P9 Towards distributed image reconstruction for radio interferometers

Jonas Schwammberger

November 7, 2019

Abstract

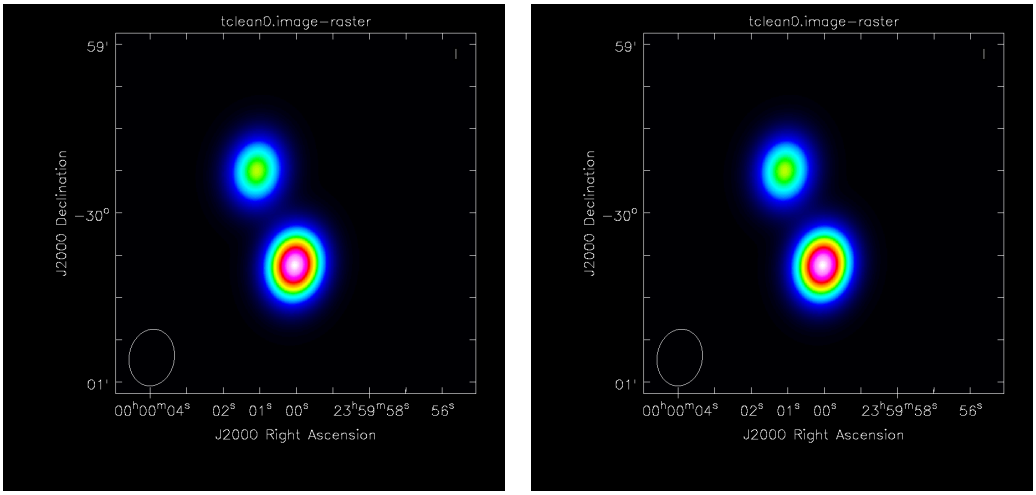
Contents

1 The image reconstruction problem of radio interferometers

In Astronomy, one goal is to find ever smaller objects in the sky. For this purpose, we build instruments with higher angular resolution. The instruments angular resolution depends on two factors: On the diameter of the antenna-dish or mirror, and on the observed wavelength. With longer wavelengths we need bigger dishes/mirrors to achieve a similar angular resolution.

This is an issue for Radio Astronomy. The long radio wavelengths require huge dishes for a high angular resolution. Of course there is a practical limit on the antenna-dish diameter we can build. The famous Arecibo observatory is one of the largest single-dish radio telescopes with a diameter of 305 meters. Antennas with such a large diameter become difficult to steer accurately, let alone the construction costs. We have reached the practical limits of single-dish telescopes. If we require higher angular resolution, we need to look at another type of instrument: The radio interferometer. They use several smaller antennas together, acting like a single large dish. An interferometer can achieve angular resolutions which are comparable to dishes with a diameter of several kilometers.

But there are drawbacks: The interferometer does not measure the sky in pixels. It measures the sky in Fourier space. As such, an interferometer produces amplitude and phase for each Fourier component that was measured. The observed image has to be reconstructed from the Fourier measurement. The measured Fourier components are called visibilities in the Radio Astronomy literature. From this point forward, we will call the measured Fourier components visibilities. The Figure 1 shows an example of the image reconstruction problem. The Figure 1a shows the measurements in the Fourier space, and the figure 1b shows the observed image of the sky, with two stars close to each other. The image reconstruction has to find the observed image 1b from the measurements 1a.



(a) Measurements of the interferometer in the Fourier space.

(b) Observed image of the sky, showing two stars.

Figure 1: The image reconstruction problem, the observed image has to be reconstructed from the Fourier measurements.

At first glance, we might believe that the image reconstruction is trivial: The interferometer measures Fourier components, and efficient algorithms for the inverse Fourier transforms are well-known. However, two properties of the measured Fourier components make the image reconstruction difficult: The measurements are both noisy and incomplete.

The atmosphere of the earth is one source that introduces noise. It adds noise to the amplitude and phase of each measured Fourier component. The atmosphere changes over time and can under the right circumstances introduce a high level of noise compared to the signal. The image reconstruction should be able to

find the observed image from potentially very noisy Fourier measurements.

The interferometer measures an incomplete set of Fourier components. Note that the Figure 1a shows the Fourier space, which has missing components. The interferometer can only measure a limited set of Fourier components. The reconstruction algorithm has to find the observed image even though important Fourier components are missing from the measurements.

These two difficulties, the noise and the incomplete measurements, lead to the fact that there are many different candidate images that fit the measurements. This is known as an inverse problem. We want to find the observed image, even though all we have are imperfect measurements. From the measurements alone, we cannot decide which candidate is the truly observed image. However, we have additional knowledge that simplifies the inverse problem: We know it is an image of the sky, which consists of stars, hydrogen clouds, etc. By including prior knowledge in the reconstruction, we can find the most likely image given the measurements.

The question remains is: How close is the most likely image to the observed one? Is exact reconstruction possible where the most likely and observed image are equal? Surprisingly the answer is yes. It is possible in theory[1, 2], and was shown in practice on low noise measurements[3, 4]. However, not all algorithms perform equally well when the noise level in the measurements is high. Also, computing resources required for each algorithm can vary significantly. In short, a reconstruction algorithm has three opposing goals:

1. Produce a reconstruction which is as close to the truly observed image as possible.
2. Robust against even heavy noise in the measurements.
3. Use as few computing resources as possible.

No reconstruction algorithm performs equally well on all three goals. One of the most widely used reconstruction algorithms is CLEAN [5, 6]. It has shown to be robust against heavy noise and, depending on the observation and is one of the oldest algorithms still in use today. As such, it was developed before the advent of distributed and GPU-accelerated computing. Today's new radio interferometers produce ever more measurements. The recently finished MeerKAT radio interferometer produces roughly 80 million Fourier measurements each second. Astronomers wish to reconstruct an image from several hours worth of measurement data. Reconstructing an image from this data volume requires GPU and distributed computation. But how to use GPU and distributed computing effectively is still an open problem.

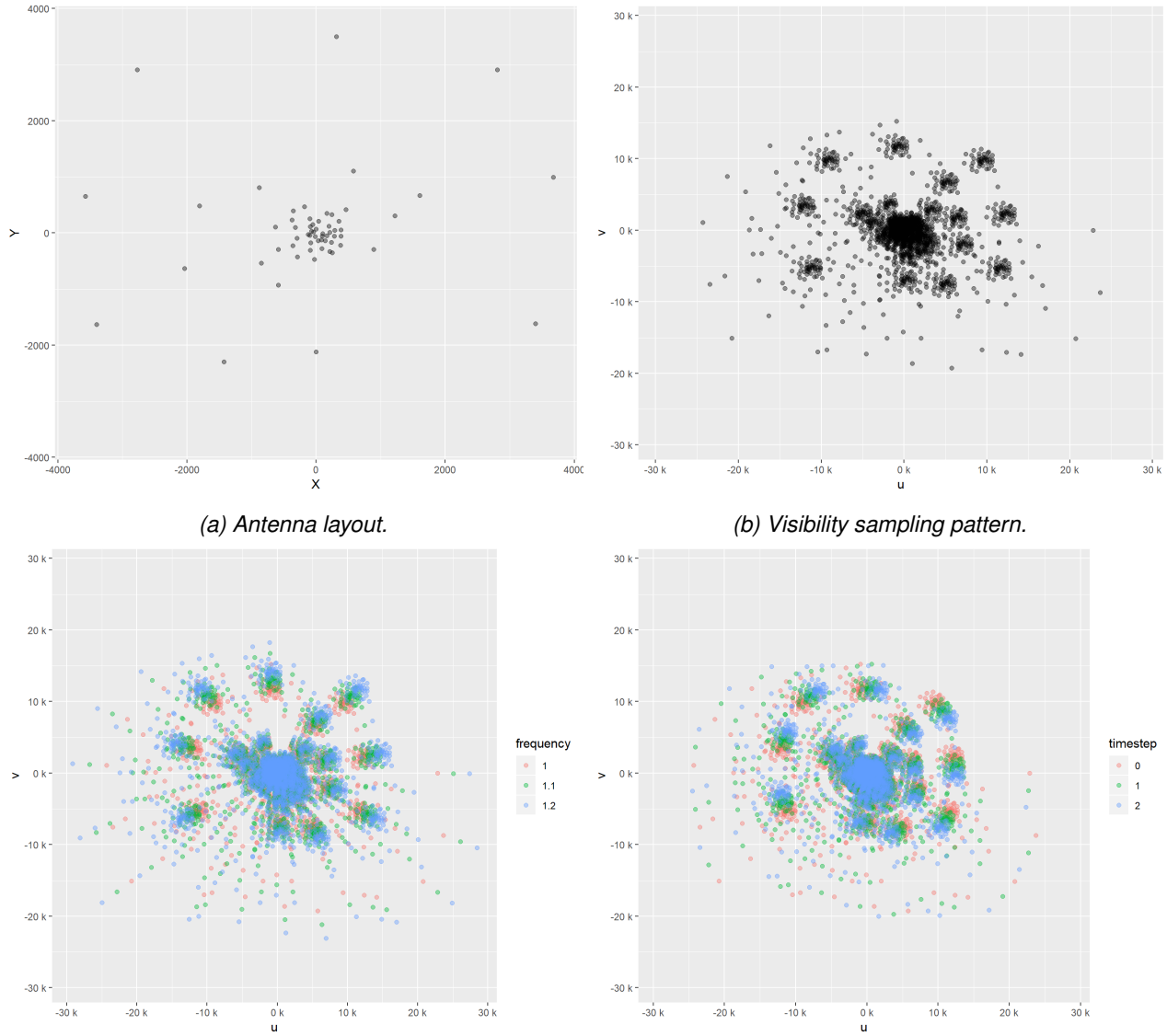
Coordinate descent methods have been successfully applied in other inverse problems, such as reconstruction of CT scans[7], or X-Ray imaging[8]. GPU accelerated[9] and distributed[10] variants have been developed. To our knowledge, coordinate descent methods have not been explored for the inverse problem in Radio Astronomy.

In this work, we develop our own proof-of-concept image reconstruction algorithm based on coordinate descent methods. We apply the reconstruction on a real world MeerKAT observation provided by SARAO. We explore the possible speedups we can achieve by using GPU and distributed computation. The algorithm is implemented platform independent in .netcore.

The rest of this work is structured as follows. First in section 2, we give an introduction to radio interferometric imaging, and give the theoretical background to why a reconstruction can even achieve a higher resolution than the instrument. Next we present the current state-of-the-art in image reconstruction for radio interferometers in section 3. Then we derive a basic image reconstruction algorithm based on coordinate descent in section 4, and show how we can use GPU acceleration and distribution to speed up reconstruction.

2 Introduction to radio interferometric imaging

A radio interferometer consists of several antennas. Each antenna pair measures a visibility in Fourier space. Each measurement consists of an amplitude and phase at a location at a u and v location. The distance between the antennas, which we call the baseline, defines what point in the Fourier space gets sampled. The Figure 2a shows the antenna layout of the MeerKAT radio interferometer, and the Figure 2b shows the measurement points in Fourier space. Short baselines sample points close to the origin, and contain the low-frequency Fourier components. They contain information about large areas of the images. Longer baselines measure points further away from the origin. They sample the high-frequency Fourier components. They contain information about edges, and other small structures in the image.



(a) Antenna layout.

(b) Visibility sampling pattern.

(c) Visibilities added from multiple channels.

(d) Visibilities added from multiple timesteps.

Figure 2: Sampling regime of the MeerKAT radio interferometer.

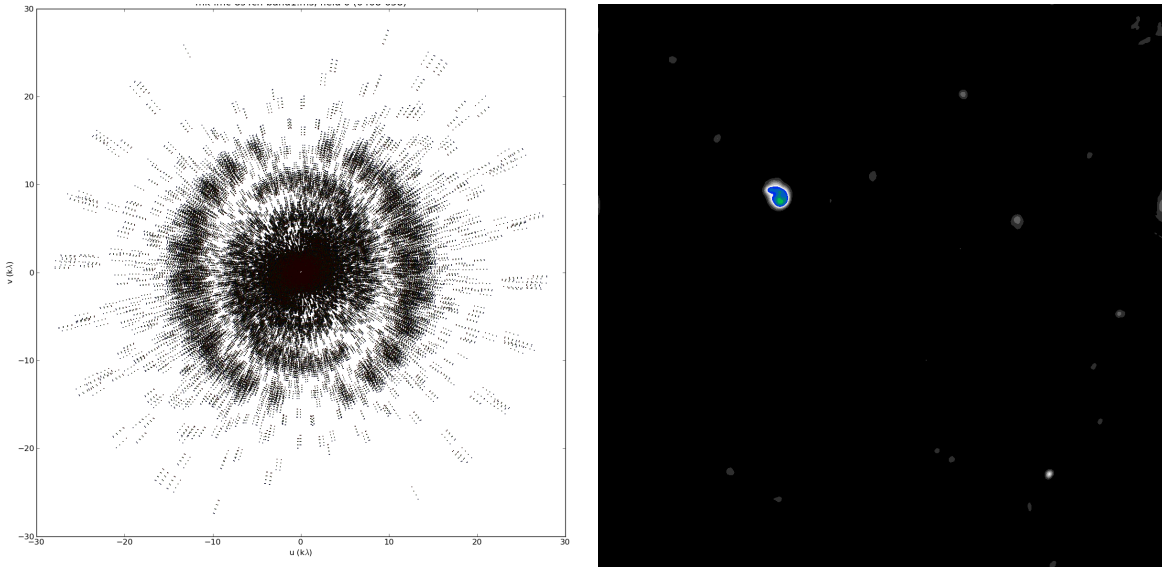
The sampling pattern of the MeerKAT interferometer is not uniform in the Fourier space. We have areas which are densely sampled, and areas which are sparsely sampled. Note that we only have a few samples of the high-frequency Fourier components. We are missing measurements from a large portion of the Fourier space.

Radio interferometers use two ”tricks” to measure more points in the Fourier space. Radio interferometers

measure the sky in different radio channels simultaneously. We can add the visibility measurements from different channels together, shown in Figure 2c. Each channel measures the Fourier space using the same pattern, but scaled by the radio frequency.

The second trick is to use the earth's rotation to sample different points in the Fourier space. The earth's rotation also rotates the sampling pattern in Fourier space, shown in Figure 2d, and we can sample the Fourier space at new locations.

The MeerKAT radio interferometer measures 2016 visibilities, for each channel, at each timestep. It has 20 thousand radio channels. The time resolution can be as low as half a second. This results in roughly 80 million visibility measurements per second. In radio astronomy, we want to reconstruct several hours worth of visibility measurements.



(a) Real-world visibilities combined from different channesl and timesteps.

(b) Reconstruction of the visibility measurements.

Figure 3: Example of an image reconstruction for Fourier measurements of the MeerKAT radio interferometer

It is easy to see that the visibility measurements from MeerKAT quickly fills up the hard disk and the Fourier space with samples. The Figure 3a shows a fraction of the visibility samples from a real-world observation. The visibilities are combined from multiple channels and multiple timesteps. Although we have a large number of visibilities, we still have areas of the Fourier space without a sample. Although we have a large number of samples, the visibilities are still an incomplete. From the measurements alone, we cannot reconstruct the image shown in Figure 3a.

This is known as an ill-posed inverse problem in the literature. A problem is considered ill-posed when:

1. No solution exists.
2. There are solutions, but no unique solution exists.
3. The solution behavior does not change continuously with the initial condition (For example: a small change in the measurements lead to a very different reconstructed image).

Image reconstruction for radio interferometer is an inverse problem, because we want to find the image which the interferometer observed in Fourier space. It is ill-posed in general, because there are many different images fitting the measurements.

Note that we can reduce the resolution of the reconstructed image until the problem becomes well-posed. The

Nyquist-Shannon sampling theorem states the case of radio interferometers: The highest highest Fourier-frequency we measure should be more than twice the highest frequency in the image. The center of the Fourier space in Figure 3a is densely sampled. We can reconstruct a low-resolution image that only needs the information from the densely sampled center.

However, this would reduce the effective resolution of the reconstruction. If we can solve the ill-posed inverse problem, we would be able to retrieve the observed image at a higher resolution than possible with the Nyquist-Shannon sampling theorem. As it turns out, this is possible to solve the ill-posed inverse problem by including prior information. We use a numerical optimization algorithm and find the optimal image, which is both consistent with the measurements and consistent with our prior knowledge. This is known in signal processing as compressed sensing [1, 2] in the literature. The theory of Compressed Sensing shows that, under the right prior information, we are guaranteed to reconstruct the observed image at a higher resolution than under the Nyquist-Shannon sampling theorem.

2.1 The theory of Compressed Sensing

We introduce the theory of Compressed Sensing for the problem of radio interferometric image reconstruction. As we have mentioned compressed sensing image reconstruction involves a numerical optimization algorithm to find the optimal solution which is consistent with our measurements and consistent with our prior knowledge. As we will see later, the theory of compressed sensing guarantees us exact reconstruction under certain assumptions.

Let us formulate the image reconstruction as an optimization problem.The image reconstruction wants to to find the image which is as close to the measurements as possible. Or more formally, we want to minimize the euclidean distance between the visibility measurements V and the reconstructed image x. We write it as an objective function:

$$\underset{x}{minimize} \ \|V - Fx\|_2^2 \tag{2.1}$$

We can reconstruct the image by finding the optimum of the objective function (2.1). The objective function is convex, meaning it has only one global minimum, and we can use the class of convex optimization algorithms to search the minimum. However, our measurements V are incomplete, meaning we do not have all the data we need for reconstruction. This means our objective function (2.1) does not ”point” to the observed image. It still has a global minimum, but observed image is not guaranteed to be near the global minimum.

A side note: We are guaranteed to find the observed image at the minimum of (2.1) is when the measurements fulfill the Nyquist-Shannon sampling theorem. In that case, we can find the minimum by calculating the inverse Fourier transform: $x = F^{-1}V$. We can still calculate the inverse Fourier transform when we are dealing with incomplete measurements, but it does not result in the observed image.

However, the objective (2.1) only includes information about the measurements. As we have mentioned before, we have prior knowledge about the image. We know it is likely to contain stars. Stars are radio-emissions which are concentrated around a single pixel. In that case, most pixels of the image will be zero, except for the locations where the interferometer has located stars. In other words, we know that the image is sparse. We can add a regularization to the objective function (2.1) and force the reconstructed image to be sparse. This results in the modified objective function:

$$\underset{x}{minimize} \ \|V - Fx\|_2^2 + \lambda \|x\|_1 \tag{2.2}$$

Note the two terms in the objective (2.2): We have the same term from our measurements, which we call

the ”data term”. But we also have an additional ”regularization term”, which is the L1 norm[1] and forces our reconstruction to be sparse. The parameter λ represents how much emphasis we put on the regularization. The new objective function is still convex, it still has a global minimum. The regularization term simply shifted the global minimum to a different location when compared to the first objective (2.1). Now the question is: How likely is our modified objective (2.2) to point at the observed image? The theory of Compressed Sensing tells us that it depends on the image content of the observed image. If it only consists of stars, we are practically guaranteed to find the observed image at the minimum of (2.2), even though we are dealing with incomplete measurements.

We use the words 'practically guaranteed' because the theory of Compressed Sensing does give us guarantees to find the observed image at the minimum of (2.2) under certain assumptions. The issue is these assumptions are hard to verify for any given reconstruction problem. In reality, it is often easier to empirically show that the regularization works. For example, by creating a super-resolved[2] reconstruction from visibility measurements [3]. The assumptions do have important implications for instrument design. This project however is focused on the numerical optimization algorithm. We do not have an influence on the visibility measurements, for our intends and purposes, the measurements are fixed. That is why we first give an intuitive explanation to when the observed image is at the minimum of the objective (2.2), and what this means for the ill-posed image reconstruction in Section 2.1.1. Then we give an overview of the formal guarantees in Section 2.1.2 and the implications for the visibility measurements.

2.1.1 Intuitive explanation of Compressed Sensing

We go back to

Intuitive

Why it works

Here we see the core assumption for the theory of Compressed Sensing to apply: We know what the observed image looks like, and can model it adequately as a regularization term. This is not always true. We do only know what the observed image looks like to a certain degree, and the regularization term does not model the observed image perfectly. In our example, we assume the image is sparse, because it only contains point sources. This is not always true in reality, as radio interferometers also measure extended emission like hydrogen clouds, which span over several pixels. The observed image will not be as sparse as before. In that case we can still try to find the observed image with an L1 regularization, but we may need more visibility measurements for the reconstruction.

To guarantee the observed image is at the There are more assumptions involved about the properties of the visibility measurements.

There are other regularization methods that model both point sources and extended emissions [11, 12]. With these regularizations, we can reconstruct the observed image from fewer measurements, or find a super-resolved image from the same measurements. This means that the number of samples which are necessary for reconstruction depend on the regularization we use.

The theory of Compressed Sensing can a To solve the ill-posed inverse problem, we have an implicit data modeling task involved. We need less visibility measurements the better we can model the observed image.

