Journal of proteome research

pubs.acs.org/jpr

Article

mzMLb: A Future-Proof Raw Mass Spectrometry Data Format Based on Standards-Compliant mzML and Optimized for Speed and Storage Requirements

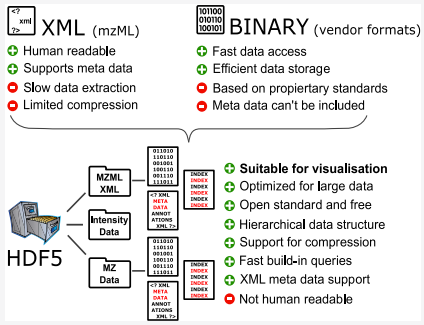
Ranjeet S. Bhamber, Andris Jankevics, Eric W. Deutsch, Andrew R. Jones, and Andrew W. Dowsey*

Cite This: J. Proteome Res. 2021, 20, 172−183

Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: With ever-increasing amounts of data produced by mass spectrometry (MS) proteomics and metabolomics, and the sheer volume of samples now analyzed, the need for a common open format possessing both file size efficiency and faster read/write speeds has become paramount to drive the next generation of data analysis pipelines. The Proteomics Standards Initiative (PSI) has established a clear and precise extensible markup language (XML) representation for data interchange, mzML, receiving substantial uptake; nevertheless, storage and file access efficiency has not been the main focus. We propose an HDF5 file format "mzMLb" that is optimized for both read/write speed and storage of the raw mass spectrometry data. We provide an extensive validation of the write speed, random read speed, and storage size, demonstrating a flexible format that with or without compression is faster than all existing approaches in virtually all cases, while with compression is comparable in size to proprietary vendor file formats. Since our approach uniquely preserves the XML encoding of the metadata, the format implicitly supports future versions of mzML and is straightforward to implement: mzMLb's design adheres to both HDF5 and NetCDF4 standard implementations, which allows it to be easily utilized by third parties due to their widespread programming language support. A reference implementation within the established ProteoWizard toolkit is provided.



KEYWORDS: *Proteomics Standards Initiative, mzML, mass spectrometry, proteomics, metabolomics, data compression, HDF5*

INTRODUCTION

Through an extensive industry-wide collaborative process, in 2008, the Proteomics Standards Initiative (PSI) established a standardized Extensible Markup Language (XML) representation for raw data interchange in mass spectrometry (MS),[1] "mzML," further building upon concepts defined in earlier formats mzData and mzXML.[2] mzML is now the pervasive format for interchange and deposition of raw mass spectrometry (MS) proteomics and metabolomics data.[3] However, to provide a detailed, flexible, consistent, and simple standard for the sharing of raw MS data, it was designed around a generic ontology for its representation at the expense of inefficient storage and file access. Two data types are contained within raw mass spectrometry (MS) data sets: (a) numeric data, i.e., mass over charge and spectral/chomatographic intensities; and (b) metadata related to instrument and experimental settings. mzML encodes these data types within a rich, schema-linked XML file, where the metadata is accurately and unambiguously annotated using the PSI-MS controlled vocabulary[4] (CV). However, one of the bottlenecks of mzML's design is that it is a text-based XML file format and all numeric spectrum data are converted into text strings using Base64 encoding.[5] Optionally, the numeric data can be zlib[6] compressed before encoding, but nevertheless, the sizes of the output files are still 4- to 18-fold higher than the original proprietary vendor format.

A number of technologies[6−8] have been developed by various laboratories to address the inherent performance/practical difficulties of utilizing the mzML format for large-volume biological sampled, high-throughput data analysis. The first approach to address the performance and file size issues of mzML was mz5.[6] At the core of mz5 is HDF5[9] (Hierarchical Data Format version 5), originally developed by the National Center for Supercomputing Applications (NCSA) for the storage and organization of large amounts of data. HDF5 is a binary format but is similar to XML in the sense that files are self-describing and allow complex data relationships and dependencies. An HDF5 file allows multiple data sets to be stored within it in a hierarchical group structure akin to folders and files on a file system. The two primary objects represented in HDF5 files are "groups" and "data sets." Groups are

Received: March 24, 2020
Published: August 31, 2020

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183

container constructs that are used to hold data sets and other groups. Data sets are multidimensional arrays of data elements of a specific type, e.g., integer, floating point, characters, strings, or a collection of these organized as compound types. Both objects support metadata in the form of attributes (key-value pairs) that can be assigned to each object; these attributes can be of any data type. Using groups, data sets, and attributes, complex structures with diverse data types can be efficiently stored and accessed. Each data set can optionally be subdivided into regular "chunks" to enable more efficient data access, as chunks can be loaded and stored in HDF5's cache implementation for subsequent repeated access. By changing the chunk size parameter, it is possible to adjust HDF5 for different applications, e.g., fast random access where file size does not matter, or larger chunks for an overall smaller compressed file size.

Compared with mzML, mz5's implementation in HDF5 yields an average file size reduction of 54% and increases linear read and write speeds 3−4-fold.[6] However, mz5 involves a complete reimplementation of mzML accomplished through a complex mapping of mzML tags and binary data to compound HDF5 data sets that mimic tables in a relational database. This structure would need to be explicitly altered to accommodate future versions of mzML. The mapping also precludes a Java implementation using the HDF5 Java application programming interface (API) as compound structures are extremely slow to access with this API. Moreover, some implementation choices are not supported by the Java API at all, specifically the variable-length nested compound structures mz5 uses to describe scan precursors.

The mzDB format[7] uses an alternative database paradigm, the lightweight SQLite relational database. mzDB's main mechanism of increasing random read performance is in organizing data in small two-dimensional blocks across multiple consecutive spectra (i.e., along both the m/z and retention time axis), enabling a quick reading of XICs. In comparison with XML formats, mzDB saves 25% of storage space when compared to mzML, and data access times are improved by 2-fold or more, depending on the data access pattern. Due to its unique data indexing and accessing scheme, three different software libraries have been created to handle MS data sets, two of which are designed to create and handle MS data-dependent acquisition (DDA), the first "pwiz-mzDB" and the second "mzDB-access". The third instance named "mzDB-Swath" is specifically designed for the data-independent acquisition (DIA) MS−SWATH technique. In addition, mzDB does not compress the text metadata, which is stored in dedicated "param_tree" fields in XML format with specific XML schema definitions (XSDs). mzDB also stores raw data sets uncompressed, but compression can be achieved through an SQLite extension; however, this extension requires a commercial license for both compression and decompression, and comparative results were not presented in the manuscript. As an alternative, a "compressed fitted" mode is proposed, which uses Gaussian Mixture Models to determine the centroid and left/right half width at half-maximum (LHW/RHW) of each peak for reconstruction. This approach has significantly better sensitivity to low-intensity and overlapping peaks than conventional centroiding, but some errors may result in this process, challenging the use of this approach as a permanent record of the raw data for sharing and archival. In summary, mzDB is an excellent format for use as a backend for processing and visualizing mass spectrometry data, as in the authors' recent Proline software package.[10] As it is not fully integrated into ProteoWizard and there is currently no other mechanism to convert mzDB files back to mzML or other formats, it is currently less suitable as a data sharing and archival format; hence, we do not compare it to mzMLb further in this paper.

The imzML data format[11] is predominantly aimed at storing very large mass spectrometry imaging data sets and does so through modest modifications to the mzML format. At the core of this approach is the splitting of XML metadata from the binary encoded data into separate files (*.imzML for the XML metadata and *.ibd for the binary data) and linking them unequivocally using a universally unique identifier (UUID). imzML also introduces new controlled vocabulary (CV) parameters designed specifically to facilitate the use of imaging data. These additional imzML CV parameters, including x/y position, scan direction/pattern, and pixel size, are stored in the *.obo file, following the OBO format 1.2 (which is a text-based format used to describe the CV terms). The approach is designed to enable easier visualization of the data using third-party software.

