

Compressing 3D Measurement Data under Interval Uncertainty

Olga Kosheleva¹, Brian Usevitch¹, and
Edward Vidal, Jr.²

¹ Department of Electrical and Computer Engineering
University of Texas, El Paso, TX 79968, USA, olgak@cs.utep.edu

² Army Research Laboratory, White Sands Missile Range,
NM 88002-5501, USA, evidal@ar1.army.mil

Abstract. The existing image and data compression techniques try to minimize the mean square deviation between the original data $f(x, y, z)$ and the compressed-decompressed data $\tilde{f}(x, y, z)$. In many practical situations, reconstruction that only guaranteed mean square error over the data set is unacceptable.

For example, if we use the meteorological data to plan a best trajectory for a plane, then what we really want to know are the meteorological parameters such as wind, temperature, and pressure along the trajectory. If along this line, the values are not reconstructed accurately enough, the plane may crash – and the fact that on average, we get a good reconstruction, does not help.

In general, what we need is a compression that guarantees that for each (x, y) , the difference $|f(x, y, z) - \tilde{f}(x, y, z)|$ is bounded by a given value Δ – i.e., that the actual value $f(x, y, z)$ belongs to the interval

$$[\tilde{f}(x, y, z) - \Delta, \tilde{f}(x, y, z) + \Delta].$$

In this paper, we describe new efficient techniques for data compression under such interval uncertainty.

1 Formulation of the Problem

Compression is important. At present, so much data is coming from measuring instruments that it is necessary to compress this data before storing and processing. We can gain some storage space by using lossless compression, but often, this gain is not sufficient, so we must use lossy compression as well.

Successes of 2-D image compression. In the last decades, there has been a great progress in image and data compression. In particular, the JPEG2000 standard (see, e.g., [8]) uses the wavelet transform methods together with other efficient compression techniques to provide a very efficient compression of 2D images.

Within this standard, we can select different bitrates (i.e., number of bits per pixel that is required, on average, for the compressed image), and depending on the bitrate, get different degrees of compression.

When we select the highest possible bitrate, we get the lossless compressions that enables us to reconstruct the original image precisely. When we decrease the bitrate, we get a lossy compression; the smaller the bitrate, the more the compressed/decompressed image will differ from the original image.

2-D data compression. In principle, it is possible to use these compression techniques to compress 2D measurement data as well.

Compressing 3-D data: layer-by-layer approach. It is also possible to compress 3D measurement data $f(x, y, z)$ – e.g., meteorological measurements taken in different places (x, y) at different heights z .

One possibility is simply to apply the 2D JPEG2000 compression to each horizontal layer $f(x, y, z_0)$.

Compressing 3-D data – an approach that uses KLT transform: general idea. Another possibility, in accordance with Part 2 of JPEG2000 standard, is to first apply the KLT transform to each vertical line. Specifically, we:

- compute the average value $\bar{f}(z) = N^{-1} \cdot \sum_{x,y} f(x, y, z)$ of the analyzed quantity at a given height z , where N is the overall number of horizontal points (x, y) ;
- compute the covariances between different heights:

$$V(z_1, z_2) = N^{-1} \cdot \sum_{x,y} (f(x, y, z_1) - \bar{f}(z_1)) \cdot (f(x, y, z_2) - \bar{f}(z_2));$$

- find the eigenvalues λ_k and the eigenvectors $e_k(z)$ of the covariance matrix $V(z_1, z_2)$; we sort these eigenvectors into a sequence $e_1(z), e_2(z), \dots$ so that $|\lambda_1| \geq |\lambda_2| \geq \dots$;
- finally, we represent the original 3D data values $f(x, y, z)$ as a linear combination

$$f(x, y, z) = \bar{f}(z) + \sum_k a_k(x, y) \cdot e_k(z)$$

of the eigenvectors $e_k(z)$.

An approach that uses KLT transform: details. How can we implement this approach?