The theory of Compressed Sensing gives us a formula to how many samples we need for the reconstruction. Sadly, these are based on hard-to-verify assumptions.

[1] Sum of absolute values of the pixels

[2] The radio interferometer also has a resolution limit. Super-resolved reconstructions were able to find structures which are smaller than the limit of the instrument.

Where is the limit Ther is one, but we cannot calculate the limit in practice. It is based on assumptions hard-to-verify assumptions. In reality, it is often easier just to find the reconstruction and do an empirical proof. Our regularization works, because it produces good results.

First assumption: we know what the observed image looks like. Not always true, radio interferometers also measure hydrogen clouds, which span an area over several pixels. We can retrieve an image below the Nyquist-Shannon sampling rate. Where is the limit? how many samples do we need to retrieve the image? In short: There is a limit, but it is based on even harder to prove assumptions. We quickly introduce them for completion's sake.

In practice, we know this works, because it was shown to work.

This project does not dive deeper into the prior, and how good it models the observed image. We are diving into numerical optimization algorithms and how we can retrieve the observed image with a given regularization.

So when we have a good prior, the objective points to the observed image even though we do not have enough samples according to the Nyquist-Shannon theorem. What the theory of Compressed Sensing can tell us is how many samples we need until the objective points to the observed image.

Sadly, we cannot verify that the assumptions hold in practice. is that we cannot verify these assumptions in practice.

Depends on the image content and the prior we use. Also, the theory of Compressed Sensing difficult bounds in practice.

Theory of Compressed Sensing Pretty well, if the image only contains stars. Theory of compressed sensing Incoherence. We gain maximum information about the non-zero locations with each visibility measurement.

All we need for Compressive Sensing: A reconstruction space which our image is sparse. A measurement space which is incoherent to our reconstruction space.

Then the number of samples we need does not depend on the resolution of the image, but on the number of non-zero components in our reconstruction space.

2.1.2 Formal Guarantees

RIP What is the RIP Meaning of F. How many random columns of F are uncorrelated? Remember that we have redundant information, the whole matrix F does have columns which are correlated.

At what point are we guaranteed? The matrix A needs to fullfil the Restricted Isometry Porperty (RIP) [1, 2]. Approximately orthonormal on sparse signals. (If we randomly choose ten columns of F, how much do they correlate. We want as little correlation as possible.) Calculate the RIP is NP-hard[13]. Approximations are also difficult to compute[14]. So we can only talk about how likely a given matrix fullfils the RIP. Matrix where each element is sampled from a random Gaussian distributîon has the highest change.

Random samples in the Fourier space also have a high chance to fulfill the RIP [15].

Not possible for Radio interferometers, because they sample in the Fourier space. Not random.

There are also extended emissions, clouds etc. Resulting in a lot of non-zero pixels. There may be better sparse spaces for radio astronomy.

But also RIP may be too strict. Exact reconstruction can also work under less strict conditions[16] active field of research.

We do not know if the image we reconstruct is the observed one. Calculating good bounds are hard. Demonstrated super-resolution performance.

2.1.3 General Compressed Sensing reconstruction formulation

Use everything with the L1 norm. More general formulation

F is fixed, because we cannot change the interferometer.

We do just need a sparse space.

T

So far, we discussed how the interferometer measures in Fourier space, and we wish to find the observed image that matches the measurements. In other words, we wish to find a solution to a system of linear equation (**??**), where V are the measurements, x is the observed image and F is the Fourier transform matrix. We also discussed that F can be complicated in practice, but is still essentially a linear operator. Meaning we know how the inverse Fourier transform matrix F^{-1}, and the question arises: Why can't we solve the equations (**??**) by calculating the inverse Fourier transform? Or, why does $x = F^{-1}V$ not lead to the observed image?

The answer is, the equations (**??**) do not have a unique solution, which makes the problem ill-posed. The problem is considered ill-posed when it has one of the following properties:

1. No solution exists.
2. There are solutions, but no unique solution exists.
3. The solution behaviour does not change continuously with the initial condition (For example: a small change in the measurements lead to a very different reconstructed image).

From linear algebra, we know that an under-determined system of linear equations, i.e. when (**??**) has more pixels than visibility measurements, then the problem is under-determined and there may be a potentially infinite number of solutions to the system. Under-determined systems arise in many similar fields, as for example in X-Ray imaging of the sun[8]. However, radio interferometers measure a large number of visibilities. We generally have more visibilities than pixels. This means the image reconstruction problem (**??**) for radio interferometers is actually over-determined.

From linear algebra, we know that an over-determined system either has one or zero solutions. At first glance it may be counter-intuitive why there are many possible solutions to (**??**) for radio interferometers. The reason why lies in two properties of the measurements: They are noisy and incomplete.

The radio interferometer measures noisy visibilities, meaning each amplitude and phase of a measurement is influenced by an unknown noise factor. Finding a reconstructed image is the same as finding the de-noised versions of the visibility measurements. This alone would make the problem ill-posed, but the visibilities actually have a second property that makes them ill-posed: incompleteness.

When we look back at figure 3a, it is clear to see that the interferometer does not sample the visibilities in a uniform way. There are regions with a high sample density. The density decreases when we move further away from the center. The higher frequency visibilities get fewer and fewer samples. This means we are missing data for crucial measurements for the reconstruction.

Also note that the question whether the measurements are incomplete essentially comes down to the image resolution of the reconstruction: Since we are missing high-frequency measurements and we can choose the resolution of the image, we can also reduce the resolution of the reconstructed image until the missing frequencies become negligible. However, if we can solve the ill-posed inverse problem, we can reconstruct an image at a higher resolution from the same measurements. The question is how do we solve the ill-posed inverse problem?

2.1.4 Adding a regularization

For ill-posed inverse problems, there are two viewpoints for the same idea. From the viewpoint of optimization, we can solve the ill-posed image reconstruction problem (**??**) by adding a regularization. The regularization creates a system of linear equations with a unique solution. From the viewpoint of Bayesian Statistics, we include prior knowledge about the image, and therefore search the most likely image given the measurements. For this project, both terms describe the same idea and we use regularization and prior interchangeably. We know that the reconstructed image from radio interferometers contain a mixture of stars (point sources located in a single pixel) and extended emissions like hydrogen clouds. By adding this prior knowledge to the reconstruction problem, we can find the most likely image given the measurements. As we will see, under the right prior, we can create a reconstruction algorithm that is almost guaranteed to find the truly observed image in theory.

There are different ways to include regularization in the reconstruction problem. In this project, we use the following method: We cast the image reconstruction problem into an optimization objective consisting of a data fidelity term and a regularization term. A reconstruction algorithm therefore consists of an optimization objective, a prior function and an optimization algorithm.

$$\underset{x}{minimize}\ \|V - Fx\|_2^2 + \lambda P(x) \tag{2.3}$$

The objective (2.3) has a data term $\|V - Fx\|_2^2$, which forces the most likely image to be as close to the measurements as possible, and the regularization term $P(x)$, which penalizes unlikely images according to our prior function. The parameter λ represents how much we penalize unlikely images and by extend, much noise we expect in the reconstruction. The parameter λ is either left to the user to define, can be estimated from the data [17].

The prior function $P()$ represents our prior knowledge about the image. It assigns a high penalty for unlikely images. In in radio interferometric image reconstructions tend to use similar prior functions as for image denoising applications. Such as: Total Variation ($\|\nabla x\|_1$) [18], L2 ($\|x\|_2$) [19] or the L1 norm in a wavelet space ($\|\Psi x\|_1$) [20].

Finally, an optimization algorithm is necessary which can optimize the objective function (2.3) with the chosen prior function $P()$. Typical choices for optimization algorithms in image reconstructions are Interior Point Methods like Simplex, Matching Pursuit [5] and ADMM[21].

A reconstruction algorithm for radio astronomy consists of an optimization objective, a prior function and an optimization algorithm. It is a three dimensional design space. Not every prior is suitable for every optimization algorithm. The choice of optimization objective influences both what prior and what optimization algorithm we can use. Although there are a different choices for the optimization objective, we limit ourselves to the objective (2.3) and explore how we can distribute the reconstruction problem.

The last question that remains is, how close are the most likely image, under a given prior, to the truly observed one? Remember that the Nyquist-Shannon sampling theorem states that our uniform sampling frequency needs to be larger than twice the highest frequency in a band-limited signal[3], and then the theorem guarantees exact reconstruction. For radio astronomy, we do not have uniformly sampled visibilities, and although we have a large number of samples, we are missing crucial parts of the Fourier space. Luckily, there is another sampling theorem that, under certain assumptions, guarantees exact reconstruction for the case of radio interferometers: The theory of compressed sensing.

[3] For example: if we want to record human voices with the highest frequency of 20 kHz, Nyquist-Shannon states our uniform sampling frequency has to be larger than 40 kHz to guarantee exact reconstruction

2.1.5 Compressive sampling of the sky

For the sake of demonstration, let us assume the radio interferometer observes a patch of sky containing ten stars. It measures an incomplete set of random Fourier components of the ten stars, and we would like to reconstruct an image of size 256^2 pixels. The emissions from stars are concentrated into a single pixel. For compressive sampling, we need to know an space in which our reconstructed image is sparse, and we need to take measurements in a different, incoherent space.

Our reconstructed image contains only zero pixels except at the ten locations of the stars, meaning the image space for this patch of the sky is already sparse. In this case, we do not need any additional sparse space like wavelets. We can reconstruct directly in the sparse image space, and we have the first requirement met for compressive sampling.

The next requirement is that the measurement and reconstruction space (which is the image in this example), are as incoherent as possible.

Yes, incoherence.

$$\underset{x}{minimize} \ \|V - Fx\|_2^2 + \lambda \|x\|_1 \tag{2.4}$$

Intuitively, the number of samples depend on the information content, not on the bandwidth. thi

At what point are we guaranteed? The matrix A needs to fullfil the Restricted Isometry Porperty (RIP) [1, 2]. Approximately orthonormal on sparse signals. (If we randomly choose ten columns of F, how much do they correlate. We want as little correlation as possible.) Calculate the RIP is NP-hard[13]. Approximations are also difficult to compute[14]. So we can only talk about how likely a given matrix fullfils the RIP. Matrix where each element is sampled from a random Gaussian distributîon has the highest change.

Random samples in the Fourier space also have a high chance to fulfill the RIP [15].

Not possible for Radio interferometers, because they sample in the Fourier space. Not random.

There are also extended emissions, clouds etc. Resulting in a lot of non-zero pixels. There may be better sparse spaces for radio astronomy.

But also RIP may be too strict. Exact reconstruction can also work under less strict conditions[16] active field of research.

2.1.6 Reconstruction guarantees in the real world

What does this all mean for image reconstruction? For us, we design reconstruction algorithms and cannot influence the matrix F. We do not know if it actually

The theory of compressed sensing gives us a framework. There is a data modelling task, and a task for finding efficient algorithms.

So there is a data modelling task in finding a good sparse prior. We also do not know the proper sparse space in which radio interferometric images. We know several spaces, Curvelets [11] Starlets [12], Daubechies wavelets [22]. As of the time of writing, it is currently unknown which leads to the best reconstruction. We can also learn dictionaries.

The task for efficient algorithms is finding the best way to optimize the objective with a given prior on real hardware. Fast convergence. Also difficult, because in practice the choice of prior can also influence convergence.

Compressed sensing based reconstruction algorithm. In radio astronomy.

2.2 Noise, approximations and other difficulties in radio interferometry

We give a short introduction into how the electromagnetic wave gets measured by the interferometer, turned into visibilities and finally processed into an image. Figure 4 shows a radio source and its electro-magnetic (em) wave arriving at the antennas of the interferometer. It then shows the three processes involved to arrive at an image: Correlation, calibration and image reconstruction.

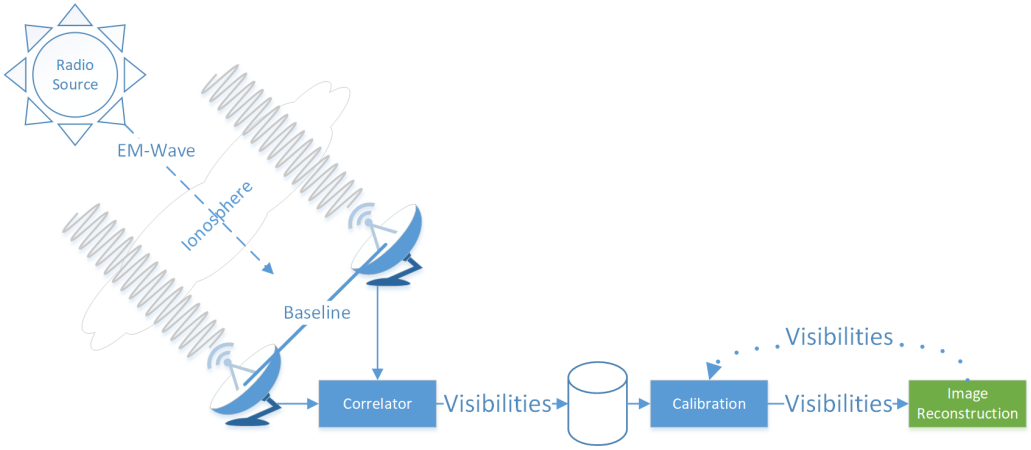


Figure 4: Radio interferometer system

First, we have a source in the sky that is emitting em-waves in the radio frequency. The waves travel to earth, through the earth's ionosphere and finally to our interferometer. Along its path, the e-m waves may get distorted from various sources. For example, it may receive a phase shift by the ionosphere.

Then, the em-wave arrives at our interferometer. We call each antenna pair a baseline. Each baseline will end up measuring a single visibility. The distance between the antennas and their orientation to the em-wave will determine where we sample the uv-plane. Short baselines measure the uv-plane close at the origin, while long baselines sample the uv-space further away from the origin. Remember that the samples away from the uv-origin contain the information about edges and other details of our image. With a longer baseline the interferometer measures more highly resolved details, regardless of the antenna dish-diameter[4]. The figure 4 shows the em-wave arriving at a single baseline of the interferometer. Each antenna picks up its version of the em-wave and transfers it to the correlator.

The correlator then takes the feed of each antenna and correlates the signals, which results in the amplitude and phase of the visibility component. Amplitude and phase for each visibility are measured for a short time range (i.e. fractions of a second up to several seconds). At this point, the visibilities are saved to disk for further processing. The radio interferometer produces a visibility measurement for each baseline, for each time range, for each frequency channel of the instrument. Because a single observation can take up to several hours, measured with several thousand frequency channels, radio interferometers produce an almost arbitrary large number of visibilities.

Calibration

Image Reconstruction

2.2.1 The measurement equation

As we discussed so far, the radio interferometer measures visibilities of the sky image, and we wish to find the observed image from the measurements. Put formally, we wish to invert the following system of linear

[4]Remember that this is the reason why we build radio interferometers. We do not need impossibly large dish diameters for a high angular resolution. We just need large distances between smaller antennas.

equations (2.5), where V is the visibility vector[5], F is the Fourier transform matrix and I is the pixel vector of the observed image.

$$V = FI \tag{2.5}$$

We wish to find the observed image I, while we only know the visibility vector V and the Fourier transform matrix F. This is what we call the measurement equation. In most context for this project, looking is an adequate view of the image reconstruction problem. We will show why we cannot find the observed image I by simply calculating the inverse Fourier transform. However, when we need to efficiently apply the Fourier transform, we need to know F in more detail. As we will see, radio interferometers have some difficulties hidden in the Fourier transform matrix, which are difficult to handle efficiently. First, let us abandon the vector notation of (2.5), and represent the measurement equation with integrals (2.6).

$$V(u,v) = \int\int I(l,m)e^{2\pi i[ul+vm]}\,dl\,dm \tag{2.6}$$

This is essentially the same problem. The main difference is that we do not represent the Fourier transform as a matrix F, but as integrals $\int\int e^{2\pi i[ul+vm]}$, where u, v are the coordinates in Fourier space and l, m are the angles away from the image center. A single pixel represents the intensity of the radio emission from the direction l, m. Note that the measurement equation (2.6) shows the fact that the visibilities are measured in a continuous Fourier space. If the Fourier space would also be discrete, we could replace the integrals with sums.

However, the measurement equation (2.6) is inaccurate in the sense that it ignores many effects that distort the signal. For example, it does not account for the distortion by the ionosphere, or the distortion introduced by real-world antennas. The measurement equation (2.6) shown here does not represent the real world. But depending on the instrument and the observation, these distortions may be negligible, and the measurement equation (2.6) is a good approximation.

When there is a distortion source that cannot be ignored, it has to be modelled in the measurement equation. As such there is no unified measurement equation for all radio interferometric observations, let alone radio interferometers. The equation shown in (2.6) can be seen as the basis that gets extended as necessary[23, 24, 25, 26].

For example, the measurement equation (2.6) is only accurate for small field of view observations, when l and m are both small angles. For wide field of view observations, we need to account for the fact that the visibilities have a third term w, and we arrive at the wide field of view measurement equation (2.7).

$$V(u,v,w) = \int\int \frac{I(l,m)}{c(l,m)} e^{2\pi i[ul+vm+w(c(x,y)-1)]}\,dl\,dm\,, \quad c(l,m) = \sqrt{1-l^2-m^2} \tag{2.7}$$

The third w-term has two effects on the measurement equation. It introduces a phase shift in the Fourier transform $e^{2\pi i[\ldots+w(c(l,m)-1)]}$, and a normalization factor of the image $\frac{I(l,m)}{c(l,m)}$. Note that when the angles are small, i.e. $l^2 + m^2 \ll 1$ then the wide field of view measurement equation (2.7) reduces to our original (2.6). This is another way of saying that for small field of views, the measurement equation (2.6) is a good approximation under the right conditions.

In this project, we use the wide field of view measurement equation (2.7). But as we mentioned in the beginning of this section, for most contexts, it is not important whether we ignore the w-term of the visibilities or not. It is important when we design an efficient implementations for applying the wide field of view Fourier

[5] We use the lower-case v to denote the axis in the Fourier space uvw, and the upper-case letter to denote the visibility vector.

transform, because the w-term keeps us from using the Fast Fourier Transform (FFT). In every other case, we can ignore this technicality. Because even more complicated measurement equation still have a linear relationship between visibilities and image [23, 24, 25, 26]. We can view the whole reconstruction problem as a system of linear equations (2.5), where the matrix F takes care of how exactly the measurements and pixels relate in this case.

2.3 Introduction into optimization/RI reconstruction algorithms

2.3.1 Image reconstruction as deconvolution

2.3.2 CLEAN deconvolution algorithm

2.3.3 The Major/Minor cycle

3 State of the Art image reconstruction

Image reconstruction pipelines are split in two taks. Gridding and deconvolution. We introduce here the two latest gridding algorithms, w-stacking in 3.1 and Image Domain Gridder 3.2.

Deconvolution we discuss CLEAN and MORESANE.

Not all algorithms are here from

3.1 w-stacking Gridder

3.2 Image Domain Gridding Algorithm

3.3 Deconvolution

3.3.1 CLEAN

Various Improvements and speedups[6, 27, 28, 29], but the core algorithm is still this.

3.3.2 MORESANE

3.4 Coordinate Descent

4 Coordinate descent deconvolution

In this section we describe the basic idea behind coordinate descent methods in general, and derive our coordinate descent deconvolution algorithm. In essence, we create our own deconvolution algorithm which can be used instead CLEAN in the Major/Minor cycle architecture using coordinate descent methods. We call the coordinate descent algorithm ”basic”, because it can be seen as a special case of more complicated coordinate descent methods[30, 31], which we will discuss later in Section 8. Here, we derive efficient CPU- and GPU- implementation and show how we can use MPI for a naive distributed deconvolution algorithm.

Coordinate descent methods are a family of numerical optimization algorithms, that are used for convex objective functions. They share one common idea: Most of our problems become simple when we reduce the number of dimensions. Deconvolving a whole image in our case is difficult, but deconvolving a single pixel is easy. As we will show in this Section, we can derive a closed form solution[6] for deconvolving a single pixel. The basic coordinate descent algorithm then iterates overall pixels, possibly several times, until we converge to a deconvolved result.

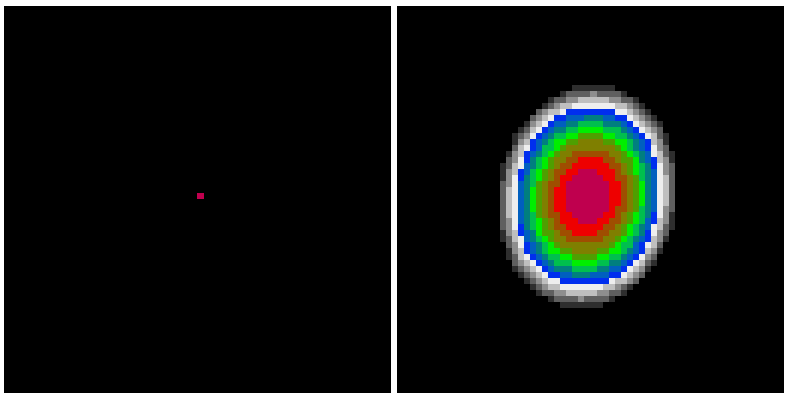
This is the idea behind coordinate descent methods. By reducing the dimensions of the problem, we can often find an optimization algorithm where each iteration is ”cheap” to compute. In these cases, coordinate descent methods produce competitive results when compared to other widely used methods like gradient descent.

