithm produces very sharp results, comparable or better than anything else available in the literature. However, one key observation is that minimization in L^1 norm is computationally expensive, and any serious attempt to use this method should come up with a way to make the minimization algorithm fast and capable of solving very large problems. We are going to consider two possible approaches to address this concern: first, local L^1-minimization, which allows parallel implementation; second, use of a different method to solve the minimization problem, namely, the Augmented Lagrangian method. Next three sections discuss these directions.

2. L^1 MINIMIZATION METHODS

This chapter discusses some of the available methods to solve the discrete minimization problem 1.23.

2.1 Interior Point Method

One way to solve eq. 1.23 is to minimize $\|v\|_{L^1}$ such that the constraint $Au-b-v = 0$ is satisfied for a $v \in \mathbb{R}^n$. Interior Point method utilizes this idea by applying Newton method to the resulting linear programming constraints [3]. The step size and direction of move in this algorithm are chosen so that constraints are satisfies at each iteration. The Interior Point method reads [4]:

input: A, b, x, λ; μ, ϵ;

$r = b - Ax$;

$a = (|r|_1 - r^t\lambda)/m; \quad y_i = |r_i| + a,\ i = 1, \ldots, m$;

while $\boxed{(|r|_1 > (1+\epsilon)\, r^t\lambda)}$ **do**

  $t^{-1} = (y^t 1 - r^t\lambda)/(2m\mu)$;

  $s_1 = y + r; \quad s_2 = y - r$;

  $d_1 = (1-\lambda)/(2s_1); \quad d_2 = (1+\lambda)/(2s_2)$;

  $d = 4d_1d_2/(d_1+d_2)$;

  $v = t^{-1}(s_2^{-1} - s_1^{-1}) + (d_2 - d_1)/(d_1+d_2)[1 - t^{-1}(s_1^{-1} + s_2^{-1})]$;

  $w = A^t v$;

  $\boxed{\Delta x = (A^t \mathrm{diag}(d) A)^{-1} w}$;

  $v = A\Delta x$;

  $\Delta y = [-1 + t^{-1}(s_1^{-1} + s_2^{-1}) + (d_1 - d_2)v]/(d_1+d_2)$;

  $\Delta\lambda = -\lambda + t^{-1}(s_2^{-1} - s_1^{-1}) - (d_1+d_2)v + (d_1 - d_2)\Delta y$;

  $\sigma = \max\{\tau \in (0,2] : -1 \le \lambda + \tau\Delta\lambda \le 1,$

$$y + \tau\Delta y \geq r - \tau v,\ y + \tau\Delta y \geq -r + \tau v\}$$

16: $\sigma = \min\{1,\ 0.99\sigma\}$;

17: $x = x + \sigma\Delta x; \quad y = y + \sigma\Delta y; \quad r = r - \sigma v; \quad \lambda = \lambda + \sigma\Delta\lambda;$

18: **end while**

19: **output:** $x,\ \lambda$;

2.2 Augmented Lagrangian Method

This method solves the unconstrained problem by introducing a term that mimics Lagrange multipliers [5]. To derive this method we rewrite the minimization problem $\min_u \|Au - b\|_{L^1}$ by introducing auxiliary variable $v := Au - b$. It reads

$$\|v\|_{L^1}, \quad \text{subject to } Au - b - v = 0 \tag{2.1}$$

Now we consider the Lagrangian

$$\mathfrak{L}(u, v, \lambda) := \|v\|_1 + \frac{\mu}{2}\|Au - b - v\|_2^2 - \langle\lambda,\ Au - b - v\rangle \tag{2.2}$$

where λ is a Lagrange multiplier vector, and μ is a parameter chosen suitably. The dual functional is obtained by minimizing over u and v for fixed λ.

$$g(\lambda) = \operatorname*{argmin}_{u,v}\ \mathfrak{L}(u, v, \lambda) = \min_{u,v}\ \left\{\|v\|_1 + \frac{\mu}{2}\|Au - b - v\|_2^2 - \langle\lambda,\ Au - b - v\rangle\right\} \tag{2.3}$$

which upon completing the square gives:

$$g(\lambda) = \operatorname*{argmin}_{u,v}\ \mathfrak{L}(u, v, \lambda) = \min_{u,v}\ \left\{\|v\|_1 + \frac{\mu}{2}\|Au - b - v - \lambda/\mu\|_2^2 - \frac{1}{2\mu}\|\lambda\|_2^2\right\} \tag{2.4}$$

The minimization is performed in two steps:

1. $u^{n+1} = \underset{u}{\operatorname{argmin}}\ \mathfrak{L}(u, v^n, \lambda^n)$ while v^n is kept fixed.

2. $v^{n+1} = \underset{v}{\operatorname{argmin}}\ \mathfrak{L}(u^{n+1}, v, \lambda^n)$ while u^{n+1} is kept fixed.

Step 1 will be:

$$u^{n+1} = \underset{u}{\operatorname{argmin}}\ \left\{ \frac{\mu}{2} \| Au - b - v^n \ - \lambda^n/\mu \|_2^2 - \frac{1}{2\mu} \|\lambda^n\|_2^2 \right\} \tag{2.5}$$

Thus, to find u^{n+1}, we just solve the least square problem:

$$u^{n+1} = (A^T A)^{-1} A^T \left(b + v^n + \frac{\lambda^n}{\mu} \right) \tag{2.6}$$

Step 2 reads:

$$v^{n+1} = \underset{v}{\operatorname{argmin}}\ \left\{ \|v\|_{L^1} + \frac{\mu}{2} \| Au^{n+1} - b - v - \lambda^n/\mu \|^2 - \frac{1}{2\mu} \|\lambda^n\|^2 \right\} \tag{2.7}$$

Notice that the problem is separable in each coordinate. The minimizer is

$$v^{n+1} = S_{1/\mu}(Au^{n+1} - b - \lambda^n/\mu) \tag{2.8}$$

where

$$S_\delta(u) = \begin{cases} 0, & |u| < \delta \\ u - \operatorname{sign}(u)\delta & \text{otherwise} \end{cases} \tag{2.9}$$

is a shrinkage operator. Finally, for a given u and v, performing gradient descent method on the dual yields the update on λ:

$$\lambda^{n+1} = \lambda^n - \gamma\mu(Au^{n+1} - b - v^{n+1}) \tag{2.10}$$

The iterative scheme to solve the problem can be summarized as follows:

1: **input:** A, b, u, μ, λ, γ, ϵ

2: **while** $\|u^{n+1} - u^n\| > \epsilon$ **do**

3: keep v^n fixed, $u^{n+1} = (A^T A)^{-1}(b + v^n + \frac{\lambda^n}{\mu})$

4: keep u^{n+1} fixed, $v^{n+1} = S_{1/\mu}(Au^{n+1} - b - \frac{\lambda^n}{\mu})$

5: $\lambda^{n+1} = \lambda^n - \gamma\mu(Au^{n+1} - b - v^{n+1})$

6: **end while**

7: **output:** u;

Note that contrary to the Interior Point method, the Augmented Lagrangian method does not offer a stopping criterion based on the error (see line 4 of algorithm 19 and line 2 of algorithm 7).

2.3 Sub-gradient Method

This method is a generalization of the Steepest Descent method in convex optimization to non-differentiable convex functions. When the objective functional is differentiable, the Sub-gradient method uses the same search direction as the method of Steepest Descent for unconstrained problems. But when the objective functional is not differentiable, unlike the steepest Decent method the Sub-gradient method can provide a solution. This method is specially suitable for convex minimization problems with very large number of dimensions, because it requires little storage. For non-differentiable convex $f(u) = |Au - b|_1$, This method reads:

1: **input:** A, b, u

2: **while** $\boxed{\|u^{n+1} - u^n\| > \epsilon}$ **do**

3: $g^n = A^T sign(Au^n - b)$

4: $t^{n+1} = \frac{\delta}{n^\gamma}$

5: or

6: $t^{n+1} = \frac{f(u^n) - f*}{\|g^n\|_2{}^2}$

7: $u^{n+1} = u^n - t_{n+1} g^n$

8: **end while**

9: **output:** u;

where g is in fact sub-gradient of f. Note that similar to the Augmented Lagrangian method, this method does not provide a stopping criterion based on error.

It is also worth mentioning that the Interior Point method has a proof of polynomial time complexity when linear systems are solved exactly [6]. Although the Augmented Lagrangian method convergence has been established [7], in general one cannot truly predict the number of steps this method needs to converge. However, the lower iteration cost of the Augmented Lagrangian method compared to the Interior point method makes it much easier to use in practice.

2.4 Practical Considerations

Note the Interior Point method requires computation of inverse of $A^T DA$ (line 11 of 19) where D becomes ill-conditioned over time. To achieve a better performance, we avoid doing an exact solve, and use the Conjugate Gradient method to compute this inverse. We might also treat $A^T DA$ term in two different ways. One is to carry out the matrix-matrix products each time and explicitly form a new matrix $B := A^T DA$ and pass B to linear solver, or refrain from doing the multiplication and only use the matrix-vector product. Recall that implementation of the Conjugate Gradient method only require the matrix-vector product and not the whole system matrix. Nevertheless, in former setting, since we have access to matrix B entirely, we might use more complicated preconditioners based on B structure. In the latter setting, our choice of preconditioners is limited to whatever can be constructed with mere knowledge of A and D. In practice, the former setting proves to be more

efficient.

For an efficient implementation of the Augmented Lagrangian method, one does not need to use the Conjugate Gradient method to find an approximate solution; rather, one can form $B := A^t A$ once, and compute its LU decomposition $B = LU$. Then step 3 of the Augmented Lagrangian algorithm becomes two direct efficient solves of order $O(n^2)$.