- The first two steps are straightforward.
- The third step – computing eigenvalues and eigenvectors – is a known computational problem for which many standard software packages provides an efficient algorithmic solution.
- So, to specify how this method can be implemented, we must describe how the last step can be implemented, i.e., how we can represent the original 3D data as a linear combination of the eigenvectors.

First, why is such a representation possible at all? It is known (see, e.g., [8]), that in general, the eigenvalues of a covariance matrix form an orthonormal basis in the space of all N_z -dimensional vectors $e = \{e(z)\} = (e(1), e(2), \dots, e(N_z))$. By definition of a basis this means, in particular, for every (x, y) , the corresponding difference vector $d_{x,y}(z) \stackrel{\text{def}}{=} f(x, y, z) - \bar{f}(z)$ can be represented as a linear combination of these eigenvalues, i.e., that for every z , we have

$$d_{x,y}(z) = f(x, y, z) - \bar{f}(z) = \sum_k a_k(x, y) \cdot e_k(z),$$

where by $a_k(x, y)$, we denoted the coefficient at $e_k(z)$ in the corresponding expansion. From this formula, we get the desired representation of $f(x, y, z)$.

How can we actually compute this representation? Since the vectors $e_k(z)$ form an orthonormal basis, we can conclude that for the expansion of the vector $d_{x,y}$, each coefficient $a_k(x, y)$ at $e_k(z)$ is a dot (scalar) product of the vector $d_{x,y}$ and the k -th vector $e_k(z)$ from the basis, i.e., that

$$a_k(x, y) = d_{x,y} \cdot e_k \stackrel{\text{def}}{=} \sum_z d_{x,y}(z) \cdot e_k(z).$$

Substituting the expression for $d_{x,y}(z)$ into this formula, we conclude that

$$a_k(x, y) = \sum_z (f(x, y, z) - \bar{f}(z)) \cdot e_k(z).$$

Comment. Actually, instead of using this explicit (thus fast-to-compute) formula, we can use even faster formulas of the so-called *fast KLT transform* (see, e.g., [8]).

KLT transform: result. As a result of the KLT approach, we represent the original 3-D data as a sequence of the horizontal *slices* $a_k(x, y)$:

- the first slice $a_1(x, y)$ corresponds to the main (1-st) eigenvalue;
- the second slice $a_2(x, y)$ corresponds to the next (2-nd) eigenvalue;
- etc.,

with the overall intensity decreasing from one slice to the next one.

Next, to each of these slices $a_k(x, y)$, we apply a 2D JPEG2000 compression with the appropriate bit rate b_k depending on the slice k .

Decompressing 3-D data: KLT-based approach. To reconstruct the data, we so the following:

- First, we apply JPEG2000 decompression to each slice; as a result, we get the values $\tilde{a}_k^{[b_k]}(x, y)$.
- Second, based on these reconstructed slices, we now reconstruct the original 3-D data data as $\tilde{f}(x, y, z) = \bar{f}(z) + \sum_k \tilde{a}_k^{[b_k]}(x, y) \cdot e_k(z)$.

This approach is tailored towards image processing – and towards Mean Square Error. The problem with this approach is that for compressing measurement data, we use image compression techniques. The main objective of image compression is to retain the quality of the image. From the viewpoint of visual image quality, the image distortion can be reasonably well described by the mean square difference MSE (a.k.a. L^2 -norm) between the original image $I(x, y)$ and the compressed-decompressed image $\tilde{I}(x, y)$. As a result, sometimes, under the L^2 -optimal compression, an image may be vastly distorted at some points (x, y) – and this is OK as long as the overall mean square error is small.

For data compression, MSE may be a bad criterion. When we compress measurement results, however, our objective is to be able to reproduce each individual measurement result with a certain guaranteed accuracy.

In such a case, reconstruction that only guaranteed mean square error over the data set is unacceptable: for example, if we use the meteorological data to plan a best trajectory for a plane, what we really want to know are the meteorological parameters such as wind, temperature, and pressure along the trajectory.

If along this line, the values are not reconstructed accurately enough, the plane may crash – and the fact that on average, we get a good reconstruction, does not help.