Unlike mz5, mzDB, and imzML, Numpress[8] is an encoding scheme for mzML and not a new or modified file format; its main focus on improving the file size is based on a novel method to compress the binary data in the mzML file before Base64 encoding (note: it does not compress the XML metadata). It accomplishes this by encoding the three common numerical data types present in mzML (mass to charge ratios—m/z, intensities, and retention times) using a variety of heuristics. The first, Numpress Pic (numPic), is intended for ion count data (e.g., from time of flight) and simply rounds the value to the nearest integer for storage in truncation form. The second, Numpress Slof (numSlof), is for general-intensity data and involves a log transformation followed by a multiplication by a scaling factor and then conversion and truncation to an integer. This ensures an approximately constant relative error; the authors demonstrate that choosing the threshold to yield a relative error of $<2 \times 10^{-4}$ did not noticeably affect downstream analysis results. The third approach, Numpress Lin (numLin), is intended specifically for m/z values and uses a fixed-point representation of the value, achieved by multiplying the data by a scaling factor and rounding to the nearest integer. Likewise, a relative error of approximately $<2 \times 10^{-9}$ was deemed not to unduly affect downstream processing. Taken together, Numpress was shown to reduce mzML file size by around 61%, or approximately 86% if the Numpress spectral output was additionally zlib compressed.

In the proposed mzMLb format, we adopt the HDF5 format[9] also used by mz5, which is well-established for high-volume data applications. However, rather than using a complex and inflexible mapping between mzML and HDF5, we propose a simple hybrid format where the numeric data are stored natively in HDF5 binary while the metadata is preserved as fully PSI standard mzML and linked to the binary in a manner similar to imzML—but stored within the same HDF5 file. Furthermore, we use only core features of HDF5, making our format compatible with NetCDF4[12] readers and writers (including their native Java library). This enables third-party bioinformatics tool developers to import and export data written in mzMLb using libraries already available on a wide variety of platforms and programming languages in a straightforward way. Taking advantage of the inbuilt HDF5 functionality, we also implement a simple predictive coding

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183

method that enables efficient lossy compression that results in file sizes comparable to Numpress but is much easier to implement. Alternatively, Numpress compressed data can be stored in mzMLb without modification. We provide a reference implementation for mzMLb fully integrated into the popular ProteoWizard toolset, available at https://github.com/biospi/pwiz. We demonstrate the advantages of mzMLb using ProteoWizard's bidirectional mzML, mz5, and Numpress implementations to provide a fully objective benchmark comparison.

METHODS

The fundamental design of mzMLb is shown in Figure 1, with the full specification given in the Supporting Information. As

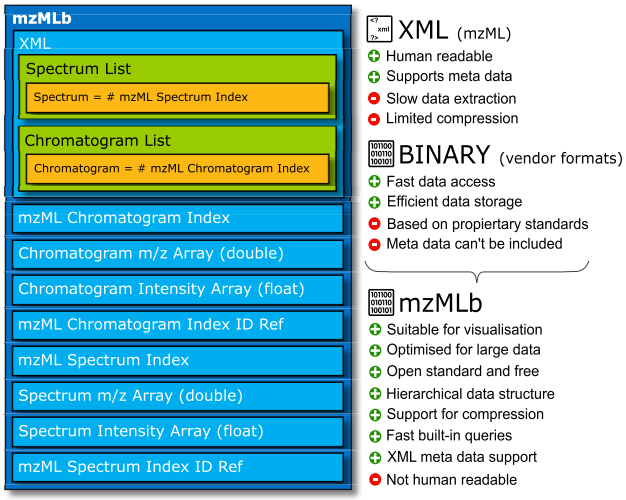


Figure 1. mzMLb internal data structure. All data is stored using standard HDF5 constructs, PSI-standard mzML is maintained, and full XML metadata is stored, along with binary data in separate HDF5 data sets. Storage of the chromatogram and spectral data (scan start times, *m*/*z*'s, and intensities) is flexible and self-described in terms of floating-point precision and layout, relying simply on the data set name and offset being specified within the <binary> tag for each chromatogram and spectrum in the mzML XML metadata.

illustrated, an mzMLb HDF5 file is composed of data sets for different data types (numerical and text-based) contained within an mzML file. In this example with our ProteoWizard implementation, the data is stored in four HDF5 data sets: chromatogram start scan times (chromatogram_MS_1000595_double); chromatogram intensities (chromatogram_MS_1000515_float); spectrum *m*/*z*'s (spectrum_MS_1000514_double); and spectrum intensities (spectrum_MS_1000515_float). These data sets are accompanied by native HDF5 version mirroring the indexed mzML schema (e.g., mzML_chromatogramIndex and mzML_chromatogramIndex_idRef). It illustrates how mzMLb utilizes the advantages of mzML (XML) and proprietary binary vendor formats by combining the positive values of both approaches while mitigating the negative traits.

The mzML XML metadata is stored inside a HDF5 character array data set "mzMLb." This is identical to the mzML format except for the following: (a) The binary data is not stored within the <binary> tags; instead, the binary tag provides attributes for the name of the HDF5 data set containing the binary data, and the offset within the HDF5 data set where the data is located. This mechanism is also used in imzML and results in valid mzML. (b) If mzML spectrum and chromatogram indices are desired (i.e., an <indexed mzML> block in mzML), they are represented instead by native HDF5 data sets "mzML_spectrumIndex" and "mzML_chromatogramIndex," which are one-dimensional arrays of 64-bit integers pointing to the start byte of each spectrum/chromatogram in the mzMLb. In addition, spectrum/chromatogram identifiers, spot ID (an identifier for the spot from which this spectrum was derived, if a matrix-assisted laser desorption ionization (MALDI) or similar run), and scan start time indices can be specified as further HDF5 data sets (see the Supporting Information).

All numerical data that is Base64 encoded in mzML (*m*/*z*'s, intensities, etc.) is instead stored in mzMLb as native HDF5 data sets, either as floating-point (32-bit or 64-bit) or as a generic byte array if Numpress encoded. As each <binary> tag in the mzMLb data set specifies the name of the data set containing the data, each mzMLb implementation has the freedom to organize the binary data as it wishes. Since offsets can be specified, data from multiple spectra can also be colocated within the same HDF5 data set as long as they are of the same data type. This enables mzMLb to harness efficiency gains from HDF5 chunk-based random access and caching schemes and also reduces the file size as data will then be compressed across spectra (which is not possible in mzML). In our ProteoWizard reference implementation of mzMLb, chromatogram and spectrum data are kept apart, but otherwise all data for specific controlled vocabulary parameters (CVParam) are stored in the same data set. For example, in the data set in Figure 1, spectrum intensity values for all spectra are stored in the data set "spectrum_MS_1000515_-float."

We also implemented a simple coding scheme that combines data truncation, a linear prediction method, and use of HDF5's inbuilt "shuffle" filter to improve the results of a subsequent compression step. The aim of this approach is to exploit the way numerical floating-point data is represented in binary natively on modern computing hardware, resulting in much better compression ratios. The method is lossy but like Numpress is designed only to add relative error at very small parts-per-million that does not affect downstream processing. Compared to Numpress, it is much easier to implement by third-party developers as the encoding and decoding can be implemented in a single line of code.

To fully appreciate its function and implementation, a basic understanding of how decimal real numbers are represented as binary floating-point numbers is required. A number in double-precision (64-bit) or single-precision (32-bit) binary floating-point[13] format consists of three parts: a sign, an exponent, and a mantissa, as represented in Figure 2. The sign bit represents a negative or positive number if set or unset, respectively (blue binary bit in Figure 2). The exponent bits represent the scale of the number and hence specify the location of the decimal point within the number (orange binary bits in Figure 2). Finally, the mantissa (green binary bits in Figure 2) expresses the fractional part of the number—the number of bits in the mantissa hence gives you the number of significant figures. Having more bits in the exponent (11 bits in double precision compared to 8 bits in single precision) allows you to represent a wider range of numbers, whereas more bits in the mantissa (e.g., there are 52 bits in double precision vs 23 bits in single precision) allows