Unlike the Interior Point and the Augmented Lagrangian methods, the Subgradient method does not involve any solution of linear system of equations, and it only requires matrix-vector multiplications. However, since this method requires a very large number of iterations to converge in our particular application (order of 10^5 or more) we were not able to effectively use it.

3. GLOBAL L^1-MINIMIZATION RESULTS

3.1 Correctness

In this section we present some basic tests that reproduce the results from [4] and expand them for the Augmented Lagrangian method.

3.1.1 Surface Reconstruction

This test case demonstrates reconstruction of a piecewise smooth surface based on point-wise data. Each point-wise value is associated with a Q-measurement of the surface at hand. The data is obtained from point values of the following function

$$u(x,y) = f(\max\{|x-1/2|, |y-1/2|\}), \qquad (x,y) \in \Omega := [0,1]^2, \tag{3.1}$$

where

$$f(r) = \begin{cases} 5/3 & r \in [0, 1/8] \\ 1 & r \in (1/8, 5/16] \\ 16(1/2-r)/3 & r \in (5/16, 1/2]. \end{cases} \tag{3.2}$$

Note that $u(x,y)$ is discontinuous at $\Gamma_1 = \{r = 1/8\}$ and its gradient has jumps across $\Gamma_2 = \{r = 5/16\}$ and $\Gamma_3 = (\{x = y\} \cup \{x + y = 1\}) \cap \{5/16 \le r \le 1/2\}$. The graph of u looks like an Aztec pyramid (see Figure 3.1). Our goal is to reconstruct a non-oscillatory approximation of u from point-wise values on a uniform Cartesian mesh composed of $1/h \times 1/h$ square cells.

Figure 3.1 shows a reconstruction for $1/h = 16$. Contrary to the L^2 least square reconstruction the result is smooth yet non-oscillatory.

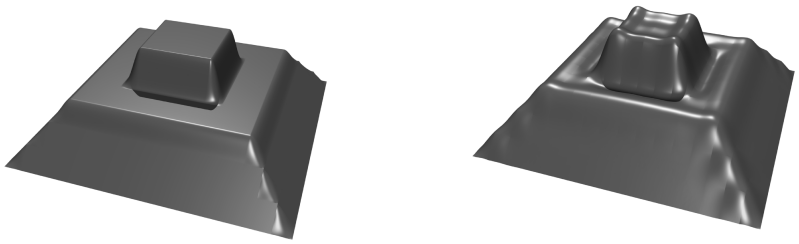

Figure 3.1: L^1 (left) and L^2 surface reconstruction (right) on a 16×16 mesh with Q-measurements, $\alpha = 3$, and $\beta = \infty$.

3.1.2 Super-resolution

In these series of test the global L^1 minimization algorithm is applied to images. The goal is to enhance the resolution of an under-resolved or aliased gray-scale images. Each pixel value in the given image is associated by an S-measurement, and the resulting constraints are chosen to be exactly satisfied.

Figure 3.2 shows results for the standard Lenna test image. We have down-sampled the 512×512 gray-scale original image to a 128×128 image by averaging 4×4 pixel blocks, and then the down-sampled image is reconstructed with the proposed L^1 minimization algorithm using the Interior Point and the Augmented Lagrangian methods on a uniform 128×128 mesh. Figure 3.3 shows how the global L^1 minimization algorithm performs on another test image. We are going to refer to this image as the peppers test image. The parameters for the Interior Point method are $\epsilon = 0.01$, and $\mu = 10$, and the parameters for the Augmented Lagrangian method are $\gamma = 1$, and $\mu = 10$. In each case the resulted images are compared to the standard bi-cubic reconstruction.

Figure 3.2: The Lenna test image. Original image (top left); down-sampled image (top right); standard bi-cubic reconstruction (middle left); global L^1-reconstruction using the Interior Point method (middle right); global L^1-reconstruction using 10 steps of the Augmented Lagrangian method (bottom left); and global L^1-reconstruction using the Augmented Lagrangian method reached to the same functional the Interior Point method does (bottom right).

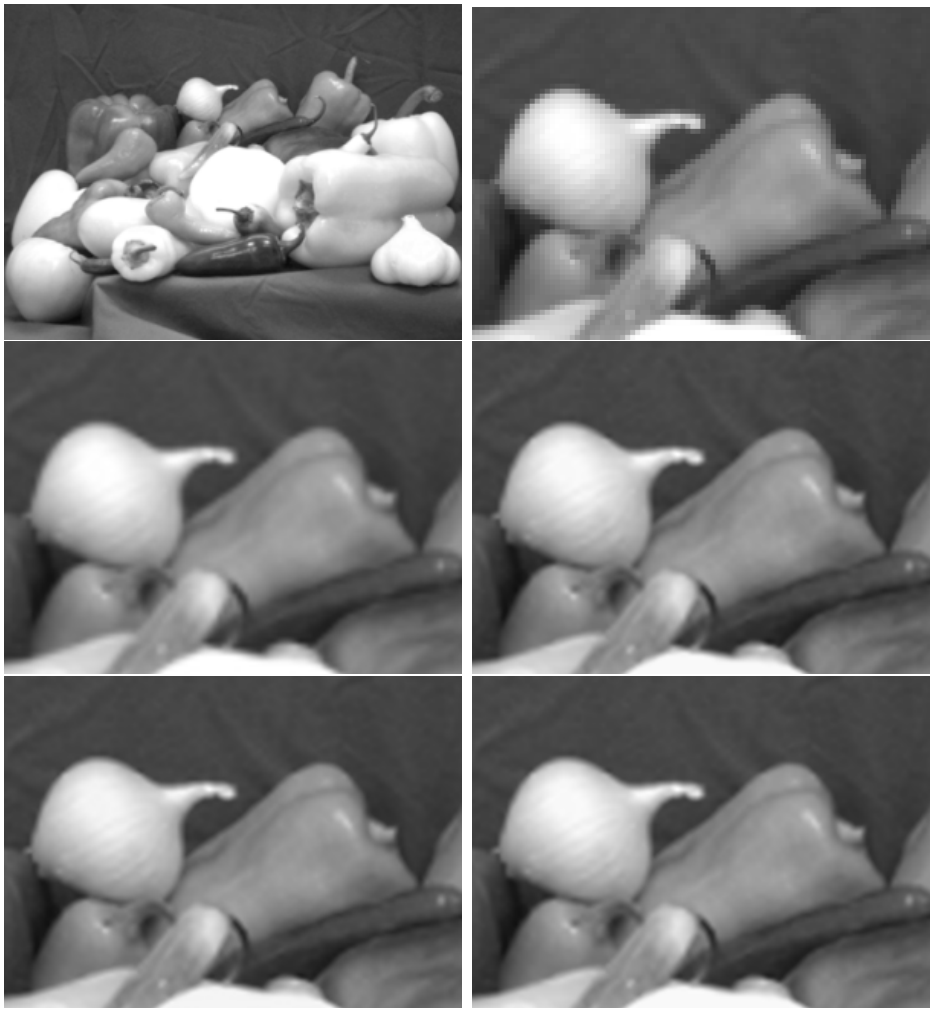

Figure 3.3: The peppers test image. Original image (top left and top right); standard bi-cubic reconstruction (middle left); global L^1-reconstruction using the Interior Point method (middle right); global L^1 reconstruction using 10 steps of the Augmented Lagrangian method (bottom left); global L^1-reconstruction using Augmented Lagrangian method reached to the same functional the Interior Point method does (bottom right).

As expected, the average constraints satisfaction error $\Delta({u_h}^*)$ (where ${u_h}^*$ is global minimizer) was zero for all the tests with interpolatory measurements.

3.2 Performance Analysis

Tables 3.1 and 3.2 the performance of the Interior Point and the Augmented Lagrangian methods are compared for the pyramid tests. As usual, the Interior Point method's parameters are $\epsilon = 0.01$, and $\mu = 10$, and the Augmented Lagrangian method's parameters are $\gamma = 1$, and $\mu = 10$. The initial solution in each method is chosen to be zero. Since the Augmented Lagrangian method does not provide a specific stopping criterion, the Augmented Lagrangian algorithm in each test case is iterated until the resulting functional matches to the Interior Point method. We observe that for this test case, the time Complexity for the Interior Point and Augmented Lagrangian methods are $\sim O(n)$ and $\sim O(n^{1.3})$ respectively.

$1/h$	CPU Time	$J(u_h)$	IP steps	CG steps
16	14.77	80.1941	16	7317
32	45.88	108.9296	15	6070
64	138.08	164.8261	16	6837
128	536.08	275.8485	17	7744
256	2481.13	497.3505	18	8662

Table 3.1: The Interior Point method performance for the pyramid test with Q-measurements, $\alpha = 3$, and $\beta = 5$

$1/h$	CPU Time	$J(u_h)$	AL steps
16	3.02	80.1507	111
32	6.89	108.8750	100
64	25.78	164.7916	103
128	160.74	275.8037	127

Table 3.2: The Augmented Lagrangian method performance for the pyramid test with Q-measurements, $\alpha = 3$, and $\beta = 5$

Another key observation is that depending on the accuracy required, the Augmented Lagrangian can ben faster. Table 3.3 compares performance of the two methods for the Lenna test image and different accuracy requirements controlled by tolerance ϵ defined in 19.

tolerance ϵ	0.5	0.2	0.1	0.01	0.001
Interior Point Method	57.18	136.12	207.46	734.32	3604.47
Augmented Lagrangian Method	60.74	75.04	92.94	278.77	1372.61

Table 3.3: Time required to reach the same functional controlled by ϵ for the Interior Point and the Augmented Lagrangian methods. Times are in seconds.

Figure 3.4 shows the functional $J(u_h)$ reached by the two L^1 minimization method for comparable CPU times in seconds for both the Lenna and the peppers test images. The very first step of the Augmented Lagrangian takes a relatively long time. This is due to the LU decomposition performed at the very beginning of this algorithm.