An appropriate criterion for data compression. What we need is a compression that guarantees the given accuracy for all pixels, i.e., that guarantees that the L^∞ -norm $\max_{x,y,z} |f(x, y, z) - \tilde{f}(x, y, z)|$ is small.

What we need is data compression under interval uncertainty. In other words, what we need is a compression that guarantees that for each (x, y) , the difference $|f(x, y, z) - \tilde{f}(x, y, z)|$ is bounded by a given value Δ – i.e., that the actual value $f(x, y, z)$ belongs to the interval $[\tilde{f}(x, y, z) - \Delta, \tilde{f}(x, y, z) + \Delta]$.

Comment. In engineering applications of interval computations, “interval uncertainty” usually means that the problem’s parameters are uncertain *parameters*, known only with interval uncertainty.

In the above data compression problem, we have a *non-parametric* problem, so the traditional engineering meaning of interval uncertainty does not apply. In our problem, by interval uncertainty, we mean guaranteed bounds on the loss of accuracy due to compression.

Need for optimal data compression under interval uncertainty. There exist several compressions that provide such a guarantee. For example, if for each slice, we use the largest possible bitrate – corresponding to lossless compression – then $\tilde{a}_k(x, y) = a_k(x, y)$ hence $\tilde{f}(x, y, z) = f(x, y, z)$ – i.e., there is no distortion at all.

What we really want is, among all possible compression schemes that guarantee the given upper bound Δ on the compression/decompression error, to find

the scheme for which the average bitrate $\bar{b} \stackrel{\text{def}}{=} (1/N_z) \cdot \sum_k b_k$ is the smallest possible.

In some cases, the bandwidth is limited, i.e., we know the largest possible average bitrate b_0 . In such cases, among all compression schemes with $\bar{b} \leq b_0$, we must find a one for which the L^∞ compression/decompression error is the smallest possible.

What we have done. In this paper, we describe new efficient (suboptimal) techniques for data compression under such interval uncertainty.

2 New Technique: Main Ideas, Description, Results

What exactly we do. Specifically, we have developed a new algorithm that uses JPEG2000 to compress 3D measurement data with guaranteed accuracy. We are following the general idea of Part 2 of JPEG2000 standard; our main contribution is designing an algorithm that selects bitrates leading to a minimization of L^∞ norm as opposed to the usual L^2 -norm.

Let us start our analysis with a 2-D case. Before we describe how to compress 3-D data, let us consider a simpler case of compressing 2-D data $f(x, y)$. In this case, for each bitrate b , we can apply the JPEG2000 compression algorithm corresponding to this bitrate value. After compressing/decompressing the 2-D data, we get the values $\tilde{f}^{[b]}(x, y)$ which are, in general, slightly different from the original values $f(x, y)$.

In the interval approach, we are interested in the L^∞ error

$$D(b) \stackrel{\text{def}}{=} \max_{x,y} \left| \tilde{f}^{[b]}(x, y) - f(x, y) \right|.$$

The larger the bitrate b , the smaller the error $D(b)$. When the bitrate is high enough – so high that we can transmit all the data without any compression – the error $D(b)$ becomes 0.

Our objective is to find the smallest value b for which the L^∞ error $D(b)$ does not exceed the given threshold Δ . For the 2-D case, we can find this optimal b_{opt} by using the following iterative *bisection* algorithm. In the beginning, we know that the desired bitrate lies between 0 and the bitrate B corresponding to lossless compression; in other words, we know that $b \in [b^-, b^+]$, where $b^- = 0$ and $b^+ = B$.

On each iteration of the bisection algorithm, we start with an interval $[b^-, b^+]$ that contains b_{opt} and produce a new half-size interval still contains b_{opt} . Specifically, we take a midpoint $b_{\text{mid}} \stackrel{\text{def}}{=} (b^- + b^+)/2$, apply the JPEG2000 compression with this bitrate, and estimate the corresponding value $D(b_{\text{mid}})$. Then:

- If $D(b_{\text{mid}}) \leq \Delta$, this means that $b_{\text{opt}} \leq b_{\text{mid}}$, so we can replace the original interval $[b^-, b^+]$ with the half-size interval $[b^-, b_{\text{mid}}]$.

