Testing compression algorithms for macromolecular diffraction images

Kaden Badalian
Watson School of Engineering and Applied Sciences, Binghamton University, Binghamton, NY 13902

Herbert J. Bernstein
School of Chemistry and Materials Science, Rochester Institute of Technology, Rochester, NY 14623

Robert M. Sweet
NSLS II, Brookhaven National Laboratory, Upton, NY 11973

Abstract

Macromolecular Crystallography is the most effective method to determine the three-dimensional structures of large biological molecules. Collecting diffraction-image data rapidly and accurately is imperative to this technique, but the new generation of x-ray pixel-array-detectors, such as the DECTRIS Eiger, makes this task challenging because these new detectors produce very high volumes of data that can't be transferred or stored efficiently. In order to conserve resources, these images need to be compressed quickly and at a high compression ratio with little to no data loss. In order to test and compare different compression algorithms, I wrote programs that timed the compressions, compared the output file size, and compared the original image to the decompressed image both as a whole and focusing on the peaks. I tested lossless compressions, which lose no data and have lower compression ratios, and "lossy" compressions, which sacrifice some data but have higher compression ratios. One type of compression that maintains peaks of an image, features that are vital even when it is "lossy," is called wavelet compression. For this reason, I focused my "lossy" compression testing on the open-source wavelet compression software called Epsilon, which became the basis for the preferred "lossy" compression software. For the lossless compression, I tested a JPEG2000 compression which uses wavelet encoding, and an open-source compression called FLIF which uses arithmetic encoding, a type of entropy encoding. FLIF seems to outperform JPEG2000 as it compresses the images into smaller files and takes less than a third of the time to encode. For these reasons, FLIF would make a reasonable choice to investigate for the lossless compression of diffraction images. Testing these programs has demonstrated the importance of investigating all possibilities in computer science and has developed my ability to do so.

I. Background

A. Synchrotrons and Macromolecular Crystallography

Macromolecular Crystallography (MX) is the most effective method to determine the three dimensional structures of large biological molecules. Such molecules include proteins, viruses, and nucleic acids. These large molecules can be crystallized under regulated conditions, but they tend to form small, imperfect, and weakly diffracting crystals that make collecting accurate data difficult. In order to overcome this, synchrotrons are used because of their extreme brightness.

A synchrotron light source is a facility with a small synchrotron, used to get the electron to a speed close to the speed of light, and a large synchrotron called the storage ring. Once the electron is released from the small synchrotron into the storage ring, the energy (electromagnetic radiation) released by the electron is captured by tangential beamlines where experiments take place. For MX, the most heavily used electromagnetic radiation is x-ray. An MX beamline will have the macromolecule crystal of interest in the sample position followed by a pixel-array-detector to capture the diffraction pattern of the crystal.

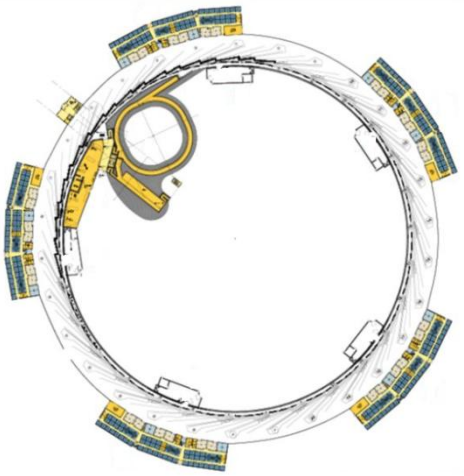


Figure 1. NSLS-II

The crystal is rotated and a diffraction image is captured at all angles. These diffraction patterns are then used by software to produce a three-dimensional model of the macromolecule. This information can then be used to advance knowledge of the relationship between structure and function, protein folding, and evolutionary relationships. One major application of this information is rational drug design.

