BMC Bioinformatics

SOFTWARE Open Access

Image-centric compression of protein structures improves space savings

Luke Staniscia[1] and Yun William Yu[1,2*]

*Correspondence: ywyu@cmu.edu

[1] Department of Mathematics, University of Toronto, Toronto, ON, Canada
[2] Ray and Stephanie Lane Computational Biology Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Background: Because of the rapid generation of data, the study of compression algorithms to reduce storage and transmission costs is important to bioinformaticians. Much of the focus has been on sequence data, including both genomes and protein amino acid sequences stored in FASTA files. Current standard practice is to use an ordinary lossless compressor such as gzip on a sequential list of atomic coordinates, but this approach expends bits on saving an arbitrary ordering of atoms, and it also prevents reordering the atoms for compressibility. The standard MMTF and BCIF file formats extend this approach with custom encoding of the coordinates. However, the brand new Foldcomp tool introduces a new paradigm of compressing local angles, to great effect. In this article, we explore a different paradigm, showing for the first time that image-based compression using global angles can also significantly improve compression ratios. To this end, we implement a prototype compressor 'PIC', specialized for point clouds of atom coordinates contained in PDB and mmCIF files. PIC maps the 3D data to a 2D 8-bit greyscale image and leverages the well developed PNG image compressor to minimize the size of the resulting image, forming the compressed file.

Results: PIC outperforms gzip in terms of compression ratio on proteins over 20,000 atoms in size, with a savings over gzip of up to 37.4% on the proteins compressed. In addition, PIC's compression ratio increases with protein size.

Conclusion: Image-centric compression as demonstrated by our prototype PIC provides a potential means of constructing 3D structure-aware protein compression software, though future work would be necessary to make this practical.

Keywords: Protein structure, Compression, Image

Background

For over half a century, determining protein structure has been a primary means of understanding function and behavior [1, 2]. After proteins are characterized by researchers using various methods such as X-ray crystallography, NMR spectroscopy, and cryo-electron microscopy, various files are generated describing the protein and stored in online repositories such as the Protein Data Bank [3, 4]. One such file, the FASTA file, contains strings of characters representing the amino acids that make up the protein and its variants [5]. Other files, such as PDB (Protein Data Bank format) and mmCIF (macromolecular Crystallographic Information File) files, contain structural information about the protein [6]. Although the Protein Data Bank is no longer growing exponentially, the number of new structures deposited is still quite formidable [4]; furthermore, the recent publication of AlphaFold predicted structures has increased total available structures by orders of magnitude [7].

FASTA files are used for storing both protein and genomic sequence information, and much work has been done to create customized sequence compression algorithms. It bears mentioning that the genomic sequence compression literature has recently seen significantly more activity with the advent of next-generation sequencing [8–11], and many protein sequence compressors take advantage of that work. For protein sequences, [12] introduce a single and double pass version of a amino acid sequence compressor for FASTA files that makes use of substitution matrices. MFCompress was introduced by [13], and converts the amino acid sequences to their corresponding DNA bases, divides the data into three streams, and compresses the resulting streams. CoMSA is another compression algorithm for FASTA files introduced by [14] based on a generalized Burrows-Wheeler transform. Similarly to MFCompress, The Nucleotide Archival Format (NAF) introduced by [15] is another compressor that works on amino acid sequences converted to their corresponding DNA bases by dictionary encoding this transformed string.

In addition to directly transforming and compressing the sequences in FASTA files, a significant amount of research has gone into read-reordering algorithms for genomic sequences in the BEETL [16], SCALCE, [17], MINCE [18], and more. These methods are applicable when FASTA (and the related FASTQ) files are used to store multiple small fragments ('reads') of sequences; next-generation sequencing produces these reads in no particular order, so the reads can be safely reordered without losing important information. When properly performed, this reordering can significantly improve the compression ratio of standard compressors.

On the other hand, the primary data component of PDB and mmCIF protein structure files is a point cloud of coordinates belonging to the atoms that make up the protein. In the standard formats, each atom has its own separate ASCII-formatted line entry in the file that contains the type of atom, type of amino acid to which it belongs, atom and amino acid identifiers, followed by three floating point Cartesian coordinates, along with other information. The coordinates are measured in units of Angstroms Å, where $1\,\mu\text{m} = 10{,}000$ Å [19]. Unlike their FASTA counterparts, comparatively less work has been done to create compressors customized for the structural data contained in PDB and mmCIF files, though there have been a number of recent tools/formats like MMTF [20], BCIF [21], and the brand new Foldcomp [22]. We note with especial interest Foldcomp, which introduces a new paradigm for compressing atomic coordinates using local angles, which is a radical shift from what both MMTF and BCIF do. In this manuscript, we explore yet another different direction in the form of image-centric compression and global angle computations.

Valasatava et al. [23] did a deep investigation on compressing 3D coordinates of atoms in proteins by investigating a full gamut of compression techniques. Their final recommendation was to apply "intramolecular compression", which aims to reduce the size of each protein via three steps: encoding, packing, and entropy compression. The encoding step transforms floating point coordinates into alternate representations, such as Integer, Delta, Predictive, Wavelet, and Unit Vector encodings. Integer encoding as described by [23] multiplies the floating point location coordinates by a power of 10 and rounds the result to the nearest integer. This encoding strategy is lossy when not all decimal places of precision are kept in the integer encoded value, but it can be lossless when used in MMTF and BCIF with a sufficiently large power of 10. However, some amount of loss of precision can be acceptable because of both measurement error, and due to the natural uncertainty of exact atom locations in a protein—PIC will use a lossy variant of integer encoding. Going back to [23], the authors suggest that after packing the encoded coordinate vectors using either recursive indexing or variable packing, the resulting packed coordinates are entropy encoded using standard methods like gzip [24] or brotli [25], which are both combinations of LZ77 dictionary based encoding and Huffman encoding.

However, [23]'s investigation focused primarily on compression of atomic coordinates as sequential objects stored within a text file, treating the data as sequential, much like in FASTA files without reordering. However, unlike protein/genomic sequences, 3D atomic point clouds are not naturally sequential, and the sequence of atoms listed is purely an artifact introduced by using a sequential file format to store the atoms. Thus, preserving the order of the atoms as listed out in a PDB or mmCIF file is largely irrelevant for the purposes of compression, so long as the original information can be reconstructed. Given the background above, one logical next step would be to perform a principled reordering of the atoms to improve compressibility, similar to the technique used by read-reordering algorithms (where again, the order of reads output by the sequencer is inconsequential). The remaining question is of course how to perform that reordering, as point clouds are very different from sequenced genomic reads in underlying structure.