- If $D(b_{\text{mid}}) > \Delta$, this means that $b_{\text{opt}} > b_{\text{mid}}$, so we can replace the original interval $[b^-, b^+]$ with the half-size interval $[b_{\text{mid}}, b^+]$.

After each iteration, the size of the interval halves. Thus, after k iterations, we can determine b_{opt} with accuracy 2^{-k} .

3-D problem is difficult. In the 3-D case, we want to find the bitrate allocation b_1, \dots, b_{N_z} that lead to the smallest average bit rate b among all the allocations that fit within the given interval, i.e., for which the L^∞ compression/decompression error does not exceed the given value Δ : $D(b_1, b_2, \dots) \leq \Delta$.

For each bitrate allocation, we can explicitly compute this error, but there are no analytic formulas that describe this dependence, so we end up having to optimize a complex function with a large number of variables b_i .

Such an optimization is known to be a very difficult task, because the computational complexity of most existing optimization algorithms grows exponentially with the number of variables. There are theoretical results showing that in general, this growth may be inevitable; to be more precise, this problem is known to be NP-hard; see, e.g., [9].

The source of our main idea: use of enclosures in interval computations. To solve our problem, we use the experience of interval computations; see, e.g., [4].

In many areas of science and engineering, we are interested in the value of a physical quantity y that is difficult (or even impossible) to measure directly. To measure such quantities, we find auxiliary easier-to-measure quantities x_1, \dots, x_n that are related to y by a known algorithm $y = g(x_1, \dots, x_n)$ – an algorithm that, in general, computes a complex functions with a large number of variables. Then, we measure x_i , and apply the algorithm f to the results $\tilde{x}_1, \dots, \tilde{x}_n$ of measuring x_i . As a result, we get an estimate $\tilde{y} = g(\tilde{x}_1, \dots, \tilde{x}_n)$ for y .

Since the measured values \tilde{x}_i are, in general, slightly different from the actual values x_i , a natural question is: what is the error of the resulting estimate?

In many practical situations, the only information that we have about the measurement errors $\Delta x_i \stackrel{\text{def}}{=} \tilde{x}_i - x_i$ of direct measurements is the upper bounds Δ_i on $|\Delta x_i|$ guaranteed by the manufacturer of the measuring instrument. In such situations, the only information that we have about the actual value x_i is that x_i lies in the interval $\mathbf{x}_i \stackrel{\text{def}}{=} [\tilde{x}_i - \Delta_i, \tilde{x}_i + \Delta_i]$. In this case, the only information that we have about y is that y belongs to the range

$$\mathbf{y} = g(\mathbf{x}_1, \dots, \mathbf{x}_n) \stackrel{\text{def}}{=} \{g(x_1, \dots, x_n) \mid x_1 \in \mathbf{x}_1 \ \& \ \dots \ \& \ x_n \in \mathbf{x}_n\}.$$

It is known that computing this range exactly is an NP-hard problem; see, e.g., [5]. Crudely speaking, NP-hard means that we cannot have an algorithm that always finished in reasonable time and that always produces the exact range.

The objective of interval computation is find *guaranteed* bounds for the actual value of y . Since we cannot find the exact range \mathbf{y} , researchers in interval computations design algorithms that provide us with an *enclosure* $\mathbf{Y} \supseteq \mathbf{y}$ for the actual range.

Our main idea: using the upper estimate (enclosure) for the optimized error function. In our case, the problem is, e.g., to find, among all bitrate allocations (b_1, b_2, \dots) with $\bar{b} \leq b_0$, the one for which the L^∞ compression/decompression error $D(b_1, b_2, \dots)$ is the smallest possible.

Since it is difficult to minimize the original function $D(b_1, \dots)$, we find easier-to-optimize upper estimate $\tilde{D}(b_1, b_2, \dots) \geq D(b_1, b_2, \dots)$ and then find the values b_i that minimize $\tilde{D}(b_1, \dots)$. As a result, we find an allocation b_i guaranteeing that $\tilde{D}(b_1, \dots) \leq \tilde{D}_{\min}$ and thus, that $D(b_1, \dots) \leq \tilde{D}_{\min}$.