For a deconvolution algorithm in radio astronomy, we need three parts: An optimization algorithm, a regularization, and an optimization objective. We use coordinate Descent as the optimization algorithm, take Elastic Net as the regularization and use the following objective function:

$$\underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - x * PSF\right\|_2^2 + \lambda ElasticNet(x) \tag{4.1}$$

As we have shown before, the objective function consists of two parts. We will derive the ”basic” coordinate descent algorithm that optimizes the objective (9.1) in Section 4.2. First, let us explain what the elastic net regularization does.

4.1 Elastic net regularization



(a) Effect of the pure L1 norm (λ = 1.0) on a single point source.

(b) Effect of the pure L2 norm (λ = 1.0) on a single point source.

Figure 5: Effect of the L1 and L2 Norm separately.

This regularization is a mixture between the L1 and L2 regularization. The Figure 11 shows the effect of the L1 and L2 norm on a single star. The L1 regularization forces the image to contain few non-zero pixels as

[6] Deriving a formula which we can implement in a few lines of code.

possible. It encodes our prior knowledge that the image will contain stars. The L2 regularization on the other hand ”spreads” the single star across multiple pixels. This forces the image to represent extended emissions, like hydrogen cloud, with a large number of non-zero pixels (the L1 norm tends to break extended emissions apart, only using a handful of non-zero pixels). The L2 norm was already used in other image reconstruction algorithms in radio astronomy[19], with the downside that the resulting image will not be sparse.

Elastic net mixes these two norms together, becoming ”sparsifying L2 norm”. It retains the sparsifying property of the L1 norm, while also keeping extended emissions in the image. Formally, elastic net regularization is defined as the following:

$$ElasticNet(x, \alpha) = \alpha \|x\|_1 + \frac{1-\alpha}{2} \|x\|_2 \tag{4.2}$$

Elastic net has three properties which make it an interesting regularization for coordinate descent: It was shown to speed up convergence rates compared to the pure L1 or L2 norm[34], is a separable function, and has a closed form solution. The first property was not further investigated in this work. The second property, separability, means that we can calculate the regularization for each pixel independently of each other, and we still arrive at the same result. Lastly, we can find a simple formula for each pixel that applies the elastic net regularization:

$$ElasticNetClosedForm(x, \lambda, \alpha) = \frac{max(x - \lambda * \alpha, 0)}{1 + \lambda(1-\alpha)} \tag{4.3}$$

The closed form solution (4.3) of the elastic net regularization is also a mixture of the closed form solutions of the L1 and L2 norm. The closed form solution of the L1 norm is shrinkage: $max(x - \lambda, 0)$, we reduce the pixel value by λ and clamp negative values. For the L2 norm, we divide the pixel value: $\frac{x}{1+\lambda}$.

Note that the shrink operation in this project always clamps negative pixels to zero. We constrain the image to only contain zero or positive pixel values. This has become a widely used constraint in radio interferometric image reconstruction and may lead to improved image quality[35].

Elastic net is the regularization we use throughout this work. It is separable (we can calculate it for each pixel independently) and has an easy to compute closed form solution.

4.2 Deriving the basic coordinate descent deconvolution algorithm

In this section we derive the basic coordinate descent deconvolution algorithm, which minimizes the objective (9.1). Coordinate descent methods have a tendency to need a more iterations to converge compared to other methods like gradient descent. However, when a single iteration is cheap to compute, they can be faster in practice[36]. The elastic net regularization has an easy to compute closed form solution (4.3), and a single iteration is cheap to compute.

We call the coordinate descent algorithm described here ”basic”. Other coordinate descent algorithms in the literature[30, 31, 37] can be seen as generalizations of the ”basic” algorithm. The basic algorithm optimizes a single pixel at each iteration, while other algorithms can optimize one or several pixels.

In this section, we derive the basic coordinate descent algorithm that optimizes a single pixel in each iteration, and iterates over all pixels several times, with a specific strategy, until convergence. There are three types of iteration strategy we can choose:

1. Random: where we choose a pixel to optimize uniformly at random.
2. Greedy: where we first choose the pixel which minimizes our objective the most
3. Cyclic: where we choose a subset of pixels and cycle through them until convergence.

The iteration strategy is not important for convergence. We can for example create a mixture of the different strategies and the algorithm would still converge to the optimum. However, the strategy we choose has an impact on how many iterations we need until convergence. For example: if the image consists of a single star in the center of the image, a greedy strategy would first optimize the pixel at the center, while a random strategy may waste the computing resources in checking every other pixel several times before finally landing on the center. In this implementation, we chose the greedy strategy. Each iteration takes the best possible step towards the optimum. We arrive at the following coordinate descent algorithm in pseudo code:

```
1  dirty = IFFT(Gridding(visibilities))
2  residuals = dirty
3
4  x = new Array
5  objectiveValue = SUM(residuals * residuals) + P(x)
6  oldObjectiveValue = objectiveValue
7
8  do
9  {
10   oldObjectiveValue = objectiveValue
11
12   //the core of the algorithm
13   pixelLocation = GreedyStrategy(residuals)
14   oldValue = x[pixelLocation]
15   optimalValue = oldValue + Minimize(residuals, psf, pixelLocation)
16   optimalValue = ApplyElasticNet(optimalValue, lambda, alpha)
17
18   //housekeeping
19   x[pixelLocation] = optimalValue
20   residuals = residuals - PSF * (optimalValue - oldValue)
21   objectiveValue = 0.5 * SUM(residuals * residuals) + lambda * ElasticNet(x, alpha)
22 } while (oldObjectiveValue - objectiveValue) < epsilon
```

The core of the algorithm consists of the three functions: $GreedyStrategy()$, $Minimize()$ and $ApplyElasticNet()$. The function $GreedyStrategy()$ will be discussed in Section 5.3. The function $ApplyElasticNet()$ was already described in equation (4.3). The $Minimize()$ function is responsible for minimizing the data term of our objective (9.1). Because we only minimize a single pixel, we are dealing with a one dimensional minimization problem and can derive a closed form solution for it.

When we only have one pixel to minimize, the data term of our objective (9.1) reduces itself to a parabola. We derive the standard parabola form in (4.4), where $\langle x, y \rangle$ is the inner product(element-wise multiplication followed by a sum over all elements):

$$\begin{aligned} Minimize(pixel) &= \|I_{res} - PSF * pixel\|_2^2 \\ Minimize(pixel) &= (I_{res} - PSF * pixel)^2 \\ Minimize(pixel) &= \langle I_{res}, I_{res} \rangle - 2\langle I_{res}, PSF \rangle * pixel + \langle PSF, PSF \rangle * pixel^2 \\ Minimize(pixel) &= \langle PSF, PSF \rangle * pixel^2 - 2\langle I_{res}, PSF \rangle * pixel + \langle I_{res}, I_{res} \rangle \end{aligned} \tag{4.4}$$

Now finding the optimal value for the pixel is the same as finding the optimal value of the parabola:

$$\begin{aligned} f(x) &= a * x^2 \\ &+ b * x \\ &+ c \end{aligned} \qquad \begin{aligned} Minimize(pixel) &= \langle PSF, PSF \rangle * pixel^2 \\ &- 2\langle I_{res}, PSF \rangle * pixel \\ &+ \langle I_{res}, I_{res} \rangle \end{aligned} \tag{4.5}$$

$$x_{min} = \frac{-b}{2a} \qquad pixel_{min} = \frac{-(-2\langle I_{res}, PSF \rangle)}{2\langle PSF, PSF \rangle}$$

This means we can find the optimum value of a single pixel by following the formula in (4.5). Note that the PSF in formula (4.4) and (4.5) is shifted to the pixel position we wish to optimize.

We derived the closed form solution (4.5) by looking at the data objective as a parabola. When we take another look at the closed form solution with Calculus in mind, we can see that the numerator $-2\langle I_{res}, PSF \rangle)$ is actually the same as calculating the gradient for this pixel, and the denominator $\langle PSF, PSF \rangle$ is the Lipschitz constant.

Intuitively, the Lipschitz constant describes how fast a function $f(x)$ changes with x. If $f(x)$ changes slowly, we can descend larger distances along the gradient without the fear for de-convergence. In short, it is a data-defined step size. Because our function $Minimize()$ is simply a parabola, the gradient together with the Lipschitz constant point to the optimum of our function.

Because the minimizer (4.5) of our coordinate descent algorithm calculates the gradient, one might ask what the differentiates the coordinate descent method from gradient descent. The main difference is that coordinate descent uses the gradient of a single pixel (or subset of pixels in other versions), in each iteration, while gradient descent generally uses the gradients of all pixels in each iteration. Conceptually, coordinate descent does is not bound to use the gradient. We could also minimize the pixel with a line-search algorithm, trying different values for the pixel, and it is still a coordinate descent method.

This is how the basic coordinate descent deconvolution algorithm works. But as it is described here, one iteration is too expensive to be practical. The $Minimize()$ function calculates both the gradient and the Lipschitz constant in each iteration and the residual update in line 20 requires a convolution. We can drastically improve the runtime costs by caching intermediate results, and using approximations.

4.3 Efficient implementation of basic coordinate descent deconvolution

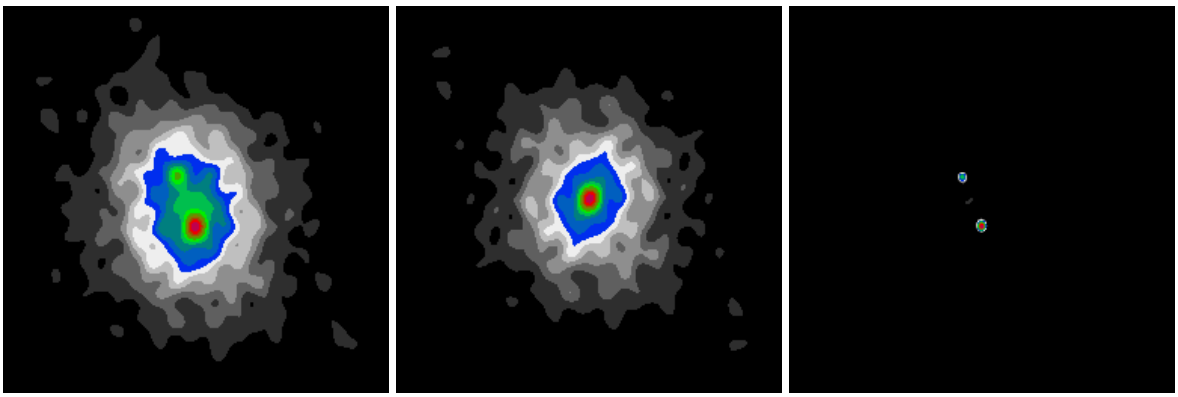
In Section 4.2, we derived the basic coordinate descent deconvolution algorithm. There are several ”tricks” to speed up each iteration. We can cache intermediate results, and exploit the convolution to efficiently calculate the inner products of the basic algorithm We discuss:

1. Edge handling of the convolution
2. Pre-calculation of the Lipschitz constants
3. Efficient greedy strategy
4. Pre-calculation of gradients
5. Efficient update of gradients

Gradient calculation is the most time consuming step. We can exploit the convolution to efficiently pre-calculate, update and approximate the gradients for each pixel. This will be discussed in detail in this Section. To our knowledge, we are the first to explore ways to approximate the gradient in radio interferometric image

reconstruction, and their effect on parallel and distributed deconvolution. As we will se in the later sections, approximating the gradients can help us to distribute the deconvolution.

Visual aid:



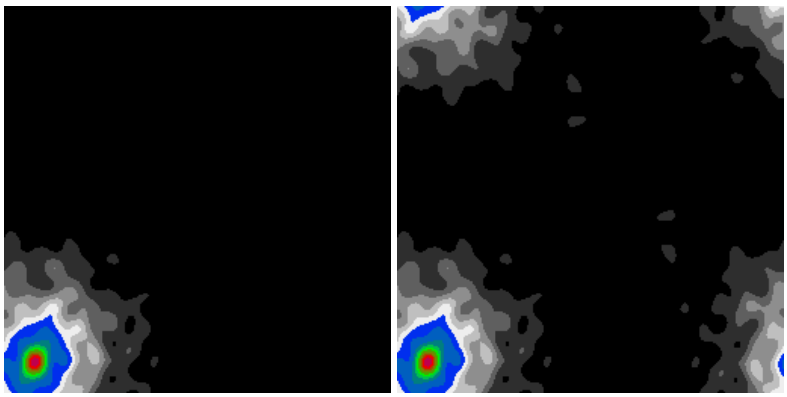
(a) Dirty Image. *(b) Point Spread Function.* *(c) ElasticNet deconvolution*

Figure 6: Example problem with two point sources.

First, we dive into the implementation of the convolution operator and the pre-calculation of the Lipschitz constants, and then we discuss the gradient calculation in detail.

4.3.1 Edge handling of the convolution

As the reader is probably aware, there are several ways to define the convolution in image processing, depending on how we handle the edges on the image. Two possibilities are relevant for radio interferometric image reconstruction: Circular and zero padded.



(a) Zero padded convolution. *(b) Circular convolution.*

Figure 7: Comparison of the two convolution schemes.

Circular convolution assumes the image ”wraps” around itself. If we travel over the right edge of the image, we arrive at the left edge. The convolution in Fourier space is circular. Remember: A convonlution in image space is a multiplication in Fourier space, and vice versa. When we convolve the reconstructed image x with the PSF using circular convolution, then non-zero pixels at the right edge of the image ”shine” over to the left edge. This is physically impossible.

Zero padding assumes that after the edge, the image is zero. Non-zero pixels at the right edges of the image do not influence the left edge after convolution. This is the physically plausible solution. However, the zero

padded convolution is more expensive to calculate. We either have to calculate the convolution in image space, which is too expensive for large kernels, or apply the FFT on a zero-padded image. Either way, it is more expensive than the circular convolution.

In designing a deconvolution algorithm, we have the choice between the circular and the zero-padded convolution scheme. Circular convolution is more efficient to calculate, while zero-padded convolution is closer to the reality. Both choices are possible. The PyMORESANE reconstruction algorithm [38] leaves this choice to the user. We decided on using the zero-padded convolution. This choice influences other parts of the coordinate descent deconvolution algorithm, like how we can efficiently calculate the Lipschitz constants.

4.3.2 Pre-calculation of the Lipschitz constants

Lipschitz constants are $\langle PSF, PSF \rangle$. We simply multiply the PSF with itself and sum up the values. However, we are using the zero-padded convolution. This means the PSF for pixels at the edges is not only shifted, but also cropped. In other words, every pixel has a different Lipschitz constant depending on how much the PSF gets cropped by the image edges.

Note that it is not an issue for convergence: The Lipschitz constant describes the largest step we can take without overshooting the target. We can always make smaller steps, but may pay it with more iterations. The Lipschitz constant of the edge pixels is always lower than the center. The coordinate descent algorithm does converge, but needs more iterations for pixels at the edges of the image. Luckily, there is a way to re-use intermediate results, and efficiently calculate the Lipschitz constant for each pixel in the image.

The first observation is that the image edges always create a rectangular crop of the PSF. To calculate the Lipschitz constant, we square and sum up all the values that lie inside the rectangle. This can be exploited with a scan algorithm: We store the PSF as a running sum of squares.

```
var scan = new double[,];
for (i in (0, PSF.Length(0))
{
  for (j in (0, PSF.Length(1))
  {
    var iBefore = scan[i - 1, j];
    var jBefore = scan[i, j - 1];
    var ijBefore = scan[i - 1, j - 1];
    var current = PSF[i, j] * PSF[i, j];
    scan[i, j] = current + iBefore + jBefore - ijBefore;
  }
}
```

$scan[0, 13]$ contains the sum of the squared PSF values from index $(0, 0)$ up to and including index $(0, 13)$. The last element, $scan[PSF.Length(0) - 1, PSF.Length(1) - 1]$ contains the sum of squares over the whole PSF. In short, we have stored the sum of all possible rectangles starting from index $(0, 0)$. If a part of the PSF is cropped, we can look up $scan$ and find out by how much it affects the total sum.

Up to 4

4.3.3 Efficient Greedy strategy

Calculate what each update would lead to what objective. Expensive to calculate.

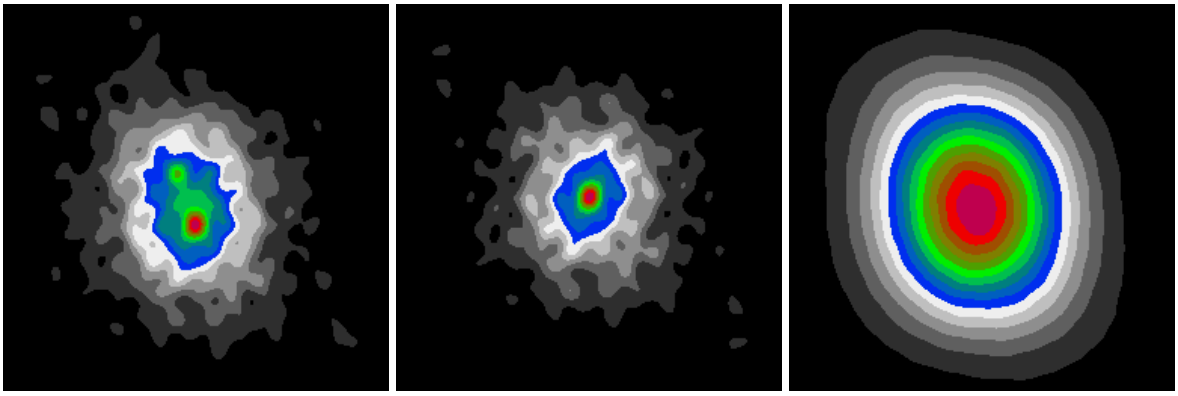
However, we can use another strategy, we use the biggest change in pixel value. A lot cheaper to compute Biggest change in pixel value is in toy examples the same as the best pixel.

Not sure if this is always true.

4.3.4 Pre-calculation of gradients

In each iteration, we need to know the gradient for all pixels. We need to calculate the inner product $\langle I_{res}, PSF \rangle$ for each pixel. Typically, the PSF and the image have the same number of pixels, which leads to a quadratic number of operations to calculate all gradients.

Luckily, we can use the Fourier transform to speed up the calculation. Notice that the inner product $\langle I_{res}, PSF \rangle$ is equivalent to calculating the correlation of the residuals with the PSF ()$I_{res} \star PSF$). The convolution and correlation operators are related: The convolution is equal to a correlation with a flipped kernel. Since a convolution in image space is a multiplication in Fourier space, we can calculate the PSF correlation efficiently in Fourier space.



(a) Dirty Image. *(b) Point Spread Function.* *(c) Gradient for each pixel.*

Figure 8: Example of the gradient calculation.

The Figure 15 shows the process for the first step of the coordinate descent deconvolution. We start with the dirty image. We calculate the correlation of the PSF with the dirty image and arrive at the map of gradients. Figure 15c shows the gradient for each pixel.

4.3.5 Efficient update of gradients

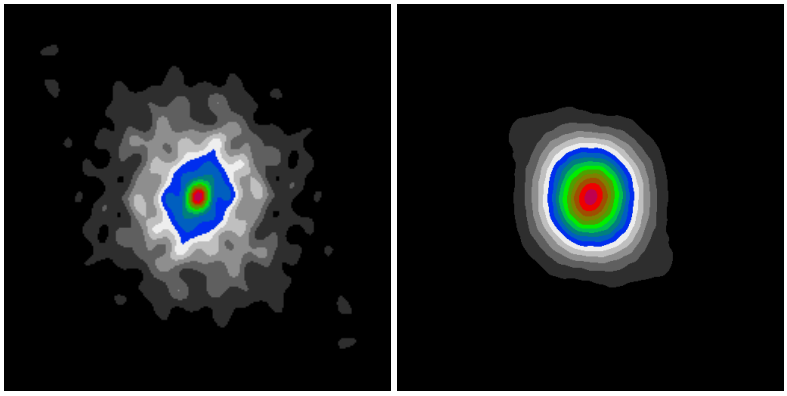
The naive coordinate descent implementation minimizes a single pixel, and updates the residuals by subtracting the PSF at the correct location (The first line of (4.6)). It then calculates the new gradient for each pixel(Second line of (4.6)) in each iteration. This is not necessary. We only need to calculate the correlation in the first iteration. All later iterations update the map gradients directly without the Fourier transform.

$$\begin{aligned} residuals &= residuals - PSF * (optimalValue - oldValue) \\ gradients &= residuals \star PSF \end{aligned} \tag{4.6}$$

The trick is to combine both lines of the naive update (4.6). We correlate each term of the first line with the PSF, and we arrive at the update rule (4.7).

$$gradients = gradients - (PSF \star PSF) * (optimalValue - oldValue) \tag{4.7}$$

The noteworthy part of (4.7) is that we update the gradients directly, but instead of using the PSF, we take the product of $(PSF \star PSF)$, of the PSF correlated with itself. Also note that we do not need to keep the residuals in memory. All we need is the map of gradients.



