The Chinese University of Hong Kong, Shenzhen

MAT3007

Midterm Project Report

Image Inpainting and Mosaics

Working Group: The Optimizers

Author:	*Student Number:*
Chen Tonglei	118010022
Wu Zhanxin	118010336
Cao Yuji	117010007

Each group member contributed equally in this midterm project

Contents

1 Part I - Total Variation Minimization **2**
1.1 Problem Reformulation . . . 2
1.2 Proof of Reformulation . . . 3
1.3 Result Comparison between Two Solvers (Dual-Simplex and Interior-Point) . . . 3
1.4 Result Comparison between Different Tolerance . . . 6
1.5 Conclusion . . . 7

2 Part I - Sparse Reconstruction **7**
2.1 Reformulation . . . 7
2.2 Dual Problem . . . 8
2.3 Analysis . . . 9
2.3.1 General Restruction . . . 9
2.3.2 Iteration Tolerance Influence . . . 10
2.3.3 δ Influence . . . 11
2.4 Denoising Setting . . . 13
2.5 Conclusion . . . 13

3 Part II - Generating Mosaics: **14**
3.1 Model Formulation . . . 14
3.2 Dual Problem . . . 16
3.3 Relationship between Mosaics of Grey-scale and Color Image . . . 17
3.4 Our Own Tiles and Possible Extensions . . . 18
3.5 Conclusion . . . 19

4 Appendix **20**
4.1 Part I - Total Variation Minimization . . . 20
4.2 Part I - Sparse Reconstruction . . . 21
4.2.1 General Reconstruction . . . 21
4.2.2 Tolerance Influence . . . 22
4.2.3 δ Influence . . . 22
4.2.4 Denoising Setting . . . 23

1 Part I - Total Variation Minimization

In this part, the method of total variation minimization is used to inpaint the damaged image. It minimizes the image gradient just around the damaged pixel. In this part, the performance of two solvers in linprog are compared. And the effect of Constraint Tolerance selection on the results is also studied.

To initialize D, the first $mn - m$ row is to calculate $\delta_1(X_{(i+1)j} - X_{ij})$, since the bottom most line of pixels do not have δ_1. the next $mn - n$ row is corresponding to $\delta_2(X_{i(j+1)} - X_{ij})$, since the right most line of pixels do not have δ_2. Meanwhile, the pixel X_{lk} corresponds to $D_{(l-1)m+k}$ and the optimal solution $x_{(l-1)m+k}$.

1.1 Problem Reformulation

$$\min \quad \|Dx\|_1 \quad \textbf{s.t.} \qquad A\mathbf{x} = \mathbf{b} \tag{1}$$

It can be reformulated to (to prove the reformulation, see section "Proof of reformulation"):

$$\begin{aligned} \min \quad & \mathbf{y} + \mathbf{z} \\ \textbf{s.t.} \quad & \mathbf{y} - \mathbf{z} = D\mathbf{x} \\ & A\mathbf{x} \quad = \mathbf{b} \end{aligned}$$

Add the constrains of x that $0 \leq x \leq 1$. It can be written as the following form.

$$\begin{aligned} \min \qquad & \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \mathbf{z} \\ \mathbf{x} \end{pmatrix} \\ \textbf{s.t.} \quad & \begin{pmatrix} I_\kappa & -I_\kappa & -D \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \mathbf{z} \\ \mathbf{x} \end{pmatrix} = \mathbf{0} \\ & \begin{pmatrix} \mathbf{0} & \mathbf{0} & -A \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \mathbf{z} \\ \mathbf{x} \end{pmatrix} = \mathbf{b} \\ & \mathbf{0} \leq \begin{pmatrix} \mathbf{y} \\ \mathbf{z} \\ \mathbf{x} \end{pmatrix} \leq \mathbf{1} \end{aligned}$$

where $\mathbf{y},\mathbf{z} \in R^\kappa$, I_κ is the identity matrix that $\kappa \times \kappa$, $\mathbf{0}$ is the matrix that all the element are 0.

The dual is

$$\begin{aligned} \min \quad & \begin{pmatrix} \mathbf{0} & b^T & \mathbf{1} \end{pmatrix} \begin{pmatrix} \mathbf{p} & \mathbf{b} \end{pmatrix} \\ \mathbf{s.t} \quad & \begin{pmatrix} I_\kappa & 0 \\ -I_\kappa & 0 \\ -D^T & -A^T \end{pmatrix} \mathbf{p} + I\mathbf{q} \leq \begin{pmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \end{pmatrix} \\ & \mathbf{p}\ free \\ & \mathbf{q} \leq \mathbf{0} \end{aligned}$$

1.2 Proof of Reformulation

Let $Dx = d$. For i-th row of matrix $\mathbf{y}, \mathbf{z}, \mathbf{d}$, it can be proved that:

$$\begin{aligned} \min \quad & y_i + z_i \\ \text{s.t.} \quad & y_i - z_i = d_i \\ & y_i \geq 0, z_i \geq 0 \end{aligned} \tag{2}$$

which is equivalent to $\min|Dx|$. Assume that the optimal solution is

$$s* = \begin{cases} y_0 > 0 \\ z_0 > 0 \end{cases}$$

However, at this circumstance, another solution can always be find that $y = y_0 - \alpha \geq 0, z = z_0 - \alpha \geq 0$ ($\alpha$is a small positive number) such that $y+z = y_0+z_0-2\alpha < y_0-z_0$, thus y_0 and z_0 must not be optimal solution. There exists contradiction. Therefore, in the optimal solution, either $y = 0$ or $z = 0$. Then the optimal function value will be d_i or $-d_i$, which equals to $|d_i|$.

In order to get $\|Dx\|_1$, multiple an identity matrix to $y + z$ as well as both side of $y - z = d_i$. What's more, d_i is δ_i or δ_2, which is the difference of neighbor two pixels ($-1 \leq d_i \leq 1$). Therefore, the constraints of y and z are $0 \leq y \leq 1$, $0 \leq z \leq 1$.

1.3 Result Comparison between Two Solvers (Dual-Simplex and Interior-Point)

The program linprog provides two solvers to solve the linear problems. Table 1 are the results tested in MATLAB by calculating results using two methods.(Duality gap here is the difference of optimal function value of dual and primal problem.)