However, after the first step the Augmented Lagrangian method quickly catches on. After certain step the Augmented Lagrangian method is more accurate than the Interior Point method run for a similar time in both test cases. Note that here the Interior Point method reduces the L_1 norm at a relatively slow linear rate. On the other hand Augmented Lagrangian method reduces L_1 norm at an initial relatively fast rate but they slow down at fine tuning.

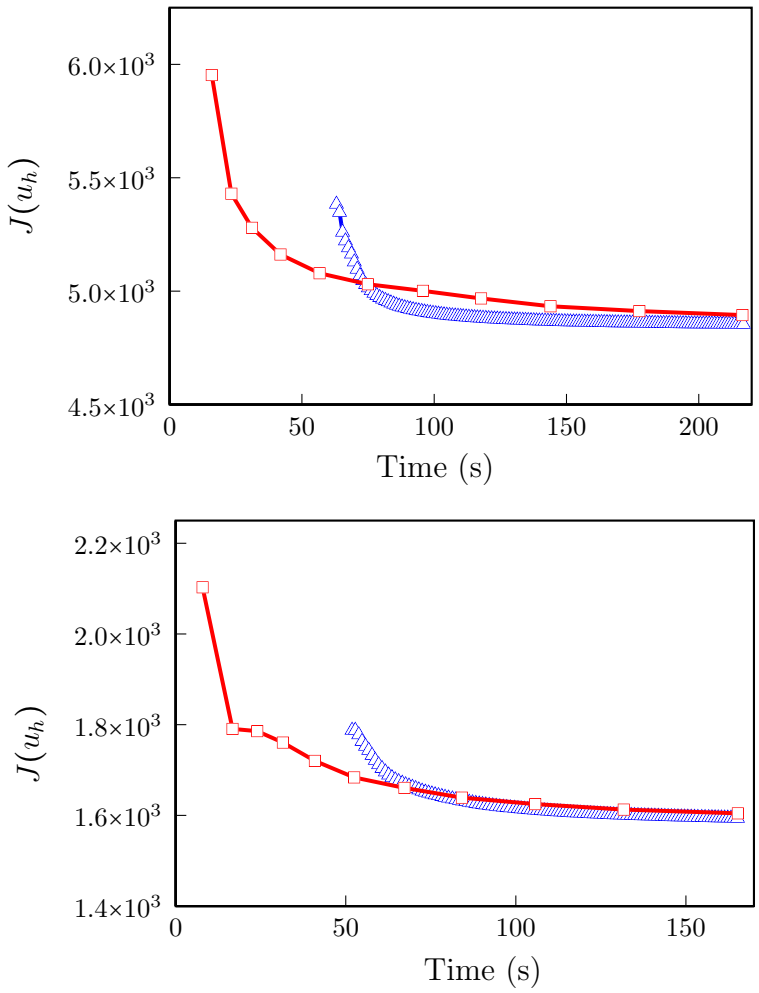

Figure 3.4: $J(u_h)$ versus elapsed CPU time for the Lenna (top) and the peppers (bottom) test images. Red squares and blue triangles represent the Interior point and the Augmented Lagrangian steps respectively. Consecutive squares and triangles represent achieved functional in each step of the Interior Point and the Augmented Lagrangian methods. Since the functional at the very first step of the Interior Point method is still large the plot is started from the second step for the Interior Point method in each case. Note that the very first step of the Augmented Lagrangian method is at 60 seconds mark because this method spends the first 60 seconds computing the LU decomposition.

4. LOCAL L^1-MINIMIZATION AND PARALLELIZATION

In the first chapter we suggested that performing L^1 minimization in parallel is pivotal to any effective use of our reconstruction method. There are two basic ideas to achieve this goal: First, one can solve the exact global problem in parallel. That is to store the system matrix in parallel, to perform parallel matrix-vector multiplication and dot product, and to utilize parallel solvers. This option is easy to implement, and many modern libraries are available (deal.II and Trilinos are two examples). However, this approach proves to need extensive communication and is not easy to make scalable. A second idea is to reduce the functional 1.3 by consecutive local minimizations rather than by one global minimization. This approach is partly motivated by the very nature of the functional 1.3 (see figure 4.1). There are two specific ways to do a local minimization: 1) Decrease the functional by local steps. This leads to a domain decomposition strategy. That is, instead of computing the minimizer over the whole domain at once, we divide the domain of interest into sub-domains and compute local L^1 minimizers in each sub-domain. This local L^1 minimizing technique is used in a Jacobi or Gauss-Seidel iterative scheme to reach the global minimizer. This option requires less communication than the first approach, is suitable to implement on GPUs, and is easier to make scalable. 2) Perform an approximate L^1 minimization in the sense that the functional is only decreased locally in each step. This means the overall functional might not necessarily decrease in each step, but certain local functionals defined on local sub-domains do. This option requires even less communication, and expected to be the most scalable.

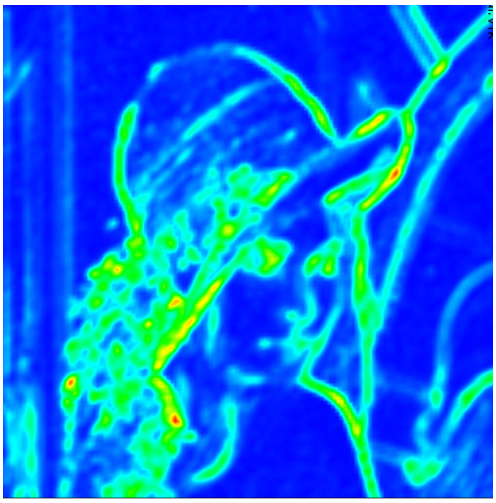

Figure 4.1: The functional 4.10 localization for the initial $\mathbb{Q}_3$ interpolation of the Lenna test image prior to any minimization. Warmer colors represent a larger value. Notice the localized nature of the functional.

Note that it is not at all clear that the global minimization problem introduced in previous section is equivalent to these domain specific local minimizations. However, we conjecture that for certain classes of surfaces u (possibly piecewise linear), the global and local minimization problems are equivalent. A formal statement of this conjecture is as follows. Consider the L^1 minimization problem defined in first chapter. We seek a $u \in X$ that reconstructs a surface based on a set of measurements. Consider the continuous approximation space ${X_h}^c$ where the original global minimization problem was defined on

$$ {X_h}^c = \{u \in \mathcal{C}^0(\overline{\Omega}) : u|_T \in \mathbb{Q}_3 \ \ \forall T \in \mathcal{T}_h\} \tag{4.1} $$

The solution space was defined

$$ {Y_h}^c \subset {X_h}^c = \{u \in {X_h}^c : d_i(u) = \varpi_i \quad \forall i\} \tag{4.2} $$

for the interpolation, and ${Y_h}^c = {X_h}^c$ for the relaxation case. Let $\mathcal{P}_1 : X \to X_h$ be a set valued mapping from u to the set of all minimizers of the global minimization problem. Define the discontinuous space ${X_h}^d$ as

$$ {X_h}^d = \{u \in W^{1,1}(\Omega) : u|_T \in \mathbb{Q}_3 \ \ \forall T \in \mathcal{T}_h\} \tag{4.3} $$

Also define discontinuous solution space as

$$ {Y_h}^d \subset {X_h}^d = \{u \in {X_h}^d : d_i(u) = \varpi_i \quad \forall i\} \tag{4.4} $$

for the interpolation, and ${Y_h}^d = {X_h}^d$ for the relaxation case. Now consider a new minimization problem in the discontinuous solution space Y_h.

$$ {u_h}^d = \text{argmin}_{v \in {Y_h}^d} J(v) \tag{4.5} $$

where J is the familiar functional defined in the first chapter. Let $\mathcal{P}_2 : X \to X_h$ be a set valued mapping from u to the set of all minimizers of this new minimization problem. Here, the minimization in discontinuous solution space represents any local minimization that might produce a solution ${u_h}^d$ with discontinuities between local domains. We conjecture that for a suitable u there exists an operator $\mathcal{O} : {X_h}^d \to {X_h}^c$, such that

$$ \|u - \mathcal{O}\mathcal{P}_2 u\|_{L^1} \leq C\|u - \mathcal{P}_1 u\|_{L^1} \tag{4.6} $$

where $C > 0$ is a constant, and the operator $\mathcal{O} : {X_h}^d \to {X_h}^c$ produces a continuous solution from discontinuous one by suitable averaging, etc (see diagram below).

$$
\begin{array}{ccc}
X & \xrightarrow{\mathcal{P}_1} & {X_h}^c \\
\downarrow \mathcal{P}_2 & & \\
{X_h}^d & \xrightarrow{\mathcal{O}} & {X_h}^c
\end{array}
$$

Finally, notice that it is not clear if certain stopping criterion in a local minimization algorithm is actually equivalent to the same stopping criterion for global problem. Since in practice one can hardly distinguish between consecutive iterations of the local minimization algorithm, we define convergence for the local algorithm as

$$\frac{J_h(u_h) - J_h(u_h^*)}{J_h(u_h^*)} < 0.01 \tag{4.7}$$

where u_h^* is a global minimizer.

4.1 Domain Decomposition

For sake of simplicity first assume the domain Ω is partitioned into two subdomains Ω_1 and Ω_2 (which consist of multiple cells), and a closed interface I is separating them (See figure 4.2).