(a) Point Spread Function. *(b) Gradient update:* $(PSF \star PSF)$.

Figure 9: Example problem with two point sources.

Calculate the PSF correlation with itself. Calculate the PSF correlation once, and use it to update the gradient map directly.

Edges again. We have the problem that, when the PSF is cutoff by the edges of the image, we would need to calculate $(PSF \star PSF)$ again. This is too expensive. Does not change dramatically, unless the pixel we optimize is at the full edge.

What happens in the worst case: we get stuck on a single pixel and do not move any further. Worst case we waste iterations. A major cycle then fixes this problem.

4.4 Similarities to the CLEAN algorithm

Comparison to CLEAN. Similar algorithm, but we descend in the actual gradient direction.

4.5 Pseudo-code of the basic, optimized algorithm

putting it all together

GPU implementation

4.6 Distributed coordinate descent with MPI

How do we distribute the major cycle. We need to distribute every step, Gridding, FFT and Deconvolution.

Gridding, Large number of input data. This needs to be distributed We use the Image domain gridding introduces in sectionand use it as the basis for the distributed gridding.

The FFT is generally not worth distributing, if we can keep all the data in memory. When the gridding is done, in our setup, the grid is small enough to keep in memory. (cite distributed fftw)

Deconvolution is also worth distributing. CLEAN depending on the observation is the second most time consuming step. But gridding tends to be easier to distribute, so in some observations it is the most time consuming step. Split the image into patches and deconvolve each patch. Sadly not possible, we need communication. how we communicate is important.

We use a distributed Gridding and a distributed deconvolution. Which leads us to the following architecture.

Where each node is one computer, i.e. has its own, possibly multiple cpus and its shared memory. Split the input visibilities onto nodes. Do the gridding locally on each node Communicate the grid inverse FFT on one node. Communicate the patches of the image. Deconvolve each patch and communicate

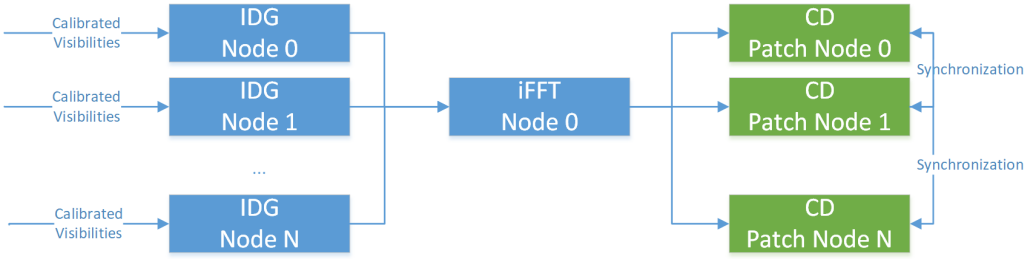


Figure 10: Distributed architecture for half a major cycle

4.7 GPU implementation of coordinate descent

ILGPU

5 Coordinate descent methods

In this section we describe the basic idea behind coordinate descent methods in general, and derive a serial coordinate descent deconvolution algorithm. This algorithm replaces CLEAN in the Major/Minor cycle architecture. The algorithm we describe here is serial in the sense that each step of the algorithm has to finish before the next step can be started. Each individual step can use multiple processors, as we will show with a GPU-accelerated implementation. Later in this work, in Section 8, we will introduce more sophisticated parallel coordinate descent methods.

Remember that a deconvolution algorithm in radio astronomy has three components: A numerical optimization algorithm, an objective function and a regularization. We use a serial coordinate descent method for the optimization algorithm. The objective function is:

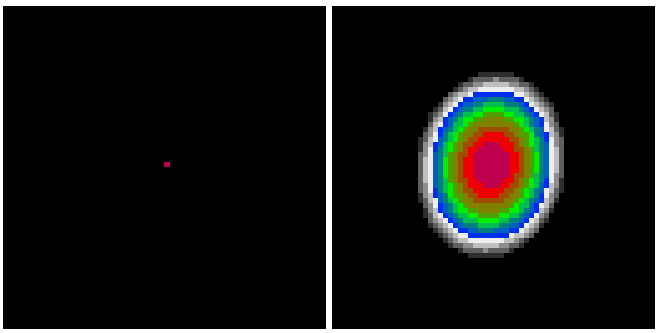
$$\underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - x * PSF\right\|_2^2 + \lambda ElasticNet(x) \tag{5.1}$$

The objective consists of two parts: The data term $\|I_{dirty} - x * PSF\|_2^2$ and the regularization term $ElasticNet(x)$. The data term forces the image to be as close to the measurements as possible which forces the image to be as close to the measurements as possible, the regularization term forces the image to be as consistent as possible with our prior knowledge. The parameter λ is a weight that either forces more or less regularization. It is left to the user to define λ for each image.

And finally, the regularization we use is elastic net. We first go into more detail what the elastic net regularization is and how it influences the image. We then derive the serial coordinate descent method that minimizes the objective (9.1) in Section 5.2, and continue with its efficient implementation.

5.1 Elastic net regularization

This regularization is a mixture between the L1 and L2 regularization. The L1 regularization is simply the absolute value of all pixels, and the L2 norm is the squared sum of all pixels. he Figure 11 shows the effect of the L1 and L2 norm on a single star. The L1 regularization forces the image to contain few non-zero pixels as possible. It encodes our prior knowledge that the image will contain stars. The L2 regularization on the other hand ”spreads” the single star across multiple pixels.



(a) Effect of the pure L1 norm (λ = 1.0) on a single point source.

(b) Effect of the pure L2 norm (λ = 1.0) on a single point source.

Figure 11: Effect of the L1 and L2 Norm separately.

The L1 regularization alone models an image that consists of points sources. For extended emissions like hydrogen clouds, the L1 regularization often leads the reconstructed image to become a cluster of point sources, instead of a real extended emission.

The L2 norm alone was already used in other image reconstruction algorithms in radio astronomy[19], with the downside that the resulting image will not be sparse. I.e. all pixels in the reconstruction will be non-zero, even though all they contain is noise.

Elastic net mixes the L1 and L2 norm together, becoming ”sparsifying L2 norm”. It retains the sparsifying property of the L1 norm, while also keeping extended emissions in the image. Formally, elastic net regularization is defined as the following regularization function:

$$ElasticNet(x, \alpha) = \alpha \|x\|_1 + \frac{1-\alpha}{2} \|x\|_2 \tag{5.2}$$

The parameter α is between 0 and 1, and mixes the two norms together. A value of 1 leads to L1 regularization only, and a value of 0 leads to L2 only. The elastic net regularization has two properties, which are relevant later for the serial coordinate descent deconvolution: It is separable, and has a proximal operator.

Separability means that we can calculate the elastic net regularization penalty independently for each pixel. We arrive at the same result if we evaluate (5.2) for each pixel and sum up the results, or if we evaluate (5.2) for the whole image. This is an important property when one tries to minimize the the elastic net penalty (as we will with the serial coordinate descent deconvolution algorithm). We can also calculate how much any pixel change reduces the elastic net penalty independently its neighbors.

The proximal operator of elastic net allows us to minimize the regularization penalty. Notice that the elastic net regularization (5.2) is not differentiable (the L1 norm is not continuous). We cannot calculate a gradient, and cannot use methods like gradient descent to minimize the regularization penalty. However, it has a proximal operator defined:

$$ElasticNetProximal(x, \lambda, \alpha) = \frac{max(x - \lambda\alpha, 0)}{1 + \lambda(1-\alpha)} \tag{5.3}$$

In our deconvolution problem, we can apply the proximal operator (5.3) on each pixel, and we minimize the elastic net penalty. Again, the proximal operator can be applied on each pixel independently, as neighboring pixels do not influence its result.

The elastic net regularization is separable. We can calculate its penalty for each pixel independently of its neighbors. As such, its proximal operator is also independent of the neighbors. It is the only regularization we use in this project. We now derive a coordinate descent based deconvolution algorithm that uses the proximal operator to efficiently reconstruct an image.

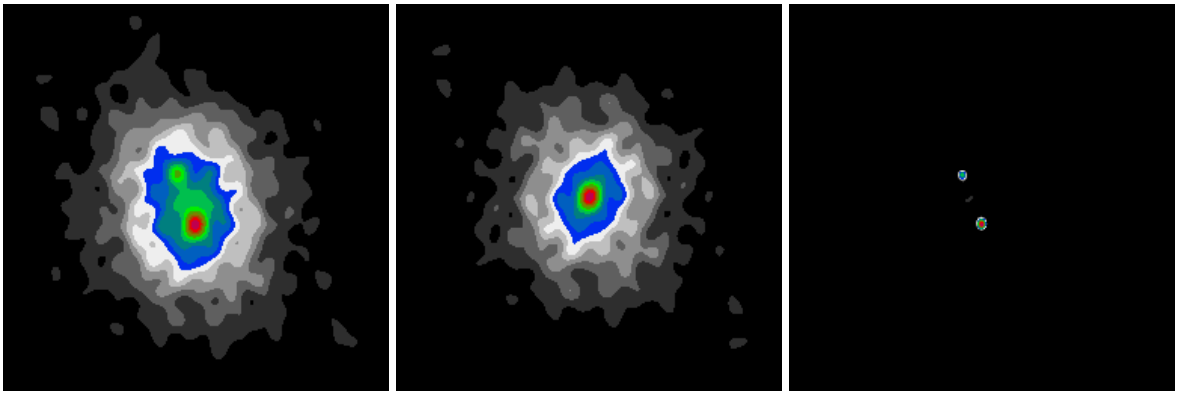
A side note on the proximal operator used in this project (5.3): The numerator always clamps negative pixels to zero. This is a conscious design decision. In radio astronomy, it is usual to constrain the reconstruction to be non-negative (because we cannot receive negative radio emissions from any direction). It is widely used in radio astronomy image reconstruction and may lead to improved reconstruction quality [35].

5.2 Serial coordinate descent deconvolution

Our serial coordinate descent deconvolution algorithm minimizes the deconvolution objective (9.1). It is a convex optimization algorithm that optimizes a single pixel (coordinate) at each iteration. Each iteration consists of two steps. Step 1: Find the best pixel to optimize. Step 2: Calculate the gradient for this pixel, take a descent step and apply the elastic net proximal operator. We repeat these steps in each iteration until coordinate descent converges to a solution.

We demonstrate the serial deconvolution algorithm with the help of a simulated MeerKAT reconstruction problem of two point sources. Figure 12a shows the dirty image of two point sources, and Figure 12b the PSF.

The deconvolved image with elastic net regularization is shown in Figure 12c. I.e. Figure 12c is the optimum x of the objective function (9.1).



(a) Dirty Image. *(b) Point Spread Function.* *(c) Elastic net reconstruction*

Figure 12: Example problem with two point sources.

The serial coordinate descent method finds the optimum over several iterations. Note that we termed the algorithm ”serial”, because step 1 (finding a pixel) first has to finish before we can continue with step 2 (optimizing the current pixel). There is always only one pixel which gets optimized at any given time.

Also note that our implementation calculates the gradient for the current pixel. This may raise the question: What exactly is the difference between gradient- and coordinate descent is that gradient descent optimizes all pixels in each iteration, while coordinate descent optimizes (generally) a single pixel at a time[7]. Also, Coordinate descent methods are not bound to use the gradient. It could use a line search approach, where we try different values and decide on the one leading to the lowest objective value.

Because coordinate descent methods only optimize a single pixel at a time, they generally need a large number of iterations to converge compared to other methods. But when each iteration is cheap to compute, coordinate descent methods can converge to a result in less time than competing methods[32, 33]. We discuss the efficient implementation in Section 5.3. First, we explain each step in the serial coordinate descent algorithm in more detail.

5.2.1 Step 1: Choosing single a pixel

Our serial coordinate descent algorithm uses a greedy strategy. From all possible pixels, it searches the pixel whose gradient has the largest magnitude. In each iteration, it chooses the pixel which reduces the objective function the most. There are two other strategies that are used in coordinate descent: Random and Cyclic.

A random strategy chooses, as the name implies, each pixel at random. Usually, the pixels are chosen from a uniform distribution. The greedy strategy leads to cheaper iteration compared to the greedy strategy, because we do not check the gradient of each pixel.

A cyclic strategy iterates over a subset of pixels until the subset converges. It then chooses another subset. Each iteration of the cyclic strategy is also cheaper than the greedy strategy.

For our image deconvolution problem, we choose the greedy strategy. Even though it is more expensive, it tends to be faster to converge. Our reconstructed image is sparse, meaning most pixels in the image will be zero. The greedy strategy tends to iterate on pixels which will be non-zero in the final reconstructed image. While the random or cyclic strategy add pixels in the intermediate image that will eventually have to

[7] There are block coordinate descent methods that optimize a block of coordinates at each iteration. They are also discussed together with parallel coordinate descent methods in Section 8

be removed. For the deconvolution problem, removing pixels from an intermediate solution seems to slow down convergence significantly. We explore the different strategies systematically for the parallel coordinate descent methods.

5.2.2 Step 2: Optimizing a single pixel

At this point, the greedy strategy has selected a pixel at a location to optimize. Now, our deconvolution objective (9.1) is reduced to a one dimensional problem:

$$\underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - x_{location} * PSF\right\|_2^2 + \lambda ElasticNet(x_{location}) \tag{5.4}$$

Side note: The reason why this reduces nicely to one dimension is the elastic net regularization is separable. I.e. the regularization can be calculated independently of the surrounding pixels.

Optimizing the one dimensional problem (5.4) is a lot simpler. In essence, we calculate the gradient for the pixel at the selected location, and apply the proximal operator of elastic net. First, let us look at how the gradient is calculated and ignore the regularization. The gradient arises from the data term of the one dimensional objective (5.4) ($\|I_{dirty} - x_{location} * PSF\|_2^2$). After simplifying the partial derivative, we arrive at the calculation:

$$\begin{aligned} residuals &= I_{dirty} - x * PSF \\ gradient_{location} &= \langle residuals, PSF_{location} \rangle \\ Lipschitz_{location} &= \langle PSF_{location}, PSF_{location} \rangle \\ pixel_{opt} &= \frac{gradient_{location}}{Lipschitz_{location}} \end{aligned} \tag{5.5}$$

First, we calculate the residuals by convolving the current solution x with the PSF. Then, the gradient for the selected pixel location is the inner product(element-wise multiplication followed by a sum over all elements) of the residuals and the PSF, shifted at the current location. calculate the gradient for the selected location. The next step is to calculate the Lipschitz constant at the current location. Finally, we arrive at the optimal pixel value by dividing the gradient by the Lipschitz constant. Note, that we currently ignore the elastic net regularization.

The Lipschitz constant describes how fast a function $f(x)$ changes with x. If $f(x)$ changes slowly, we can descend larger distances along the gradient without the fear for divergence. The Lipschitz constant can be looked at as a data-defined step size.

An interesting point is that the update rule $pixel_{opt} = \frac{gradient_{location}}{Lipschitz_{location}}$ finds the optimal pixel value, if the pixel is independent. Remember that our objective function is convex. The data term of our one dimensional objective (5.4) actually forms a parabola, with the parameters: $x^2\langle PSF, PSF\rangle - 2x\langle resdiuals, PSF_{location}\rangle + c$. Dividing the gradient of the pixel location with the Lipschitz constant is identical to calculating the optimum of the parabola $\frac{-b}{2a}$, where $b = -2gradient_{location}$ and $a = Lipschitz_{location}$.

This means if our reconstruction problem has point sources which are far away, such that their PSFs do not overlap, then the update rule finds the optimal value for each point source with one iteration. But when the PSFs overlap as in our example problem, shown in Figure 12, then we need several iterations over the same pixel until coordinate descent converges.

Including the elastic net regularization
So far, we ignored the regularization. We can calculate the optimal pixel value without elastic net regulariza-

tion. The last step is to combine the proximal operator of the elastic net regularization (5.3) with the gradient calculation, and we arrive at the following update step:

$$pixel_{opt} = \frac{max(gradient_{location} - \lambda\alpha, 0)}{Lipschitz_{location} + (1-\alpha)\lambda} \tag{5.6}$$

This update rule now finds the optimal pixel value with elastic net regularization. If the PSFs do not overlap, we still only need one iteration per source.

5.2.3 Inefficient implementation pseudo-code

Now we put together our serial coordinate descent algorithm, and show where the bottleneck lies. In each iteration, the serial coordinate descent algorithm selects the pixel with the maximum gradient magnitude, and optimizes the selected pixel with the update rule (5.6).

```
1  dirty = IFFT(GridVisibilities(visibilities))
2  residuals = dirty
3
4  x = new Array
5  objectiveValue = 0.5* Sum(residuals * residuals) + ElasticNet(x)
6
7  do
8    oldObjectiveValue = objectiveValue
9
10   //Step 1: Search pixel
11   pixelLocation = GreedyStrategy(residuals, PSF)
12   oldValue = x[pixelLocation]
13   shiftedPSF = Shift(PSF, pixelLocation)
14
15   //Step 2: Optimize pixel
16   gradient = Sum(residuals * shiftedPSF)
17   lipschitz = Sum(shiftedPSF * shiftedPSF)
18   tmp = gradient + oldValue * lipschitz
19   optimalValue = Max(tmp - lambda*alpha) / (lipschitz + (1 - alpha)*lambda)
20
21   //housekeeping
22   x[pixelLocation] = optimalValue
23   residuals = residuals - shiftedPSF * (optimalValue - oldValue)
24   objectiveValue = 0.5 * Sum(residuals * residuals) + lambda * ElasticNet(x, alpha)
25 while (oldObjectiveValue - objectiveValue) < epsilon
```

The actual update step (line 19) is cheap to compute. We are only dealing with 4 one dimensional variables. The expensive calculations are the inner products. The gradient calculation, the Lipschitz constant and the objective value. The residuals and PSF generally contain millions of pixels. Calculating the inner product of those becomes expensive.

Also note that the greedy strategy needs to calculate the gradient for each pixel. As it is, the greedy strategy has a quadratic runtime complexity.

5.3 Efficient implementation

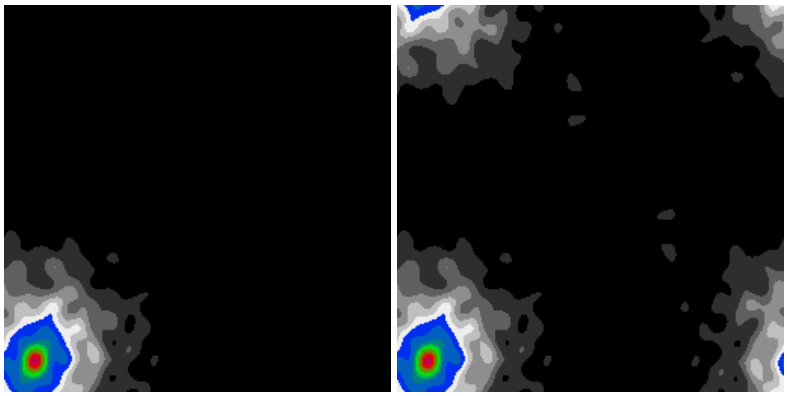
The bottleneck of the serial coordinate descent algorithm are all the inner products that need to be calculated in each iteration. In each iteration, we need to know the gradient for every pixel, and the Lipschitz constant of the current pixel. Luckily, we can cache a map of gradients, where we save the gradient for every pixel and

skip all of the inner products associated with the gradient. Also, we can efficiently calculate and cache the Lipschitz constants. We can greatly reduce the runtime cost for each iteration.

This section shows the implementation details on how we can calculate the map of gradients and the Lipschitz constant efficiently. But first we need to define another implementation detail: How we handle the edges of the convolution.

5.3.1 Edge handling of the convolution

As the reader is probably aware, there are several ways to define the convolution in image processing, depending on how we handle the edges on the image. Two possibilities are relevant for radio interferometric image reconstruction: Circular and zero padded.



(a) Zero padded convolution. *(b) Circular convolution.*

Figure 13: Comparison of the two convolution schemes.

Circular convolution assumes the image ”wraps” around itself. If we travel over the right edge of the image, we arrive at the left edge. The convolution in Fourier space is circular. Remember: A convonlution in image space is a multiplication in Fourier space, and vice versa. When we convolve the reconstructed image x with the PSF using circular convolution, then non-zero pixels at the right edge of the image ”shine” over to the left edge. This is physically impossible.