Table 1: The Table of number of iterations, duality gap for different images and masks using TV-model. The Constraint Tolerance is $1e^{-3}$.

Image	Mask	Method	Iterations	Duality gap
buildings	random30	interior-point	9	3.32e-11
buildings	random30	dual-simplex	131317	0.0016
circles	random70	interior-point	16	1.4828e-06
circles	random70	dual-simplex	540420	0.0039
eagle	random70	interior-point	18	7.39e-07
eagle	random70	dual-simplex	823459	0.0039

It can be easily found that the results calculated using interior-point method have much smaller iterations and much smaller duality gap than the results calculated using dual-simplex method.

To show the comparison of PSNR and cpu-time using the two methods more clearly, more data(The raw data are in *Appendix*) are plotted in the line chart(Figure 1 and Figure 2). ("e" means image "eagle", "c" means image "circle", "b" means image "buildings".

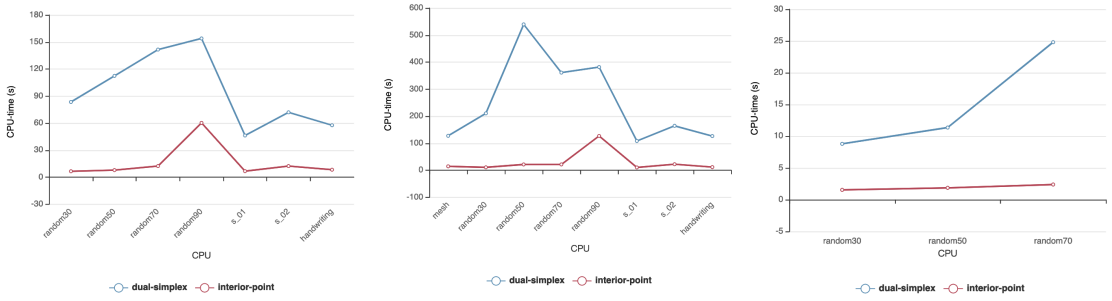

Figure 1: The line chart of cpu-time by using dual-simplex method (blue line) and interior-point method (red line). Left: The tested image is circles. Middle: The tested image is eagle. Right: The tested image is buildings.

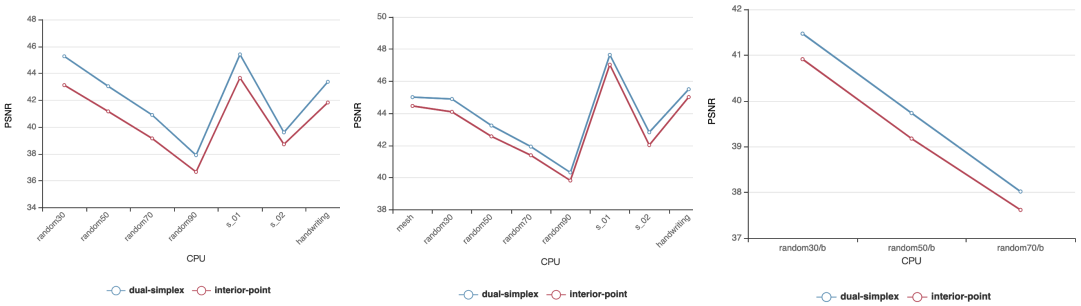

Figure 2: The line chart of PSNR by using dual-simplex method (blue line) and interior-point method (red line). Left: The tested image is circles. Middle: The tested image is eagle. Right: The tested image is buildings.

By observing Figure 1, it is obvious that in all tests, testing by using the interior-point method is faster than testing by using the dual-simplex method. Figure 2 shows that in all tests, the result using the dual-simplex method has lower PSNR than the result using the interior-point method.

In conclusion, in the settings of this problem, interior point method is better than dual-simplex.

Figure 3 and Figure 4 shows two examples of image recovering. The test image for Figure 3 is eagle, and the test image for Figure 4 is circles. The mask for both of the figures is random70. The constraint tolerance here is $1e^{-3}$.

Figure 3: Left: damaged image (grey-scale). Middle: Recovered image using the TV-model and a dual simplex method.(PSNR = 41.3945) Right: Recovered image using the TV-model and an interior point method.(PSNR = 41.9345)

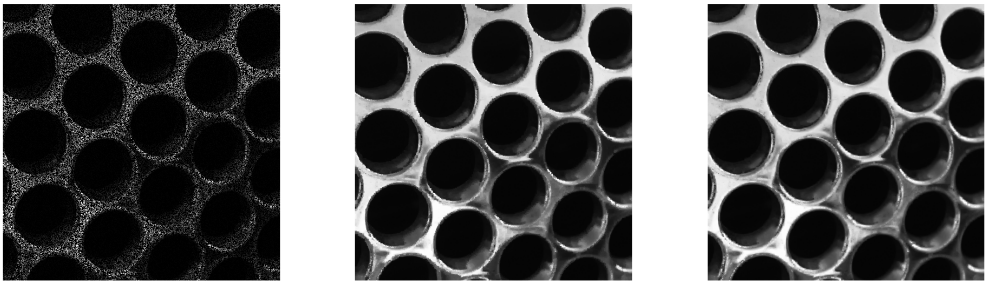

Figure 4: Left: damaged image (grey-scale). Middle: Recovered image using the TV-model and an dual simplex method(PSNR = 39.1564). Right: Recovered image using the TV-model and an interior point method(PSNR = 40.9017).

1.4 Result Comparison between Different Tolerance

Pick 512_512_circles.png and mask 512_512_random70.png as an example, different constraint tolerance were tested. The result is shown in Table 2.
Since when tolerance is $1e^{-4}$ or lower, and the interior-point method is used, the iterations exceed the MaxIterations and the time used is exceed 200 sec in average, these choices are not taken into consideration. What's more, the tolerance of $1e^{-2}$ and lower value is not valid for dual-simplex method. Therefore, it is also not taken into consideration.

From the table, it can be found that as long as the result can be get, the bigger the constraint tolerance is, the more cpu-time it needs to get the optimal solution. As for the number of iterations, there is no significant change in it. When using a dual-simplex method, setting the constraint tolerance to $1e^{-6}$, the iterations will increase a little. Meanwhile, the influence of PSNR is neglectable, which means that the influence of constraint tolerance to the quality of the recovered image is neglectable. For the interior-point method, if the constraint tolerance is too low, for example, less than or equal to $1e^{-4}$, the number of iterations will increase rapidly compared to the number of iterations that use bigger iterations.