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183

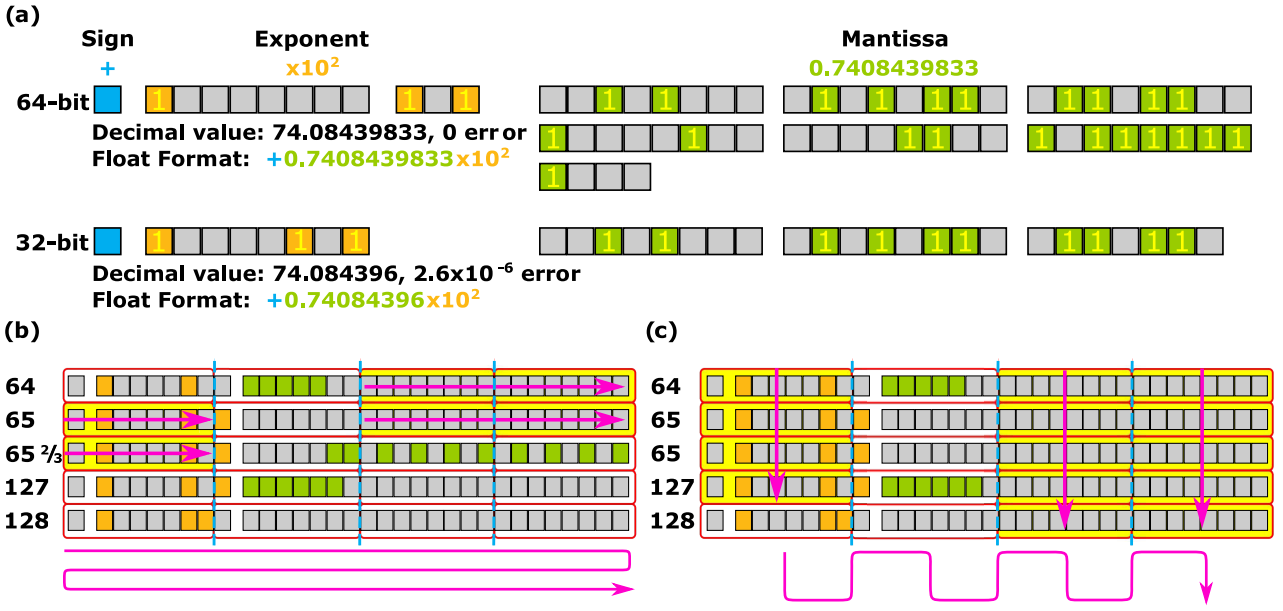


Figure 2. (a) Visual representation of IEEE 754 double-precision (64-bit) floating-point format and IEEE 754 single-precision (32-bit) floating-point format; zeros are represented by empty boxes and ones are populated. (b) Array of floating-point numbers stored conventionally; yellow bytes can be compressed. (c) Same array truncated and stored using the HDF5 shuffle filter leads to higher compressibility. The pink arrows represent the order in which data is compressed; by reshuffling the order, a higher compression ratio can be achieved.

Table 1. Effect of Changing the Number of Bits Representing the Mantissa in a Floating-Point Number and the Associated Error[a]

Mantissa Size	Mantissa Binary	Truncated Decimal	Error Parts per million
52	1001000000010101100110110010000100000011001011111110	400.08439833	0.00000
46	10010000000101011001101100100001000000011001011	400.084398329996	8.80887e-09
39	100100000001010110011011001000010000001	400.084398329724	6.90786e-07
32	10010000000101011001101100100001	400.084398329258	1.85469e-06
27	100100000001010110011011001	400.084398269653	0.00015
24	10010000000101011001101 1	400.084396362305	0.00492
23	10010000000101011001101	400.084381103516	0.04306
21	100100000001010110011	400.084350585938	0.11933
20	10010000000101011001	400.084228515625	0.42445
17	10010000000101011	400.083984375	1.03467
16	1001000000010101	400.08203125	5.91645
14	10010000000101	400.078125	15.68002
13	1001000000010	400.0625	54.73428

[a]The mantissa of a double-precision (64-bit) floating-point number (52 bits in the mantissa) and the mantissa of a single-precision (32-bit) floating-point number (23 bits in the mantissa) both are highlighted in green accordingly.

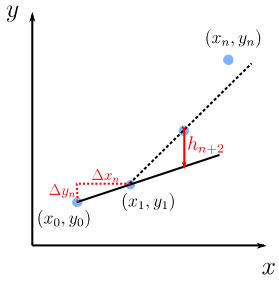
more precision. If full precision is not required, then a large number of bits are stored unnecessarily, resulting in unnecessary memory and storage use. This is the case for a significant amount of the numerical data stored in conventional mzML files.

We exploit this fact by implementing a simple lossy truncation scheme based on reducing the numbers of mantissa bits used in the floating-point format to represent m/z and intensity values by zeroing insignificant bits, with an example shown in Table 1. Here, we can see that we do not observe an appreciable drop in the parts-per-million accuracy of the decimal number until after we remove 21 bits from the mantissa, and it can be seen how zeroing more and more bits increases the error as we pass the single-precision (23 bits) mantissa level.

To translate our truncation approach into improved zlib[14] compression, it is necessary to employ HDF5 byte shuffling. In most formats, floating-point numbers are stored consecutively on disk, so zeroed mantissa bits appear in short bursts, as shown in Figure 2b. The HDF5 shuffle filter rearranges the byte ordering of the data so that it is stored transversely rather than longitudinally, as shown in Figure 2c. This leads to large numbers of consecutive zeros that can be compressed extremely well.

Moreover, further gains are possible by transforming the data so that consecutive values or sets of values are identical, as zlib is designed to compress away repeated patterns. Toward this goal, the mz5 format uses a "delta" prediction scheme that stores the difference between consecutive data points, rather than the data points themselves. This results in floating-point bit patterns (Figure 2) that are less likely to change between consecutive data points and hence are more likely to be compressed. We present an improved technique termed "mzLinear" that extends this approach to a linear extrapolation predicting each data point from the two previous data points, with only the error between the prediction and the actual value

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183



$$
\begin{aligned}
h_{n+2} &= y_{n+2} - y_n + \frac{dy_n}{dx_n}(x_n - x_{n+2}) \qquad (1)\\
n &= 0, 1, 2, \ldots, N-2\\
\Delta x_n &= x_{n+1} - x_n\\
\Delta y_n &= y_{n+1} - y_n
\end{aligned}
$$

Figure 3. mzLinear; linear predictor implemented in mzMLb, where $m/z_n = y_n$ and the index $i_n = x_n$, both $h_0 = 0$ and $h_1 = 0$ as the first value are stored in the new array and a linear equation can always be derived to intersect the first two points. However, for the rest of the data points, h_{n+2}, where $n = 0, 1, 2, \ldots, N-2$, is calculated by a linear predictor equation based on the previous two points and N is the total number of m/z values.

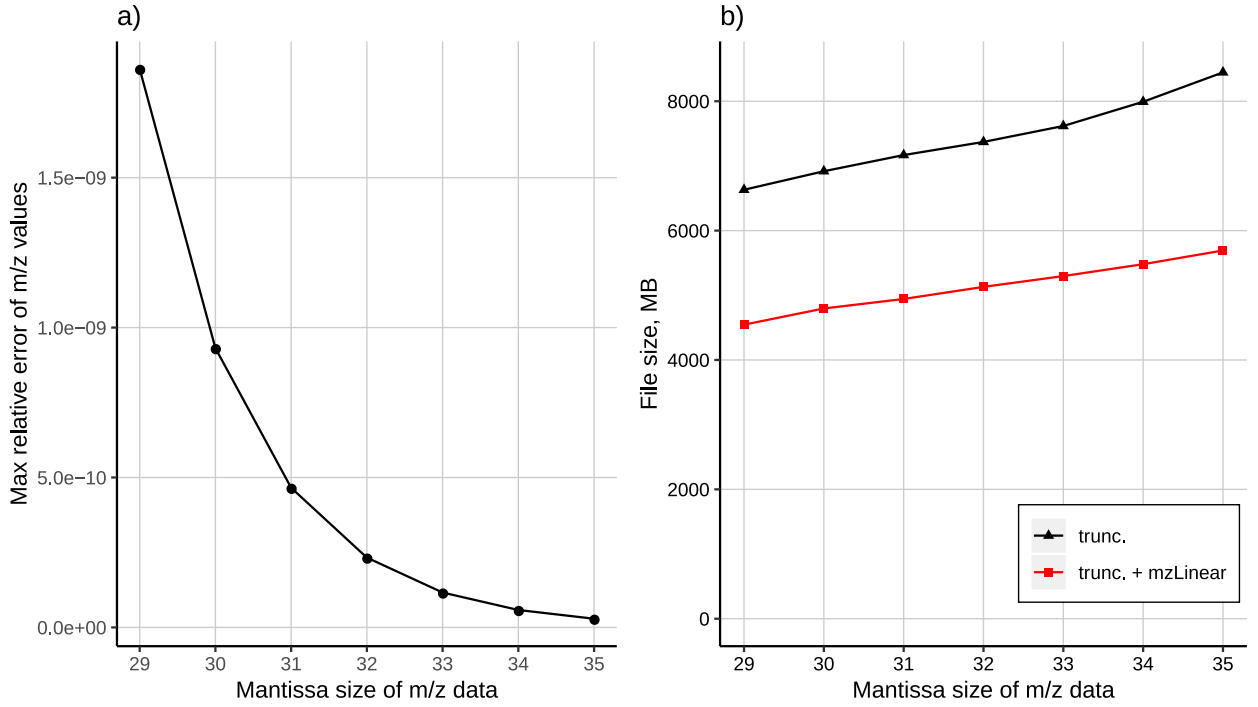


Figure 4. mzMLb; mantissa truncation of the AgilentQToF data file, with truncation error and file size for both mzLinear enabled and disabled.

stored. As there is often a quadratic relationship across m/z values (for example, since there is a quadratic relationship between time-of-flight and m/z for a standard time-of-flight analyzer), the aim of mzLinear is to result in an approximately constant prediction error across the m/z range, which will compress extremely well. In comparison, delta prediction on quadratic data would lead to prediction errors that rise linearly with m/z. The technique and equation to calculate the stored error Δh is depicted in Figure 3, with the plot showing a numerical series of m/z values exhibiting a quadratic relationship and how the prediction error Δh remains constant for each value.