Zero padding assumes that after the edge, the image is zero. Non-zero pixels at the right edges of the image do not influence the left edge after convolution. This is the physically plausible solution. However, the zero padded convolution is more expensive to calculate. We either have to calculate the convolution in image space, which is too expensive for large kernels, or apply the FFT on a zero-padded image. Either way, it is more expensive than the circular convolution.

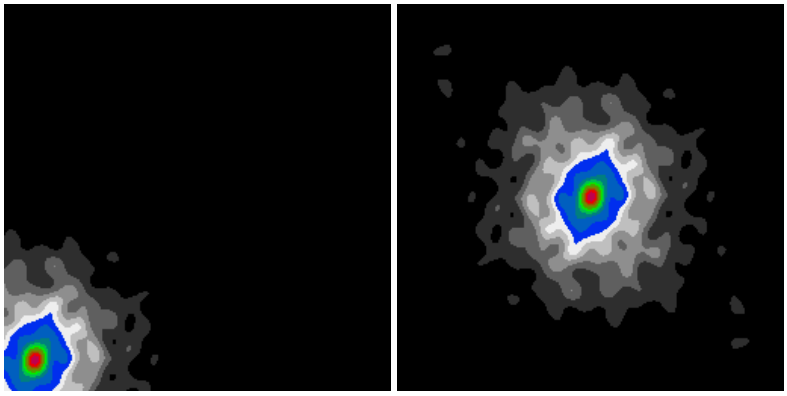
In designing a deconvolution algorithm, we have the choice between the circular and the zero-padded convolution scheme. Circular convolution is more efficient to calculate, while zero-padded convolution is closer to the reality. Both choices are possible. Some implementations leave this choice to the user [38]. We decide on using the zero-padded convolution. This choice influences how we calculate the Lipschitz and gradients efficiently.

5.3.2 Efficient calculation of the Lipschitz constants

Quadratic runtime.

We can do better We can be linear.

Scan algorithm



(a) Shifted PSF. *(b) Sum of squared values.*

Figure 14: Comparison of the two convolution schemes.

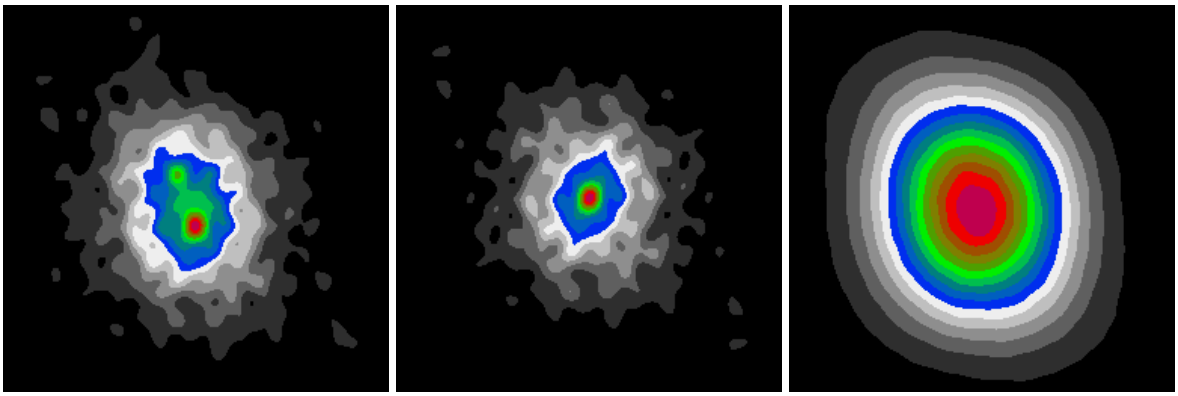
```
1  var scan = new double[,];
2  for (i in (0, PSF.Length(0))
3  {
4  for (j in (0, PSF.Length(1))
5  {
6  var iBefore = scan[i - 1, j];
7  var jBefore = scan[i, j - 1];
8  var ijBefore = scan[i - 1, j - 1];
9  var current = PSF[i, j] * PSF[i, j];
10 scan[i, j] = current + iBefore + jBefore - ijBefore;
11 }
12 }
```

Four evaluations of $scan[]$

5.3.3 Using a map of gradients

replace the residuals

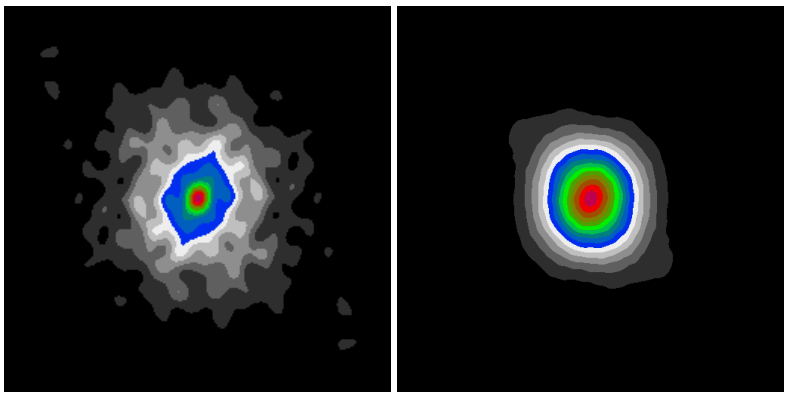
Efficient calculation



(a) Dirty Image. *(b) Point Spread Function.* *(c) Gradient for each pixel.*

Figure 15: Example of the gradient calculation.

Efficient update



(a) Point Spread Function. *(b) Gradient update:* $(PSF \star PSF)$.

Figure 16: Example problem with two point sources.

6 PSF approximation for parallel and distributed deconvolution

For a distributed deconvolution, we would like to deconvolve the image with as little communication as possible. This largely depends on the size of the PSF when compared to the overall image. If the PSF is for example $\frac{1}{16}$ of the total image size, we have patches of the image which are completely independent of each other. Sadly, this is not true for radio interferometers. The PSF is generally the same size as the image. We cannot deconvolve any part of the image independently of each other.

However, we have two effects of modern radio interferometers, that produce an "approximately" smaller PSF: First, we have an increasing number of visibilities. They create a PSF that increasingly resembles a Gaussian function in the center, and the rest approaches zero. And secondly, we reconstruct images with a wide field-of-view. Although the PSF is not zero the further away we move from the center, its values approach zero.

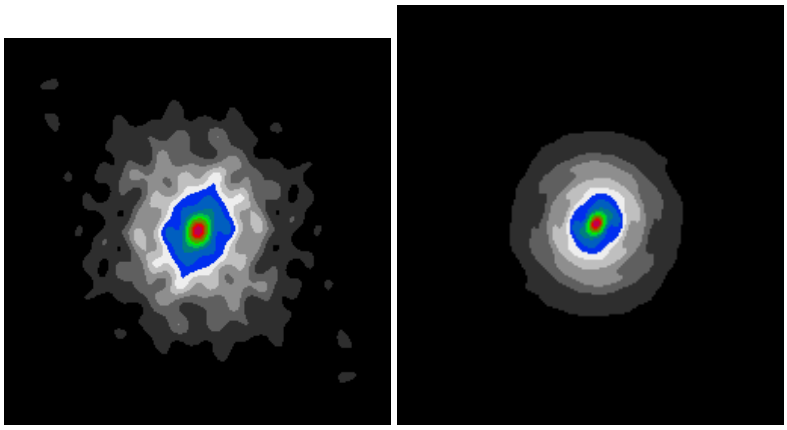


Figure 17: PSF arising from an increasing number of visibilities.

In short, with an ever increasing number of visibilities and field-of-view, the influence of far-away image sections become negligible. We can approximate the deconvolution with a fraction of the true PSF. To our knowledge, we are the first to propose such approximation methods. In this Section, we present our approximation methods. In Section 7.3, we empirically demonstrate the validity of our approximations on a real-world MeerKAT observation. In Section 8, we show more sophisticated coordinate descent methods that can exploit the smaller PSF.

6.1 Intuition for approximating the PSF

Our basic coordinate descent algorithm chooses a pixel to minimize, calculates its gradient and descends in that direction. The gradient calculation reduces itself to a correlation of the residuals with the PSF at the pixel location. In other words, we need the PSF to calculate a gradient. If we only use parts of the PSF for the calculation, we essentially approximate the gradient for the pixel. Because the PSF only has significant non-zero values in the center, we should be able to ignore most of the values and still have an adequate gradient approximation.

Furthermore, our basic coordinate descent algorithm reconstructs inside the Major/Minor cycle framework. The framework is designed to handle only an approximate PSF in the deconvolution (Remember: the w-term changes the PSF depending on the position in the image). The framework may be able to deal with further PSF approximations, namely with a PSF that is reduced in size, which makes the distributed deconvolution simpler.

The approximation methods essentially work by "cutting off" the less significant PSF values and only use a rectangle around the center. If we cut off the PSF by a factor of $\frac{1}{4}$ (If the PSF is 1024^2 pixels in size, we only use a rectangle of 256^2 of the center), we get pixels in the reconstructions that are not influenced

by each other. Starting from a cut-off factor of $\frac{1}{16}$, we can split the image into eight patches, four of which are independent of each other. This would allow us to run up to four basic coordinate descent algorithms in parallel, simplifying the distribution.

Indeed, it is possible to approximate the PSF with only a fraction of its true size, as we will demonstrate empirically in Section 7.3. But an approximation of the PSF may lead to other problems in the reconstruction:

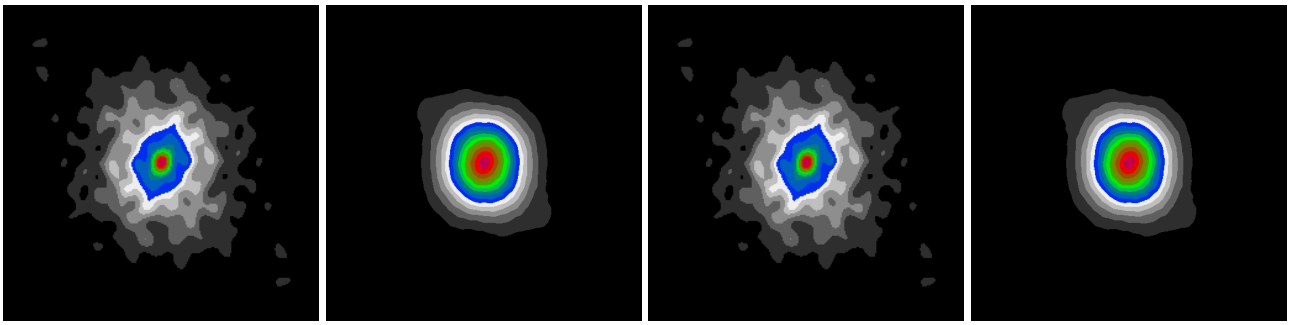
- Needs additional Major cycles to converge.
- Slow down convergence speed of coordinate descent.
- Not guaranteed to converge to the same image.

The Major cycle corrects the errors the approximate PSF introduces. The more inaccurate the PSF is, the more Major cycles we may need to converge. As we already discussed, a Major cycle is an expensive operation. Our PSF approximation should lead to as few (if any) additional Major cycles.

For the basic coordinate descent algorithm, an approximate PSF may slow down the convergence speed. In each iteration, the basic algorithm finds the optimal value for the current pixel. With an approximate PSF, we may need several iterations on the same pixel (over several major cycles) until we arrive at the same value. In short, an approximate PSF can slow down the convergence speed of coordinate descent.

Depending on how we approximate the PSF, we may not have any guarantee that we arrive at the same result. We developed two approximation methods: Method 1 uses an approximation in the gradient update step of coordinate descent. Method 2 solves an approximate deconvolution problem with only a fraction of the PSF. Only method 1 is guaranteed to converge to the same solution (with enough major cycles), but is slower to converge than method 2. Depending on the method we use, we can remedy some of the problems that approximating the PSF introduces. But there seems to be a trade-off to be made. We have not found a method that works best in every aspect.

6.2 Method 1: Approximate gradient update



(a) PSF *(b) Update map* *(c) $cut(PSF)$* *(d) Approx. update*

Figure 18: Approximation of gradient update.

The basic coordinate descent method first correlates the residuals with the PSF. It pre-calculates the gradient for each pixel. Then, in each iteration, it directly updates the map of gradients with the product of $PSF \star PSF$ (the PSF correlated with itself). In this approximation method, we start from the same pre-calculated map of gradients, but use an approximate update. The first coordinate descent iteration of this approximation method is identical to coordinate descent with the full PSF. With each coordinate descent iterations, the gradient map becomes more inaccurate. But with enough major cycles, this method converges to the same result as when the full PSF is used.

The question is, how do we approximate the product of $PSF \star PSF$. As we have seen before, the product of $PSF \star PSF$ also approaches zero away from the center (an example was shown in Figure 16 in Section

4.3.5). A naive way to approximate the update step is to cut off"the insignificant value and only use a rectangle of the center, which is a fraction of the total image size. For example: An image of size 1024^2 also has a PSF the size of 1024^2. The product $PSF \star PSF$ actually has the size of 2048^2 pixels due to the correlation. We can try to approximate the product by only using the center rectangle $\frac{1}{8}$ of the total size, (256^2 pixels). This approximation works, but leads to artifacts during deconvolution: The image will be reconstructed in visible "blocks" which are the size of the center fraction we use.

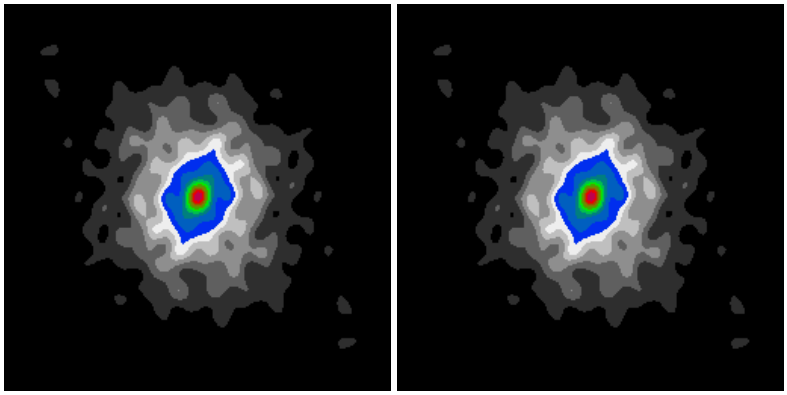
We use the approximation shown in equation (6.1), where $cut()$ is the function that cuts away everything but the center rectangle of the PSF. This is a better approximation than cutting the product of $PSF \star PSF$ directly, and leads to faster convergence.

$$PSF \star PSF \approx cut(PSF) \star cut(PSF) \tag{6.1}$$

The reason why (6.1) is a better approximation lies in the reason why we update the gradients with the product of $PSF \star PSF$ in the first place: It is the combination of two separate operations, removing the PSF from the residuals at the current position, and recalculating the correlation with the PSF. If we cut away parts of the product $PSF \star PSF$ directly, we implicitly update the residuals with a different PSF. But when we approximate the product by (6.1), we ensure that the implicit removal of the PSF from the residuals is equal to $cut(PSF)$.

In our implementation, we use one more trick to improve the approximation: we scale the product of $cut(PSF) \star cut(PSF)$ to have the same maximum as the original product $PSF \star PSF$. The approximation has a lower maximum value than the original. Over several coordinate descent iterations, we run into the danger of overestimating the pixel values. By scaling the product of $cut(PSF) \star cut(PSF)$ to the same maximum, we end up with a better approximation of the true gradient update.

6.3 Method 2:Approximate deconvolution



(a) PSF on log scale *(b) PSF Approximation*

Figure 19: Approximate deconvolution with a fraction of the PSF.

The main problem with Method 1 is that the map of gradients becomes less accurate with more coordinate descent iterations. This method sovles this problem by using an approximate deconvolution instead, but it loses the guarantee to converge to the same result as the full PSF.

This method cuts off the insignificant part of the PSF, and only uses the center rectangle of it for the whole deconvolution problem. As such, the coordinate descent method solves an approximate deconvolution problem

shown in (6.2).

$$\underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - x * cut(PSF)\right\|_2^2 + \lambda ElasticNet(x) \tag{6.2}$$

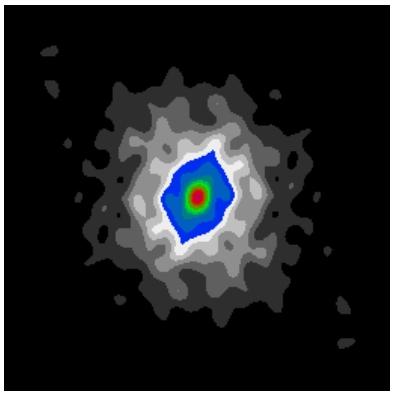
In essence, it uses the same basic coordinate descent method, but ignore large parts of the PSF. It pre-calculates the gradient map by correlating the residual image with $cut(PSF)$, and update the gradient map with the product of $cut(PSF) \star cut(PSF)$. The main difference between approximate gradient update and approximate deconvolution is: In approximate gradient update starts with the same gradient map as the original. The approximate deconvolution does not. With this method, the gradient map does not become more inaccurate with more coordinate descent iterations.

The approximate deconvolution objective (6.2) is not guaranteed to "point" to the same solution as the original. It may in reality point to a very different solution. But thanks to the Major cycle, the image retrieved by optimizing (6.2) is always "close" to the original solution.

Nevertheless, this approximation method introduces an error in the final reconstructed image. The obvious error it introduces is it under-estimates the true pixel values. Pre-calculating the gradient map with $cut(PSF)$ under-estimates gradient magnitudes, and by extend the pixel values.

To combat the under-estimation of pixel values, we reduce the regularization parameter λ for the approximate deconvolution problem. Since we cut off parts of the PSF, we also reduce the Lipschitz constant (sum of the squared PSF values) used in the approximate deconvolution. We reduce the λ parameter by the same factor that the Lipschitz constant gets reduced. This ensures that the approximate deconvolution and the original deconvolution arrive at the same pixel value for a point source. But it does not completely remove the issue for extended emissions.

6.4 Major Cycle convergence and implicit path regularization



(a) PSF cutoff

Figure 20: Max sidelobe PSF.

Problem of using a reduced PSF. We have sidelobes.

Danger of adding them in the deconvolution, just to remove them in the next major cycle (when it is clear that they were bogus). With more extreme approximations, this leads to oscillations (add bogus pixels in one Major cycle, remove them in the next, just to add them again in the cycle after). And in most extreme cases, it can even lead to divergence.

Sidelobe equals to largest absolute PSF value we cut off

We know what the biggest sidelobe is we cut off. That way we can estimate what magnitude of pixel we can be sure that it is bogus.

$$\begin{aligned} psfSidelobe &= MAX(PSF - cut(PSF)) \\ gradients &= residuals \star PSF \\ gradientLimit &= MAX(gradients) * psfSidelobe \end{aligned} \tag{6.3}$$

Lower bound estimate. At which point we are sure it is bogus. This is only true for point sources. problem of correlation, extended emissions. With extended emissions in the image, we still take up bogus pixels.

Estimate a correlation factor.

Coordinate Descent Path regularization optimization [34]

[27] and the question on how many iterations per major cycle Putting it all together

We have the Minor Cycle, which is easy to converge.

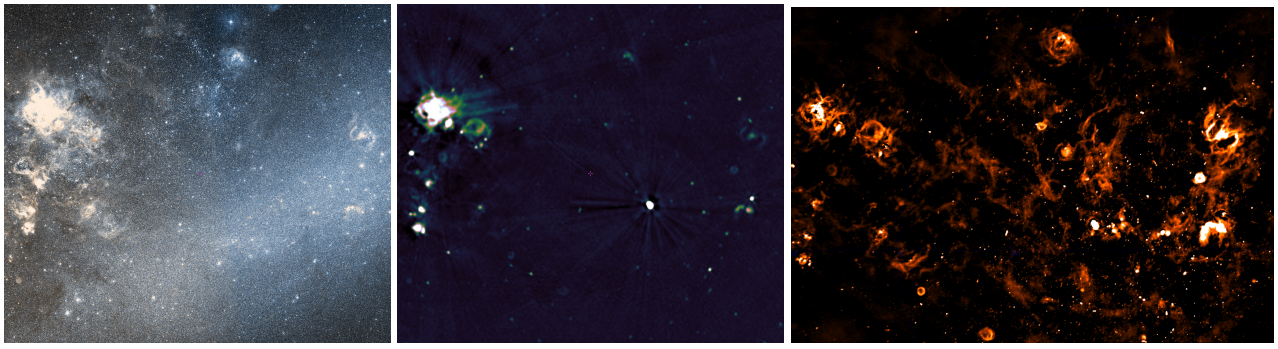
Danger that CD takes too many pixel into a Major Cycle. Lower bound per iteration, PSF sidelobe can still be too low, danger when many psf sidelobes overlap

7 Tests on MeerKAT LMC observation

The Large Magellanic Cloud (LMC) is a galaxy is the second or third closest galaxy to the Milky Way. Figure 21 shows the LMC in both optical and radio wavelenghts. The radio wavelengths was observed by the VLA radio interferometer[39] at 843MHz. In the optical wavelengths, the abundance of stars are clearly visible. The LMC is close enough to earth for individual stars are visible. But it also contains a large number of supernova remnants, gas clouds, and other extended emissions, which shine bright in the radio wavelengths.