In conclusion, a proper big constraint tolerance can increase the efficiency of calculating and keep the quality of calculation.

When it comes to the possible explanations of the findings above, the reason might be that the constraints and optimal function still stays the same. The change of tolerance only affect the stop condition, thus the optimal solution will not change. However, if the constraint tolerance is too small, it might be harder to decide the convergence, so the iteration will be harder to stop.

Table 2: The table of times(s), Constraint Tolerance, PSNR and iterations when testing different constraint tolerance using image 512_ 512_ circles.png and mask random70 by using TV-model

Time(s)	Tolerance	PSNR	iterations	method
12.393072	1.00e-03	40.9017	16	interior-point
11.903649	1.00e-02	40.9015	16	
11.758896	1.00e-01	40.9015	16	
142.383543	1.00e-06	39.1557	540462	dual-simplex
141.62164	1.00e-05	39.1564	540420	
138.133246	1.00e-04	39.1564	540420	
130.780332	1.00e-03	39.1564	540420	

1.5 Conclusion

After testing the model and analyzing the results, the Total Variation Minimization model has a good performance while recovering damaged images. It is obvious that the interior-point method can obtain higher quality images more quickly comparing to the dual-simplex method. The change of Constraint Tolerance will not change the PSNR of the recovered images.

2 Part I - Sparse Reconstruction

2.1 Reformulation

In this part, we focus on the ℓ_1-regularized image reconstruction problem. We use a block-wise discrete cosine transformation as sparse basis for the image and try to solve the inpainting task. This motivates the choice of the ℓ_1-norm in the model and therefore we can minimize the Ψ to make the pixels of undamaged parts of image u and the corresponding reconstruction close.

In this problem, we reformulate the model in two different ways. Since both of them have its own advantages in inpainting, we will show two different reformulations in this part and analyze their difference in section 2.3.

The reformulations are as follows:

Reformulation 1

$$\begin{aligned}
\min_{x,y,z\in\mathbb{R}^{mn}} \quad & \begin{pmatrix}\mathbf{1} & \mathbf{1} & \vec{0}\end{pmatrix}\begin{pmatrix}y\\z\\x\end{pmatrix} \\
\text{s.t.} \quad & \begin{pmatrix}I & -I & -\Psi\end{pmatrix}\begin{pmatrix}y\\z\\x\end{pmatrix} = \begin{pmatrix}\mathbf{0}\end{pmatrix} \\
& \begin{pmatrix}\mathbf{0} & \mathbf{0} & A\\ \mathbf{0} & \mathbf{0} & -A\end{pmatrix}\begin{pmatrix}y\\z\\x\end{pmatrix} \leq \begin{pmatrix}b+\delta\\ \delta-b\end{pmatrix} \\
& y, z \geq 0
\end{aligned}$$

Reformulation 2

$$\begin{aligned}
\min_{x,y,z\in\mathbb{R}^{mn}} \quad & \begin{pmatrix}\mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0}\end{pmatrix}\begin{pmatrix}y\\z\\x\\g\end{pmatrix} \\
\text{s.t.} \quad & \begin{pmatrix}I & I & -\Psi & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & -A & I\end{pmatrix}\begin{pmatrix}y\\z\\x\\g\end{pmatrix} = \begin{pmatrix}\mathbf{0}\\ \mathbf{0}\end{pmatrix} \\
& y, z \geq 0, \quad g \in [b-\delta, b+\delta]
\end{aligned}$$

2.2 Dual Problem

Then, we derive the associated dual of the linear optimization formulation.

$$\begin{aligned}
\max_{t_1,t_2,t_3\in\mathbb{R}^{mn}} \quad & \begin{pmatrix}\mathbf{0} & b+\delta & b-\delta\end{pmatrix}\begin{pmatrix}t_1\\t_2\\t_3\end{pmatrix} \\
\text{s.t.} \quad & \begin{pmatrix}I & \mathbf{0} & \mathbf{0}\end{pmatrix}\begin{pmatrix}t_1\\t_2\\t_3\end{pmatrix} \leq \begin{pmatrix}\mathbf{1}\end{pmatrix} \\
& \begin{pmatrix}-\Psi & A & -A\end{pmatrix}\begin{pmatrix}t_1\\t_2\\t_3\end{pmatrix} = \mathbf{0} \\
& t_2, t_3 \leq 0
\end{aligned}$$

From the constraint of dual problem, we can find that $A(t_2 - t_3) = \Psi t_1$. To simplify the dual problem, we use this equation to substitute t_1 and obtain the dual problem with less variables.

$$\begin{aligned}
\max_{t_2,t_3\in\mathbb{R}^{mn}} \quad & \begin{pmatrix}b+\delta & b-\delta\end{pmatrix}\begin{pmatrix}t_2\\t_3\end{pmatrix} \\
\text{s.t.} \quad & A(t_2 - t_3) \leq \Psi\mathbf{1} \\
& t_2, t_3 \leq 0
\end{aligned}$$

2.3 Analysis

Now, we try to solve the primal problem by using the interior-point solver. The following is the performance of the model.

2.3.1 General Restruction

In this part, we experiment on the two different reformulations (1 and 2) to solve the primal problem under the same circumstance. The results are shown in follows.

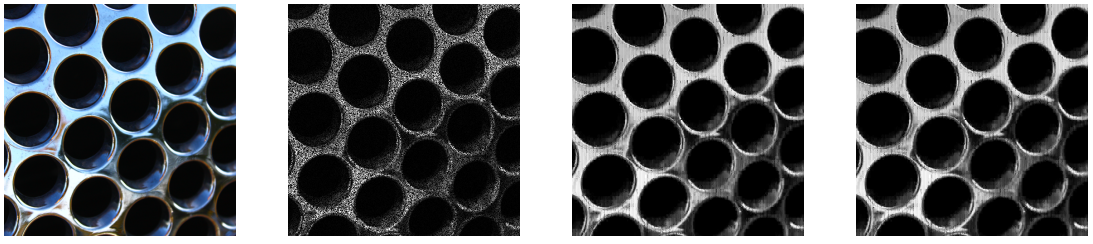

Figure 5: From left to right. First: original image. Second: damaged image (grey-scale). Third: Recovered image using reformulation (1). Forth: Recovered image using reformulation (2).