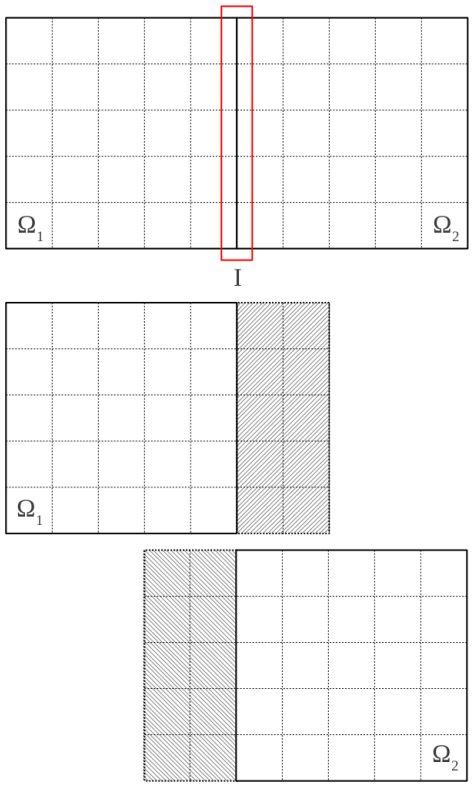

Figure 4.2: Decomposition into two domains and resulting sub-problems.

Let $\mathcal{T}_h^i$ be meshes on Ω_i composed of open quadrilaterals (or cells) for $i = 1, 2$. The set of interior edges of Ω_i are denoted by $\mathcal{F}_h^i$. Let $\{\phi_j\}_{j=1}^{\hat{n}}$ be a basis for X_h. Let $\{\phi_j^1\}$ and $\{\phi_j^2\}$ be respectively the basis elements in $\{\phi_i\}$ that are compactly supported in Ω_1 and Ω_2, and $\{\phi_j^I\}$ the elements that have a support in both Ω_1 and Ω_2. Then any $u_h \in X_h$ can be expressed as

$$u_h = \sum_{j=1}^{\hat{n}} x_j \phi_j = \sum x_j^1 \phi_j^1 + \sum x_j^2 \phi_j^2 + \sum x_j^I \phi_j^I = {u_h}^1 + {u_h}^2 + {u_h}^I. \tag{4.8}$$

where ${u_h}^1 := \sum x_j^1 \phi_j^1$, ${u_h}^2 := \sum x_j^2 \phi_j^2$, and ${u_h}^I := \sum x_j^I \phi_j^I$. In other words, we decomposed u_h into a part that is not zero only on Ω_1, one that is not zero only in Ω_2, and a part

on the interface of the domains (4.2). Then the functional (4.10) can be written as

$$J(u_h) = J_1(u_h^1 + u^I) + J_2(u_h^2 + u^I) + \int_I \left| [\![\partial_{\mathbf{n}} (u_h^1 + u_h^2 + u_h^I)]\!] \right|, \tag{4.9}$$

where

$$\begin{aligned} J_i(u_h^i + u^I) = \sum_{T \in \mathcal{T}_h^i} \int_T \left(|\partial_{xx}(u_h^i + u_h^I)| + 2|\partial_{xy}(u_h^i + u_h^I)| + |\partial_{yy}(u_h^i + u_h^I)| \right) \\ + \alpha \sum_{F \in \mathcal{F}_h^i} \int_F \left| [\![\partial_{\mathbf{n}} (u_h^i + u_h^I)]\!] \right|, i = 1, 2 \end{aligned}$$

Rather than minimizing the functional (1.3) globally, we try to find the best u_h^1 and u_h^2 in their respective domains as follows

1: Initialize u_h

2: **while** desired error reached **do**

3: Form u_h^1, u_h^2, and u_h^I from u_h

4: Fix u_h^2,

compute ${u_h^1}^* + {u_h^I}^* = \text{argmin}_{u_h^1, u_h^I} J_1(u_h^1 + u^I) + J_2(u_h^2 + u^I) + \int_I \left| [\![\partial_{\mathbf{n}} (u_h^1 + u_h^2 + u_h^I)]\!] \right|$

5: $u_h = {u_h^1}^* + u_h^2 + {u_h^I}^*$

6: Form u_h^1, u_h^2, and u_h^I from u_h

7: Fix u_h^1,

compute ${u_h^2}^{**} + {u_h^I}^{**} = \text{argmin}_{u_h^2, u_h^I} J_1(u_h^1 + u^I) + J_2(u_h^2 + u^I) + \int_I \left| [\![\partial_{\mathbf{n}} (u_h^1 + u_h^2 + u_h^I)]\!] \right|$

8: $u_h = u_h^1 + {u_h^2}^{**} + {u_h^I}^{**}$

9: **end while**

One may use the global algorithm developed in last section to perform step 4 and 7 in parallel. First consider step 4. Let K be the two cells layer in Ω_2 adjacent to the interface of two sub-domains Ω_1 and Ω_2 (See figure 4.2). Since we assume u_h^2 is

fixed in step 4, $(u_h^2 + u^I)$ which is computed on Ω_2, is fixed everywhere other than K, the two cell layer adjacent to the interface of two sub-domains. Thus there is no need to consider the whole Ω_2 to compute $J_2(u_h^2 + u^I)$ in step 4, and it suffices to only perform the minimization on the interface I as well as in K. There are going to be two set of measurement functionals involved in step 4. 1) A set of measurements inside Ω_1, and on I capturing the given data. 2) A set of Q-measurements imposing the u_h nodal values strictly on the ghost layer intended to make sure what minimized is the functional J. This set of measurements require the value of any node which is on the two cell layer K and not on I, to be set by a Q-measurement from u_h^2 (which is fixed during step 4). One can apply the global minimization algorithm to Ω_1 along with the ghost layer with these two set of measurements to find the minimizers $u_h^{1\,*}$ and $u_h^{I\,*}$.

A similar argument applies to step 7 and computation of $J_1(u_h^1 + u^I)$. Step 7 can be done by considering a two cells ghost layer, introducing two set of measurements, and applying the global minimization algorithm.

Note that in above example the sub-domains are visited by a Gauss-Seidel sweep to ensure the functional at each step is strictly smaller than the step before. In practice a red-black Gauss-Seidel or a Jacobi sweep may be used at the expense of loss of this property. Further, in case of interpolatory S-measurements, when the solution for a local minimization is used to update the global solution u_h, the end result is inevitably not going to satisfy the measurement constraints. Indeed, whenever a cell is updated such that it strictly satisfy an S-measurement the neighboring untouched cells are going to be altered and lose their interpolatory properties. Hence, unlike the global L^1 minimization, the interpolatory S-measurement are not going to be strictly satisfied by the end of each iteration of a local minimization. However, all the local minimization algorithms introduced in this thesis are able to decrease the average

constraints satisfaction error Δ introduced in the first chapter in each iteration. The choice of the initial solution does not matter either. The initial u_h in our method was the $\mathbb{Q}_3$ interpolation of the surface defined by given data. Since this particular choice does not satisfy S-measurements one might think that using an initial solution which is corrected to strictly satisfy the S-measurements might help. However, the domain decomposition algorithm was able to quickly decrease the rather large initial constraint satisfaction error of the initial $\mathbb{Q}_3$ interpolation to a small value.

Generalization of above algorithm to p domains is straightforward. One again needs to consider a two cell layer around each domain, and mark the points inside the layer that are not on the interface as measurement points. The measured value for these point should be communicated from neighboring domains.

The local minimization algorithms can be implemented in parallel. In a parallel setting the local sub-problems are solved concurrently on different processors, and the final result is communicated between the processors afterwards. To measure the performance of a parallel code the speed-up is defined to be $S_p = T/T_p$ where T is the best solution time the sequential code can achieve, and T_p is the time the parallel code with p processors requires to solve the problem.

4.1.1 Possible Unwanted Oscillations

The above construction is not always guaranteed to converge to the correct minimizer when interpolatory Q-measurements are used. To illustrate this point consider the following example on a one dimensional uniform mesh $\mathcal{T}_h$ on $[0,1]$. Let $X_h = \{u \in \mathcal{C}^0(\overline{\Omega}) : u|_T \in \mathbb{Q}_3 \ \forall T \in \mathcal{T}_h\}$. Define

$$J(u_h) = \sum_{T \in \mathcal{T}_h} \int_T |\partial_{xx} u_h| + \alpha \sum_{i=1}^{3} |[\![\partial_{\mathbf{n}} u_h]\!](x_i)|, \quad u_h \in X_h, \tag{4.10}$$

where $x_i = i/4$, $i = 0 \ldots 4$. Find function $u_h \in X_h$ such that $J(u_h)$ is minimized and $u_h(x_i) = f(x_i)$ where $f(x) := 1$ for $x > 1/2$ and $f(x) := 0$ for $x \leq 1/2$ (figure 4.3). Let $\{\phi_j\}_{j=0..12}$ be the nodal basis for the $\mathbb{Q}_3$ finite space. Then u_h can be expressed as $u_h = \sum_{j=1}^{13} a_j \phi_j$. We apply the domain decomposition algorithm to this problem by considering Q-measurements at each node and decomposing $\Omega = [0, 1]$ into 2 equal sub-domains Ω_1 and Ω_2, and setting the interface at $I = 1/2$. Consider step 4 of the domain decomposition algorithm where we perform the minimization at Ω_1 while we keep Ω_2 fixed. It is easy to see that for the minimizer $u_h^{1\,*} = \sum_{j=1}^{5} a_j^* \phi_j$, the nodal value a_5^* might be non-zero because the minimization algorithm is forced to make $u(x_2) = 0$, and consequently depending on α value it might choose to make the jump term small rather than the second derivative term.

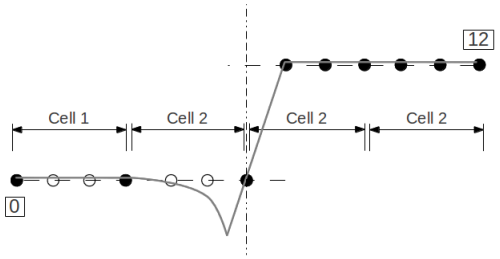

Figure 4.3: Application of the domain decomposition algorithm to one dimensional reconstruction on a mesh comprised of four cells and two sub-domains. This figure illustrates the minimization on Ω_1 while the solution is kept fixed in Ω_2. Solid points represent fixed nodal values, while hollow points represent the nodal values to be determined by the minimization algorithm. Degrees of freedoms are indexed from 0 to 12.