To resolve this question, we turn to an alternate paradigm for compressing point cloud data sets proposed in the field of LIDAR (light detection and ranging) imaging. Houshiar and Nüchter [26] proposed a new compression algorithm for the 5D point cloud data generated by LIDAR scans of real-world scenes. The LIDAR scans produced tuples of data points containing coordinates of a point in space in the scene, along with reflectance and colour data of the surface at that location. Their compression algorithm converts the Cartesian coordinates to spherical coordinates, maps the angular coordinates to the axes of an image, and the radial component, colour, and reflectance data to pixel's fields at the mapped location. The radial component, colour, and reflectance data are written to the R, G, and B components respectively of a single coloured image as well as the greyscale intensity field of three separate consecutive images. The resulting images were compressed using PNG, JPEG 100 (lossless, perfect quality JPEG), JPEG2000, no compression TIFF, LZW TIFF, and Pack Bits TIFF lossless image compressors. The authors of [26] found that compressing three greyscale images using the PNG compressor performed the best in terms of compression ratio.

In this manuscript, we take inspiration from the next-generation sequencing read-reordering literature and combine the intramolecular compression techniques of [23] with the image-centric methods of [26]. In "Implementation" section, we outline our new compression algorithm, PIC, for the structural protein data contained in PDB and mmCIF files. Design choices and methodology are examined in detail followed by a pseudo-code outline of the compression algorithm. In "Results" section, we give compression results for the atomic coordinates of 20 proteins of a variety of different sizes compressed using both PIC and gzip and show PIC outperforms gzip in terms of compression ratio for proteins over 20,000 atoms in size. We also give the images that constitute the compressed files for a few of the compressed proteins. Furthermore, although PIC is not a full compressor as it does not compress metadata, for the sake of completeness, we also compare PIC file sizes against full compression software MMTF, BCIF, and Foldcomp. In "Discussion" section, we highlight some trends in the compression results and make note of the advantages of the PIC compressor over gzip for structural protein data compression.

```
1: Read PDB/mmCIF file
2: Compute global centroid $\mu$ and the maximum radial component $r^*$
3: for each atom in the protein do
4:     Translate atom's Cartesian coordinates by $-\mu$
5:     Convert the translated coordinates to spherical coordinates
6:     Compute target encoding position $(x, y)$ and keep the elevation angle $\theta$
7:     Queue $[(x, y), \theta]$
8: end for
9: while the queue is not empty do
10:    Initialize a new image
11:    for each data point in the queue do
12:        if $(x, y)$ is not available then
13:            Compute $(x^*, y^*)$
14:        end if
15:        if an encoding position is available in the current image then
16:            Pack $\theta$ based on the encoding position
17:            At the encoding position, write packed $\theta$
18:            Remove the data point from the queue
19:        end if
20:    end for
21: end while
22: Crop the resulting image(s)
23: Compress the cropped image(s) using the PNG compressor on the highest compression ratio setting
24: Construct the parameter file with the image count, compression parameters, and $\mu$ to enable
    decompression
```

Algorithm 1 PIC compression algorithm

Implementation

The PIC compression algorithm has has three main components, namely mapping each atom to a position in an image, encoding information at that position, and compressing the resulting image. A high-level overview is given in Algorithm 1 and Fig. 1, and details are furnished in the following text.

Mapping

Cartesian coordinates of atoms stored in the protein's PDB or mmCIF file are extracted and the global centroid μ of all the coordinates is computed. The coordinates are translated by $-\mu$ so that the global centroid becomes the new reference point or origin for the coordinates. This transformation minimizes the instances of collisions when mapping the coordinates to the image. To decompress the images, μ is stored along with the images.

The translated coordinates are then transformed to spherical coordinates. Each spherical coordinate component is rounded to a precision of one decimal place. Valasatava et al. [23] noted that experimental measurements that produce the Cartesian coordinates determine an atom's position with a degree of uncertainty, greater than $0.2\mathring{A}$. This