The LMC is a region with a large number of sources at different brightness. In the lower-right quadrant of the radio-image 21b, we see the bright emission of the supernova remnant N132D, the brightest radio source in the LMC. But around the N132D are faint emissions from gas-clouds. This means faint emissions may get lost next to N132D. We need a deconvolution algorithm to uncover these faint emissions.

We received a MeerKAT observation of the LMC from SARAO for the purpose of algorithm testing. At the time of writing, the MeerKAT instrument is still being tested. The observation is only representative in the data volume. The observation is calibrated, and averaged down in both frequency and time. The averaging reduces both the disk space and the runtime costs of the gridding step. Nevertheless, the observation takes up over 80 GB of disk space (roughly $\frac{1}{30}$ of the original data). A CLEAN reconstruction of the calibrated observation is shown in Figure 21c.



(a) Optical wavelength *(b) Radio wavelength at 843MHz.* *(c) Wide band radio image by MeerKAT.*

Figure 21: Section of the Large Magellanic Cloud (LMC)

The MeerKAT observation covers a wide band of radio frequencies. The lowest frequency in the MeerKAT observation is 894 MHz, and the highest frequency is 1658 Mhz. Imaging the whole frequency band requires a wide band deconvolution algorithm. In wide band imaging, several images at different frequencies get deconvolved as an image cube. Wide band imaging again multiplies the amount of work that has to be done for reconstruction, as now we cannot deconvolve a single image, but have to deal with a whole image cube.

Wide band imaging is not possible within the time frame of this project. We take a narrow band subset of 5 channels from the original data (ranging from 1084 to 1088 MHz, about 1 Gb in size) for reconstruction. We also reduce the field-of-view to a more manageable section. Figure 22 shows the LMC image section we are using together with a CLEAN reconstruction of the narrow band data.

At the center of our image section 22 we see the N132D supernova remnant. We partially see the faint extended emissions, although they are close to the noise level. This is known as a high-dynamic range reconstruction. We have strong radio sources mixed together with faint emissions, which are only marginally above the noise level of the image.

The total field-of-view of our image section is roughly 1.3 degrees(or 4600 arc seconds). Our reconstruction has 3072^2 pixel with a resolution of 1.5 arc seconds per pixel. this is still a wide field-of-view reconstruction problem. We have to account for the effects of the w-term to achieve a high-dynamic range reconstruction.

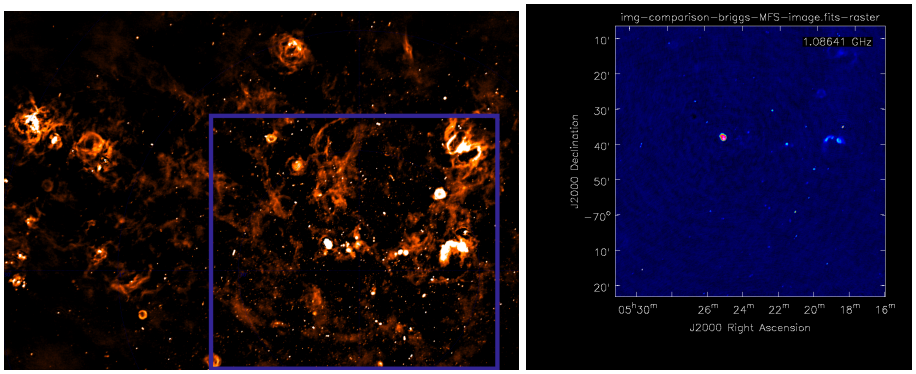


Figure 22: Narrow band image section used.

In our test reconstruction, we need to account for w-term correction and high-dynamic range. We have excluded wide-band imaging as not feasible within the time frame of this project. In Section 7.1 we compare the reconstructions of CLEAN with our coordinate descent based algorithm on the LMC observation. The next Section 7.2 presents the speedup we achieve with coordinate descent by using our distributed or GPU-accelerated implementations.

In Section 7.3 we show the core result of this project. Namely what effect has an approximate PSF on the deconvolution problem and whether we can use it to further distribute the problem. The answer to that question is affirmative: We can approximate the PSF, and we can exploit it to further distribute the deconvolution. But we need more sophisticated coordinate descent algorithms to fully benefit from it.

7.1 Comparison with CLEAN reconstructions

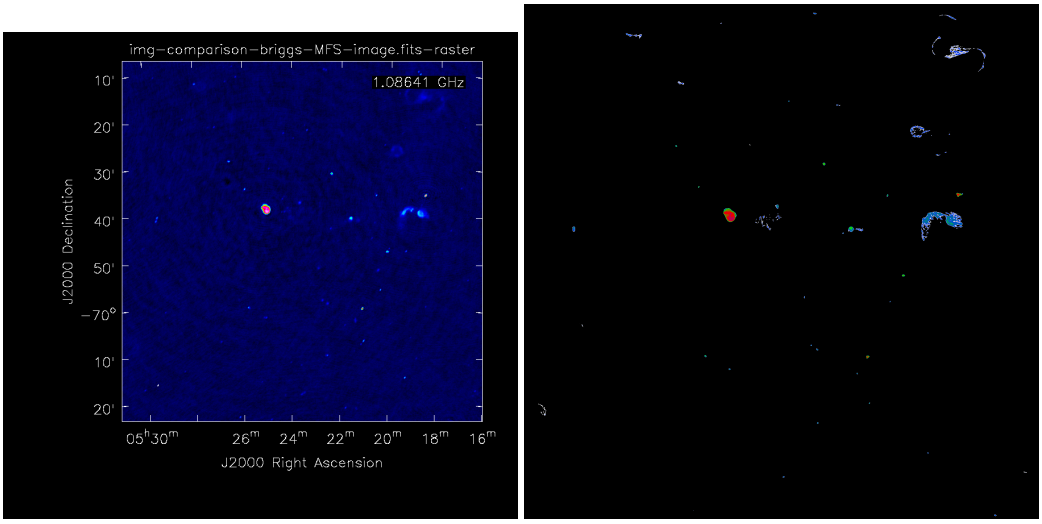
We use the WSCLEAN [40] implementation of multi-scale CLEAN. We compare our coordinate descent reconstruction with two CLEAN reconstructions, oe with naturally weighted visibilities and one with briggs weighted visibilities.

There are three main visibility weighting scheme for the gridder that lead to different PSFs from the same measurements: Natural, uniform, and Briggs[41]. Natural weighting scheme leads to an image with a lower noise level, but a wider PSF. Uniform weighting leads to a higher noise level, but to a PSF whgich is more concentrated around a single pixel. Briggs weighting is a scheme combines the best from both worlds, receiving an image with acceptable noise level while getting a more concentrated PSF. As such it is widely used in radio astronomy image reconstruction. Our gridder implements the natural weighting scheme only. Nevertheless our coordinate descent algorithm is able to retrieve structures similar to the briggs-weighted multi-scale CLEAN reconstruction, even though coordinate descent has to work with a wider PSF.

Figure 23 shows the reconstruction of both briggs-weighted multi-scale CLEAN and the naturally weighted coordinate descent reconstruction. CLEAN used 6 major cycles and 14 thousand minor cycle iterations. Our coordinate descent implementation converged after 5 major cycles and needed 100 thousand iterations to converge.

Coordinate descent needs a large number of iterations to converge when compared to multi-scale CLEAN. Note that a coordinate descent iteration is cheaper to compute than one iteration of multi-scale CLEAN. Also note that because we are searching for structures close to the noise level of the image, coordinate descent often adds pixels belonging to the noise in one major cycle, just to remove them in the next one. Path regularization[34] can combat this problem, and gets further investigated in the following Section 7.3.

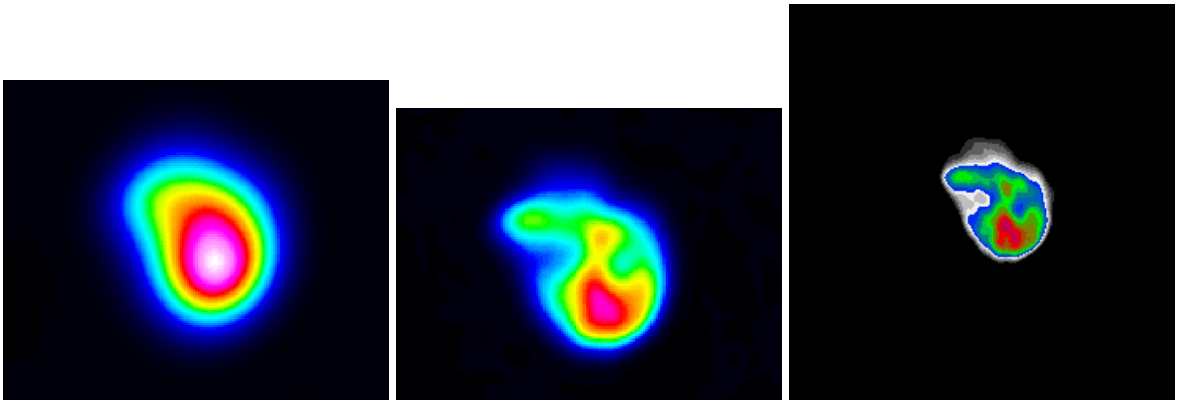
Both algorithms detect the three extended emissions at the right side of the image. They detect various point sources at the same location. Coordinate descent and multi-scale CLEAN arrive at a roughly similar



(a) Briggs weighted multi-scale CLEAN. *(b) Naturally weighted coordinate descent.*

Figure 23: Comparison of the whole image

result. Coordinate descent detects similar structures in the N132 supernova remnant, as the briggs-weighted CLEAN, but also includes calibration errors in its reconstruction of the faint extended emissions.



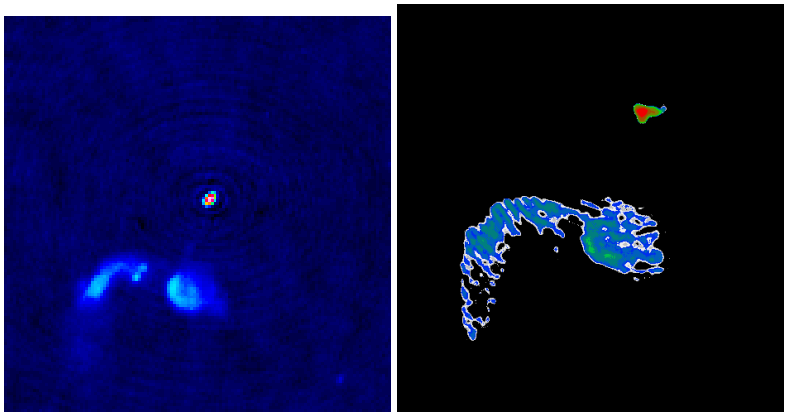
(a) CLEAN Natural weighting. *(b) CLEAN Briggs weighting.* *(c) CD Natural weighting.*

Figure 24: N132 comparison

Figure 24 compares the naturally-weighted CLEAN, briggs CLEAN and coordinate descent on the N132 supernova remnant. The naturally-weigted CLEAN and coordinate descent use the same PSF for the deconvolution. But coordinate descent finds structures in N132 similar to the briggs-weighted CLEAN. Coordinate descent arrived at a plausible higher-resolved reconstruction of N132.

Calibration errors on the other hand negatively influence the coordinate descent reconstruction. Figure 25 shows a cutout of the right hand section of the reconstruction, where a faint extended emission is next to a point source with calibration errors. Multi-scale CLEAN is able to differentiate between the ”ripples” from the calibration error, and the signal from the extended emission. Coordinate descent with the elastic net regularization includes the ripples into the reconstructed image.

The only way to exclude the ripples from the reconstruction is to increase the regularization parameter λ., such as no pixel gets included which is not above the noise level + calibration error in the image. However, that would lead to other sources being ”regularized away” in other regions of the image, which do not have a severe calibration error close by.



(a) Briggs CLEAN *(b) Coordinate Descent*

Figure 25: Influence of calibration errors

7.2 Coordinate descent acceleration with MPI or GPU

Describe hardware

Distributed with MPI

GPU implementation

Measurement of the speedup.

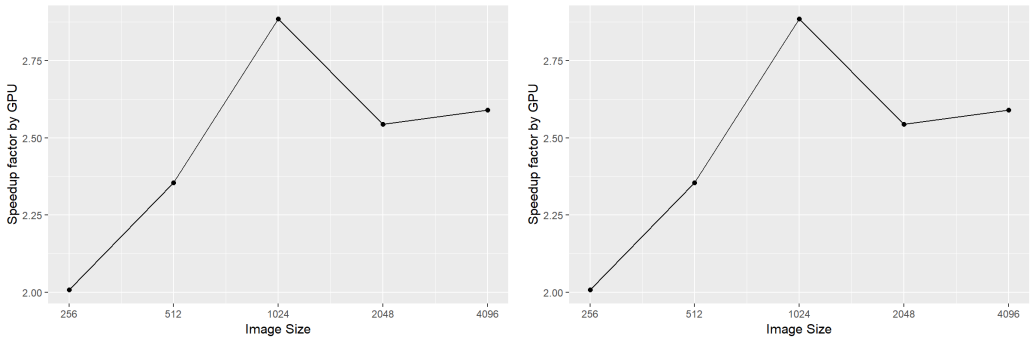


Figure 26: Speedup by using MPI or GPU acceleration

We cannot use MPI combined with the GPU. The MPI implementation uses a communication step in each coordinate descent iteration (communicating which pixel to optimize with MPI Allreduce).

7.3 Effect of approximating the PSF

As we described in Section 6, the PSF for deconvolution is as big as the image. For wide field-of-view observations of MeerKAT, the PSF is approximately a gaussian with decreasing pixel values the further we move from the center. Most of the values in the PSF are close to zero. The question is, what effect has an approximate deconvolution with a smaller PSF? If we can approximate the deconvolution with a small enough PSF, we can solve patches of the image independently of each other. However, the PSF approximation may need more major cycles to converge.

The effect of approximating the PSF are not clear. We know that thanks to the w-term in the visibilities, the PSF is not constant over the image. We already need several major cycles to converge. With a good

approximation of the PSF, we may speed up the individual iterations of coordinate descent without needing more major cycle.

We presented two methods to approximating the PSF for the deconvolution in Section 6. Method 1 updates only a fraction of the gradients, and Method 2 uses a fraction of the PSF for deconvolution. We test both methods on the LMC data and explore what effects the approximations have on the reconstruction.

7.3.1 Method 1: Approximate gradient update

Our coordinate descent method updates the map of gradients after each iteration. Method 1 starts with the same map of gradients as the original, but then only updates a fraction of the gradients in each iteration. It updates a rectangle of the most significant gradients.With each coordinate descent iteration, the map of gradients gets less accurate. Because we do an approximate update of gradients, this method should converge to the same result as the original with enough major cycles.

At the beginning of each major cycle, we calculate the objective value of the current solution. We compare the objective value and the wall-clock time of the original and the approximate gradient update. This is a minimization problem, meaning the lowest objective is the most accurate reconstruction (according to the elastic net regularization).

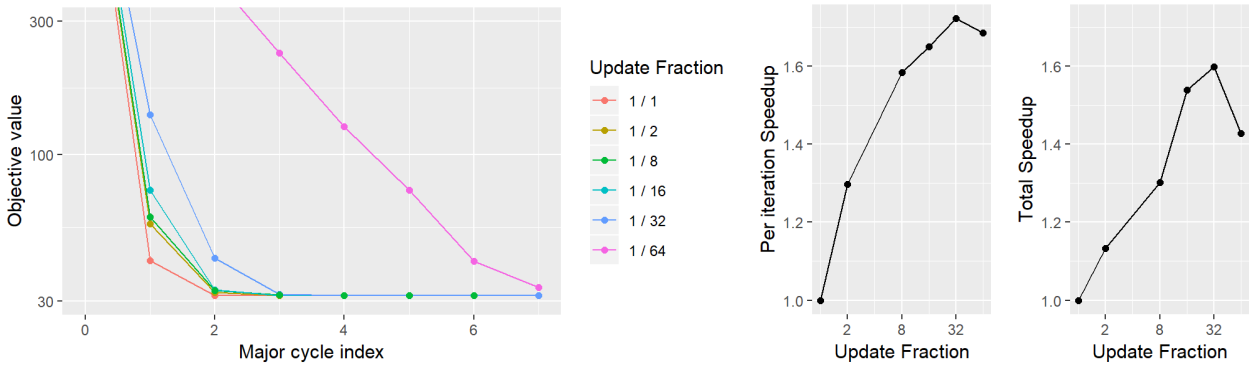


Figure 27: Effect of only updating a fraction of the gradients.

The smallest fraction of gradient updater, for which coordinate still converges is $\frac{1}{64}$ of the total update. Meaning, we can update only a rectangle of 48^2 pixels, or an image patch of 72 arc-seconds in size. However, it is obvious from the Figure 27 that coordinate descent needs more major cycles to converge with such an extreme approximation.

With less extreme approximations, we also reduce the number of necessary major cycles. The approximation of $\frac{1}{32}$ needs two more major cycle, while all other need the one more as the original method[8]. The objective values of the approximations end within 0.03% of the original(the objective value of the approximations are higher by a factor of 0.0003).

Figure 27 also shows two speedup comparisons. The speedup from the approximation is harder to quantify. For one, each iteration of coordinate descent becomes cheaper, because we only update a small part of the gradient map. This is the per iteration speedup. The second speedup Figure compares the overall time spent in deconvolution. Although each iteration becomes cheaper with the approximation, we may need more iterations to converge. Also, we have an implicit path regularization in the approximation. Therefore, a speedup per coordinate descent iteration does not necessarily lead to an overall speedup.

As we discussed in section In Section 6.4, we have a limit in each major cycle to how far we can trust our approximation. That is why in the first major cycle coordinate descent gets started with a higher λ regularization

[8] 6 cycles to converge, and one extra cycle which is here to measure the final objective

than specified. In each successive major cycle, we reduce the λ parameter until we reach the specified value. This is an implicit form of path regularization, and may help speeding up the convergence rate of coordinate descent in general [34]. In our case, we need path regularization to ensure convergence over several major cycles (Aggressive approximations like $\frac{1}{64}$ start to oscillate over several major cycle). Notice that the overall speedup in Figure 27 is lower than the speedup we get per iteration. This suggests path regularization as implemented here does not speed up convergence rate.

Overall with gradient update approximations give us a speedup factor of around 1.5. Depending on how aggressive we approximate, We need one or two more major cycles than the original coordinate descent deconvolutions. This puts gradient update approximations on par with multi-scale CLEAN in terms of major cycles.

7.3.2 Method 2: Approximate deconvolution

In this method, we use a fraction of the total PSF for deconvolution. This method solves a different deconvolution problem, where the PSF is for example only $\frac{1}{8}$ the size. The downside is that method 2 is not guaranteed to converge to the same optimum. Nevertheless, we solve the approximate deconvolution problem with several different fractions of the PSF and compare how close the approximate solution is to the original.

As before, we measure the true objective value for the approximate solution at the beginning of each major cycle iteration. The Figure 28 compares the approximate deconvolution to the original.

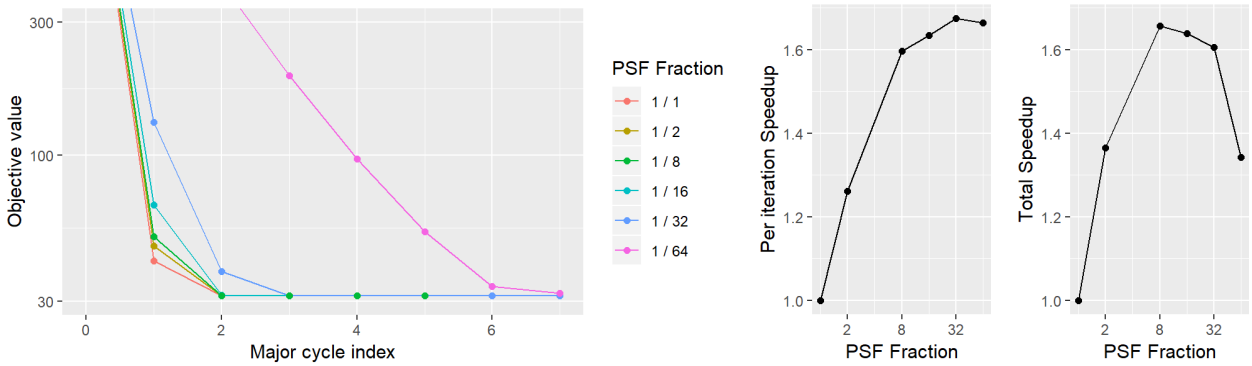


Figure 28: Effect of the L1 and L2 Norm separately.