This simple example clearly shows how overshoots and undershoots can happen when interpolatory Q-measurements are used. These oscillations are an inherent property of above algorithm, and will not be removed by doing additional iterations. Rather, these oscillations are going to be communicated to neighboring sub-domains in the next iteration of the domain decomposition algorithm and make the algorithm

eventually unstable. However, a key observation is that these oscillations only occur when boundary of two domains meet a discontinuity, and they are located at the cell adjacent to the boundary. Thus instead of considering a layer of two cell wide K, it is reasonable to allow a larger K comprised of two parts, K_1 a layer of one cells in Ω_2 right next to I, and K_2 an outer layer of two cells (see figure 4.4). In K_2 every nodal value is fixed similar to the original domain decomposition algorithm, but in K_1 the degree of freedoms are not fixed and they are allowed to be determined by local minimization algorithm. Since oscillations will potentially happen in K_1, at each iteration we simply discard the computed solution in K_1 and replace it by values from Ω_2. Hence, the continuity condition is kept intact while unwanted oscillations are avoided.

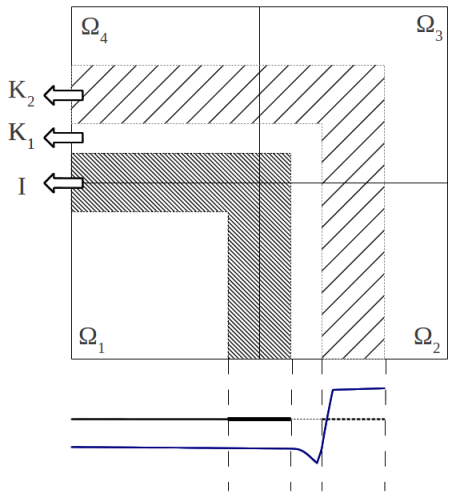

Figure 4.4: Application of the domain decomposition algorithm with four subdomains. Figure illustrates the minimization on Ω_1 while the solution is kept fixed in the other domains. Note the extended overlaps K_1, and K_2, as well as the interface I. A diagram shows where the jump might happens.

One might relax the continuity condition by prescribing a penalty β_{ghost} for interpolation on the ghost layer. This might also help to avoid unwanted oscillations at

boundary of two domains. When S-measurements are used this unwanted oscillation are not an issue. Our tests show that an optimal configuration for interpolatory S-measurements is I and K_2 to be 1 and 2 cells wide respectively, and K_1 is omitted.

4.2 Approximate Local L^1 Minimization

For sake of simplicity first consider a domain composed of only two sub-domains (which consist of very few or even only one cell). A closed interface I is separating the two sub-domains. There is also a layer of ghost cells enclosing the original domain which its values are copied from the exterior cells on the original domain. Denote the two sub-domains by Ω_1 and Ω_2. (See figure 4.5). Let $\{\phi_j\}_{j=1}^{\hat{n}}$ be a basis for X_h. Let $I(\Omega_i)$ the indices for basis functions with non-zero support on Ω_i, and $I_{adj}(\Omega_i)$ be the indices for basis functions with non-zero support on adjacent cells to Ω_i excluding those in $I(\Omega_i)$ for $i = 1, 2$. Then any $u_h \in X_h$ can be decomposed into two parts:

$$u_h = \sum_j x_j \phi_j = \sum_{j \in I(\Omega_i)} x_j \phi_j + \sum_{j \in I_{adj}(\Omega_i)} x_j \phi_j =: w^i + w^i_{adj} \tag{4.11}$$

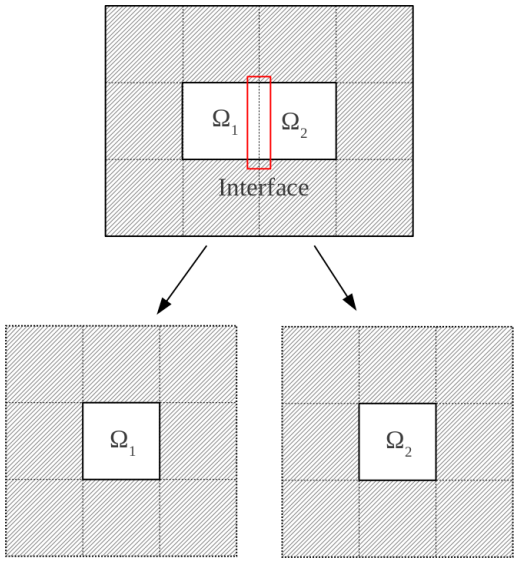

Figure 4.5: Approximate local L^1 minimization with two domains and resulting subproblems.

Let $\mathcal{T}_h^i$ be the set of meshes on Ω_i composed of open quadrilaterals (or cells) for $i = 1, 2$. The set of all edges of Ω_i (including exterior ones) is denoted by $\widetilde{\mathcal{F}}_h^i$. We Define a new approximate functional K_i for each domain Ω_i as

$$K_i(u_h) = \sum_{T \in \mathcal{T}_h^i} \int_T \left(|\partial_{xx} u_h| + 2|\partial_{xy} u_h| + |\partial_{yy} u_h|\right) + \alpha \sum_{F \in \widetilde{\mathcal{F}}_h^i} \int_F |[\![\partial_{\mathbf{n}} u_h]\!]_F|\,, \quad u_h \in X_h, \tag{4.12}$$

Now instead of minimizing the functional (1.3) globally, we seek to minimize each K_i individually as follows

1: Initialize $u_h = \sum_j x_j \phi_j$

2: **while** desired error reached **do**

3: Form $v_h^i := \sum_j y_j^i \phi_j$, by setting

$y_i^i = x_j$ for $j \in I(\Omega_i) \cup I_{adj}(\Omega_i)$

$y_j^i = 0$ otherwise.

4: Decompose v_h^i as $v_h^i = w^i + w_{adj}^i$

5: For $i = 1, 2$, fix w_{adj}^i,

compute $v_h^{i\,*} = \sum_j y_j^{i\,*} \phi_j = w^{i\,*} + w_{adj}^i = \text{argmin}_{w^i} K_i(w^i + w_{adj}^i)$

6: Form $u_h^* := \sum_j x_j^* \phi_j$ by setting

$x_j^* = y_j^{i\,*}$ for $j \in I(\Omega_i) \backslash I_{int}$ for $i = 1, 2$

$x_j^* = \frac{1}{2}(y_j^{1\,*} + y_j^{2\,*})$ for $j \in I_{int}$

$x_j^* = x_j$ otherwise.

7: **end while**

To perform step 5 in parallel one can use the global algorithm developed in last section for $i = 1, 2$. Similar to the Domain Decomposition algorithm there are going to be two set of measurement functionals involved. 1) A set of measurements inside Ω_i and on I capturing the given data. 2) A set of Q-measurements imposing the u_h nodal values on the ghost layer. The latter set of measurements require values of all nodes on the ghost layer to be set from w_{adj}^i. One can apply the global minimization algorithm to Ω_1 along with the ghost layer with these two set of measurements to find the minimizers $v_h^{i\,*}$ and update the minimizer u_h^* accordingly.

Since Ω_i's are small, and the Augmented Lagrangian method offers much better performance for smaller domains, a fixed number of the Augmented Lagrangian iterations can be used to solve the associated L^1 minimization problem at the expense of loss of some accuracy. Improvements in accuracy of resulting reconstruction can be deferred to further iterations of above algorithm to a great extent. However, there is a lower bound for the number of steps that can be used, and if one uses less, the approximate local algorithm will not produce an acceptable result.

Note that in above example the sub-domains are visited by a Jacobi sweep. This method does not guarantee that J is decreased at each step and it only decreases the approximate local K functionals. Moreover, although this algorithm is not going

to strictly satisfy the interpolatory S-measurements by end of each iteration, similar to the domain decomposing algorithm, it is capable of decreasing the constraints satisfaction error Δ.

Again, generalization of above algorithm to p domains (and p processors) is straightforward. One needs to consider a layer around each domain, and mark the points inside the layer that are not on the interface as measurement points.

Also notice that if the ghost layer is more than one cell wide, this algorithm is effectively a variation of the domain decomposition algorithm with small domains.

One can relax the interpolatory constraints at the ghost layer by prescribing a penalty β_{ghost} for interpolation. This might help the approximate local algorithm to achieve lower functionals. However, our numerical tests show this approach increases the constraints satisfaction error Δ significantly and slows down the algorithm. Furthermore, use of the different sweep (for example Gauss Seidel) can lead to end results with better functional and constraints satisfaction error.

5. LOCAL L^1 MINIMIZATION RESULTS

In this chapter we are going to consider the Lenna and the peppers test images, and the pyramid test introduced in chapter 3. For the Lenna test image, we again down-sample the 512×512 gray-scale original image to a 128×128 image by averaging 4×4 pixel blocks. The down-sampled image is reconstructed with the domain decomposition 4.1 and the approximate local 4.2 algorithms introduced in previous chapter on a uniform 128×128 mesh. For the peppers test image, the local reconstruction algorithms are used to enhance the resolution of a low resolution 128×128 image to a 512×512 one.

Two different methods are available to solve the local problems. The Interior point and the Augmented Lagrangian methods are both used and their performance is compared to the global algorithm. For the reference the global L^1 minimization algorithm was able to achieve $J(u_h^*) = 4848.47$ in 2207.1 seconds for the Lenna test image, and $J(u_h^*) = 1581.13$ in 1219.7 seconds for the pepper test image with $\epsilon = 0.01$, and $\mu = 10$. The domain decomposition algorithm should reach a functional as low as $(1 + 0.01)J(u_h^*)$ to meet the convergence criterion 4.7. It must effectively decrease the constraints satisfaction error too.