To demonstrate mzMLb across a broad spectrum of proteomics and metabolomics data sets used in different laboratories, we selected a wide variety of MS techniques and instruments from varying vendors. The data sets are depicted in the Supporting Information Table S1; files 1−4 are from ref 8, data files 5−8 are from ref 15, 9 is from ref 16, 10 is from ref 17 11 is from ref 18, and finally 12 is from ref 19. We tested mzMLb across different MS types, including SWATH-DIA, DDA, and selected reaction monitoring (SRM) data, and from the major vendors including Thermo, Agilent, Sciex, and Waters. Our implementation of mzMLb has been integrated into the open-source cross-platform ProteoWizard software libraries and tools and is available from https://github.com/biospi/pwiz. Hence, the proprietary raw vendor files can be directly converted into mzMLb using the "msconvert" tool.

RESULTS

We first analyze the performance and generalizability of our truncated mzLinear coding method for m/z accuracy. Figure 4 shows the effects of the change of mantissa on the data set, "AgilentQToF"; it can be seen that increasing truncation decreases the file size while having minimal effect on accuracy. The effectiveness of the mzLinear prediction clearly improves the zlib compression rates significantly across the range of possible truncations, as it is able to exploit the quadratic nature of the m/z to time-of-flight relationship.

The procedure was performed on all data sets tested, and the mantissa values were chosen such that the error induced by truncation would be less than or comparable to Numpress' default values of $<2 \times 10^{-4}$ and $<2 \times 10^{-9}$ relative errors for the intensities and m/z values, respectively, which according to[8] are small enough so as to have no effect on the output of results on the downstream of a given workflow. The result of

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183

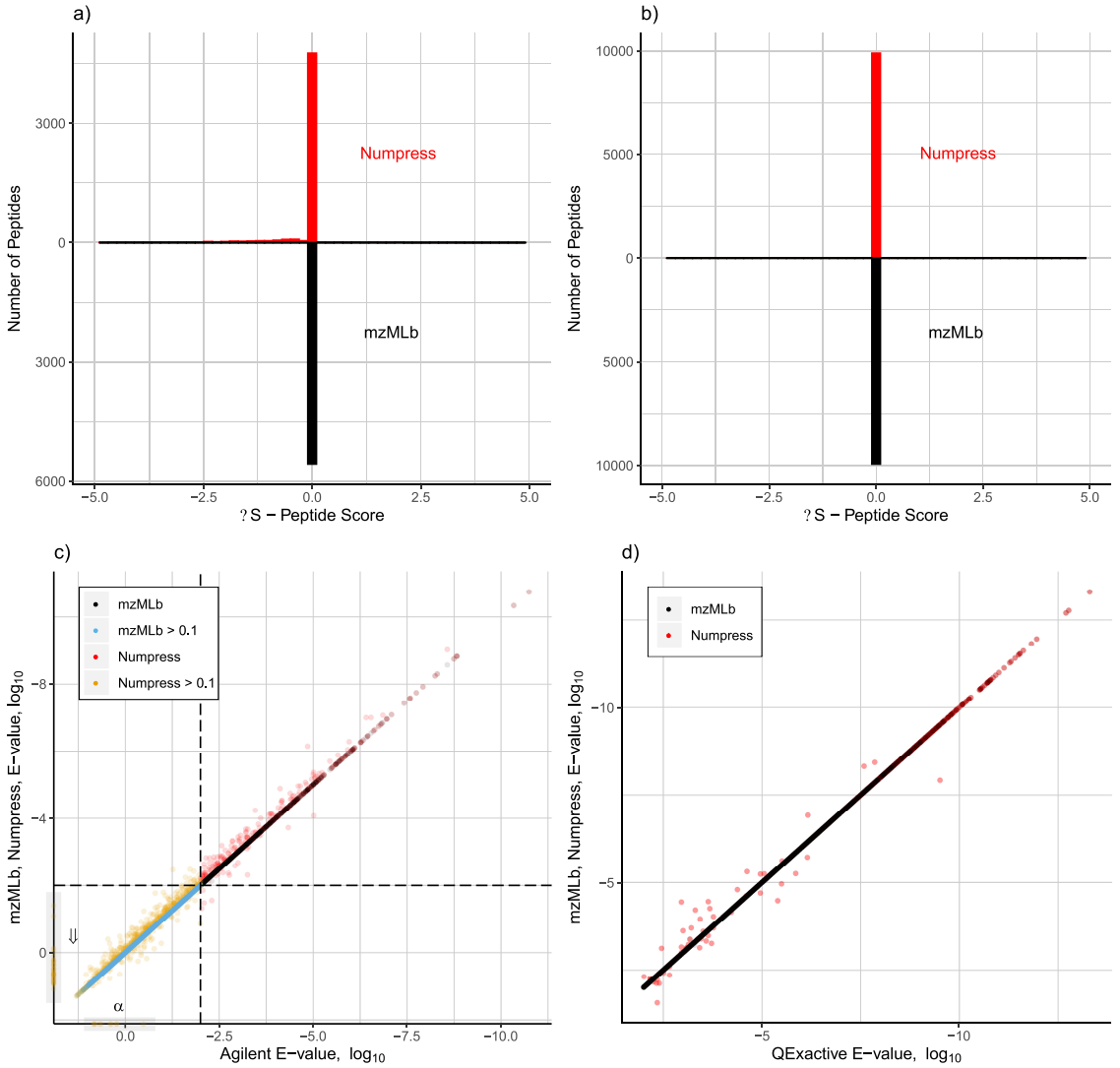


Figure 5. Mascot peptide PSM search results of the original data set against both Numpress and mzMLb compression for AgilentQToF and QExactive datafiles. The top two plots show the number of peptides found in Numpress and mzMLb against the original data with the *x*-axis representing the deviation (ΔS) of the peptide score from the original. (a) For the Agilent file, here were can clearly see a number of peptide scores deviating from the original score for the Numpress case. (b) Results for the QExactive file, where the number of peptides deviating in the Numpress case is much less when compared to the number of matching peptides. In both cases, mzMLb outperforms Numpress and has virtually no peptides deviating from the original. The bottom two plots show the relative *E*-value performance of both mzMLb and Numpress against the original data set, with (c) depicting the results for Agilent and (d) for the QExactive data file.

the relative errors can be seen in the Supporting Information, Table S2, where mzMLb produces higher compression ratios and hence smaller file sizes.