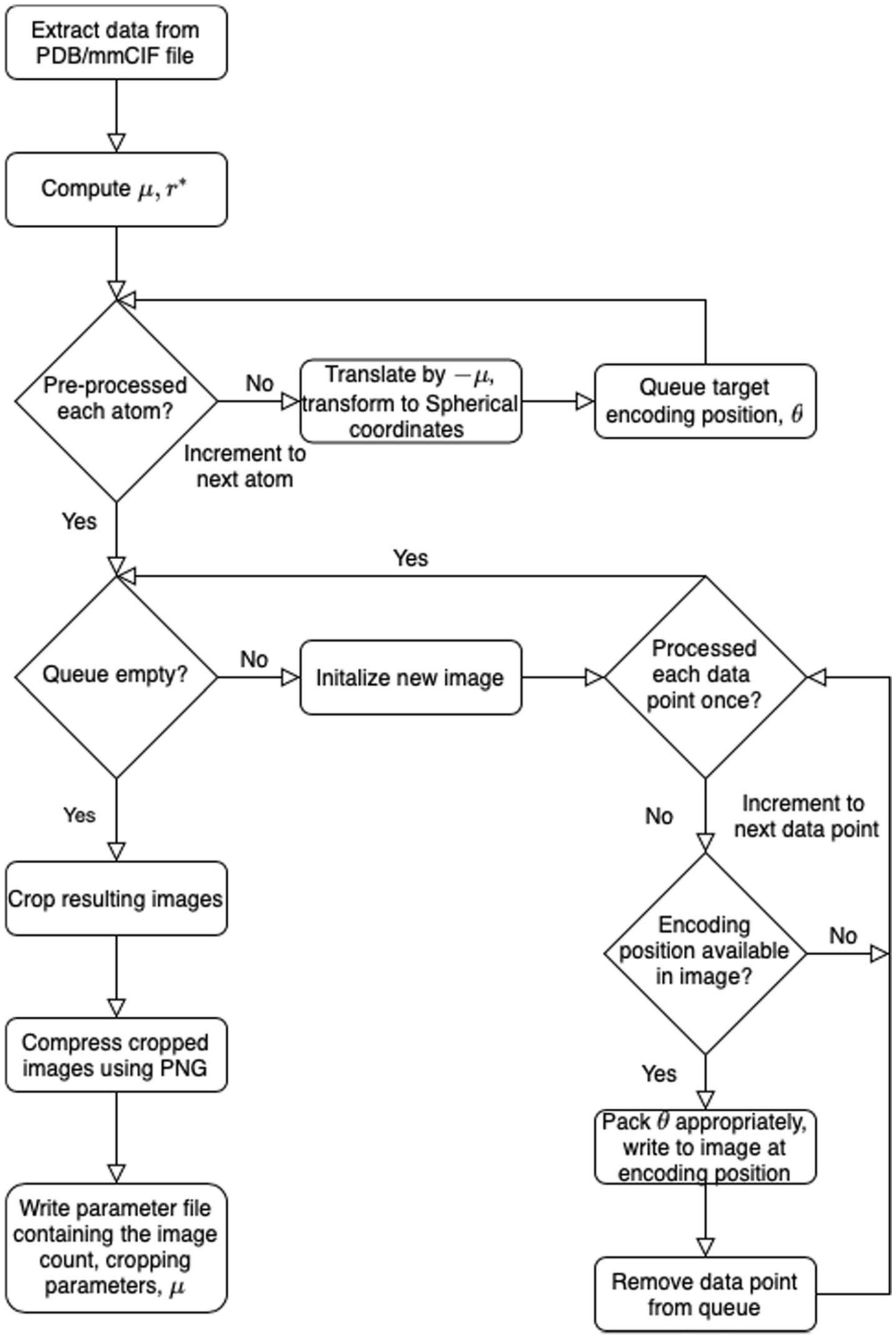
Extract data from PDB/mmCIF file
Compute μ, r^*
Pre-processed each atom?
No
Translate by $-\mu$, transform to Spherical coordinates
Queue target encoding position, θ
Increment to next atom
Yes
Yes
Queue empty?
No
Initalize new image
Processed each data point once?
Yes
Crop resulting images
No
Increment to next data point
Encoding position available in image?
No
Compress cropped images using PNG
Yes
Pack θ appropriately, write to image at encoding position
Write parameter file containing the image count, cropping parameters, μ
Remove data point from queue

Fig. 1 Flow chart diagram of the PIC compression algorithm. μ is the global centroid of all atoms used to center the image, and r^* is the maximal radial component that needs to be stored after centering. The basic intuition is to store atoms and their coordinate data in a pixel corresponding to the radial coordinates, and then compress with PNG

allows for the exploitation of lossy compression to store the coordinates only up to a tenth of an $\AA$, which is generally sufficient to preserve the essential structural information provided by lossless representation.

The radial r and azimuth ϕ spherical coordinate components of each atom are positionally encoded to the horizontal and vertical axis of an eight bit pixel greyscale image as follows

$$(x, y) = (10r, \varepsilon\phi)$$

where ε is a user-defined parameter that sets the number of pixels per azimuth angle degree. Letting r^* be the maximal radial component across all spherical coordinates, the dimensions of the resulting image are $10r^* \times 360\varepsilon$. Further note that while $x \in \mathbb{Z}_{\geq 0}$, y is not necessarily an integer. However, ε is chosen such that $360\varepsilon, 8y \in \mathbb{Z}_{\geq 0}$ for all y and $\varepsilon \geq 1.25$. This ensures there is at least one bit available per tenth of an azimuth angle degree and each y coordinate has an integer bit-level position on the vertical axis. In this way, we view each column in the image as a bit string that is being written to.

Care must be taken when choosing ε. Setting ε too large will produce a large image, degrading the compression ratio. On the contrary, choosing a small ε will induce more collisions when data is mapped to the image. This results in increased compression time, as alternate data storage locations need to be considered. A decrease in the compression ratio may also be experienced in this case as more data points will need pointers to their intended locations and additional images may need to be populated to store all the required data.

The remaining elevation angle θ is stored in the image's pixel intensity values beginning at the data point's (x, y) encoding position in the image. Further details on how the elevation angle is formatted or *packed* and stored in the image is described in "Packing" section. This encoding scheme was selected as it positionally encodes the spherical coordinates r and ϕ with the largest range of values and encodes the smallest ranging coordinate θ in the image's pixel's intensity values. Thus each coordinate takes up the fewest amount of pixels when encoded into the image, allowing for more data to be stored in the image before another image needs to be generated.

In the event that a data point is mapped to a position that does not have availability to hold all the required information, an alternate encoding position (x^*, y^*) is determined systematically. A position (x, y) has availability if all bits at positions between and including (x, y) and $(x, y + l - 1/8)$, where l is the length of the encoded elevation angle in bytes, have not had data previously written to them. Beginning at the data point's target encoding position (x, y), the positions $(x, y + i/8 \mod 360\varepsilon), 0 < i < 8 \cdot 360\varepsilon = 2880\varepsilon$ are scanned subsequently to find the first position with availability. This position is the alternate encoding position. All encoding positions (x, y) also satisfy $y < y + l - 1/8 < 360\varepsilon$, ensuring no data points begin at the bottom of the image and finish at the top to enable proper decompression of the image.

If (x^*, y^*) is the alternate encoding position for a data point with target position (x, y), and $y \leq y^* < y + 0.1\varepsilon$, the encoded elevation angle is stored begining at (x^*, y^*) as is. Otherwise, a pointer p is encoded and stored along with the encoded elevation angle at (x^*, y^*). p points to the largest $y' \in \{i/8 | 0 \leq i < 2880\varepsilon\}$ that satisfies $y \leq y' < y + 0.1\varepsilon$, namely $y' = y + 0.1\varepsilon - 1/8$. The stored pointer is the integer $p = 8(y^* - y')$. Note that $p > 0$ as $y^* > y'$. The decompressor then knows that the intended azimuth angle for the data point is that belonging to the position $(y^* - p/8) = y'$.

In the case that an alternate encoding position cannot be found in the current image, another image is generated, if not already done by a previous data point. The above mapping procedure is repeated in that image to locate an encoding position for the data point. This process repeats until an encoding position is determined for each atom's coordinate.