Table 3: The difference of PSNR, cpu-time and iteration between reformulation (1) and reformulation (2)

Method	PSNR	CPU time	iteration
Reformulation (1)	40.952	80.986402	22
Reformulation (2)	40.9518	72.689053	29

We can find both these two reformulations have good performance on damaged image with mask random 30, 50, 70. The reformulation (1) has a bit higher PSNR and performs better in inpainting, but its running time is longer than reformulation (2) (Refer to Appendix). However, when we consider a damaged image with mesh or handwriting mask, we can observe that these two reformulations perform much differently. The reformulation (1) can choose small δ and get high PSNR in around 120 seconds, while reformulation (2) has to take large δ, and takes over triple time to get solution. In conclusion, these two different reformulations have their own advantages and they both perform well generally. However, for different situation and requirements, we can choose the specific one to get better results.

In the following part, we will test the influence of different iteration tolerance and δ. Since reformulation (2) cannot handle small δ, we will use reformulation (1) to do the following tests.

2.3.2 Iteration Tolerance Influence

To find the better solution, we try to adjust the iteration tolerance.

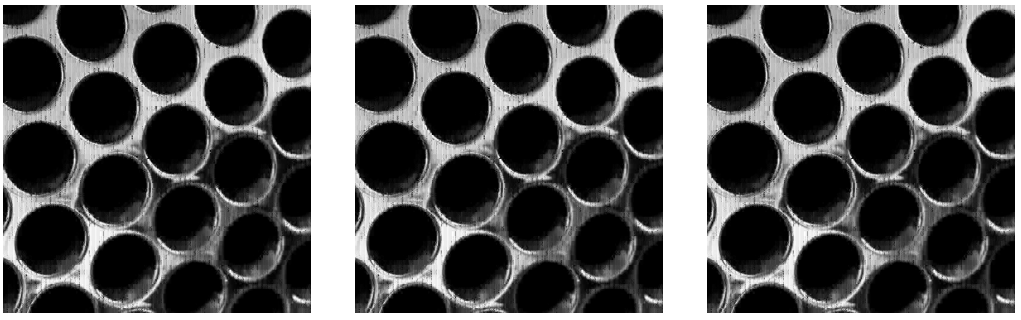

Figure 6: Left: Recovered image with tolerance e^{-2}. Middle: Recovered image with tolerance e^{-3}. Right: Recovered image with tolerance e^{-4}.

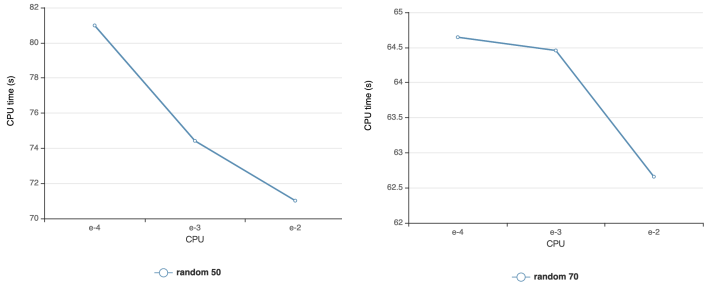

Figure 7: Left: The cpu-time for different iteration tolerance with mask random 50. Right: The cpu-time for different iteration tolerance with mask random 70.

We try different termination tolerance, e^{-2}, e^{-3}, e^{-4} and keep other parameters same. The PSNR does not change, which means the reconstructed image from these three tests are similar. For example, the PSNR for images in Figure 6 are all 40.9520. We can also observe that these three images are similar. However, the running time varied for different iteration tolerance. Generally, the running time and iteration times decreased when we lower termination tolerance. It improves the efficiency of the model without decreasing the accuracy.

Since the iteration tolerance is a threshold which, if crossed, stops the iterations of a solver. The iteration tolerance will not affect the objective function and constraints. Therefore, even if we change the iteration tolerance, we can still find the same optimal solution by more or less iterations. However, if the iteration tolerance is too small, a solver may fail to recognize when it has converged, and can continue futile iterations. In conclusion, to find an optimal solution with an acceptable running time, we need to adjust iteration tolerance to be suitable.

2.3.3 δ Influence

Now we consider the influence of different δ on the model performance.

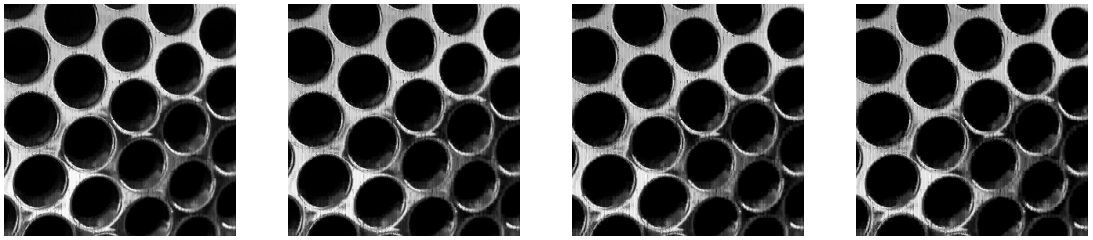

Figure 8: From left to right. First: Recovered image with δ 0.03. Second: Recovered image with δ 0.06. Third: Recovered image with δ 0.08. Forth: Recovered image with δ 0.10.

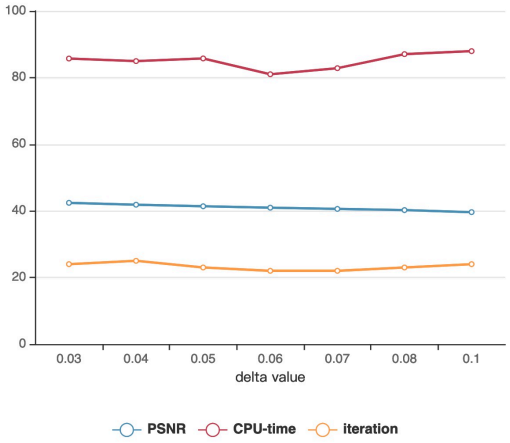

Figure 9: The PSNR, cpu time and iteration information with different delta value

To analyze the influence of δ, similarly, we keep other parameters same. In the above table, we can find that the PSNR decreased when we increased the δ value. However, the PSNR is all around 40 and the model is still considered as good performance. It implies that δ is vital in inpainting accuracy. The relationship between δ and PSNR can be explained by the model. Since δ represents the difference between original image and reconstructed image the model can accept, lower δ means that the model need to find better inpainting and then result in higher PSNR naturally.