B. New Detectors

Collecting data fast and accurately is imperative for MX. This necessity encourages companies to make new detectors that better capture MX experiments. In NSLS-II at Brookhaven National Laboratory, the detector used since its release in 2007 was the DECTRIS Pilatus. This detector has about six million pixels, each being 172 x 172 micrometers, and for the uses at Brookhaven National Lab, a frequency of 25 Hz. Along with this detector came the need to compress diffraction images because the computers and networks could not keep up with this influx of raw data. Diffraction images are compressed for two main reasons: to minimize transfer time and to conserve storage. Storing this much raw data can become very costly. This expense is reduced if the data is compressed. Trying to move too much data through a network or onto a hard drive disk can not only result in data loss, but can delay other flows of data on the same networks or storage systems. Using compressions on the data reduces these risks.

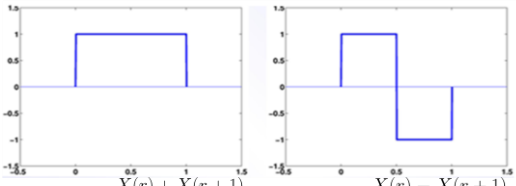
The Pilatus detector is currently being replaced by the DECTRIS Eiger which has about eighteen million pixels, each being 75 x 75 micrometers, and a possible frequency of 133 Hz. This increase of data creates the need for a more powerful compression. This is the basis for the first part of my project: to find a compression that compresses diffraction images with high compression ratios and little to no difference between the raw and compressed image.

C. Compressions

There are two very broad types of compressions: lossless and "lossy." A lossless compression doesn't lose any information, but the compression ratio isn't as high as in a "lossy" compression. A "lossy" compression sacrifices data to achieve this higher compression ratio. It is disputed whether "lossy" compression should be used on MX data. Some will argue that no

data should be altered in the diffraction images. Others will claim that some "lossy" compression algorithms maintain the integrity of the important parts of the diffraction image, the peaks/spots. Compression algorithms that use the wavelet transform seemed like they would be effective on diffraction images because the signals in diffraction images are the peaks which are brief, making the localized wavelet used in the wavelet transform seem appropriate.

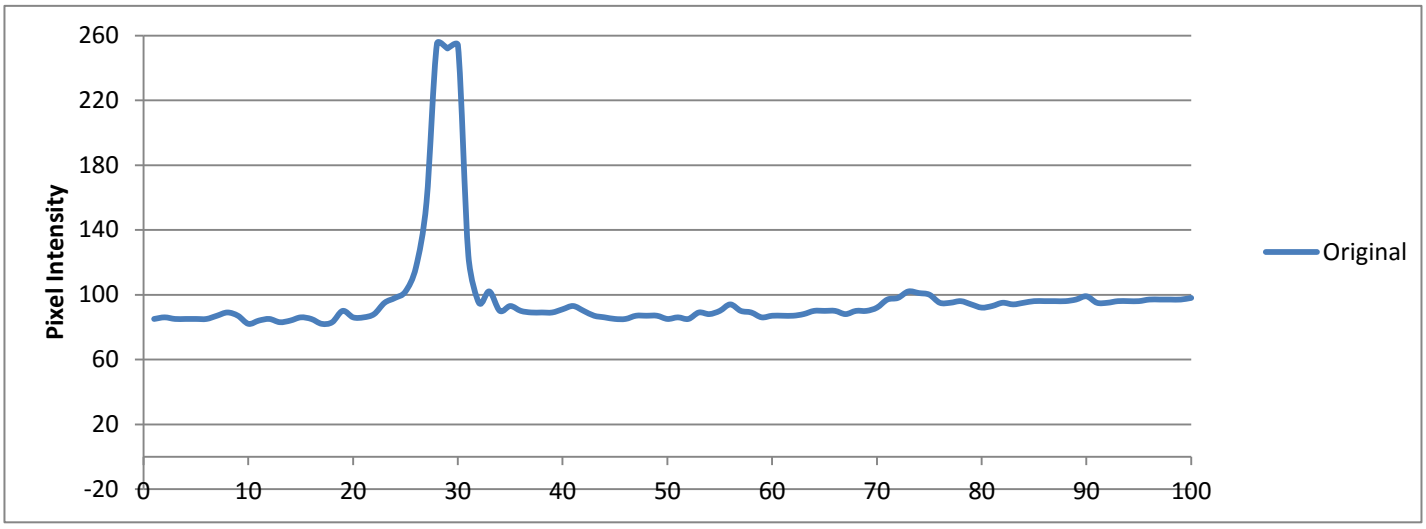
The wavelet transform starts out by choosing a mother wavelet. One example of a mother wavelet is the Haar wavelet. Two forms of the mother wavelet, the high and low pass filter, are convoluted with the data to be compressed. The low-pass filter smoothes the data so it can be stored easier. According to the Nyquist sampling theorem, about half of this data is required to faithfully reconstruct all of it. The high-pass filter essentially takes the difference between the original data and the new smoother data, so the original data can be reconstructed. The high-pass data is typically very small numbers and can be stored in few bits. This process is repeated recursively. The Wavelet Transform becomes "lossy" by throwing away the high-pass data after a few passes because it will eventually become negligible. This information that is thrown away pertains to the features in the background, so the “lossy” transformation sacrifices the background detail and adds noise, but the spots in the diffraction image are maintained. Here is an example of a wavelet transform using the Haar wavelet:



$(1) M_l X(r) = \frac{X(r) + X(r+1)}{2}$ $(2) M_h X(r) = \frac{X(r) - X(r+1)}{2}$

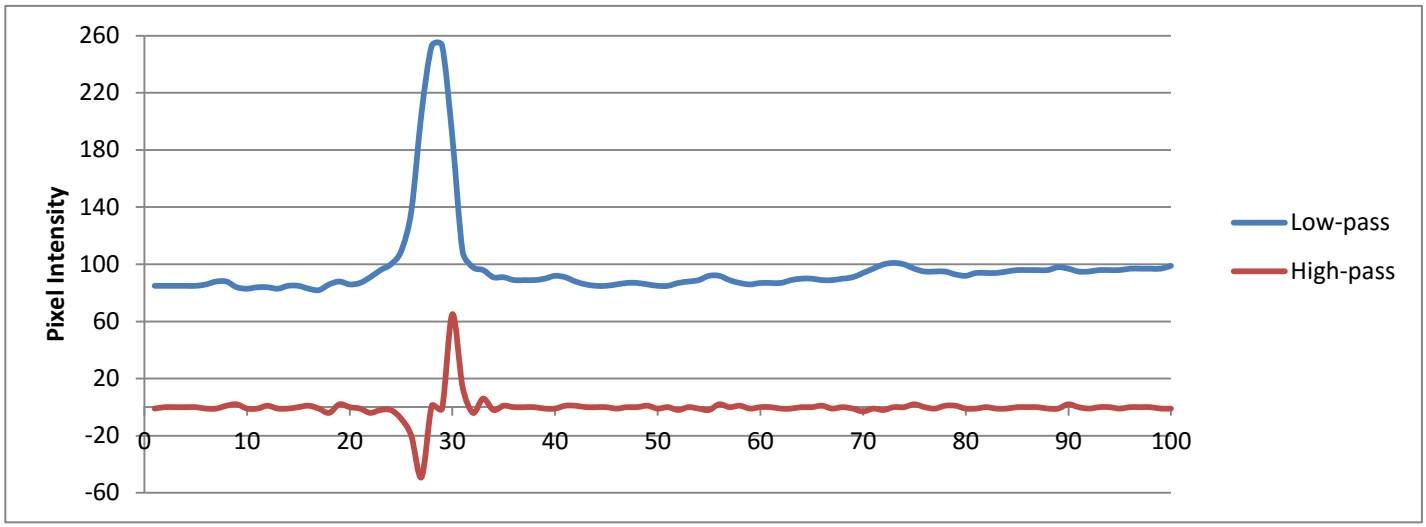
Figure 2. Haar Wavelet. Low-pass filter on left and high-pass filter on right.