5.1 The Domain Decomposition Algorithm

5.1.1 Correctness

In these series of tests the given 128×128 image is decomposed into 64 subdomains of size 16×16 and each domain is solved in parallel in a Jacobi scheme. Similar to chapter 3 data is given as interpolatory S-measurements. I is two cells wide, $K1$ is zero cells wide, and $K2$ is two cells wide. For the Lenna test, five different experiments are conducted: 1) an Interior Point solver with $\epsilon = 0.01$, and $\mu = 10$; 2)

an Interior Point solver with $\epsilon = 0.05$, and $\mu = 10$; 3) an Augment Lagrangian solvers with $\gamma = 1$, and $\mu = 10$ run for 10; 4) an Augment Lagrangian solvers with $\gamma = 1$, and $\mu = 10$ run for 100 steps; 5) an Augment Lagrangian solvers with $\gamma = 1$, and $\mu = 10$ run for 500 steps. For the peppers test, the experiments 1 and 4 are reported.

Table 5.1 shows for the Lenna test the functional, the average constraints satisfaction error, and the accumulative time required by the parallel code after each iteration when the Interior Point solvers are used. These results show that the domain decomposition algorithm converges in sense of inequality 4.7 in the third iteration for local solves with $\epsilon = 0.01$, and in the fourth iteration for local solves with $\epsilon = 0.05$. Hence, the speed-up for 64 processors is $S_{64} = \frac{2207.1}{43.67} = 50.5$ and $S_{64} = \frac{2207.1}{19.86} = 111.1$ in the two cases respectively. In both cases, the domain decomposition algorithm is also able to reduce the constraints satisfaction error from roughly 7.4×10^{-3} for the initial $\mathbb{Q}_3$ interpolation to approximately 10^{-5}. These results reveal that one gains impressive speed-ups by fast but less accurate solution of local sub-problems in the domain decomposition algorithm. The lost accuracy can be compensated with a few more iterations. This also reconfirms that the same stopping criterion is not equivalent for local and global minimizations.

Note that these speed-up figures are conservative, as we obtain them by performing at least one more Jacobi iteration than one typically needs in practice (see figure 5.1).

		The Interior Point method with ϵ = 0.01		
		$J(u_h)$	$\Delta(u_h)$	Wall Time (s)
Iteration	1	5936.26	0.000176	11.05
	2	4928.40	0.000021	26.87
	3	4871.51	0.000009	43.67

		The Interior Point method with ϵ = 0.05		
		$J(u_h)$	$\Delta(u_h)$	Wall Time (s)
Iteration	1	5997.77	0.000180	3.67
	2	4953.41	0.000021	10.14
	3	4890.23	0.000008	14.64
	4	4878.04	0.000004	19.86

Table 5.1: Results of the domain decomposition algorithm for the Lenna test image using the Interior Point method. Boxes indicate converge in sense of 4.7.

Table 5.2 shows a similar result for the Augmented Lagrangian solvers. These results reveal that the domain decomposition algorithm does not converge in sense of 4.7 when the local Augmented Lagrangian solvers are not run for enough number of steps. Nonetheless, the results from even very few steps of local Augmented Lagrangian solutions might be pleasing enough to eye for most super-resolution applications (figure 5.1).

		10 steps of the Augment Lagrangian method		
		$J(u_h)$	$\Delta(u_h)$	Wall Time (s)
Iteration	2	5105.16	0.000022	6.75
	3	5060.54	0.000011	8.57
	10	5044.15	0.000008	17.95
	16	5042.31	0.000008	27.13

		100 steps of the Augment Lagrangian method		
		$J(u_h)$	$\Delta(u_h)$	Wall Time (s)
Iteration	2	4975.05	0.000024	10.65
	3	4932.73	0.000012	14.15
	10	4907.77	0.000004	38.31
	16	4906.15	0.000004	58.42

		500 steps of the Augment Lagrangian method		
		$J(u_h)$	$\Delta(u_h)$	Wall Time (s)
Iteration	2	4963.35	0.000024	20.48
	8	4898.4989	0.000006	73.13
	9	$\boxed{4895.6053}$	0.000005	80.08
	10	4886.43	0.000005	91.21

Table 5.2: Results of the domain decomposition algorithm for the Lenna test image using the Augment Lagrangian method.

Figure 5.1: The Lenna test image. Down-sampled image (top left); global L^1-reconstruction using the Interior Point method with $\epsilon = 0.01$ (top right); third Jacobi iteration of the domain decomposition algorithm using the Interior Point method with $\epsilon = 0.01$ (middle left); third Jacobi iteration of the domain decomposition algorithm using the Interior Point method with $\epsilon = 0.05$ (middle right); third Jacobi iteration of the domain decomposition algorithm using the Augmented Lagrangian method run for 10 steps (bottom left); third Jacobi iteration of the domain decomposition algorithm using the Augmented Lagrangian method run for 100 steps (bottom right).

Table 5.3 shows the functional and the accumulative time required by the parallel

code after each iteration for two set of experiments for the peppers test. These results reveal that while the domain decomposition algorithm always converges when the Interior Point method is used for local minimization, it might fail to converge when the Augmented Lagrangian method is used. In fact, our tests show that the number of steps the Augmented Lagrangian method is run in each sub-domain hardly matters. The domain decomposition algorithm fails to converge for this test even if the local Augmented Lagrangian solvers are run for 2000 steps. Yet again, the final image is hardly distinguishable from the result of the global minimization algorithm (See figure 5.2). The failure to converge in this case can be attributed to the imbalance between the accuracy of local Augmented Lagrangian solutions. The Augmented Lagrangian method does not provide a real stopping criterion based on the error, and running it for a fixed number of steps at each local domain might leads to vastly different degrees of accuracies in each domain. This possibly hinders the domain decomposition algorithm overall convergence. For this very reason we are going to conduct the performance analysis of the domain decomposition algorithm only using the Interior Point method.

The speed-up in this test is impressive too. The domain decomposition algorithm converges in sense of inequality 4.7 in the third iteration for local Interior Point solves with $\epsilon = 0.01$, and in the fourth iteration for local solves with $\epsilon = 0.05$. Thus the speed-ups for 64 processors are $S_{64} = \frac{1219.7}{31.16} = 39.1$ and $S_{64} = \frac{1219.7}{19.27} = 63.3$ in the two cases respectively. In both cases, the domain decomposition algorithm is also able to reduce the constraints satisfaction error from roughly 3.8×10^{-3} for the initial $\mathbb{Q}_3$ interpolation to approximately 10^{-6}.

		The Interior Point method with $\epsilon = 0.01$		
		$J(u_h)$	$\Delta(u_h)$	Wall Time (s)
Iteration	1	2175.73	0.000091	7.70
	2	1615.28	0.000009	18.62
	3	$\boxed{1590.49}$	0.000003	31.16

		The Interior Point method with $\epsilon = 0.05$		
		$J(u_h)$	$\Delta(u_h)$	Wall Time (s)
Iteration	1	2199.76	0.000093	3.34
	2	1625.83	0.000009	8.62
	3	1597.05	0.000003	13.91
	4	$\boxed{1592.02}$	0.000001	19.27

		100 steps of the Augment Lagrangian method		
		$J(u_h)$	$\Delta(u_h)$	Wall Time (s)
Iteration	2	1671.17	0.000011	9.94
	3	1754.43	0.000008	12.92
	10	1732.65	0.000004	31.55
	16	1731.82	0.000004	48.51

Table 5.3: The domain decomposition algorithm for the peppers test

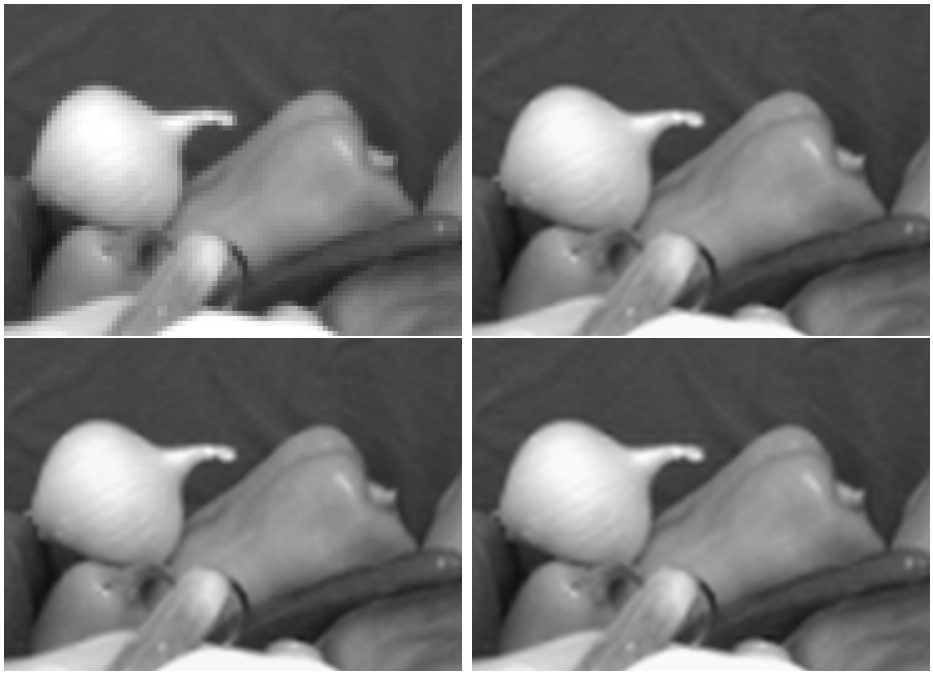

Figure 5.2: The peppers test. Original image (top left); global L^1 reconstruction using the Interior Point method with $\epsilon = 0.01$ (top right); third Jacobi iteration of the domain decomposition algorithm using the Interior Point method with $\epsilon = 0.05$ (bottom left); third Jacobi iteration of the domain decomposition algorithm using the Augmented Lagrangian method run for 100 steps (bottom right).