For cpu-time and iteration, we cannot find the monotone relationship between the δ and cpu-time. When δ increase from 0.03 to 0.06, the cpu-time and iteration decreased. When δ increase from 0.06 to 0.1, the cpu-time and iteration increased. Therefore, We can find a suitable δ with low cpu-time and iteration. The δ can help save time and keep acceptable performance.

In the same situation, the Total Variation Minimization model obtains PSNR 43.038 by interior-point method and obtains PSNR 41.166 by dual-simplex method. If we keep reducing δ in the Sparse Reconstruction model, we find that we can get PSNR 42.7612 when using $\delta = 0.25$. Therefore, the Sparse Reconstruction model can reach better results than the Total Variation Minimization model.

In conclusion, δ is negatively related to PSNR and can affect cpu-time and iteration. Based on different requirements, we can test different δ values to find a suitable solution.

2.4 Denoising Setting

Now, we add Gaussian white noise to the image and test the denoising performance. In the following denoising tests, the setting is bsz = 8 and $\sigma = 0.075$, $\delta = 0.9\sigma$.

Table 4: The PSNR of Reconstructed Image and Noisy Image

Image	Reconstructed PSNR	Noisy PSNR	PSNR difference
circle	45.6249	48.9355	-3.3106
duck	44.2570	48.8855	-4.6285

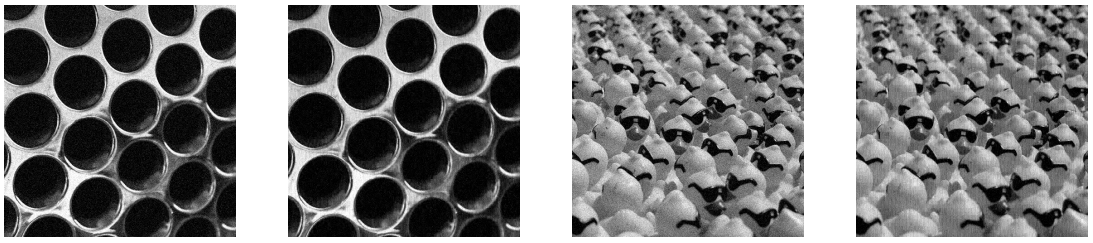

Figure 10: From left to right. First: Damaged circle image with Noise. Second: Reconstructed circle image. Third: Damaged duck image with Noise. Forth: Recovered duck image.

From the above table, we can find that the PSNR difference between reconstructed image and noisy image is around 3 - 5. Besides, the PSNR of noisy image in higher than the PSNR of reconstructed image. It is because that we try to minimize the l_1 norm in the denoising case, we can reduce the noise coming from b. However, though we reduce the noise in the reconstructed image and make it more natural than the image with noise, we lose some accurate points in the image with noise during inpainting. Therefore, the PSNR decreased when we reconstruct the image with noise. We also test denoising for different δ and σ values and the results is similar (Refer to Appendix).

2.5 Conclusion

After testing the model and analyzing the results, the Sparse Reconstruction model has a good performance in painting. When we decrease the iteration tolerance, the running time can be reduced and the PSNR will keep the same. For different δ value, the lower δ will lead to higher PSNR, but the running time will also be affected. When dealing with the noise, the sparse reconstruction model can reduce the noise and make the damaged image more natural. However, the reconstructed image will have lower PSNR than the image with noise and the difference is about 3 - 5.

3 Part II - Generating Mosaics:

In this part, we would generate mosaics of an image given a set of mosaic tiles via linear optimization approaches.

Assume we have a set of mosaics tiles of $\ell \times \ell$ and a given target image $U \in \mathbb{R}^{m \times n}$. To match the tiles and the image, we assume $\text{mod}(m, \ell) = \text{mod}(n, \ell) = 0$ and we partition the image into $\ell \times \ell$ blocks of pixels. Based on the average brightness of each block and the tile brightness values, where the average block brightness can be measured by taking the average of the pixel values.

In this way, there will be $\frac{m}{\ell} \cdot \frac{n}{\ell}$ blocks with brightness values $\beta_{i,j}$, $i = 1, \ldots, \frac{m}{\ell}$, $j = 1, \ldots, \frac{n}{\ell}$ for the image and r brightness values c_k, $k = 1, \ldots, r$ for each different tiles.

The objective here is to assign each of the mosaic tiles to a specific block of the given image so that we resemble the target image. To achieve this, we introduce the decision variable $x_{i,j,k}$ (contains the information if we place the k-th tile at block (i, j) or not) and its cost $(c_k - \beta_{i,j})^2$ for placing a tile k in block (i, j). During this approach, we have the following constraints:

- We exactly place one (of the r many) tiles in block (i, j).
- We exactly place $[\frac{m}{\ell} \cdot \frac{n}{\ell}]$ copies of each tile.
- Each optimization variable is a binary number $x_{i,j,k} \in \{0, 1\}$.

3.1 Model Formulation

$$
\begin{aligned}
\min_{x} \quad & \sum_i \sum_j \sum_k x_{i,j,k}(c_k - \beta_{i,j})^2 & \\
\text{s.t.} \quad & \sum_i \sum_j x_{i,j,k} = [\frac{m}{\ell} \cdot \frac{n}{\ell}]/r & \forall k \\
& \sum_k x_{i,j,k} = 1 & \forall i, j \\
& x \in \{0, 1\} &
\end{aligned}
$$

To further formulate this model clearer, here we provide the following notations:

- $B = \frac{m}{\ell}\frac{n}{\ell}$: representing the number of blocks.
- $C = \frac{B}{r}$: representing the number of copies of each block.