Since mzMLb's truncation relative error is always less than that of Numpress, the validation that Numpress does not noticeably affect downstream processing[8] also[8] applies to mzML. Moreover, we expand this validation by compressing the AgilentQToF and QExactive data files (shown in Table S1) and processing these files through a Mascot peptide search and protein inference workflow, the results of which can be seen in Figure 5. For the case of the AgilentQToF data, we found that Numpress was unable to produce exactly the peptide and protein lists as the original uncompressed data file. However, mzMLb was able to produce the same search and inference results for both the peptide/protein list as the original. Here, we can see that the relationship between the peptide *E*-values of both mzMLb and Numpress against the original data set; mzMLb gives an injective mapping (a straight line) vs the original peptide *E*-value, whereas the Numpress results are unable to produce the same injective relationship. The discontinuity of the Numpress results can be further illustrated by observing the peptides in the shaded regions of Figure 5c; the peptides highlighted in the enclosed box α represent the peptides that were present in both the original file and mzMLb but failed to be found in Numpress, whereas the peptides enclosed in the β region are peptides that were found in Numpress results but were not present in both the original and mzMLb results. The number of peptides deviating from the original file can be seen in Figure 5a; here, we see that Numpress did not perform quite as well as mzMLb as there are a small number of peptide score values deviating from the original data set. In Figure 5b,d, we can see the results of the same procedure on the QExactive file; here, we can see that mzMLb again produces an injective relationship with the original data set, i.e., producing the same results as the unmodified data set. Numpress in Figure 5b performs much

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183

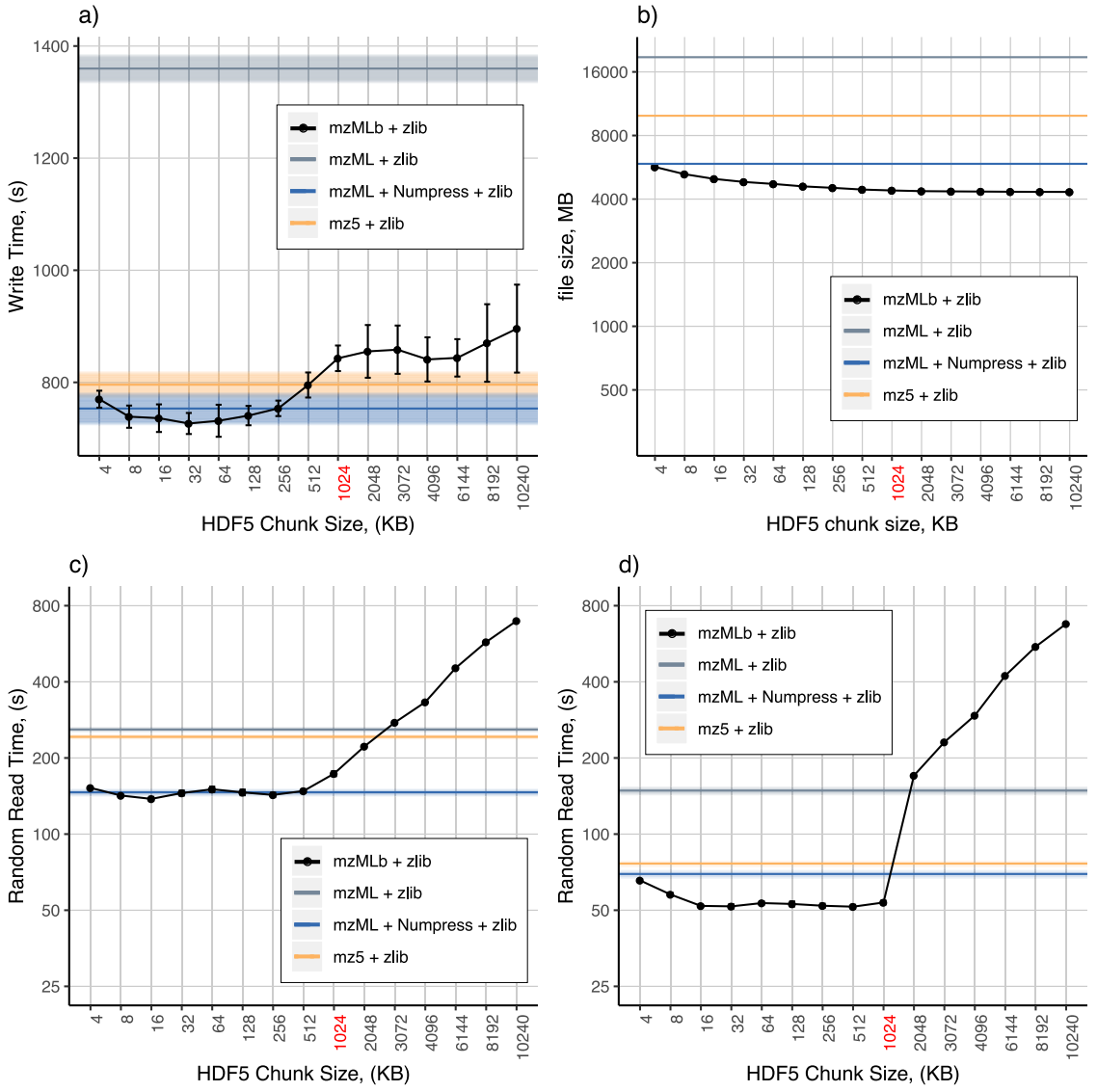


Figure 6. Chunk size optimization with mzLinear enabled; (a) mzMLb write benchmark times with varying chunk sizes, (b) file size with and without mzMLb enabled with varying sizes of HDF5 chunking, (c) random read benchmarks for singular spectrum access for full chunking size range, and (d) random read benchmarks for sequential block spectrum access for full chunking size range, with the default chunking size of 1024 highlighted in red.

better with an extremely low number of peptides showing ΔS deviation. However, Numpress is still unable to produce exactly the same results as the original in the Mascot pipeline. It does, however, perform better than the AgilentQToF case and demonstrates that the lossy compression method employed in Numpress is more susceptible to different vendor data files, whereas the mzMLb truncation scheme is more robust to data file vendor variation and able to reproduce the same results as an unmodified data file. We thus take the most conservative truncation values from Table S2 (AgilentQToF truncation values) and apply them as the mzMLb defaults.

The HDF5 binary data set chunk size can have a significant impact on access speed and file size. For the AgilentQToF file, Figure 6 compares mzML with zlib, mzML with Numpress + zlib, and mz5 with zlib, to mzMLb across a range of chunk sizes. Figure 6a demonstrates write performance on a Linux workstation; Dell T5810, Intel Xeon CPU E5-1650 v3, with 32 GB RAM and 3 TB HDD running Ubuntu Linux v18.04. To produce these results, we ran ProteoWizard msconvert 10 times converting the files from vendor format while recording the write duration. However, modern operating systems including the Linux kernel employ a sophisticated file and memory caching system; to mitigate this mechanism accelerating the multiple writes and reads of the data files being tested, we cleared the Linux memory cache after every invocation of msconvert. It can be seen for lower chunking values, mzMLb (with both mzLinear ON/OFF) outperforms the other formats, and only starts to slow for a chunk size of around 512−1024 kb. Figure 6b shows the relative compression of the files as the chunking size increases (again for both mzLinear on/off). It can be seen that at 1024 kb the benefit of increasing the chunking size for compression of data is that it quickly plateaus while the writing speed deteriorates. The file sizes of mzMLb with mzLinear perform 77% better than mzML + zlib, 25% smaller than mzMLb + Numpress + zlib, and produce a 56% increase in compression when compared to mz5 + zlib.