Packing

Elevation angles are stored in pixels' greyscale values beginning at their corresponding data point's (x, y) encoding position. Each pixel has an 8-bit intensity field. Due to the variable lengths of the binary elevation angles and use of pointers, the following packing scheme is used to store the elevation angles so they can be properly decompressed.

If no pointer is needed, the elevation angle is integer encoded as 10θ and converted into its binary representation. If the binary representation has length less than $\lceil \log_2(1801) \rceil = 11$ bits, 0 bits are added to the front until the representation is 11 bits long. Two additional bits 1 and 0 are added to the front of the resulting binary string in that order to signify the start of a new data point and to notify the decompressor the data point has no pointer, respectively.

If a pointer is required, a similar but expanded packing scheme is used. The second bit is set to 1 instead of 0 to signify to the decompressor that the data point has a pointer. The pointer p is converted to its binary representation and prefixed with 0 bits until it has length $\lceil \log_2(2880\varepsilon) \rceil$. The adjusted binary representations of the pointer and elevation angles follows the two bit prefix in that order.

For $0 \leq i < 8l$, bit i of the packed string is mapped to the bit at position $(x, y + i/8)$ in the image. This packing scheme ensures that each data point has one of two possible lengths, the exact length of which can be determined by the second bit located at $(x, y + 1/8)$. This is a key feature that allows for the proper decompression of the image.

Cropping and compression

The resulting image(s) are cropped and compressed using the PNG lossless image compressor on the highest compression ratio setting. These image(s) make up the compressed version of the protein's point cloud of atom coordinates in the PDB or mmCIF file.

Images are cropped to remove any all-black rows and columns on the edge of the image. To decompress the images, two cropping parameters are stored along with each image generated to reverse the cropping.

Other lossless image compressors investigated in [26] were also examined. Similarly to the results found by Houshiar at al., PNG was selected for use in the algorithm as it offers the highest compression ratios of the aforementioned compressors at comparable compression times.

Decompressed file

The original and decompressed files are identical up to the coordinates of the atoms. As noted in "Mapping" section, since there is a tolerance of up to $0.2\mathring{A}$ in each coordinate component, each decompressed coordinate is within a euclidean ball of radius $0.2\sqrt{3}\mathring{A}$ about the original coordinate.

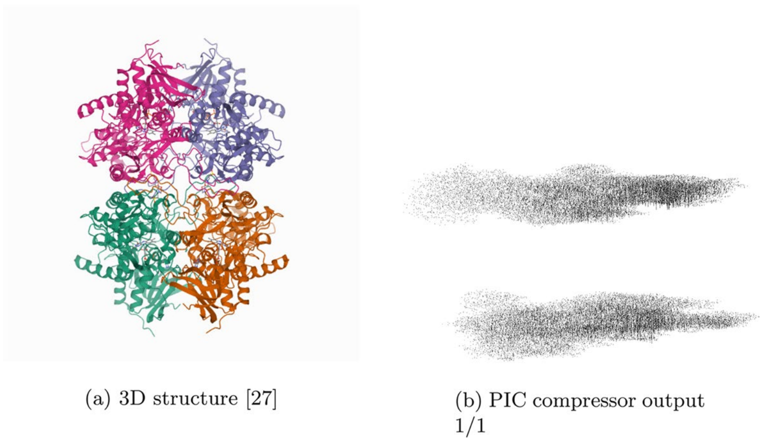

Fig. 2 3D structure [27] and PIC compressor PNG image output for 2ign. Some attributes and symmetries in the 3D structure are observed in the corresponding PIC-compressed image. The upper and lower parts of the 3D structure of protein 2ign can be seen in PIC generated image as two separate masses of black pixels, one over the other

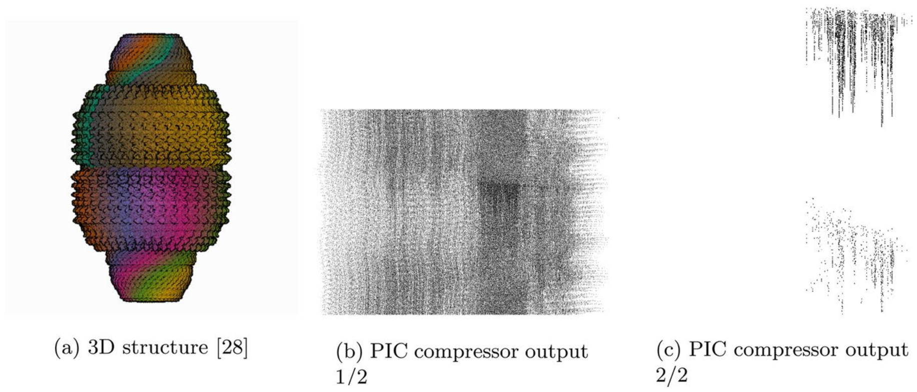

Fig. 3 3D structure [28] and PIC compressor PNG image output for 4v60. Some attributes and symmetries in the 3D structure are observed in the corresponding PIC-compressed image. The spiked edge of the 4v60 protein can be seen on the right side of the first outputted image from the PIC compressor

Results

Atomic cloud coordinate compression

We benchmarked PIC against the gzip compression after the integer compression/precision reduction of [23], the primary relevant prior work. Valasatava et al. [23] explored a variety of methods for entropy compression, but we found the differences between methods to be swamped out by the integer encoding step, and thus chose gzip as a representative method for sequential compression. We did not compare against plain gzip for that reason, as the compression ratios without the integer compression were not at all comparable. All further references to gzip are to gzip after the [23] integer compression.