- $S = r \cdot B$: representing the vector size of decision variable $x_{i,j,k}$.
- $A_1 \in \mathbb{R}^{r \times S}$: selection matrix for constraint 1.
- $A_2 \in \mathbb{R}^{B \times S}$: selection matrix for constraint 2.
- $s = \text{vec}((c_k - \beta_{i,j})^2) \in \mathbb{R}^S$: cost coefficient vector of x.

With the notations above and x being relaxed to a continuous interval $[0, 1]$, a simpler formulation is constructed:

$$\begin{aligned} \min_{x \in \mathbb{R}^S} \quad & s^T x \\ \text{s.t.} \quad & \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} x = \begin{pmatrix} C\mathbf{1} \\ \mathbf{1} \end{pmatrix} \\ & Ix \leq \mathbf{1} \\ & x \geq 0 \end{aligned} \tag{3}$$

In this way, we implement the MATLAB program to solve the problem and the results are as follows.

Table 5: Mosaics Primal Problem Results

Image	Tiles	Block size	Iteration	CPU time (s)	Result	Degenerate
640_640_tiger	circles 1 - 8	10×10	4929	0.317909	TRUE	TRUE
640_640_tiger	circles 1 - 8	20×20	1279	0.063962	TRUE	TRUE
640_640_tiger	circles 1 - 8	40×40	340	0.034442	TRUE	TRUE
640_640_tiger	symbols 1 - 8	10×10	6606	0.466921	TRUE	TRUE
640_640_tiger	symbols 1 - 8	20×20	1767	0.102302	TRUE	TRUE
640_640_tiger	symbols 1 - 8	40×40	500	0.055816	TRUE	TRUE
1024_1024_stones	circles 1 - 8	8×8	26985	4.780901	TRUE	TRUE
1024_1024_stones	circles 1 - 8	16×6	6837	0.482426	TRUE	TRUE
1024_1024_stones	circles 1 - 8	32×32	1829	0.098641	TRUE	TRUE
1024_1024_stones	symbols 1 - 8	8×8	27134	4.049609	TRUE	TRUE
1024_1024_stones	symbols 1 - 8	16×6	6841	0.428393	TRUE	TRUE
1024_1024_stones	symbols 1 - 8	32×32	1809	0.084421	TRUE	TRUE

In the table above, the TRUE in Result means the solution of relaxed problem is the same as the solution of the original integer problem. From the result, we know that the solution is degenerate and with the increasing of block size, the cpu-time and the number of iterations decreases.

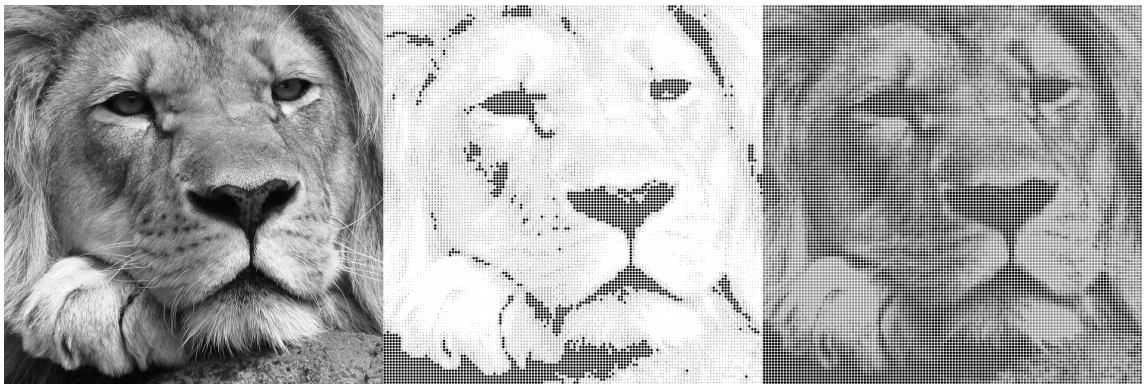

Figure 11: Generated Mosaics of 640_640_lion.png using tiles of symbols and circles with block size 5×5

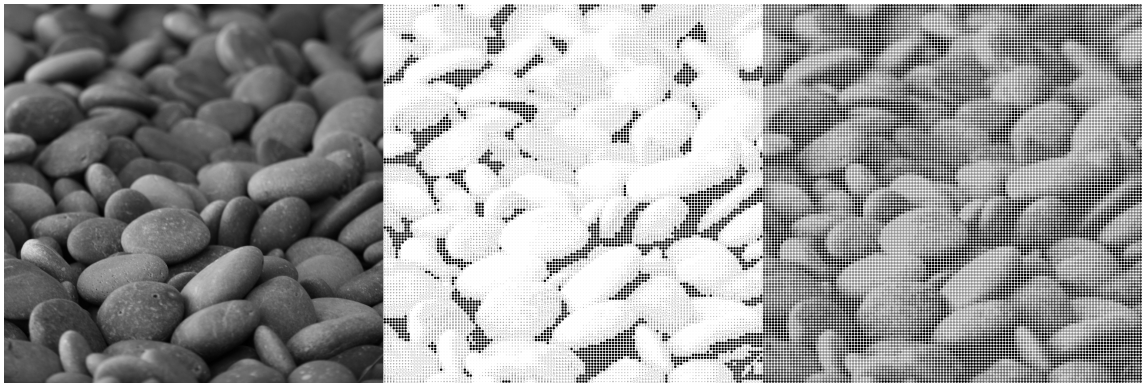

Figure 12: Generated Mosaics of 1024_1024_lion.png using tiles of symbols and circles with block size 8×8

3.2 Dual Problem

Based on the primal reformulation in Equation 3, we derive the dual problem as follows:

$$\begin{aligned} \max_{y \in \mathbb{R}^{r+B}} & \begin{pmatrix} b^T & \mathbf{1}^T \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \\ \text{s.t. } & \begin{pmatrix} A^T & I \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \leq s \\ & y_1 \text{ free} \\ & y_2 \leq 0 \end{aligned} \tag{4}$$

Using the dual problem, we solve the primal problem and compare their required cpu-time and number of iterations with the results of the primal problem. The results are demonstrated in the table below.