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183

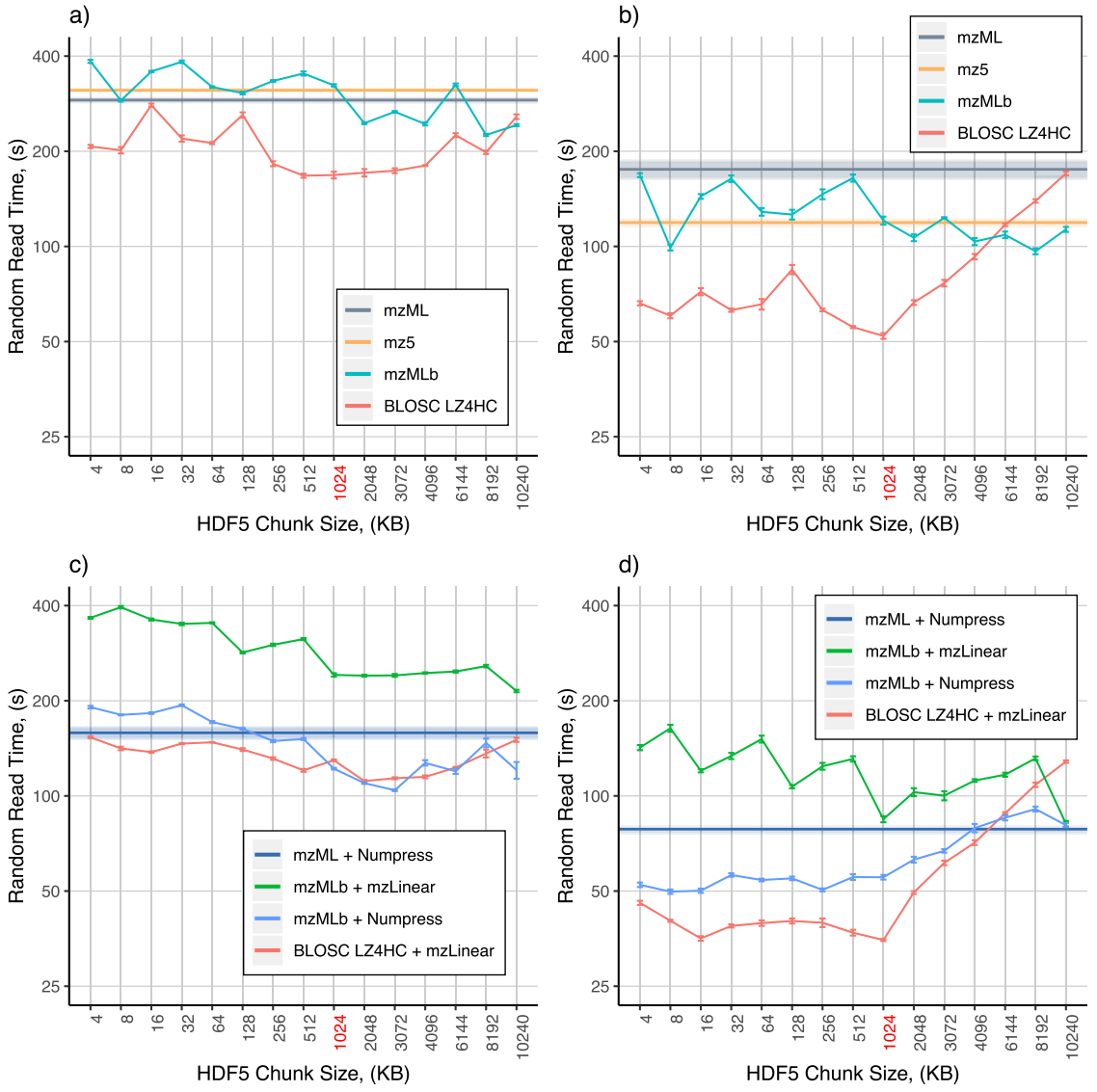


Figure 7. mzMLb; random read benchmarks for both: singular and block sequential, for uncompressed data with and without truncation and Numpress enabled; (a) lossless single-spectrum access, (b) lossless block-sequential access, (c) lossy single-spectrum access, and (d) lossy block-sequential access, with the default chunking size of 1024 highlighted in red.

To evaluate the read performance of mzMLb, we created a C ++ program readBench, which utilizes the PreoteoWizard API and its libraries to ensure the ability to read all file formats consistently under the same software implementation. This command line tool is available from https://github.com/biospi/pwiz. Here, two scenarios were considered, the first accessing a spectrum for the data set 10 000 times at random. The second involved the random reading of 10 sequential spectra selected 1000 times at random, thus giving 10 000 total spectrum accesses. These were also performed 10 times for each data point. The results are depicted in Figure 6c,d; in both cases, mzMLb outperforms the other file formats while maintaining a smaller file size. Beyond 1024 kb chunk size, the random read time drastically increases.

Subsequently, we ran the random read benchmarks again but this time without zlib compression to evaluate use cases where fast access times are paramount and file size is not important. In this test, we include mzMLb in both a lossless and lossy scenario. Here, we introduced the HDF5 BLOSC (http://blosc.org/) plugin to the validation. The aim of BLOSC is to perform modest but extremely fast decompression/compression so that the resulting read/write times are faster than using no compression at all as less data needs to be physically written to disk. It accomplishes this by: utilizing a blocking technique that reduces activity on the system memory bus; transmitting data to the CPU processor cache faster than the traditional, noncompressed, direct memory fetch approach via a memcpy operating system call; and leveraging SIMD instructions (SSE2 and AVX2 for modern CPUs) and multithreading capabilities present in multicore processors. BLOSC has a number of different optimized compression techniques including BloscLZ, LZ4, LZ4HC, Snappy, Zlib, and Zstd. Throughout these tests, we used BLOSC with LZ4HC compression, as we found it to be the most effective in terms of read and write speeds when dealing with MS data sets.

In Figure 7, we depict the results of our high-throughput results (utilizing the same data set as in Figure 6) designed to seek out the optimum solution for the fastest access to MS data, in two categories: lossless file formats (Figure 7a,b) and lossy file formats (Figure 7c,d). In both cases, we consider both

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183

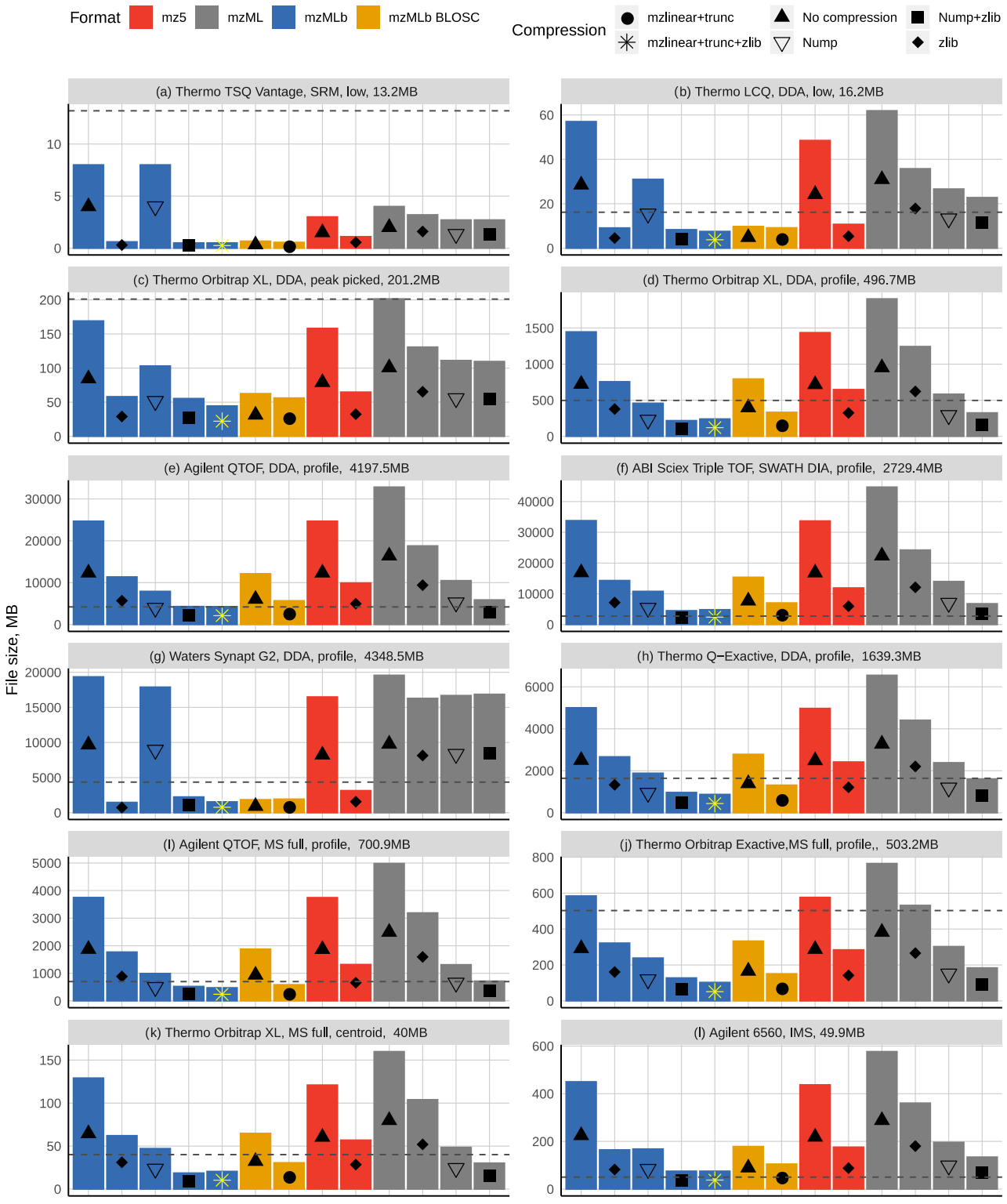


Figure 8. Summary data showing file sizes for all data sets using the three formats: mzML, mz5, and mzMLb with six different compression combinations spanning both lossless and lossy configurations. Uncompressed data files are also depicted here along with mzMLb BLOSC, demonstrating that fast access read times can be achieved without sacrificing the file size. The original vendor file sizes are represented by the horizontal dashed line.

random single-spectra access (Figure 7a,c) and random block-sequential access (Figure 7b,d). We can see that in the case of lossless compression (Figure 7a,b) mzMLb performs better than both mzML and mz5 in both single and block-sequential data access. Moreover, when we utilize mzMLb with BLOSC LZ4HC compression, we can see that it significantly outperforms both mzML and mz5 at virtually all chunking sizes and particularly performs well at around 1024 kb chunks for both single and block-sequential scans. In the case of lossy data sets (Figure 7c,d), we can see that Numpress has a

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183

significant positive impact on random read times for both single and block-sequential data access. Notably, Numpress when coupled with a chunking size in the vicinity of the optimal value performs better when contained within mzMLb rather than mzML. Nevertheless, when we utilize both mzMLb with mzLinear and BLOSC LZ4HC compression, we observe that mzMLb is significantly faster than Numpress in block-sequential data access and is comparable to Numpress within mzMLb for random single-spectra access.