Table 1 gives statistics and compression results on 20 proteins compressed using gzip and PIC where $\varepsilon = 2.5$ and the decompressed files are identical to the original with the lossy coordinate transform. Figures 2 and 3 compare the 3D structures of proteins to the images created by the PIC compressor. Figure 4 compares PIC vs

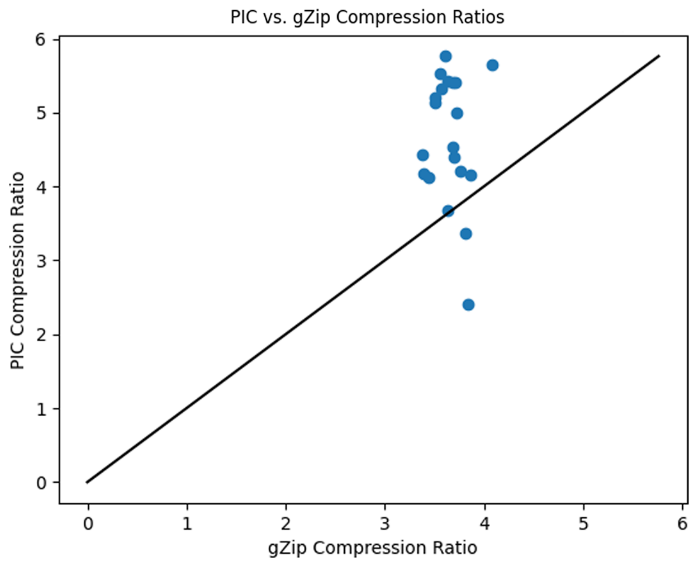

Fig. 4 PIC compression ratios plotted against gzip compression ratios (using integer precision reduction for both methods reduced to a tenth of an angstrom for comparability) for each protein compressed in Table 1. Points in the region above the diagonal indicates a protein with better compression ratios using PIC than gzip. Vice versa below the diagonal. PIC demonstrates substantially higher compression ratios for nearly all proteins tested

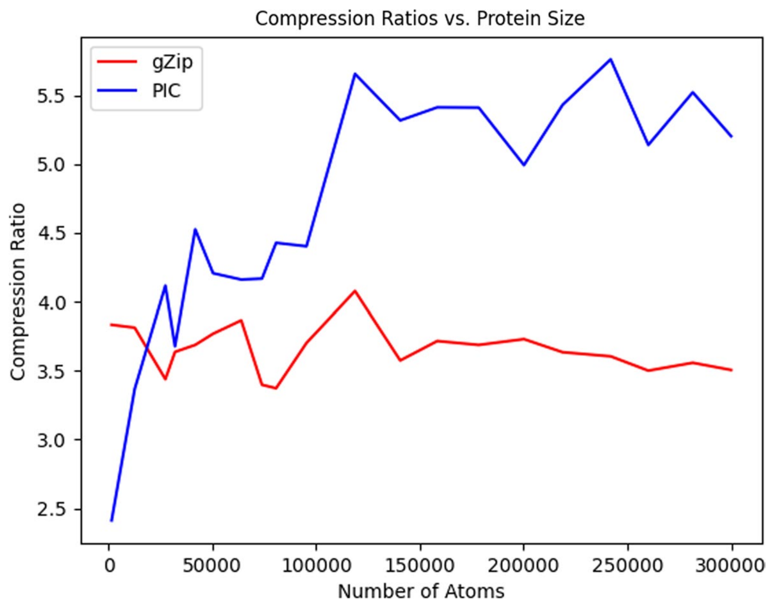

Fig. 5 PIC and gzip compression ratios (using integer precision reduction for both methods as suggested by [23] for comparability) for the proteins compressed in Table 1 plotted against the number of atoms that make up the compressed protein. All but the smallest proteins showed a higher compression ratio when using PIC; for the small proteins, the extra overhead of PIC dominates, but for any large protein, PIC performs better. Note that this comparison is fair to gzip, as instead of gzipping the original files, we only apply gzip after using the same lossy precision encoding that PIC uses; thus, the comparison here is really between sequential storage of a text file using gzip, and spherical storage using PIC

gzip compression ratios, whereas Fig. 5 visualizes some of the same results found in Table 1, but plotted against atom size.

As can be seen from Table 1, the proposed PIC algorithm has superior compression ratio performance than the standard gzip text compressor for all proteins over 20,000 atoms in size. This is seen visually in Fig. 5, as all except two points belonging to the two proteins with the fewest number of atoms lie above the diagonal, the region where PIC has better compression ratio performance. In Fig. 5, the gzip compression ratio decays while PIC's compression ratio increases with atom count.

Furthermore, unlike most compression algorithms, we can visually inspect the transformed image because it is itself a projection mapping of the original 3D structure. In Figs. 2 and 3, we show the PIC outputted images. For easier viewing, these images are inverted, five-fold contrast enhanced versions of the actual images outputted by the PIC compressor.

These results were obtained by running the *PIC.py* script in the command terminal with the "-e" option. These experiments were ran on a Ubuntu 20.04.4 LTS machine with an AMD Ryzen Threadripper 3970X 32-Core Processor and 256 GB of memory in single-thread mode without parallelization. However, the code has also been tested on an Apple MacBook Pro with a 3.5 GHz dual-core processor and 16 GB of memory, with comparable results. Thus, the code can run nearly as well on personal laptops.

Full PDB/mmCIF compression benchmark

Although PIC is designed as a prototype to showcase image-centric compression and thus only compressed the atomic point clouds, it is still instructive to compare against full compression software, such as MMTF, BCIF, and Foldcomp. In order to create a fair comparison, the total metadata space also needs to be included when comparing PIC—as such, we decided to use use MMTF to compress only the metadata, and then add that size to the size of the PIC image output. This is of course impractical for use as a compressor, but simply serves to level the playing field.

In Table 2, we compare the same benchmark proteins as in Table 1 with original PDB size, BCIF, MMTF, MMTF-lossy, PIC+MMTF-meta, and Foldcomp. MMTF-lossy notably both decreases precision to tenth of an Angstrom (same as PIC), but also only stores the C-alpha atoms, which allows them to take the least space at the cost of not storing all atoms. We were only able to get Foldcomp v0.0.5 to work on one of our proteins, 4v60, because most of our benchmark proteins had discontinuous chains, which is not supported by Foldcomp, and several of the other proteins caused segfaults. However, on both 2jan and 4v60, Foldcomp does substantially better than PIC or any of the other compressors other than MMTF-lossy.

Discussion

As expected, as atom count increases, more images are populated by PIC and more of the image space of the constructed images is used. In addition to the increased data load in only a slightly larger image width wise, this is due to an increased number of collisions as atom count increases. This causes the use of pointers, increasing the average number of bits used per data point, and, when no alternate location can be found in the current image, the population of a new image, increasing the image count.