Original:



This is a selection of 100 consecutive pixel intensities from Figure 3.

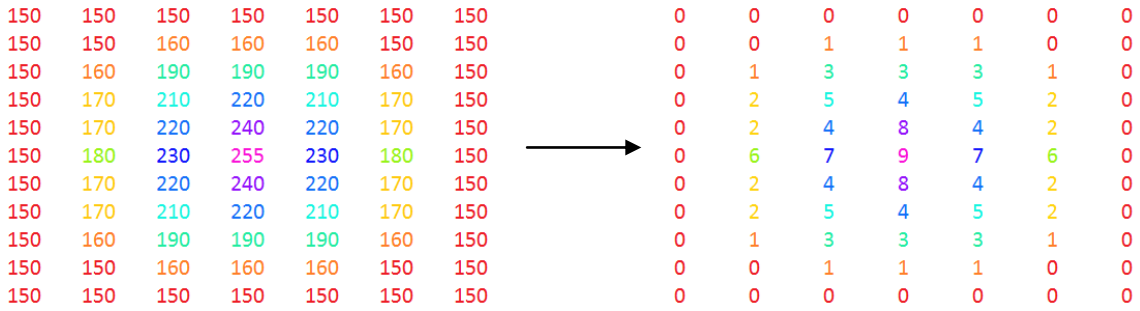
After wavelet transform:



The blue line is generated by putting the original data into equation 1 (low-pass filter) to get a smooth line that requires storing about half the data points.

The red line is produced by using the original data in equation 2 (high-pass filter) to get small numbers that require fewer bits to store.

Other compression algorithms use entropy encoding. Entropy encoding is effective on any type of image because it essentially takes the most used pixel intensities and stores them with the fewest bits possible, and stores the least common pixel intensities with the most bits necessary. Here is an example of entropy encoding:



The compression software that I tested was focused around the wavelet transform and entropy encoding.

II. Methods

A. Testing Compressions

After researching open-source compression options, this became the list of compression software that I would test:

- ImageMagick (JPEG, High Quality JPEG, PNG)
- Epsilon
- FLIF
- OpenJPEG (JPEG2000)

JPEG and High Quality JPEG is a "lossy" compression that uses the discrete cosine transform which is similar to the wavelet transform but uses cosine functions instead of wavelets. Epsilon

and JPEG200 are wavelet compressions, but Epsilon is "lossy" and JPEG2000 is lossless. FLIF and PNG are lossless compressions that incorporate types of entropy encoding.

To test the effectiveness of these compressions I wrote a Python script that compared each pixel of the raw image to the corresponding pixel in the compressed image and then output the average difference of any given pixel and the greatest difference between corresponding pixels. This script was soon revised to focus on the peaks of Figure 1. The peaks were defined as any pixel with an intensity over 25 for this 8-bit image. This definition of a peak is broad and inaccurate, which became the segue to the next part of my project: a spot finder.

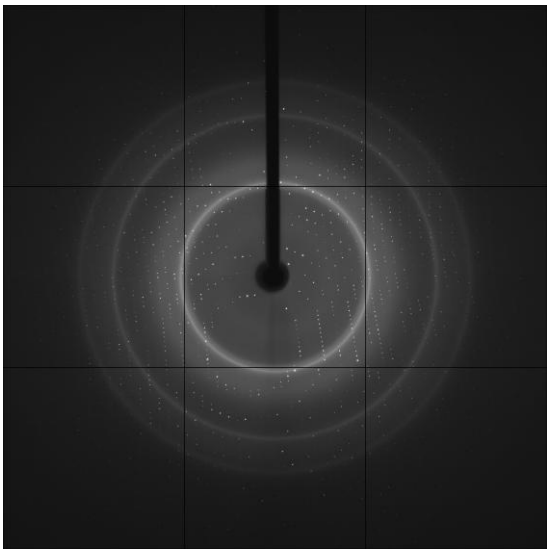


Figure 3. Diffraction Image

Other important factors of a compression algorithm are the time it takes to encode, and the output file size. To time the compression, I used the command "time" in front of the command used for encoding in Terminal.