The performance is similar to the first method at first glance. Both methods converge in a similar manner, with a similar factor of speedup. For the PSF fraction $\frac{1}{64}$ this is largely due to the same path regularization used in this method. For larger PSF's, starting from $\frac{1}{16}$, the path regularization becomes less important with this method, as it only effects the regularization parameter λ of major cycle index 0. Indeed, this method is less likely to oscillate and seems to have a more stable convergence.

The approximate deconvolution converges faster than the first method. At the start of the same major cycle, this method always has a lower objective. However, it is not guaranteed to converge to the same result. This can be seen in the fact that it never reaches the same objective value as the original. The objective value of the approximation is within 0.07% (Factor of 0.0007, roughly twice the factor of the first method). The difference gets more significant, the more extreme we chose the deconvolution approximation.

Nevertheless, the approximate deconvolution converges surprisingly close to the original solution. The question is, is this difference of 0.07% in the objective value significant? The answer to this question depends on what kind of error the approximation introduces in the image. The obvious error is that the approximation chronically under-estimates the pixel values: The maximum pixel value in N132 of the original is 0.0024 Jansky/beam, while the maximum of the $\frac{1}{16}$ approximation is 0.00235 Jansky/beam. The pixel magnitude is

important in the self-calibration regime [29] (When we take the result of the reconstruction and try to improve the calibration).

The under-approximation of pixel values is an error we cannot ignore. One naive remedy is to start with the deconvolution approximation, but switch to the original PSF after a certain number of major cycle. This would give us the guarantee to converge to the same result, but let us use an approximation at the start. We propose another solution: Combining the two approximation methods.

7.3.3 Combination of Method 1 and 2

The two approximation methods have two different shortcomings: Approximate gradient update (method 1) converges more slowly, and converges to the same result as the original. Approximate deconvolution (method 2) converges faster than method 1, but does not converge to the same result. Here, we combine the two methods to remedy the shortcomings: For the first couple of major cycles, we use approximate deconvolution, and then switch to approximate gradient update.

We compare the original, approximate gradient update, approximate deconvolution and our combination in. We use the factor $\frac{1}{16}$ for approximations, meaning we update $\frac{1}{16}$ of the gradients, and do the approximate deconvolution with $\frac{1}{16}$ of the PSF.

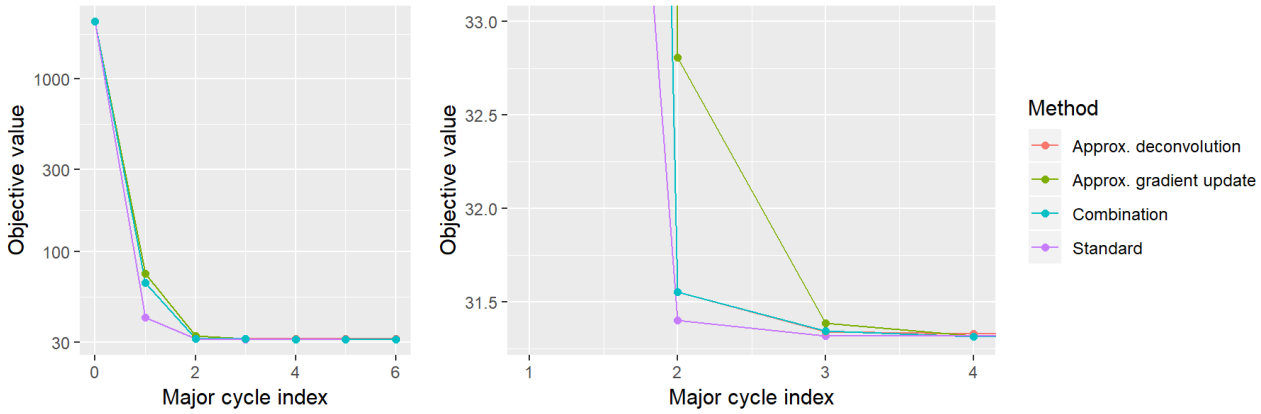


Figure 29: Comparison of the two methods using the fraction $\frac{1}{16}$ of the PSF.

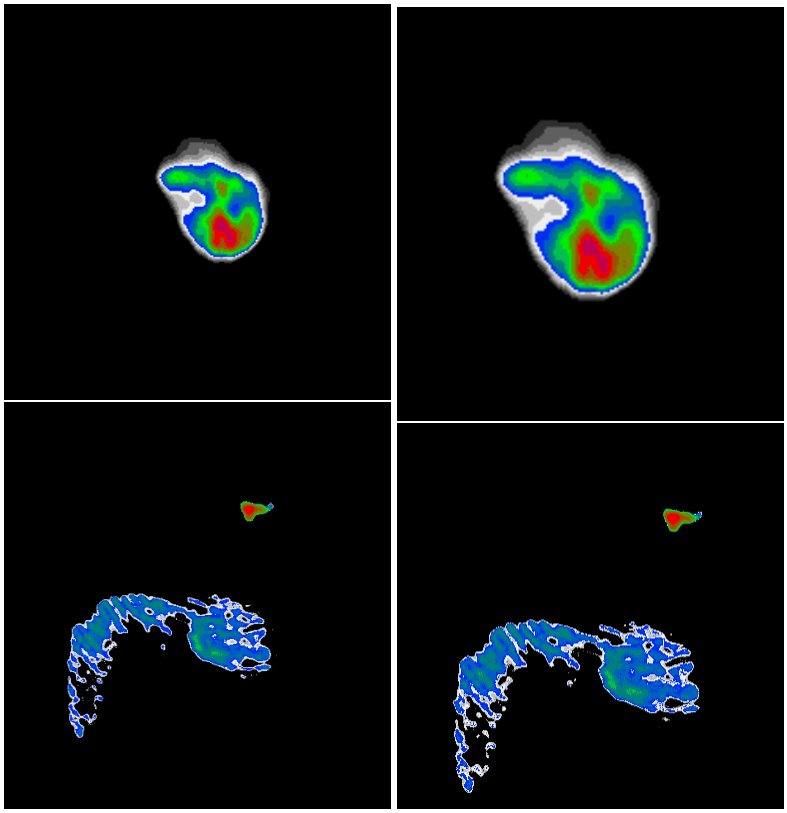
Figure 29

We switch from approximate deconvolution to gradient update after the path regularization has reached the target λ for the first time. In the case of the LMC data, the implementation uses approximate deconvolution for major cycle index 0 and 1, and then switches to approximate gradient update.

Method	Iteration Speedup	Total Speedup
Original	1.00	1.00
(Method 1) Approx. gradient update	1.66	1.55
(Method 2) Approx. deconvolution	1.67	1.68
Combination	1.69	1.68

Speedup. Combination is the fastest overall and leads to the lowest objective function. Actually beats the original by a small margin. Approximation errors are not correlated. We can use Method 1 to remove the approximation errors of Method 2.

Figure 30 compares the original result with the result of the combined approximation. Virtually identical. Same problem with calibration. Only visual difference in the extended emission with calibration errors.



(a) Original CD *(b) Combined approximation*

Figure 30: Image comparison to approximation

Difference is negligible. We can use the approximation and arrive at the same result. Speedup of a factor 1.6
How many major cycles: One more than the original. But same as multi-scale CLEAN.

7.3.4 Masking the PSF

8 Parallel coordinate descent methods

We can approximate the PSF with a small fraction of its true size.

Naive way to exploit this, we can create patches of the image, four of which are independent of each other. Deconvolve four patches in parallel. Work is not equally distributed on the different patches.

Parallel coordinate descent methods can also use the smaller PSF.

Side note: The basic coordinate descent deconvolution algorithm described in Section 4 is a serial method. It is serial in the sense that the algorithm takes a descent step and immediately updates the gradient map. The implementation of the basic algorithm uses multiple cores. Parallel coordinate descent methods take several steps at different coordinates before updating the gradients.

8.1 From serial to parallel

Parallel coordinate descent methods take several steps at different coordinates before they update the gradient map. In our deconvolution problem, this means that we risk over-estimating pixel values when we optimize pixels in parallel. Parallel pixel updates have a potentially overlapping PSF (the pixel values are correlated). The more the correlate, the higher the over-estimation gets[9]. If we update many correlated pixels in parallel, we may converge slowly, or even diverge.

Parallel coordinate descent methods have a strategy to deal with correlated pixel values: The shotgun algorithm[42] estimates the number parallel random updates that can be used without diverging. PCDM[31] estimates the expected PSF overlap between random pixels, and adjusts the step size accordingly. In this project, we use the PCDM algorithm and its accelerated variant APPROX [37]. We introduce the algorithm in Section 8.2.

Note that parallel coordinate descent methods tend to select pixels at random. This helps dealing with correlated pixel values: In our deconvolution case, the pixels which are close by to each other tend to be correlated more strongly. But the exact shape of the PSF is also a factor. If the PSFs overlap only in a region where it is close to zero, the pair of pixels also have a low correlation factor. A random selection strategy helps to keep a low correlation factor on average, no matter what the PSF looks like.

But for our deconvolution case, a random strategy brings in its own issues. Remember that the LMC has a bright supernova remnant N132D, which overshadows other sources. A deconvolution algorithm has to deconvolve the pixels belonging to N132D first. A random strategy may waste resources until it has deconvolved the fairly small subset of pixels belonging to N132D. We have adapted the PCDM algorithm for the deconvolution problem, and describe our adaption in detail in Section 8.3.

8.2 Parallel (Block) Coordinate Descent Method (PCDM)

The PCDM algorithm can be seen as a generalization of our basic coordinate descent algorithm from Section 4. The basic algorithm descends a single pixel at a time. PCDM can update one, or a whole block of pixels at each iteration. And as the name implies, it can update multiple blocks of pixels before updating the map of gradients. In this project, we use the accelerated variant of the PCDM algorithm, named APPROX [37]. We first generalize our basic coordinate descent method to block-coordinate descend. We then introduce the PCDM algorithm, show how it was adapted to the deconvolution problem and then describe the acceleration method.

[9] Imagine we optimize two pixels next to each other in serial coordinate descent: The algorithm descents, the first pixel, updates the gradient, and then descents the second pixel. The value of the second pixel will be magnitudes lower than the first, because most of the emission in that area was already explained by the first pixel. If however we update in parallel, both pixels will end up with a similar value, and both pixels try to explain the same emission.

8.2.1 Parallel updates and degree of seperability

The degree of seperability is the core concept behind parallel coordinate descent methods.

Important concept for parallel updates. We explain it with a slightly modified objective function:

$$\begin{aligned} \underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - x * PSF\right\|_2^2 + \lambda ElasticNet(x) \\ \underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - Ax\right\|_2^2 + \lambda ElasticNet(x) \end{aligned} \tag{8.1}$$

The main difference is that we represent the convolution $x * PSF$ as a matrix multiplication Ax. A columns of matrix A is simply the PSF shifted at the correct pixel location. IF A is the identity matrix, the pixels x in the reconstruction are independent from each other. Each column in A only contains zero except a single 1 where the pixel is located. The whole problem is separable to We can use as many processors as we have pixels. In that case, the deconvolution is completely separable.

This is not true in reality, the PSF is actually dense. In our case, every entry of matrix A is non-zero. But we can approximate the deconvolution with a smaller PSF, as we have shown in Section 7.3. In that case, the problem is partially separable.

Degree of separability ω: Max number of non-zero elements in a matrix columnn.

The smaller the degree of seperability ω, the closer we are to a fully separable problem.

Parallel updates are less correlated. PCDM algorithm, we can estimate the step size in a parallel environmnent.

8.2.2 From single pixel to blocks

block update strategy

closed form solution

$$block = \frac{A^T y}{A^T A} \tag{8.2}$$

$A^T y$ are again the gradients. we need an inverse of $A^T A$. And it is numerically stable for block sizes below 4^2.

A different update rule. We just update the whole block like this

$$block = \frac{A^T y}{Lipschitz} \tag{8.3}$$

We have one lipschitz constant donw below. This is just the lipschitz constant for each pixel multiplied with each other. In other words: the larger we chose the blocks to be, the smaller steps we can take.

Similarities to (F)ISTA [43] for a single block. If we chose a block size the same as the image, we arrive at the (F)ISTA algorithm. But the Lipschitz constant is enormous. If the image itself is sparse, we basically ”waste” step sizes on zero pixels. So we can take larger steps for pixels that matter with smaller step sizes.

Good step sizes get tested out later

8.2.3 PCDM in pseudo-code

PCDM for our deconvolution objective

Select random blocks Update blocks in parallel update gradient map

Last part is the random selection strategy. τ-nice sampling. I.e. take τ number of pixels uniformly at random.

Core of the algorithm is the update rule

The ESO. The Estimated Seperability overapproximation what selection strategy we use

Can be done asynchronously.

8.3 Adapting random selection strategies for deconvolution

The problem with random strategies. We have subsets of pixels that have to be deconvolved first. Even worse: A random strategy can lead to the algorithm getting ”stuck”: It randomly adds a few pixels close to a source. Now, when. we can get stuck where we first need to remove the old values. Old values potentially need several iterations. So we need to, randomly, select specific pixels multiple times before we can properly add the right pixel. Really inefficient.

Other problem of wasted iterations: First, we tend to have a lot of possible pixels to optimize. Then during the iterations, the amount of possible pixels are reduced, and most random selections lead to just a

8.3.1 Cold start

Cold start is when we do not have any pixels.

First iterations are problematic. Here we have a few pixels that reduce the objective function by a lot. Afterwards it tends to become more evenly distributed where multiple pixels are a good choice to reduce the objective function.

Our basic coordinate descent method from Section can be thought of as special case of the PCDM algorithm. Our coordinate descent method descends a single pixel at a time, and updates the gradients after each iteration. The first generalization is that we can update a block of pixels at each iteration, called block coordinate descent. We still update the gradient in each iteration. The next generalization is we update severeal blocks in parallel, and before we update the gradients.

Two extra variables, how many pixels we update together in a block, and how many blocks we update in parallel before we update the gradients.

8.3.2 Active Set heuristic

Active set heuristic used for cyclic coordinate descent. Choose a subset of pixels and optimize those until convergence. Then choose a new set.

Add every block in the active set that has a pixel we can modify. Pixels that have

8.3.3 APPROX pseudo code

8.4 APPROX implementation

9 Discussion

Problem of calibration.

CLEAN: MAsking as part for low noise. Maybe also works for Coordinate Descent

Figure 27 shows that at major cycle index 4 (the fifth major cycle), all approximations except for $frac164$ are within 0.09% of the original solution. The current implementation stops when coordinate descent converges within 1000 iterations in the current major cycle. This rule may be too strict, and our coordinate descent algorithm may be stopped earlier.

GPU Coordinate Descent

Difficult to achieve speedup with a dense PSF we approximated

9.1 Approximation of the PSF

Approximation that works. To our knowledge we are the first to explore an approximated PSF. Not relevant for all optimization algorithm, but parallel coordinate descent methods achieve a (significant) speedup.

In our test we were able to use it without needing more major cycles than multi-scale CLEAN.

The need for major cycles could be further reduced. Question of effective heuristics that work for a wide range of

Parallel deconvolutions

9.2 Calibration errors

Common experience that CLEAN based algorithms handle calibration errors better [29] than ”proper” optimization algorithms.

It is a question why. We experienced similar results, but used a simple regularization: elastic net. Other algorithms like MORESANE[4] use more sophisticated regularization.

Why does CLEAN work so well with calibration errors? Is there some sort of implicit regulariztion in the way it choses its pixels. (not in the actual regularization)

9.3 CLEAN heuristics for coordinate descent

Since CLEAN and coordinate descent methods are acting out similar things, we may be able to use CLEAN heuristics to improve coordinate descent based methods.

Masking: CLEAN uses masking to reconstruct sources below the noise level of the image. After a number of iterations, it masks all pixels which are not zero. (Ie. now all sources are detected) and clean deeper.

Similar stuff can be done for coordinate descent methods.

Greedy Coordinate descent

9.4 Hydra

our coordinate descent method is not distributed yet. Hydra exists. It is APPROX in the distributed environment. We had to adapt the implementation of APPROX for the deconvolution problem. Similar adaptions necessary for a distributed algorithm. Problems of cold start, and irrelevant pixels.

9.5 Multi frequency extension

Difficult.

Regularized inverse problem [44]. Objective function How it works, adding a new term to the objective function

$$\underset{x}{minimize}\ \frac{1}{2}\left\|I_{dirty} - X * PSF\right\|_2^2 + \lambda ElasticNet(X) + \lambda_v \left\|DX\right\|_1 \tag{9.1}$$

Where D is the Discrete cosine transform.

Does not have a proximal operator for each pixel. problem for Coordinate descent method.

Question if each iteration can be cheap.

But may be separated with respect to frequency with Lagrangian multipliers. Question if cd methods are faster.

10 Conclusion

Works

References

[1] Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. IEEE Transactions on information theory, 52(2):489–509, 2006.

[2] David L Donoho. Compressed sensing. IEEE Transactions on information theory, 52(4):1289–1306, 2006.

[3] Arwa Dabbech, Alexandru Onose, Abdullah Abdulaziz, Richard A Perley, Oleg M Smirnov, and Yves Wiaux. Cygnus a super-resolved via convex optimization from vla data. Monthly Notices of the Royal Astronomical Society, 476(3):2853–2866, 2018.

[4] Arwa Dabbech, Chiara Ferrari, David Mary, Eric Slezak, Oleg Smirnov, and Jonathan S Kenyon. Moresane: Model reconstruction by synthesis-analysis estimators-a sparse deconvolution algorithm for radio interferometric imaging. Astronomy & Astrophysics, 576:A7, 2015.

[5] JA Högbom. Aperture synthesis with a non-regular distribution of interferometer baselines. Astronomy and Astrophysics Supplement Series, 15:417, 1974.

[6] Urvashi Rau and Tim J Cornwell. A multi-scale multi-frequency deconvolution algorithm for synthesis imaging in radio interferometry. Astronomy & Astrophysics, 532:A71, 2011.

[7] Charles A Bouman and Ken Sauer. A unified approach to statistical tomography using coordinate descent optimization. IEEE Transactions on image processing, 5(3):480–492, 1996.

[8] Simon Felix, Roman Bolzern, and Marina Battaglia. A compressed sensing-based image reconstruction algorithm for solar flare x-ray observations. The Astrophysical Journal, 849(1):10, 2017.

[9] Madison Gray McGaffin and Jeffrey A Fessler. Edge-preserving image denoising via group coordinate descent on the gpu. IEEE Transactions on Image Processing, 24(4):1273–1281, 2015.

[10] Olivier Fercoq, Zheng Qu, Peter Richtárik, and Martin Takáč. Fast distributed coordinate descent for non-strongly convex losses. In 2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP), pages 1–6. IEEE, 2014.

[11] Jean-Luc Starck, David L Donoho, and Emmanuel J Candès. Astronomical image representation by the curvelet transform. Astronomy & Astrophysics, 398(2):785–800, 2003.

[12] Jean-Luc Starck, Fionn Murtagh, and Mario Bertero. Starlet transform in astronomical data processing. Handbook of Mathematical Methods in Imaging, pages 2053–2098, 2015.

[13] Andreas M Tillmann and Marc E Pfetsch. The computational complexity of the restricted isometry property, the nullspace property, and related concepts in compressed sensing. IEEE Transactions on Information Theory, 60(2):1248–1259, 2013.

[14] Abhiram Natarajan and Yi Wu. Computational complexity of certifying restricted isometry property. arXiv preprint arXiv:1406.5791, 2014.

[15] Ishay Haviv and Oded Regev. The restricted isometry property of subsampled fourier matrices. In Geometric Aspects of Functional Analysis, pages 163–179. Springer, 2017.

[16] Emmanuel J Candes and Yaniv Plan. A probabilistic and ripless theory of compressed sensing. IEEE transactions on information theory, 57(11):7235–7254, 2011.

[17] Keith Miller. Least squares methods for ill-posed problems with a prescribed bound. SIAM Journal on Mathematical Analysis, 1(1):52–74, 1970.

[18] Yves Wiaux, Laurent Jacques, Gilles Puy, Anna MM Scaife, and Pierre Vandergheynst. Compressed sensing imaging techniques for radio interferometry. Monthly Notices of the Royal Astronomical Society, 395(3):1733–1742, 2009.

[19] André Ferrari, David Mary, Rémi Flamary, and Cédric Richard. Distributed image reconstruction for very large arrays in radio astronomy. In 2014 IEEE 8th Sensor Array and Multichannel Signal Processing Workshop (SAM), pages 389–392. IEEE, 2014.

[20] Julien N Girard, Hugh Garsden, Jean Luc Starck, Stéphane Corbel, Arnaud Woiselle, Cyril Tasse, John P McKean, and Jérôme Bobin. Sparse representations and convex optimization as tools for lofar radio interferometric imaging. Journal of Instrumentation, 10(08):C08013, 2015.

[21] Rafael E Carrillo, Jason D McEwen, and Yves Wiaux. Purify: a new approach to radio-interferometric imaging. Monthly Notices of the Royal Astronomical Society, 439(4):3591–3604, 2014.