Table 1 Compression and runtime comparisons of gzip and PIC

Protein	Atom	Original	Rounded coordinates		gzip		PIC			Compression	Images		Decompression
ID	Count	File	Text	Binary	Size	CR	Size	CR	RMSD	Time	Number	Space	Time
		size (KB)	size (KB)	size (KB)	(KB)		(KB)			(min:sec)	used	used (%)	(min:sec)
2ja9	1458	163.3	24.1	6.6	**6.3**	**3.834**	10.0	2.412	0.031	0:0.1	1	[0.9]	0:0.4
2jan	12591	1101.2	206.1	56.7	**54.1**	**3.813**	61.2	3.368	0.047	0:1.3	1	[2.7]	0:2.1
2jbp	27367	2397.4	447.8	133.4	130.2	3.439	**108.8**	**4.117**	0.043	0:4.1	1	[11.1]	0:5.0
2ja8	32000	2831.2	507.6	144.0	139.6	3.637	**138.0**	**3.678**	0.043	0:5.6	1	[6.5]	0:7.0
2ign	41758	3579.2	666.7	187.9	180.8	3.688	**147.3**	**4.526**	0.069	0:9.0	1	[9.5]	0:11.4
2jd8	50351	4457.6	828.1	226.6	219.7	3.769	**196.8**	**4.207**	0.056	0:12.8	1	[7.7]	0:15.9
2ja7	63924	5605.5	1077.0	287.7	278.6	3.866	**258.8**	**4.161**	0.055	0:19.6	1	[10.2]	0:24.7
2fug	73916	6386.9	1180.7	360.3	347.5	3.398	**283.3**	**4.168**	0.060	0:26.2	1	[10.7]	0:33.3
2b9v	80710	6818.4	1279.8	393.5	379.4	3.373	**289.0**	**4.428**	0.073	0:32.2	1	[10.3]	0:39.5
2j28	95358	8152.3	1526.2	429.1	412.2	3.702	**346.6**	**4.403**	0.055	0:47.0	1	[13.7]	1:0.2
6hif	118753	12726.2	2105.2	534.4	516.2	4.078	**372.2**	**5.656**	0.062	1:30.5	2	[34.0, 0.1]	1:48.6
3j7q	140540	16027.2	2529.7	737.8	707.6	3.575	**475.6**	**5.318**	0.058	2:28.2	1	[20.3]	2:44.4
3j9m	158384	17995.2	2845.4	772.1	765.8	3.716	**525.7**	**5.413**	0.069	3:28.8	1	[21.7]	3:55.9
6gaw	178372	20825.4	3179.9	869.6	862.1	3.688	**587.6**	**5.411**	0.071	4:58.1	1	[23.5]	5:39.1
5t2a	200172	22787.6	3253.9	900.8	872.4	3.73	**651.7**	**4.993**	0.068	7:8.2	2	[31.1, 1.7]	8:59.1
4ug0	218776	24906.9	3841.4	1066.5	1056.7	3.635	**707.3**	**5.431**	0.069	8:34.2	2	[33.8, 1.7]	9:25.5
4v60	241956	24377.8	4207.8	1179.5	1167.2	3.605	**730.2**	**5.762**	0.120	9:50.8	2	[45.6, 2.1]	13:48.9
4wro	260090	35661.1	4363.1	1267.9	1246.2	3.501	**848.8**	**5.14**	0.086	13:54.0	1	[29.6]	16:6.9
6fxc	281510	31329.0	5067.1	1477.9	1424.2	3.558	**917.7**	**5.522**	0.100	15:52.9	2	[34.6, 1.0]	17:11.6
4wq1	299951	40130.9	5042.1	1462.3	1438.0	3.506	**968.8**	**5.204**	0.087	19:59.6	2	[34.7, 0.2]	22:39.0

PIC compression algorithm, $\varepsilon = 2.5$, results. Rounded Coordinates Text Size and Binary Size are the sizes of the text and binary files (in kilobytes, i.e. 1000 $\times$ bytes, rather than kibibytes), respectively, that contain only the Cartesian coordinates found in the original file, rounded to one decimal place. The binary file (which uses a variable-length encoding) is then gzipped. The gzip and PIC compression ratios (CR) are the ratios of the Rounded Coordinates Text Size to the size the gzip file and PNG image output(s) from the PIC compressor, respectively. Bolded values are the best of gzip and PIC. Compression and decompression times are for the PIC algorithm; note that our code is unoptimized, as the focus is on compression ratios, but we include these times here for completeness. As an aside, (de)compression for gzip takes negligible time for files of this size. We also include RMSD values to measure the lossiness of PIC compression. Image Space Used gives the proportion of the image space that was used to encode the protein coordinate data, or part thereof, in each image constructed by the PIC compressor (for large proteins, more than one image is needed to represent all the atoms)

Table 2 Compression of entire PDB/mmCIF files benchmark

Protein	PDB/ CIF size (KB)	BCIF Size (KB)	MMTF Size (KB)	MMTF Cα-lossy (KB)	PIC (lossy precision) Coord (KB)	MMTF-Meta (KB)	Total (KB)	RMSD (Å)	Foldcomp (Lossy precision) (KB)
2ja9	163	24	13	3	9	4	14	0.030680	segfault
2jan	1101	108	98	15	61	22	84	0.047302	26
2jbp	2397	224	214	30	108	52	161	0.043140	runtime>day
2ja8	2831	299	249	42	138	59	197	0.043373	runtime>day
2ign	3579	329	321	41	147	71	219	0.068555	runtime>day
2jd8	4457	382	384	49	196	88	285	0.056100	runtime>day
2ja7	5605	534	488	74	258	110	369	0.055422	runtime>day
2fug	6386	580	566	82	283	128	412	0.060134	runtime>day
2b9v	6817	594	608	73	288	128	417	0.072982	runtime>day
2j28	8152	738	586	70	346	50	396	0.055343	runtime>day
6hif	12726	877	894	167	372	194	566	0.061988	runtime>day
3j7q	16027	1211	1027	108	475	234	710	0.058150	runtime>day
3j9m	17995	1339	1198	160	525	282	807	0.069300	runtime>day
6gaw	20825	1584	1359	184	587	324	911	0.070872	runtime>day
5t2a	22787	1628	1396	151	651	262	914	0.068460	segfault
4ug0	24906	1827	1606	177	707	364	1072	0.069050	runtime>day
4v60	24377	1509	1618	242	730	191	922	0.119788	491
4wro	35661	2336	1902	171	848	447	1296	0.085947	runtime>day
6fxc	31328	1961	1678	181	917	103	1020	0.099960	segfault
4wq1	40130	2646	2212	216	968	523	1492	0.086623	runtime>day