In Figures 8 and S1 (including different types of instruments), we compare the file size and write performance of our new mzMLb file format against vendor raw file, mzML, mz5, and Numpress within both mzML and mzMLb. All results are the average of 10 runs. Here, we also used the optimum mzMLb chunking size of 1024 kb derived from both Figures 5 and 6, which allows mzMLb to possess both a significant compression ratio of the file size and increased performance in both reading and writing of the mass spectrometry file. Depicted in Figure 8, the colors of the markers represent the different file formats; more specifically, the red represents mz5 files, the gray the mzML files, the blue mzMLb files, and finally the orange the mzMLb files with BLOSC. The shape of the markers represents the different filters applied during the conversion process, e.g., a solid triangle represents data sets without compression, a solid diamond data sets with zlib applied, and a yellow asterisk data sets with mzLinear, truncation, zlib applied, etc. From these results, it can be seen that in all cases, the resulting mzMLb files were significantly smaller than mzML and a similar size to the vendor raw file. Moreover, from Figures 8 and S1, mzMLb can easily be tailored to different use cases (e.g., maximum compression for archiving; lower compression but faster access times for processing, both the yellow asterisk and the solid circle markers (Figure 8), representing mzLinear + trunc + zlib and Numpress + zlib, respectively) to maximize the desired performance metric.

We also demonstrate the ability of mzMLb to seamlessly integrate new mzML features without any implementation changes. The PSI is currently developing a set of recommendations for encoding data-independent acquisition and ion mobility data in mzML, which include the merging of the set of ion mobility spectra at each retention time to substantially reduce the repetition of metadata. ProteoWizard already implements this feature through msconvert switch "—combineIonMobilitySpectra." We demonstrate mzML and mzMLb conversion with and without this switch using an extremely large data set acquired using the recent Bruker diaPASEF technique;[20] the results of which are shown in Figure S2. Here, we see that even without combining ion mobility spectra, mzMLb is more effective at storing this type of data because mzMLb compresses the metadata and compresses the numerical data for multiple spectra in the same chunk. When utilizing the switch, the file size is again decreased.

CONCLUSIONS

We demonstrate that using a hybrid file format based on storing XML metadata together with native binary data within a HDF5 file, it is possible to improve the data reading/writing speed of raw MS data as well as preserve all related metadata in PSI-compliant mzML in an implicitly future-proof way. The mzMLb file format can be tailored for different applications by changing the chunk size parameter, i.e., it is possible to adjust the format for fast access where file size does not matter, e.g., visualization and processing, or a smaller compressed file size with slower reading/writing times for data archival. As a chunk can contain more than one spectrum of data, compression can occur across spectra, which is not possible in mzML.

Our ProteoWizard implementation allows mzMLb parameters to be set for the specific needs of each researcher. We have derived and validated a conservative default value for truncation that does not affect downstream analyses and show that a chunk size of 1024 kb is a good compromise for most applications, providing competitive results across a wide array of data sets. Given the wide range of use cases for mass spectrometry and the broad variety of instrumentation, an interesting future development would be the automatic optimization of mzMLb parameters for each new data set. For example, we would expect that optimal truncation depends on the mass accuracy of the instrument, while the density of the peak pattern for a spectrum would affect the optimal chunk size for Orbitrap instruments, but perhaps not for time-of-flight instruments as these tend to record background counts outside of the peak areas also.

As mzMLb utilizes HDF5, we are able to leverage transparent mechanisms for random data access, caching, partial reading or writing, and error checksums and are easily extendable through plugins to support additional filters and compression algorithms. HDF5 also allows the user to add extra information to the data file while still maintaining PSI compatibility, simply by adding extra HDF5 groups and data sets. This allows the user to store other data within the file side by side with the mzMLb data, for example, a version of the data optimized for fast visualization[21] or a blocked layout like mzDB optimized for fast extraction of XICs.

The design principles in mzMLb could be used to create a performant HDF5 implementation of PSI's in-progress mzSpecLib format for spectral libraries[22] (http://psidev.info/mzSpecLib). Existing PSI standard formats mzIdentML, mzQuantML, and mzTab for identification and quantification results could also be trivially encapsulated in HDF5, although optimum compression of numerical data would require an extended mapping as these formats do not utilize Base64 encoded data constructs.

As we use the standard features of HDF5, mzMLb is also bit-for-bit compatible with NetCDF4 (which has native Java libraries). This enables it to be easily implemented by third-party processing software, as both HDF5 and NetCDF4 are widely supported across common programming languages including Java. As of v4.5.0, NetCDF also has support to allow mzMLb files to be randomly accessed remotely over the internet (the HDF5 Group has also recently delivered their own implementation of this functionality too), opening up the potential for public repositories to provide new tools for users to efficiently query and visualize their raw data archives.

ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acs.jproteome.0c00192.

List of raw vendor MS data files used (Table S1); truncation optimization table (Table S2); file size and write times for all data formats and vendor files (Figure S1); ion mobility mzML and mzMLb compression comparison (Figure S2) (PDF)

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183

AUTHOR INFORMATION

Corresponding Author

Andrew W. Dowsey − *Department of Population Health Sciences and Bristol Veterinary School, University of Bristol, Bristol BS8 2BN, United Kingdom*; Phone: +44 (0) 117 3319193; Email: andrew.dowsey@bristol.ac.uk

Authors

Ranjeet S. Bhamber − *Department of Population Health Sciences and Bristol Veterinary School, University of Bristol, Bristol BS8 2BN, United Kingdom*; orcid.org/0000-0002-2426-4559

Andris Jankevics − *School of Biosciences and Phenome Centre Birmingham, University of Birmingham, Birmingham B15 2TT, United Kingdom*

Eric W. Deutsch − *Institute for Systems Biology, Seattle, Washington 98109, United States*; orcid.org/0000-0001-8732-0928

Andrew R. Jones − *Institute of Integrative Biology, University of Liverpool, Liverpool L69 7ZB, United Kingdom*; orcid.org/0000-0001-6118-9327

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jproteome.0c00192

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work was supported by BBSRC grants BB/M024954 and BB/R021430, MRC grant MR/N028457 to A.W.D. and A.R.J., and BBSRC grants BB/K01997X/1 and BB/R02216X/1 to A.R.J. E.W.D. also acknowledges support from National Institutes of Health grants R01GM087221, R24GM127667, U19AG023122 and from National Science Foundation grants DBI-1933311 and IOS-1922871. We thank the Proteomics Standards Initiative community for their comments and suggestions.

REFERENCES

(1) Deutsch, E. MzML: A Single, Unifying Data Format for Mass Spectrometer Output. *Proteomics* **2008**, *8*, 2776−2777.

(2) Pedrioli, P. G. A.; Eng, J. K.; Hubley, R.; Vogelzang, M.; Deutsch, E. W.; Raught, B.; Pratt, B.; Nilsson, E.; Angeletti, R. H.; Apweiler, R.; Cheung, K.; Costello, C. E.; Hermjakob, H.; Huang, S.; Julian, R. K.; Kapp, E.; McComb, M. E.; Oliver, S. G.; Omenn, G.; Paton, N. W.; Simpson, R.; Smith, R.; Taylor, C. F.; Zhu, W.; Aebersold, R. A Common Open Representation of Mass Spectrometry Data and Its Application to Proteomics Research. *Nat. Biotechnol.* **2004**, *22*, 1459−1466.