5.1.2 Parallel Code Performance Analysis

5.1.2.1 Experimental Setup

The parallel code was run on Brazos, a computing cluster at Texas A&M University. The computing power of Brazos comes from 172 computing nodes, each with two quad core Intel Xeon (Harpertown) or AMD Opteron (Shanghai) processors running at 2.5GHz with 16 to 32GB per node. Total peak performance is about 13.8 TFlops. There are a total of 300 nodes, 2400 cores, over 8TB of memory and a peak performances of 24 TFlop. The compute nodes and servers of Brazos are connected internally via a Hewlett Packard switch, with Gigabit Ethernet connections to each compute node and 10GbE connections to the login node and the data file-server.

There are some nodes connected via faster DDR infiniband too. Our test runs are limited to applications with $p < 8 \times 32 = 256$.

5.1.2.2 Scaling

To investigate the weak and strong scaling of the domain decomposition algorithm we consider the pyramid test introduced in the first chapter. The test is performed for mesh sizes $h = 1/64, 1/96, 1/128, 1/256$ and number of processors $p = 64, 144, 256$. Q-measurements are used, with $\alpha = 3$ $\beta = 5$, and $\beta_{ghost} = 4$. The Interior Point method with $\epsilon = 0.01$, and $\mu = 10$ is used to solve each local sub-domain. The speed-up is computed by comparing the run times with that of the global L^1 minimization using an interior point solver with identical parameters. This means the speed-up figures is very conservative, because one can always improve the speed-up by performing less accurate local minimizations and deferring better accuracy to more iterations. Table 5.4 shows speed-ups for different mesh sizes and number of processors. The parallel code is able to achieve the strong and the weak scalings for the pyramid surface reconstruction test.

		Mesh Size $1/h$			
		64	96	128	240
p	64	45.8	52.3	60.6	25.9
	144		57.9		79.1
	256	63.2			131.2

Table 5.4: Speed-up for different mesh sizes and number of processors.

The domain decomposition algorithm also scales strongly in case of the Lenna

test image (table 5.5).

p	Speed-up
16	14
64	48
256	150

Table 5.5: Speed-up for the Lenna test image for different number of processors.

5.2 The Approximate Local Algorithm

In these series of tests the approximate local algorithm 4.2 is applied to a 128×128 image in several different settings. For the Lenna test image, five different experiments are conducted: 1) 1×1 local domain with a one cell interpolatory ghost layer; 2) 1×1 local domain with a one cell relaxed ghost layer with $\beta_{ghost} = 4$; 3) 1×1 local domain with a 2 cells interpolatory ghost layer; 4) 2×2 local with a one cell interpolatory ghost layer. For brevity, only experiment 1 is conducted for the peppers test image.

Each case is solved using two different methods: an Interior Point solver with $\epsilon = 0.01$, and $\mu = 10$, and an Augment Lagrangian solver with $\gamma = 1$, and $\mu = 10$ run for 25 iterations. The Gauss-Seidel red-black sweep is used in all cases. The criterion for convergence is the same as the one used for the domain decomposition algorithm.

Table 5.6 shows the functional and the average constraints satisfaction error after each iteration for the Lenna test image for different configurations with a 1×1 local domain. Table 5.7 shows the same result when a larger 2×2 local domain is used. Figure 5.3 features some of the final images produced in experiments.

		1×1 local domain with a one cell interpolatory ghost layer			
		The Interior Point method		The Augmented Lagrangian method	
		$J(u_h)$	$\Delta(u_h)$	$J(u_h)$	$\Delta(u_h)$
Iteration	2	7864.57	0.001317	7980.37	0.002425
	4	5462.98	0.000219	5609.17	0.000144
	10	5104.90	0.000124	5273.51	0.000035
	20	5018.80	0.000110	5226.21	0.000024

		1×1 local domain with a one cell relaxed ghost layer			
		The Interior Point method		The Augmented Lagrangian method	
		$J(u_h)$	$\Delta(u_h)$	$J(u_h)$	$\Delta(u_h)$
Iteration	2	6143.16	0.004159	6431.09	0.003810
	10	4918.50	0.001075	4989.04	0.001556
	11	4877.85	0.001056	4971.47	0.001548

		1×1 local domain with a two cells interpolatory ghost layer			
		The Interior Point method		The Augmented Lagrangian method	
		$J(u_h)$	$\Delta(u_h)$	$J(u_h)$	$\Delta(u_h)$
Iteration	2	8023.88	0.001409	8120.94	0.001408
	4	5542.19	0.000335	5709.70	0.000373
	10	5178.71	0.000241	5320.12	0.000251
	20	5097.93	0.000231	5282.34	0.000244

Table 5.6: Results of the approximate local algorithm for the Lenna test image when 1×1 local domains are used.

		2×2 local domain with a one cell interpolatory ghost layer			
		The Interior Point method		The Augmented Lagrangian method	
		$J(u_h)$	$\Delta(u_h)$	$J(u_h)$	$\Delta(u_h)$
Iteration	2	11082.13	0.001263	11201.99	0.001256
	4	6509.28	0.000304	6664.91	0.000319
	10	5293.61	0.000081	5484.58	0.000104
	20	5108.69	0.000045	5325.78	0.000063

		2×2 local domain with a one cell relaxed ghost layer			
		The Interior Point method		The Augmented Lagrangian method	
		$J(u_h)$	$\Delta(u_h)$	$J(u_h)$	$\Delta(u_h)$
Iteration	2	7020.32	0.004415	7192.79	0.004274
	4	5422.19	0.002258	5564.18	0.002186
	10	4990.29	0.000808	5123.67	0.000999
	20	4897.80	0.000650	5075.55	0.000917

Table 5.7: Results of the approximate local algorithm for the Lenna test image when 2×2 local domains are used.

Figure 5.3: The Lenna test image. Down-sampled image (top left); global L^1-reconstruction using the Interior Point method with $\epsilon = 0.01$ (top right); fourth iteration of the approximate local algorithm for a 1×1 local domain with a one cell interpolatory ghost layer using an Augmented Lagrangian solver (middle left); tenth iteration of the approximate local algorithm for a 1×1 local domain with a one cell interpolatory ghost layer using an Augmented Lagrangian solver (middle right); tenth iteration of the approximate local algorithm for a 1×1 local domain with a one cell relaxed ghost layer using an Interior Point solver (bottom left); tenth iteration of the approximate local algorithm for a 1×1 local domain with a two cells interpolatory ghost layer using an Interior Point solver (bottom right).

Table 5.8 shows the functional and the average constraints satisfaction error after each iteration for a 1×1 local domains with an interpolatory one cell ghost layer for the peppers test image. Figure 5.4 features the final images produced in conducted tests.

		The Interior Point method		The Augmented Lagrangian method	
		$J(u_h)$	$\Delta(u_h)$	$J(u_h)$	$\Delta(u_h)$
Iteration	2	3399.18	0.000703	3473.61	0.000713
	4	1844.88	0.000052	1930.72	0.000062
	10	1673.93	0.000008	1811.96	0.000011
	20	1645.38	0.000003	1810.76	0.000009

Table 5.8: Results of the approximate local algorithm for the peppers test image. Here a 1×1 local domain with an interpolatory 1 cell ghost layer is used.

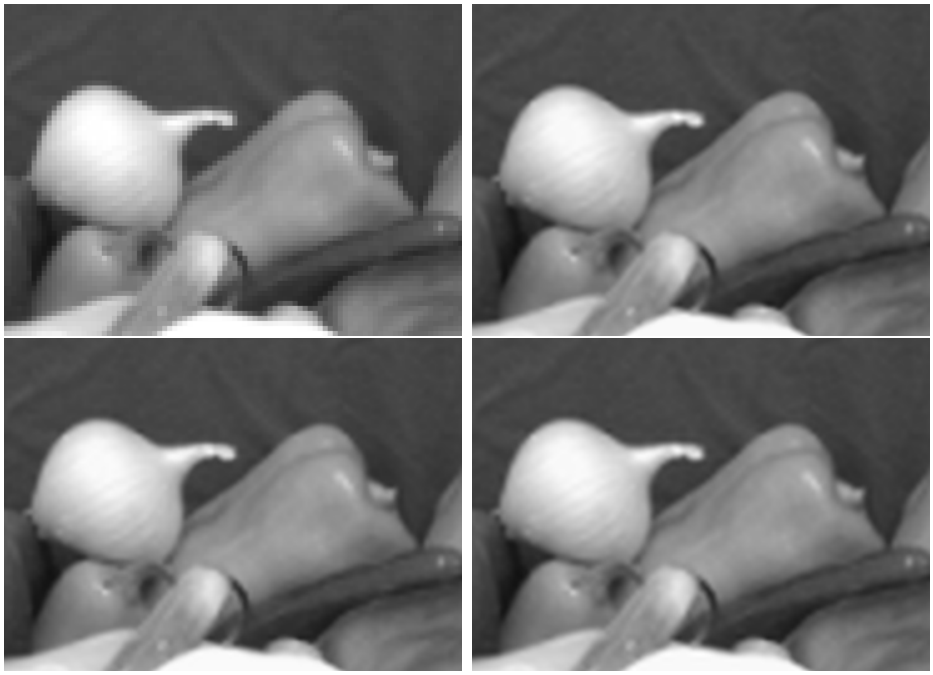

Figure 5.4: The peppers test image. Original image (top left); global L^1 reconstruction using the Interior Point method with $\epsilon = 0.01$ (top right); fourth iteration of the approximate local algorithm for a 1×1 local domain with a one cell interpolatory ghost layer using an Augmented Lagrangian solver (bottom left); tenth iteration of the approximate local algorithm for a 1×1 local domain with a one cell interpolatory ghost layer using an Augmented Lagrangian solver (bottom right).