Actual compressed file sizes. We compare PIC, MMTF, Foldcomp, and BCIF formats. However, because PIC does not compress atomic metadata, we compressed the metadata-only with MMTF and then added that to the PNG sizes from PIC. All uses of MMTF were followed-up with standard Gzip compression on the MMTF file, as is standard, whereas BCIF is already a fully compressed file format. All sizes are in kilobytes (1000 × bytes). We also include the RMSD for PIC coordinates here. Lastly, unfortunately, most of the proteins we chose for our benchmark were discontinuous, or had other quirks, so Foldcomp 0.0.5 was unable to compress them after running for 24 h. However, Foldcomp does substantially better on both 2jan and 4v60 than all but MMTF-reduced (Cα-lossy), which not only decreases precision but also only keeps the alpha carbons

Furthermore, as noted in the respective figures, our prototype PIC implementation is not optimized for speed. It is not intended as a drop-in replacement for gzip or MMTF, but is instead meant to show that image-centric compression of protein atomic point clouds can provide significant space savings. The Python implementation takes on the order of a few minutes for a single compression/decompression, which is significantly slower than the order of seconds for gzip compression.

Other values of ε investigated include $\{1.25, 5, 10\}$. Only the results for $\varepsilon = 2.5$ are shown as this value produced the best compression ratios. As stated in "Mapping" section, higher ε increased image sizes and consequently decreased the compression ratios. Setting $\varepsilon = 1.25$ increased compression times as collisions increased due to the decreased image size and alternate mapping locations needed to be considered. Compression ratios also decreased slightly as gains in the compression ratios from decreased image sizes were overcame by the additional use of pointers and higher number of images generated by the PIC algorithm. Importantly, in this prototype study, we have given results from only a single set of parameters for all sizes of proteins for principled benchmarking. In future work, it may be preferable to set parameters dynamically for each protein and to store them, as in standard practice in many file compression formats. This will play a greater role as the structures of more complex proteins are deconstructed, stored in databases, and transmitted amongst researchers. Further, as the PIC algorithm leverages the standard and widely used PNG image compressor, the algorithm can be easily implemented on a variety of platforms and systems.

Although we ran benchmarks, the comparisons against MMTF, BCIF, and Foldcomp were not especially informative for a couple of reasons. First, PIC does not compress metadata, so we had to use MMTF for that portion to create reasonable comparisons. Second, different tools made different design choices about what to focus on: MMTF-lossy also only stores the C-alpha atoms, rather than all atoms as PIC and Foldcomp do, and Foldcomp is not designed for discontinuous chains, which were common in our benchmark data set. Still, it does seem that if only C-alpha atomic coordinates are needed, MMTF-lossy is better, and where Foldcomp works, it is also better.

Conclusion

In this paper, we have introduced PIC, an new compression algorithm that leverages positional encoding techniques and the well-developed, widely available PNG image compressor to encode and compress structural protein data in PDB and mmCIF files. The algorithm encodes two of the three dimensions of an atomic coordinate from the point cloud stored in the file to a position in the image space and stores the remaining dimension in pixels' intensity values around that location. The resulting image is then compressed with the lossless image compressor PNG. We showed PIC has a compression ratio superior of that of gzip for proteins with more than 20,000 atoms, and improves with the size of the protein being compressed, reaching up to 37.4% on the proteins we examined. Although as of September 2023, only 6.7% of the structures in the RCSB PDB are over 20,000 atoms, they represent at least 21% of the database in terms of total atom count, so PIC has fair applicability. Additionally, the advent of Cryo-EM in recent years has substantially increased the number of large structures deposited in the PDB, and this growth in the number of large structures is expected to continue. The improvement in compression is also orthogonal to the lossy storage of atomic coordinates to a precision of only a tenth of an angstrom, as we compared against gzip on files with that precision level.

More important than just providing a prototype, we demonstrate in this paper that the paradigm of image-centric compression is superior in efficacy than simply applying a standard sequential compressor to the atomic point clouds. This result is consistent with examples from point-cloud compression in LIDAR imaging, read-reordering for NGS sequence compression, and also the recent Foldcomp compressor which stores internal angles instead. Importantly, this improvement in compression ratios persists even though we store the necessary metadata to undo any atom-reorderings; thus, the only lossy portion of PIC is in coordinate positions. Still, we would recommend that the ordering information be entirely discarded, as it is for LIDAR and read re-ordering—we only kept all of that information to ensure that we performed a fair comparison in our benchmarks. Were we to discard that information, PIC's benchmark results would be even stronger.

We do note that as a prototype implementation, our runtimes and file formats are not suitable for everyday use, but our hope is that future compression algorithms will be designed with our findings in mind. Ultimately, we hope that this study points the way for future image-centric (or more generally structure-aware) compression of protein structures. Indeed, the contemporaneous Foldcomp [22] makes use of internal bond angles and torsions in protein compression, which is a different means of exploiting the 3D structure than our image-centric approach, and that shows greater promise even than PIC—as the Foldcomp software matures, we expect that it will potentially be the benchmark to beat in the future.

The PIC algorithm itself, if reimplemented in a faster language, is certainly competitive on compression ratios already, and furthermore is easy to implement because the PNG image format is already implemented on many platforms. However, we mostly envision that image-centric compression will simply form a part of other more complex compression methods. As structural protein files with increasing complexity are deconstructed, added to databases, and transmitted amongst researchers, targeted compression techniques will become ever more necessary.

Abbreviations

PDB:	Protein Data Bank
mmCIF:	Macromolecular Crystallographic Information File
LIDAR:	Light detection and ranging

Acknowledgements
We would like to thank Jim Shaw, Ziye Tao, and Alex Leighton for their thought-provoking questions that inspired parts of this work.

Author contributions
LS and YWY jointly conceived of the study. LS implemented most of the code, under YWY's oversight. Both authors were fully involved in writing the manuscript.