[22] Rafael E Carrillo, Jason D McEwen, and Yves Wiaux. Sparsity averaging reweighted analysis (sara): a novel algorithm for radio-interferometric imaging. Monthly Notices of the Royal Astronomical Society, 426(2):1223–1234, 2012.

[23] Oleg M Smirnov. Revisiting the radio interferometer measurement equation-i. a full-sky jones formalism. Astronomy & Astrophysics, 527:A106, 2011.

[24] Oleg M Smirnov. Revisiting the radio interferometer measurement equation-ii. calibration and direction-dependent effects. Astronomy & Astrophysics, 527:A107, 2011.

[25] Oleg M Smirnov. Revisiting the radio interferometer measurement equation-iii. addressing direction-dependent effects in 21 cm wsrt observations of 3c 147. Astronomy & Astrophysics, 527:A108, 2011.

[26] Oleg M Smirnov. Revisiting the radio interferometer measurement equation-iv. a generalized tensor formalism. Astronomy & Astrophysics, 531:A159, 2011.

[27] BG Clark. An efficient implementation of the algorithm'clean'. Astronomy and Astrophysics, 89:377, 1980.

[28] FR Schwab. Relaxing the isoplanatism assumption in self-calibration; applications to low-frequency radio interferometry. The Astronomical Journal, 89:1076–1081, 1984.

[29] AR Offringa and O Smirnov. An optimized algorithm for multiscale wideband deconvolution of radio astronomical images. Monthly Notices of the Royal Astronomical Society, 471(1):301–316, 2017.

[30] Peter Richtárik and Martin Takáč. Distributed coordinate descent method for learning with big data. The Journal of Machine Learning Research, 17(1):2657–2681, 2016.

[31] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. Mathematical Programming, 156(1-2):433–484, 2016.

[32] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM Journal on Optimization, 22(2):341–362, 2012.

[33] Yu Nesterov. Gradient methods for minimizing composite functions. Mathematical Programming, 140(1):125–161, 2013.

[34] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33(1):1, 2010.

[35] Jason D McEwen and Yves Wiaux. Compressed sensing for wide-field radio interferometric imaging. Monthly Notices of the Royal Astronomical Society, 413(2):1318–1332, 2011.

[36] Shai Shalev-Shwartz and Ambuj Tewari. Stochastic methods for l1-regularized loss minimization. Journal of Machine Learning Research, 12(Jun):1865–1892, 2011.

[37] Olivier Fercoq and Peter Richtárik. Accelerated, parallel, and proximal coordinate descent. SIAM Journal on Optimization, 25(4):1997–2023, 2015.

[38] Jonathan S Kenyon. Pymoresane: Python model reconstruction by synthesis-analysis estimators. Astrophysics Source Code Library, 2019.

[39] DC-J Bock, MI Large, and Elaine M Sadler. Sumss: A wide-field radio imaging survey of the southern sky. i. science goals, survey design, and instrumentation. The Astronomical Journal, 117(3):1578, 1999.

[40] AR Offringa, Benjamin McKinley, Natasha Hurley-Walker, FH Briggs, RB Wayth, DL Kaplan, ME Bell, Lu Feng, AR Neben, JD Hughes, et al. Wsclean: an implementation of a fast, generic wide-field imager for radio astronomy. Monthly Notices of the Royal Astronomical Society, 444(1):606–619, 2014.

[41] Dan Briggs. High fidelity deconvolution of moderately resolved sources, 2019.

[42] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization. arXiv preprint arXiv:1105.5379, 2011.

[43] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM journal on imaging sciences, 2(1):183–202, 2009.

[44] André Ferrari, Jérémy Deguignet, Chiara Ferrari, David Mary, Antony Schutz, and Oleg Smirnov. Multi-frequency image reconstruction for radio interferometry. a regularized inverse problem approach. arXiv preprint arXiv:1504.06847, 2015.

[45] Luke Pratley, Melanie Johnston-Hollitt, and Jason D McEwen. A fast and exact w-stacking and w-projection hybrid algorithm for wide-field interferometric imaging. arXiv preprint arXiv:1807.09239, 2018.

[46] Bram Veenboer, Matthias Petschow, and John W Romein. Image-domain gridding on graphics processors. In 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 545–554. IEEE, 2017.

List of Figures

List of Tables

11 attachment

12 Larger runtime costs for Compressed Sensing Reconstructions

The MeerKAT instrument produces a new magnitude of data volume. An image with several million pixels gets reconstructed from billions of Visibility measurements. Although MeerKAT measures a large set of Visibilities, the measurements are still incomplete. We do not have all the information available to reconstruct an image. Essentially, this introduces ”fake” structures in the image, which a reconstruction algorithm has to remove. Additionally, the measurements are noisy.

We require an image reconstruction algorithm which removes the ”fake” structures from the image, and removes the noise from the measurements. The large data volume of MeerKAT requires the algorithm to be both scalable and distributable. Over the years, several reconstruction algorithms were developed, which can be separated into two classes: Algorithms based on CLEAN, which are cheaper to compute and algorithms based on Compressed Sensing, which create higher quality reconstructions.

CLEAN based algorithms represent the reconstruction problem as a deconvolution. First, they calculate the ”dirty” image, which is corrupted by noise and fake image structures. The incomplete measurements essentially convolve the image with a Point Spread Function (PSF). CLEAN estimates the PSF and searches for a deconvolved version of the dirty image. In each CLEAN iteration, it searches for the highest pixel in the dirty image, subtracts a fraction PSF at the location. It adds the fraction to the same pixel location of a the ”cleaned” image. After several iterations, the cleaned image contains the deconvolved version of the dirty image. CLEAN accounts for noise by stopping early. It stops when the highest pixel value is smaller than a certain threshold. This results in a light-weight and robust reconstruction algorithm. CLEAN is comparatively cheap to compute, but does not produce the best reconstructions and is difficult to distribute on a large scale.

Compressed Sensing based algorithms represent the reconstruction as an optimization problem. They search for the optimal image which is as close to the Visibility measurements as possible, but also has the smallest regularization penalty. The regularization encodes our prior knowledge about the image. Image structures which were likely measured by the instrument result in a low regularization penalty. Image structures which were likely introduced by noise or the measurement instrument itself result in high penalty. Compressed Sensing based algorithms explicitly handle noise and create higher quality reconstructions than CLEAN. State-of-the-art Compressed Sensing algorithms show potential for distributed computing. However, they currently do not scale on MeerKATs data volume. They require too many computing resources compared to CLEAN based algorithms.

This project searches for a way to reduce the runtime costs of Compressed Sensing based algorithms. One reason for the higher costs is due to the non-uniform FFT Cycle. State-of-the-art CLEAN and Compressed Sensing based algorithms both use the non-uniform FFT approximation in a cycle during reconstruction. The interferometer measures the Visibilities in a continuous space in a non-uniform pattern. The image is divided in a regularly spaced, discrete pixels. The non-uniform FFT creates an approximate, uniformly sampled image from the non-uniform measurements. Both, CLEAN and Compressed Sensing based algorithms use the non-uniform FFT to cycle between non-uniform Visibilities and uniform image. However, a Compressed Sensing algorithm requires more non-uniform FFT cycles for reconstruction.

CLEAN and Compressed Sensing based algorithms use the non-uniform FFT in a similar manner. However, there are slight differences in the architecture. This project hypothesises that The previous project searched for an alternative to the non-uniform FFT cycle. Although there are alternatives, there is currently no replacement which leads to lower runtime costs for Compressed Sensing. Current research is focused on reducing the number of non-uniform FFT cycles for Compressed Sensing algorithms.

CLEAN based algorithms use the Major Cycle Architecture for reconstruction. Compressed Sensing based algorithms use a similar architecture, but with slight modifications. Our hypothesis is that we may reduce the number of non-uniform FFT cycles for Compressed Sensing by using CLEAN’s Major Cycle Architecture.

12.1 CLEAN: The Major Cycle Architecture

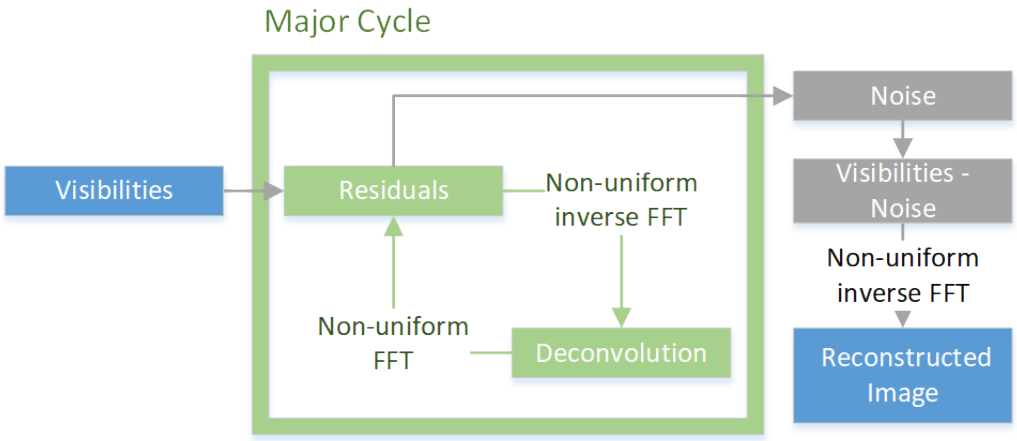


Figure 31: The Major Cycle Architecture

Figure 31 depicts the Major Cycle Architecture used by CLEAN algorithms. First, the Visibilities get transformed into an image with the non-uniform FFT. The resulting dirty image contains the corruptions of the measurement instrument and noise. A deconvolution algorithm, typically CLEAN, removes the corruption of the instrument with a deconvolution. When the deconvolution stops, it should have removed most of the observed structures from the dirty image. The rest, mostly noisy part of the dirty image gets transformed back into residual Visibilities and the cycle starts over.

In the Major Cycle Architecture, we need several deconvolution attempts before it has distinguished the noise from the measurements. Both the non-uniform FFT and the deconvolution are approximations. By using the non-uniform FFT in a cycle, it can reconstruct an image at a higher quality. For MeerKAT reconstruction with CLEAN, we need approximately 4-6 non-uniform FFT cycles for a reconstruction.

12.2 Compressed Sensing Architecture

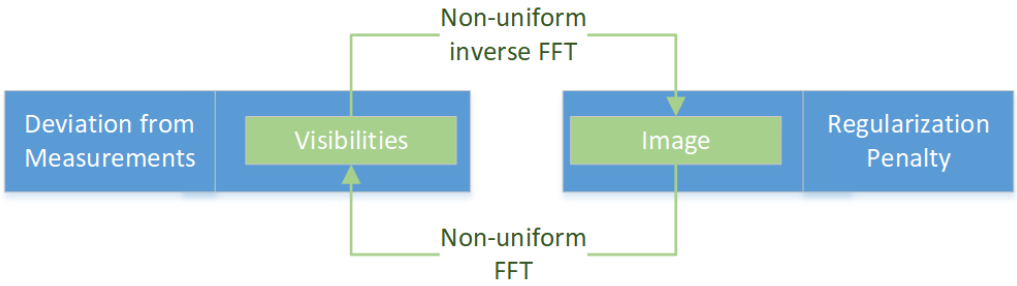


Figure 32: State-of-the-art Compressed Sensing Reconstruction Architecture

Figure 32 depicts the architecture used by Compressed Sensing reconstructions. The Visibilities get transformed into an image with the non-uniform FFT approximation. The algorithm then modifies the image so it reduces the regularization penalty. The modified image gets transformed back to Visibilities and the algorithm then minimizes the difference between measured and reconstructed Visibilities. This is repeated until the algorithm converges to an optimum.

In this architecture, state-of-the-art Compressed Sensing algorithms need approximately 10 or more non-uniform FFT cycles to converge. It is one source for the higher runtime costs. For MeerKAT reconstructions

the non-uniform FFT tends to dominate the runtime costs. A CLEAN reconstruction with the Major Cycle Architecture already spends a large part of its time in the non-uniform FFT. Compressed Sensing algorithms need even more non-uniform FFT cycle on top of the ”Image Regularization” step being generally more expensive than CLEAN deconvolution. There is one upside in this architecture: State-of-the-art algorithms managed to distribute the ”Image Regularization” operation.

12.3 Hypothesis for reducing costs of Compressed Sensing Algorithms

Compressed Sensing Algorithms are not bound to the Architecture presented in section 12.2. For example, we can design a Compressed Sensing based deconvolution algorithm and use the Major Cycle Architecture instead.

Our hypothesis is: We can create a Compressed Sensing based deconvolution algorithm which is both distributable and creates higher quality reconstructions than CLEAN. Because it also uses the Major Cycle architecture, we reckon that the Compressed Sensing deconvolution requires a comparable number of non-uniform FFT cycles to CLEAN. This would result in a Compressed Sensing based reconstruction algorithm with similar runtime costs to CLEAN, but higher reconstruction quality and higher potential for distributed computing.

12.4 State of the art: WSCLEAN Software Package

12.4.1 W-Stacking Major Cycle

12.4.2 Deconvolution Algorithms

CLEAN MORESANE

12.5 Distributing the Image Reconstruction

12.5.1 Distributing the Non-uniform FFT

12.5.2 Distributing the Deconvolution

13 Handling the Data Volume

The new data volume is a challenge to process for both algorithms and computing infrastructure. Push for parallel and distributed algorithms. For Radio Interferometer imaging, we require specialized algorithms. The two distinct operations, non-uniform FFT and Deconvolution, were difficult algorithms for parallel or distributed computing.

The non-uniform FFT was historically what dominated the runtime []. Performing an efficient non-uniform FFT for Radio Interferometers is an active field of research[40, 45], continually reducing the runtime costs of the operation. Recently, Veeneboer et al[46] developed a non-uniform FFT which can be fully executed on the GPU. It speeds up the most expensive operation.

In Radio Astronomy, CLEAN is the go-to deconvolution algorithm. It is light-weight and compared to the non-uniform FFT, a cheap algorithm. It is also highly iterative, which makes it difficult for effective parallel or distributed implementations. However, compressed sensing based deconvolution algorithms can be developed with distribution in mind.

13.1 Fully distributed imaging algorithm

Current imaging algorithms push towards parallel computing with GPU acceleration. But with Veeneboer et al’s non-uniform FFT and a compressed sensing based deconvolution, we can go a step further and create a distributed imaging algorithm.

14 Image Reconstruction for Radio Interferometers

In Astronomy, instruments with higher angular resolution allows us to measure ever smaller structures in the sky. For Radio frequencies, the angular resolution is bound to the antenna dish diameter, which puts practical and financial limitations on the highest possible angular resolution. Radio Interferometers get around this limitation by using several smaller antennas instead. Together, they act as a single large antenna with higher angular resolution at lower financial costs compared to single dish instruments.

Each antenna pair of an Interferometer measures a single Fourier component of the observed image. We can retrieve the image by calculating the Fourier Transform of the measurements. However, since the Interferometer only measures an incomplete set of Fourier components, the resulting image is ”dirty”, convolved with a Point Spread Function (PSF). Calculating the Fourier Transform is not enough. To reconstruct the from an Interferometer image, an algorithm has to find the observed image with only the dirty image and the PSF as input. It has to perform a deconvolution. The difficulty lies in the fact that there are potentially many valid deconvolutions for a single measurement, and the algorithm has to decide for the most likely one. How similar the truly observed image and the reconstructed images are depends largely on the deconvolution algorithm.

State-of-the-art image reconstructions use the Major Cycle architecture (shown in Figure 33), which contains three operations: Gridding, FFT and Deconvolution.

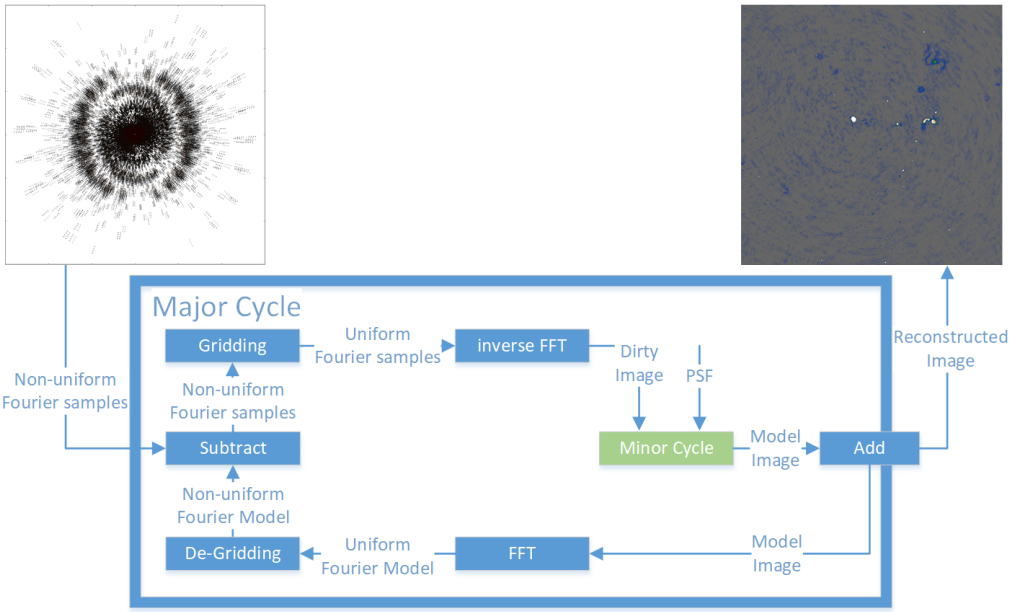


Figure 33: The Major Cycle Architecture of image reconstruction algorithms

The first operation in the Major Cycle, Gridding, takes the non-uniformly sampled Fourier measurements from the Interferometer and interpolates them on a uniformly spaced grid. The uniform grid lets us use FFT to calculate the inverse Fourier Transform and we arrive at the dirty image. A deconvolution algorithm takes the dirty image plus the PSF as input, producing the deconvolved ”model image”, and the residual image as output. At this point, the reverse operations get applied to the residual image. First the FFT and then De-gridding, arriving at the non-uniform Residuals. The next Major Cycle begins with the non-uniform Residuals as input. The cycles are necessary, because the Gridding and Deconvolution operations are only approximations. Over several cycles, we reduce the errors introduced by the approximate Gridding and Deconvolution. The final, reconstructed image is the addition of all the model images of each Major Cycle.

14.1 Distributed Image Reconstruction

New Interferometer produce an ever increasing number of measurements, creating ever larger reconstruction problems. A single image can contain several terabytes of Fourier measurements. Handling reconstruction problems of this size forces us to use distributed computing. However, state-of-the-art Gridding and Deconvolution algorithms only allow for limited distribution. How to scale the Gridding and Deconvolution algorithms to large problem sizes is still an open question.

Recent developments make a distributed Gridder and a distributed Deconvolution algorithm possible. Veeneboer et al[46] found an input partitioning scheme, which allowed them to perform the Gridding on the GPU. The same partitioning scheme can potentially be used to distribute the Gridding onto multiple machines. For Deconvolution, there exist parallel implementations for certain algorithms like MORESANE[4]. These can be used as a basis for a fully distributed image reconstruction.

In this project, we want to make the first steps towards an image reconstruction algorithm, which is distributed from end-to-end, from Gridding up to and including deconvolution. We create our own distributed Gridding and Deconvolution algorithms, and analyse the bottlenecks that arise.

14.2 First steps towards a distributed Algorithm

In this project, we make the first steps towards a distributed Major Cycle architecture (shown in figure 33) implemented C#. We port Veeneboer et al's Gridder, which is written in C++, to C# and modify it for distributed computing. We implement a simple deconvolution algorithm based on the previous project and create a first, non-optimal distributed version of it.

In the next step, we create a more sophisticated deconvolution algorithm based on the shortcomings of the first implementation. We use simulated and real-world observations of the MeerKAT Radio Interferometer and measure its speed up. We identify the bottlenecks of the current implementation and explore further steps.

From the first lessons, we continually modify the distributed algorithm and focus on decreasing the need for communication between the nodes, and increase the overall speed up compared to single-machine implementations. Possible Further steps:

- Distributed FFT
- Replacing the Major Cycle Architecture
- GPU-accelerated Deconvolution algorithm.

A state-of-the-art reconstruction algorithm has to correct large number of measurement effects arising from the Radio Interferometer. Accounting for all effects is out of the scope for this project. We make simplifying assumptions, resulting in a proof-of-concept algorithm.

15 Ehrlichkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende schriftliche Arbeit selbstständig und nur unter Zuhilfenahme der in den Verzeichnissen oder in den Anmerkungen genannten Quellen angefertigt habe. Ich versichere zudem, diese Arbeit nicht bereits anderweitig als Leistungsnachweis verwendet zu haben. Eine Überprüfung der Arbeit auf Plagiate unter Einsatz entsprechender Software darf vorgenommen werden.
Windisch, November 7, 2019

Jonas Schwammberger