(3) Martens, L.; Chambers, M.; Sturm, M.; Kessner, D.; Levander, F.; Shofstahl, J.; Tang, W. H.; Rompp, A.; Neumann, S.; Pizarro, A. D.; Montecchi-Palazzi, L.; Tasman, N.; Coleman, M.; Reisinger, F.; Souda, P.; Hermjakob, H.; Binz, P.-A.; Deutsch, E. W. MzML - a Community Standard for Mass Spectrometry Data. *Mol. Cell. Proteomics* **2010**, *10*, No. R110.000133.

(4) Mayer, G.; Montecchi-Palazzi, L.; Ovelleiro, D.; Jones, A. R.; Binz, P.-A.; Deutsch, E. W.; Chambers, M.; Kallhardt, M.; Levander, F.; Shofstahl, J.; Orchard, S.; Vizcaíno, J. A.; Hermjakob, H.; Stephan, C.; Meyer, H. E.; Eisenacher, M. HUPO-PSI Group. The HUPO Proteomics Standards Initiative- Mass Spectrometry Controlled Vocabulary. *Database* **2013**, *2013*, No. bat009.

(5) Josefsson, S. *The Base16, Base32, and Base64 Data Encodings*; RFC 4648, 2006.

(6) Wilhelm, M.; Kirchner, M.; Steen, J. A. J.; Steen, H. Mz5: Space- and Time-Efficient Storage of Mass Spectrometry Data Sets. *Mol. Cell. Proteomics* **2012**, *11*, No. O111.011379.

(7) Bouyssié, D.; Dubois, M.; Nasso, S.; Peredo, A. G.; de Burlet-Schiltz, O.; Aebersold, R.; Monsarrat, B. MzDB: A File Format Using Multiple Indexing Strategies for the Efficient Analysis of Large LC-MS/MS and SWATH-MS Data Sets. *Mol. Cell. Proteomics* **2015**, *14*, 771−781.

(8) Teleman, J.; Dowsey, A. W.; Gonzalez-Galarza, F. F.; Perkins, S.; Pratt, B.; Rost, H.; Malmstrom, L.; Malmstrom, J.; Jones, A. R.; Deutsch, E. W.; Levander, F. Numerical Compression Schemes for Proteomics Mass Spectrometry Data. *Mol. Cell. Proteomics* **2014**, *13*, 1537−1542.

(9) Folk, M.; Heber, G.; Koziol, Q.; Pourmal, E.; Robinson, D. An Overview of the HDF5 Technology Suite and Its Applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*; ACM, 2011; pp 36−47.

(10) Bouyssié, D.; Hesse, A.-M.; Mouton-Barbosa, E.; Rompais, M.; Macron, C.; Carapito, C.; Gonzalez de Peredo, A.; Couté, Y.; Dupierris, V.; Burel, A.; Menetrey, J.-P.; Kalaitzakis, A.; Poisat, J.; Romdhani, A.; Burlet-Schiltz, O.; Cianférani, S.; Garin, J.; Bruley, C. Proline: An Efficient and User-Friendly Software Suite for Large-Scale Proteomics. *Bioinformatics* **2020**, *36*, 3148−3155.

(11) Schramm, T.; Hester, Z.; Klinkert, I.; Both, J.-P.; Heeren, R. M. A.; Brunelle, A.; Laprévote, O.; Desbenoit, N.; Robbe, M.-F.; Stoeckli, M.; Spengler, B.; Römpp, A. ImzML — A Common Data Format for the Flexible Exchange and Processing of Mass Spectrometry Imaging Data. *J. Proteomics* **2012**, *75*, 5106−5110.

(12) Rew, R.; Hartnett, E.; Caron, J. In *NetCDF-4: Software Implementing an Enhanced Data Model for the Geosciences*, 22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanograph, and Hydrology, 2006.

(13) Zuras, D.; Cowlishaw, M.; Aiken, A.; Applegate, M.; Bailey, D.; Bass, S.; Bhandarkar, D.; Bhat, M.; Bindel, D.; Boldo, S. *IEEE Standard for Floating-Point Arithmetic*; IEEE Std 754-2008, 2008; pp 1−70.

(14) Deutsch, P.; Gailly, J.-L.. *ZLIB Compressed Data Format Specification*, version 3.3. RFC1950, May, 1996.

(15) French, W. R.; Zimmerman, L. J.; Schilling, B.; Gibson, B. W.; Miller, C. A.; Townsend, R. R.; Sherrod, S. D.; Goodwin, C. R.; McLean, J. A.; Tabb, D. L. Wavelet-Based Peak Detection and a New Charge Inference Procedure for MS/MS Implemented in ProteoWizard's MsConvert. *J. Proteome Res.* **2015**, *14*, 1299−1307.

(16) Decuypere, S.; Maltha, J.; Deborggraeve, S.; Rattray, N. J. W.; Issa, G.; Bérenger, K.; Lompo, P.; Tahita, M. C.; Ruspasinghe, T.; McConville, M.; Goodacre, R.; Tinto, H.; Jacobs, J.; Carapetis, J. R. Towards Improving Point-of-Care Diagnosis of Non-Malaria Febrile Illness: A Metabolomics Approach. *PLoS Neglected Trop. Dis.* **2016**, *10*, No. e0004480.

(17) Creek, D. J.; Chokkathukalam, A.; Jankevics, A.; Burgess, K. E. V.; Breitling, R.; Barrett, M. P. Stable Isotope-Assisted Metabolomics for Network-Wide Metabolic Pathway Elucidation. *Anal. Chem.* **2012**, *84*, 8442−8447.

(18) Jankevics, A.; Merlo, M. E.; de Vries, M.; Vonk, R. J.; Takano, E.; Breitling, R. Separating the Wheat from the Chaff: A Prioritisation Pipeline for the Analysis of Metabolomics Datasets. *Metabolomics* **2012**, *8*, 29−36.

(19) Zhang, X.; Romm, M.; Zheng, X.; Zink, E. M.; Kim, Y.-M.; Burnum-Johnson, K. E.; Orton, D. J.; Apffel, A.; Ibrahim, Y. M.; Monroe, M. E.; Moore, R. J.; Smith, J. N.; Ma, J.; Renslow, R. S.; Thomas, D. G.; Blackwell, A. E.; Swinford, G.; Sausen, J.; Kurulugama, R. T.; Eno, N.; Darland, E.; Stafford, G.; Fjeldsted, J.; Metz, T. O.; Teeguarden, J. G.; Smith, R. D.; Baker, E. S. SPE-IMS-MS: An Automated Platform for Sub-Sixty Second Surveillance of Endogenous Metabolites and Xenobiotics in Biofluids. *Clin. Mass Spectrom.* **2016**, *2*, 1−10.

(20) Meier, F.; Brunner, A.-D.; Frank, M.; Ha, A.; Bludau, I.; Voytik, E.; Kaspar-Schoenefeld, S.; Lubeck, M.; Raether, O.; Aebersold, R.; Collins, B. C.; Röst, H. L.; Mann, M. Parallel Accumulation − Serial

Fragmentation Combined with Data-Independent Acquisition (Dia-PASEF): Bottom-up Proteomics with near Optimal Ion Usage. *bioRxiv* **2020**, No. 656207.

(21) Zhang, Y.; Bhamber, R.; Riba-Garcia, I.; Liao, H.; Unwin, R. D.; Dowsey, A. W. Streaming Visualization of Quantitative Mass Spectrometry Data Based on a Novel Raw Signal Decomposition Method. *Proteomics* **2015**, *15*, 1419−1427.

(22) Deutsch, E. W.; Perez-Riverol, Y.; Chalkley, R. J.; Wilhelm, M.; Tate, S.; Sachsenberg, T.; Walzer, M.; Käll, L.; Delanghe, B.; Böcker, S.; Schymanski, E. L.; Wilmes, P.; Dorfer, V.; Kuster, B.; Volders, P.-J.; Jehmlich, N.; Vissers, J. P. C.; Wolan, D. W.; Wang, A. Y.; Mendoza, L.; Shofstahl, J.; Dowsey, A. W.; Griss, J.; Salek, R. M.; Neumann, S.; Binz, P.-A.; Lam, H.; Vizcaíno, J. A.; Bandeira, N.; Röst, H. Expanding the Use of Spectral Libraries in Proteomics. *J. Proteome Res.* **2018**, *17*, 4051−4060.

https://dx.doi.org/10.1021/acs.jproteome.0c00192
J. Proteome Res. 2021, 20, 172−183