Funding
This work was supported by the Ontario Graduate Scholarship Program. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), (NSERC grant RGPIN-2022-03074), and the DND/NSERC Discovery Grant Supplement for 2022.

Availability of data and materials
Project name: PIC Compression and Decompression Prototype
Project home page: https://github.com/lukestaniscia/PIC
Operating system(s): Platform independent
Programming language: Python
Other requirements: Python 3.0 or higher
License: MIT License
Any restrictions to use by non-academics: No

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 1 January 2023 Accepted: 15 November 2023
Published online: 21 November 2023

References
1. Ramachandran G. Protein structure and crystallography. Science. 1963;141(3577):288–91.
2. Ilari A, Savino C. Protein structure determination by X-ray crystallography. Bioinformatics. 2008;452:63–87.

3. Rose PW, Prlić A, Altunkaya A, Bi C, Bradley AR, Christie CH, Costanzo LD, Duarte JM, Dutta S, Feng Z, et al. The RCSB protein data bank: integrative view of protein, gene and 3D structural information. Nucleic Acids Res. 2016;45(D1):D271–81.
4. Berman HM, Kleywegt GJ, Nakamura H, Markley JL. The protein data bank at 40: reflecting on the past to prepare for the future. Structure. 2012;20(3):391–6.
5. Pearson WR. Using the FASTA program to search protein and DNA sequence databases. In: Computer analysis of sequence data. Springer; 1994. p. 307–331.
6. Westbrook JD, Fitzgerald PM. The PDB format, MMCIF formats, and other data formats. Struct Bioinform. 2003;44:159–79.
7. Varadi M, Anyango S, Deshpande M, Nair S, Natassia C, Yordanova G, Yuan D, Stroe O, Wood G, Laydon A, et al. Alphafold protein structure database: massively expanding the structural coverage of protein-sequence space with high-accuracy models. Nucleic Acids Res. 2022;50(D1):439–44.
8. Fritz MH-Y, Leinonen R, Cochrane G, Birney E. Efficient storage of high throughput DNA sequencing data using reference-based compression. Genome Res. 2011;21(5):734–40.
9. Daniels NM, Gallant A, Peng J, Cowen LJ, Baym M, Berger B. Compressive genomics for protein databases. Bioinformatics. 2013;29(13):283–90.
10. Yu YW, Yorukoglu D, Peng J, Berger B. Quality score compression improves genotyping accuracy. Nat Biotechnol. 2015;33(3):240–3.
11. Hernaez M, Pavlichin D, Weissman T, Ochoa I. Genomic data compression. Annu Rev Biomed Data Sci. 2019;2:19–37.
12. Hategan A, Tabus I. Protein is compressible. In: Proceedings of the 6th nordic signal processing symposium, 2004. NORSIG 2004. IEEE; 2004. p. 192–5.
13. Pinho AJ, Pratas D. MFCompress: a compression tool for FASTA and multi-FASTA data. Bioinformatics. 2013;30(1):117–8. https://doi.org/10.1093/bioinformatics/btt594.
14. Deorowicz S, Walczyszyn J, Debudaj-Grabysz A. CoMSA: compression of protein multiple sequence alignment files. Bioinformatics. 2018;35(2):227–34. https://doi.org/10.1093/bioinformatics/bty619.
15. Kryukov K, Ueda MT, Nakagawa S, Imanishi T. Nucleotide archival format (NAF) enables efficient lossless reference-free compression of DNA sequences. Bioinformatics. 2019;35(19):3826–8. https://doi.org/10.1093/bioinformatics/btz144.
16. Cox AJ, Bauer MJ, Jakobi T, Rosone G. Large-scale compression of genomic sequence databases with the burrows-wheeler transform. Bioinformatics. 2012;28(11):1415–9.
17. Hach F, Numanagić I, Alkan C, Sahinalp SC. Scalce: boosting sequence compression algorithms using locally consistent encoding. Bioinformatics. 2012;28(23):3051–7.
18. Patro R, Kingsford C. Data-dependent bucketing improves reference-free compression of sequencing reads. Bioinformatics. 2015;31(17):2770–7.
19. Goodsell, D.S.: PDB101: learn: guide to understanding PDB data: introduction to PDB data (n.d.). https://pdb101.rcsb.org/learn/guide-to-understanding-pdb-data/introduction.
20. Bradley AR, Rose AS, Pavelka A, Valasatava Y, Duarte JM, Prlić A, Rose PW. MMTF—an efficient file format for the transmission, visualization, and analysis of macromolecular structures. PLoS Comput Biol. 2017;13(6):1005575.
21. Sehnal D, Bittrich S, Velankar S, Koča J, Svobodová R, Burley SK, Rose AS. BinaryCIF and CIFtools—lightweight, efficient and extensible macromolecular data management. PLoS Comput Biol. 2020;16(10):1008247.
22. Kim H, Mirdita M, Steinegger M. Foldcomp: a library and format for compressing and indexing large protein structure sets. Bioinformatics. 2023;39(4):btad153.
23. Valasatava Y, Bradley AR, Rose AS, Duarte JM, Prlići A, Rose PW. Towards an efficient compression of 3D coordinates of macromolecular structures. PLoS ONE. 2017;12(3):1–15. https://doi.org/10.1371/journal.pone.0174846.
24. Deutsch P, et al. Gzip file format specification version 4.3. RFC Editor. 1996.
25. Alakuijala J, Farruggia A, Ferragina P, Kliuchnikov E, Obryk R, Szabadka Z, Vandevenne L. Brotli: a general-purpose data compressor. ACM Trans Inf Syst (TOIS). 2018;37(1):1–30.
26. Houshiar H, Nüchter A. 3D point cloud compression using conventional image compression for efficient data transmission. In: 2015 XXV international conference on information, communication and automation technologies (ICAT). 2015. p. 1–8. https://doi.org/10.1109/ICAT.2015.7340499.
27. Divne C. 2IGN: crystal structure of recombinant pyranose 2-oxidase H167A mutant. 2006. https://www.rcsb.org/structure/2IGN.
28. Kato K, Zhou Y, Tanaka H, Yao M, Yamashita M, Tsukihara T. 4V60: the structure of rat liver vault at 3.5 angstrom resolution. 2014. https://www.rcsb.org/structure/4V60.