Table 6: Comparison results of primal and dual problem

	Block size (5)	Block size (10)	Block size (20)
CPU time (Primal)	4.431769	0.418587	0.167892
CPU time (Dual)	120.35825	11.172967	0.765756
Iterations (Primal)	26250	6606	1767
Iterations (Dual)	47792	23737	5899
Duality gap	1.05×10^{-11}	6.82×10^{-13}	2.56×10^{-13}

From the table, we can see that the cpu-time of the dual problem is far more than the primal. Since the dual problem add one variable to each constraint in primal problem, in dual problem, there are more constraints than the primal problem and this may result in the slow optimization process of the dual problem. Also, from the table we can see that the duality gap between the primal and dual is rather small. Therefore, when there are plenty of constraints in the formulation, solving its dual problem may save time.

3.3 Relationship between Mosaics of Grey-scale and Color Image

After we find the mosaic of grey-scale version of a color image U, we want to explore whether the mosaic is also optimal for the color image when we include the color information in the calculation of the brightness coefficients $\beta_{i,j}$ of U, i.e., $\beta_{i,j}$ is now computed as

$$\beta_{i,j} = \frac{1}{3} \times \text{average in } \textbf{red} \text{ channel} + \frac{1}{3} \times \text{average in } \textbf{green} \text{ channel} + \frac{1}{3} \times \text{average in } \textbf{blue} \text{ channel.}$$

To investigate whether the found mosaic is still optimal for this new objective function, we refer to the knowledge of sensitivity analysis on the lectures. Basically, by using the new RGB computation method, each component of c (cost in the objective function) changes on the objective function of a color image U and its grey-scale version. For the component of c in the basis, we have

$$c_N^T - (c_B^T + \lambda e_j^T) A_B^{-1} A_N = c_N^T (c_B^T - \lambda e_j^T) A_B^{-1} A_N$$

Since c and all component in A is nonnegative, if some one component of c changes with $c_B^T - \lambda e_j^T \leq 0$, the reduced cost will not be positive. From the test, this situation indeed appears in some component as in Figure 13. Therefore, the optimal solution will change. To validate our investigation result, we tested the solution results on different figures including "640_640_lion", "640_640_jellyfish" and "640_640_rainbow", the results coincides with the investigation.

2555	0.0078
2556	0.0031
2557	0.0044
2558	-3.0475e-04
2559	5.9940e-04
2560	0.0041
2561	-0.0423
2562	-0.0451
2563	-0.0376
2564	-0.0225
2565	-0.0180
2566	-0.0258
2567	-0.0225
2568	-0.0199

Figure 13: Some coefficient difference between two RGB computation methods

3.4 Our Own Tiles and Possible Extensions

Based on this brightness substitution method, we generate my own tiles in "own_tiles" folder and take an experiment wit the problem setup. The result is illustrated below. Also, the possible extension about this problem can be using thousands of

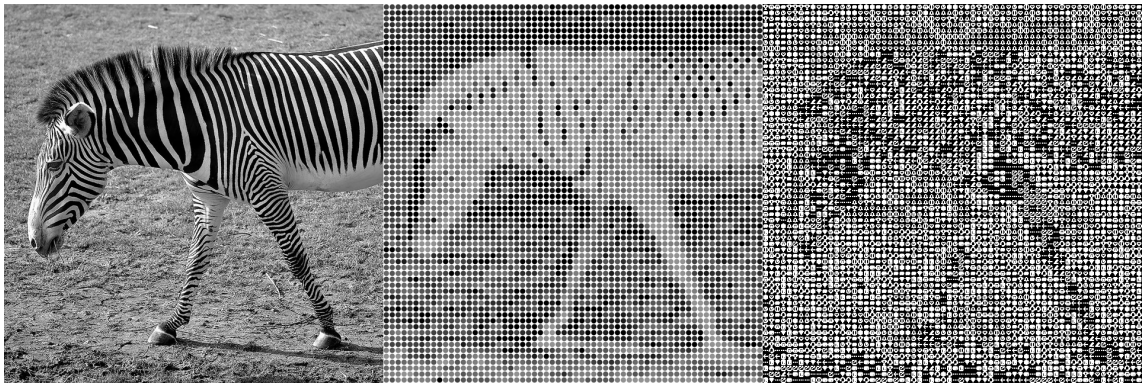

Figure 14: Generated Mosaics of 640_640_zebra.png using tiles of circles and own tiles with block size 10×10

sub-figures to resemble a large figure. For example, the figure below shows that we can use small tiles to resemble a drawing.

Figure 15: A mosaic drawing of Vincent van Gogh: *The Starry Night*

3.5 Conclusion

In this problem, we use brightness as our criterion to substitute the original image block with a mosaic tile. To place more accurately, we formulate this problem into an linear program and derive its primal and dual problem. In the results, an interesting observation is the cpu-time of dual is much larger than primal which may due to the excessive constraints in the problem. Also, we find that the mosaic substitution of grey-scale image is closely related to the transformation method from color to grey image. Different transformation may result in different solution of placement of mosaic tiles.

To extend this problem, we generate our own tiles to experiment. Additionally, this mosaic generating method can contribute to generate mosaics for larger images from smaller pictures and using tiles of different styles from the original can make the generated mosaic image more interesting.

4 Appendix

4.1 Part I - Total Variation Minimization

The cpu-time(Time in table), PSNR and the number of iterations of testing different masks and images are shown in Table N. The constraints tolerance is all $1e^{-3}$ here. All the tests in this table are done on MacBook Pro (13-inch, 2018, Four Thunderbolt 3 Ports), 2.3 GHz Quad-Core Intel Core i5 processor, 8 GB 2133 MHz LPDDR3 memory. Matlab version: MATLAB R2020a. However, the CPU temperature may influenced the cpu-time.