These results reveal that the approximate local algorithm 4.2 (with smaller domains) is inherently not as accurate as the global algorithm. Not only is the approximate local algorithm not capable of finding a minimizer with a functional value as small as the global algorithm, but also it cannot converge in sense of inequality 4.7. However, for most images the resulting reconstruction is indistinguishable from the global one in the 'eye norm', and is visually pleasing even after only a few iterations. Furthermore, relaxing the ghost layer constraints might lead to a too large average constraints satisfaction error, and might be unfavorable. Finally, our tests show that the Augmented Lagrangian Method can successfully be used to solve the local minimization problems at the expense of hardly recognizable loss of accuracy.

5.3 The Aliasing Effect

We hypothesize that the inability of the approximate local algorithm to converge is due to "the aliasing effect". That is since the high frequency ingredients in the test images are of comparable length to the local domain size, the algorithm is not able to perform an effective reconstruction. To investigate this hypothesis, we consider a 256×256 gray-scale aliased image, and down-sample it to a 64×64 image by averaging 4×4 pixel blocks (see 5.5). The down-sampled image is reconstructed using all the algorithms proposed in this thesis and the final results are compared. Since in this section we are only interested in discerning how ultimately an algorithm can handle an aliased image, we use a more accurate setting for each test. The global L^1 minimization is performed using an Interior Point solver with $\epsilon = 0.01$, and $\mu = 10$. The domain decomposition and the approximate local algorithms use an Interior Point local solver with $\epsilon = 0.01$, and $\mu = 10$. For the domain decomposition algorithm I and K_2 are chosen to be 1 and 2 cells wide respectively, K_1 is omitted, and the original domain is decomposed into sixty four 8×8 sub-domains. The end result is reported after 4 and 20 iterations for the domain decomposition algorithm, and after 10 iteration for the approximate local algorithm. To see if the inability of the approximate local algorithm to converge is related to comparable sizes of image features and local domains the algorithm is run with different local domains sizes. Table 5.9 shows the functional and the average constraint satisfaction error after each iteration in all different cases. While the domain decomposition algorithm is able to ultimately reach a functional comparable to the global L^1 minimization the approximate local algorithm fails to converge in all different cases. Figure 5.5 compares the end result in some of the select cases. This result clearly shows that 1) the global minimization, the domain decomposition algorithm, and the approximate

local algorithm are all clear improvements compared to the standard bi-cubic reconstruction; 2) the domain decomposition algorithm end result is identical to the global minimization (at least when larger sub-domains are used, and the algorithm is run for enough number of iterations); 3) the approximate local algorithm is not able to offer an improvement comparable to the global minimization (specially when smaller local domains are used). Nonetheless, the fact that improvement of the end result is possible by enlarging the local domains indicate that the inability of this method to converge can be attributed to too small local domains.

Algorithm	Local Domain	Ghost Layer	$J(u_h)$	$\Delta(u_h)$
Global L^1 Minimization	-	-	4352.55	0.0
Domain Decomposition	8×8	3 cells	4390.81	0.000013
Approximate Local	1×1	1 cell ($\beta = \infty$)	4799.21	0.000832
Approximate Local	1×1	1 cell ($\beta = 4.0$)	4662.53	0.002770
Approximate Local	2×2	1 cell ($\beta = \infty$)	4898.16	0.000281
Approximate Local	4×4	1 cell ($\beta = \infty$)	4670.38	0.000227
Approximate Local	1×1	2 cells ($\beta = \infty$)	4886.98	0.000030
Approximate Local	2×2	2 cells ($\beta = \infty$)	5155.76	0.000637
Approximate Local	4×4	2 cells ($\beta = \infty$)	4798.62	0.000399

Table 5.9: Achieved functional and constraints satisfaction error of various algorithms for the aliased test image.

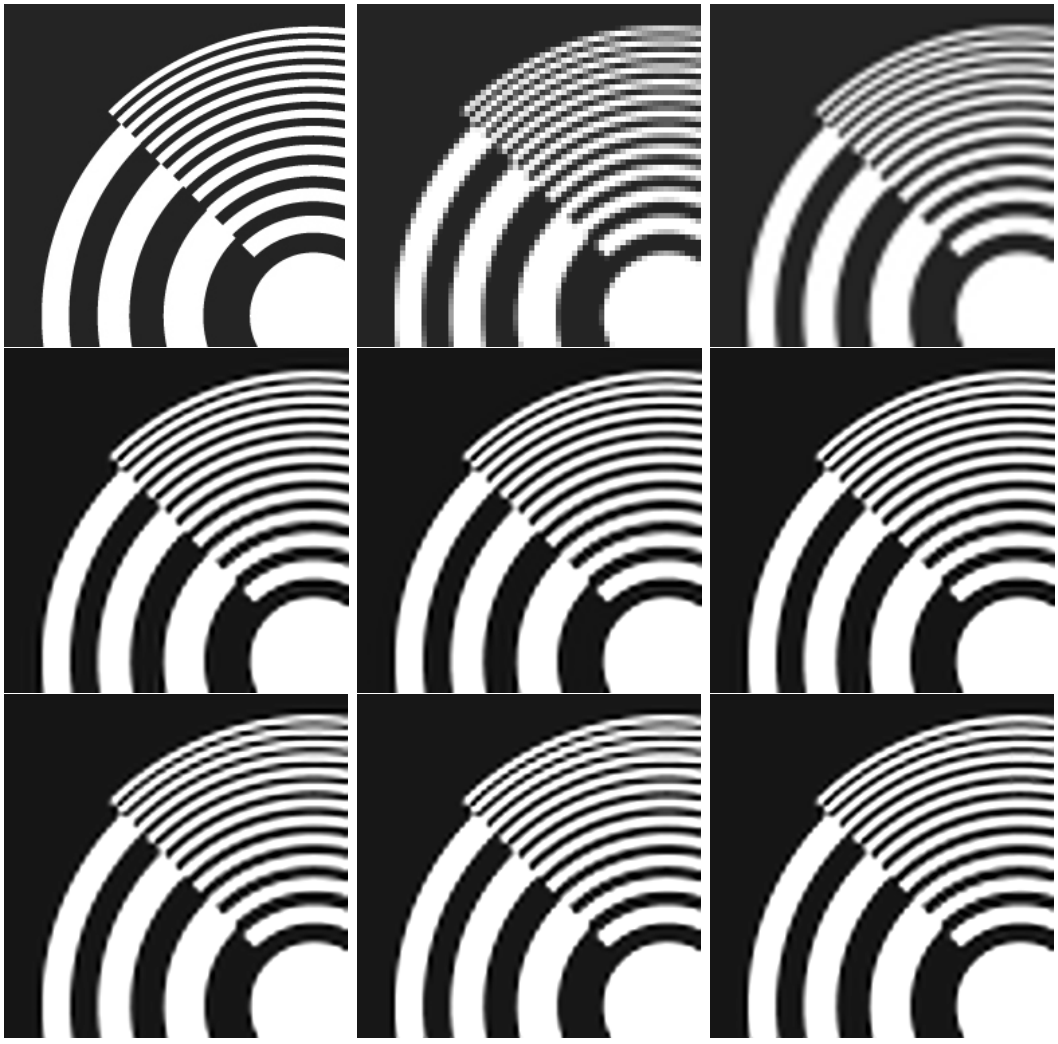

Figure 5.5: The aliased test image. The original image (top left); Down-sampled image (top center); standard bi-cubic reconstruction (top right); global L^1-reconstruction using the Interior Point method (middle left); fourth iteration of the domain decomposition algorithm (middle center); twentieth iteration of the domain decomposition algorithm (middle right); tenth iteration of the approximate local algorithm with a 1×1 local domain with an interpolatory one cell ghost layer (bottom left); tenth iteration of the approximate local algorithm with a 1×1 local domain with a relaxed one cell ghost layer (bottom center); tenth iteration of the approximate local algorithm with a 4×4 local domain with an interpolatory one cell ghost layer (bottom right).

6. CONCLUSION

In this thesis we sought to improve the performance of the global L^1 minimization algorithm in two fronts: First, local L^1-minimization; second, application of the Augmented Lagrangian method. We were able to show that 1) local solutions are faster on aggregate than solving the global problem; 2) the Augmented Lagrangian method can successfully be used to solve the local minimization problem; 3) only a few (in many cases just two or three) Jacobi or Gauss-Seidel iterations are enough to be close to the global L^1 minimizer for practical purposes; 4) while the domain decomposition algorithm (with larger domains) has a comparable accuracy to the global algorithm, the approximate local algorithm (with smaller domains) is inherently not as accurate as the global algorithm.

REFERENCES

[1] A. Chambolle and P.-L. Lions, Image recovery via total variation minimization and related problems, *Numer. Math.*, 76, pp. 167-188, 1997.

[2] J.E. Lavery Shape-preserving, first-derivative-based parametric and nonparametric cubic l^1 spline curves. *Computer Aided Geometric Design*, 23:3, pp. 276–296, 2006.

[3] S. Boyd and L. Vandenberghe Convex Optimization. Cambridge University Press, 2004, New York.

[4] B. Popov V. Dobrev, J.-.L. Guermond Surface reconstruction and image enhancement via l1-minimization. *SIAM J. Sci Comput.*, 32:3, pp. 1591–1616, 2010.

[5] J. Nocedal, and S.J. Wright Numerical Optimization (2nd ed.) Springer-Verlag, 2006, New York.

[6] R. D. C. Monteiro and I. Adler, Interior path following primal-dual algorithms. *I. Linear Programming, Math. Programming*, 44, pp. 2741, 1989.

[7] R.T. Rockafellar, The multiplier method of Hestenes and Powell applied to convex programming, *Journal of Optimization Theory and Applications*, 12, pp. 555-562, 1973.