B. Spot Finder

The spot-finding-algorithm that I used was to get the average and standard deviation of the pixel intensities in a given box around the pixel of interest, making sure to exclude outliers. To exclude the outliers, the top and bottom 10% of pixel intensities in the box were excluded from the average and standard deviation. After finding these values, the pixel of interest is defined as a spot if it is some number of standard deviations above this local average. For a

proof of concept test, I coded this algorithm in Python and, for simplicity, entirely recalculated the average and standard deviation of the box around the pixel of interest every time. This program took a very long time to run because all of the pixels in each box had to be fetched even though most of the pixels between adjacent boxes overlapped. The complexity of this algorithm was about $O(n^2)$. To avoid this, I implemented a sliding box algorithm to keep as much of the sum of the middle 80% of data as possible to calculate the average. I knew that using the traditional standard deviation equation, none of the calculations could be kept, but using the alternate standard deviation equation below, all I would have to do is add a variable that keeps the sum of the squares since I already have the sum for the average. With these changes, each pixel only has to be fetched when being added to or subtracted from a box, making the complexity O(2n).

$$\sigma = \sqrt{\frac{n * \sum x_i^2 - (\sum x_i)^2}{n * (n-1)}}$$

III. Results and Conclusions

A. Testing Compressions

Figure 3 is the image I tested the compressions on. It was originally 9.4 MB and 8-bit, so numbers are in the range of 0-255. Below is a table of the results.

Compression	**Greatest Spot Difference**	**Average Spot Difference**	**Output File Size (MB)**	**Time (s)**
JPEG	4	0.0018	0.88	0.16
High Quality JPEG	1	0.046	1.8	0.2
PNG	0	0	3.8	1.0
Epsilon	7	0.6	1.3	1.3
FLIF	0	0	2.4	1.0
JPEG2000	0	0	2.5	3.5

Out of the lossless compressions (PNG, FLIF, and JPEG2000), FLIF performed the best. Not only did it compress the image the best, but it compressed the fastest as well. From the "lossy" compressions, I will be focusing on Epsilon because the desired compression ratio can be

chosen and I won't run into copyright issues using it. For these reasons, FLIF and Epsilon will be further investigated for their use in MX.

B. Spot Finder

After testing the spot finder with different box sizes and spot definitions, I settled on a box size of 21 and defined the spots as 5 standard deviations above the average for my test image. Some variables in this algorithm are subject to change depending on the size of the spots in an image. Images with larger peaks will need either larger boxes or a larger barrier because all of the outliers won't be absorbed by the 10% barrier of a smaller box. As seen in the image to the left, the algorithm accurately finds the spots, even across difficult features of images such as the ice ring seen in this sample image. Since the algorithm seems to work pretty well, it will continue to be developed and hopefully used to identify useful images.

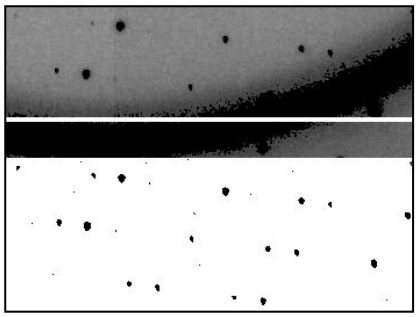


Figure 4. Part of Fig. 2 on top and output of spot finder below it

IV. References

Figure 1: Young, Leonardo. "1 BROOKHAVEN SCIENCE ASSOCIATES EFAC Review Conventional Facilities Update Marty Fallier Director for Conventional Facilities May 10, 2007." http://slideplayer.com/slide/8995828/

Figure 2: Cheng, Kileen. "Example Wavelets." https://cnx.org/contents/K5j4YUB6@4/Example-Wavelets

Figure 3: Myoglobin image courtesy of Christopher Nielsen of Area Detector Systems Corporation

V. Acknowledgements

This project was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships Program (SULI).