Table 7: The Table of cpu-time, PSNR, number of iterations for different images and masks. The Constraint Tolerance is $1e^{-3}$.

original image	mask	method	Time(s)	PSNR	iterations
circle	random30	interior-point	6.624152	45.2674	12
	random30	dual-simplex	83.508978	43.122	508040
	random50	interior-point	7.961538	43.038	14
	random50	dual-simplex	112.459355	41.166	524948
	random70	interior-point	12.393072	40.9017	16
	random70	dual-simplex	141.62164	39.1564	540420
	random90	interior-point	60.337255	37.9095	25
	random90	dual-simplex	153.974898	36.6623	553852
	s_01	interior-point	6.797567	45.406	15
	s_01	dual-simplex	46.346266	43.6575	495878
	s_02	interior-point	12.383274	39.6042	25
	s_02	dual-simplex	72.034518	38.7138	508452
	handwriting	interior-point	8.387533	43.3667	16
	handwriting	dual-simplex	57.679825	41.8269	500994
eagle	mesh	interior-point	14.874768	45.0154	19
	mesh	dual-simplex	127.570606	44.4666	782848
	random30	interior-point	11.12439	44.8996	13
	random30	dual-simplex	210.605496	44.1001	792724
	random50	interior-point	21.771421	43.2455	15
	random50	dual-simplex	539.484611	42.5718	808484
	random70	interior-point	21.853255	41.9345	18
	random70	dual-simplex	361.210292	41.3945	823459
	random90	interior-point	127.141384	40.3268	27
	random90	dual-simplex	381.740811	39.8226	840108
	s_01	interior-point	10.767516	47.6484	13
	s_01	dual-simplex	108.546431	47.0346	777545
	s_02	interior-point	22.63055	42.8257	28

Table 7 continued from previous page

original image	mask	method	Time(s)	PSNR	iterations
	s_02	dual-simplex	164.59139	42.029	787023
	handwriting	interior-point	11.972473	45.5072	15
	handwriting	dual-simplex	126.672577	45.0193	780401
buildings	random30	interior-point	1.544303	41.4697	9
	random30	dual-simplex	8.796186	40.9112	131317
	random50	interior-point	1.865589	39.734	11
	random50	dual-simplex	11.359887	39.1764	133496
	random70	interior-point	2.390084	38.0195	13
	random70	dual-simplex	24.80837	37.6179	135418

4.2 Part I - Sparse Reconstruction

4.2.1 General Reconstruction

The following table is for General Reconstruction part. We test the circle image (size 512× 512) for different masks. Most PSNR is above 40 and the cpu-time is few minutes. Therefore, the inpainting performs well.

Table 8: Reconstructed Information with Different mask on circle image

Mask	PSNR	CPU Time	Iteration
random 30	41.9906	96.86576	23
random 50	40.952	80.986402	22
random 70	38.7562	64.645817	21
random 90	34.3563	47.49016	18
mesh	40.995	113.082647	25
handwriting	41.1667	187.374098	38
sc01	41.913	123.101942	25
sc02	37.799	104.731405	25

We test the circle image (size 512× 512) for different masks using reformulation (2). We have to enlarge the δ value to get result. The related information is in the following table.

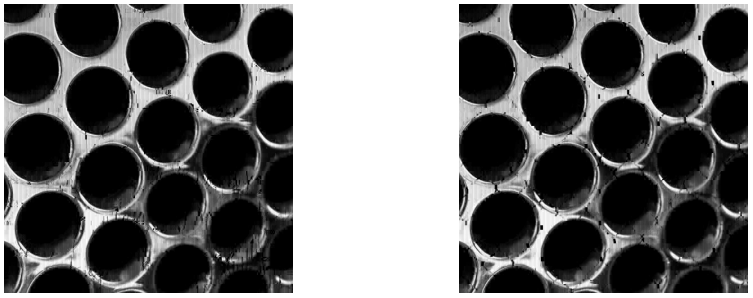

Figure 16: Left: Reconstructed circle image with mask mesh. Right: Reconstructed circle image with mask handwriting.

Table 9: Reconstructed Information using reformulation (2)

mask	δ	PSNR	CPU Time	iteration
random 30	0.06	41.9904	84.517635	32
random 50	0.06	40.9518	72.689053	29
random 70	0.07	38.5576	69.233546	26
random 90	0.075	34.2996	54.142248	21
mesh	0.1	39.6776	112.649571	42
handwriting	0.1	39.7768	98.48182	36

4.2.2 Tolerance Influence

The following table provides detailed data when we apply different tolerance on circle image.

Table 10: Reconstructed Information with Different Tolerance on circle image

mask	tolerance	δ	PSNR	CPU Time	Iteration
random 50	e^{-4}	0.06	40.9520	80.986402s	22
random 50	e^{-3}	0.06	40.9520	74.420778s	21
random 50	e^{-2}	0.06	40.9520	71.016046s	21
random 70	e^{-4}	0.06	38.7562	64.645817s	21
random 70	e^{-3}	0.06	38.7562	64.456808s	20
random 70	e^{-2}	0.06	38.7562	62.658636s	20

4.2.3 δ Influence

Table 11: Reconstruction Image Information with Different Termination Tolerance

δ	PSNR	CPU Time	Iteration
0.03	42.4053	85.706380s	24
0.04	41.8406	84.932149s	25
0.05	41.3714	85.737346s	23
0.06	40.9520	80.986402s	22
0.07	40.5744	82.810881s	22
0.08	40.2242	87.030350s	23
0.10	39.6114	87.925050s	24

4.2.4 Denoising Setting

We test the different σ and δ for denoising. The results are similar to the examples mentioned before.

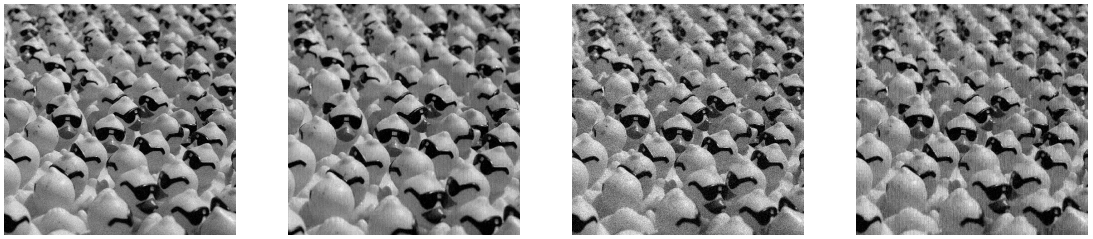

Figure 17: From left to right. First: Noisy image. Second: Recovered image with σ 0.05 and $\sigma = 0.8\delta$. Third: Noisy image2. Forth: Recovered image2 with σ 0.09 and $\sigma = 0.9\delta$.

Table 12: PSNR difference for different sigma and delta

sigma	*delta*	Reconstructed PSNR	Noisy PSNR	PSNR difference
0.05	$0.8sigma$	46.4378	50.7031	-4.2653
0.09	$0.9sigma$	43.4696	48.1238	-4.6542