Since, in general, $D(b_1, \dots) \leq \tilde{D}(b_1, \dots)$, the resulting allocation is only *sub-optimal* with respect to $D(b_1, \dots)$.

Explicit formula for the enclosure. Since we use the KLT, the difference

$$f(x, y, z) - \tilde{f}(x, y, z)$$

is equal to $\sum_k (a_k(x, y) - \tilde{a}_k^{[b_k]}(x, y)) \cdot e_k(z)$. Therefore, once we know the L^∞ -norms $D_k(b_k) \stackrel{\text{def}}{=} \max_{x, y} |a_k(x, y) - \tilde{a}_k^{[b_k]}(x, y)|$ of the compression/decompression errors of each slice, we can conclude that $|a_k(x, y) - \tilde{a}_k^{[b_k]}(x, y)| \leq D_k(b_k)$, hence, that $|(a_k(x, y) - \tilde{a}_k^{[b_k]}(x, y)) \cdot e_k(z)| \leq D_k(b_k) \cdot E_k$, where $E_k \stackrel{\text{def}}{=} \max_z |e_k(z)|$. Thus, the desired L^∞ error is bounded by $\tilde{D}(b_1, \dots) \stackrel{\text{def}}{=} \sum_k D_k(b_k) \cdot E_k$,

Resulting algorithm: derivation. In accordance with the above idea, to get the (suboptimal) bitrate allocation b_i , we must minimize the function $\tilde{D}(b_1, \dots) = \sum_k D_k(b_k) \cdot E_k$ under the condition that the $\sum_k b_k = N_z \cdot b_0$. By using the Kuhn-Tucker approach, we can reduce this problem to the unconstrained problem – of finding stationary points of the function $\sum_k D_k(b_k) \cdot E_k + \lambda \cdot \sum_k b_k - N_z \cdot b_0$. By definition of a stationary point, derivatives w.r.t. all the variables b_i should be 0s, so we end up with the equation $-D'_k(b_k) = \lambda/E_k$, where the parameters λ should be selected based on the value b_0 .

It can be easily shown that the other problem – of minimizing the average bitrate under the constraint that the compression/decompression error does not exceed Δ – leads to the same equation.

As we have mentioned, the function $D_k(b)$ decreases when b increases, so $D'_k(b) < 0$, with $D'_k(b) \rightarrow 0$ as b grows. It is therefore reasonable to represent the desired equation as $|D'_k(b_k)| = \lambda/E_k$.

What are the bounds on λ ? The larger b_k , the smaller λ .

From the above formula, we conclude that $\lambda = |D'_k(b_k)| \cdot E_k$, hence $\lambda \leq A_k \stackrel{\text{def}}{=} (\max_b |D'_k(b)|) \cdot E_k$, so $\lambda \leq \Lambda \stackrel{\text{def}}{=} \min_k A_k$.

Algorithm: description. Once we know, for each slice k , the dependence $D_k(b)$ of the corresponding L^∞ -error on the bitrate b , we can find the desired (suboptimal) values b_k as follows.

At first, we compute the above-described values A_k and A . We know that $\lambda \leq [\lambda^-, \lambda^+] := [0, A]$. We use bisection to sequentially halve the interval containing λ and eventually, find the optimal value λ .

Once we know an interval $[\lambda^-, \lambda^+]$ that contains λ , we pick its midpoint λ_{mid} , and then use bisection to find, for each k , the value b_k for which $|D'_k(b_k)| = \lambda_{\text{mid}}/E_k$. Based on these b_k , we compute the average bitrate \bar{b} . If $\bar{b} > b_0$, this means that we have chosen too small λ_{mid} , so we replace the original λ -interval with a half-size interval $[\lambda_{\text{mid}}, \lambda^+]$. Similarly, if $\bar{b} < b_0$, we replace the original λ -interval with a half-size interval $[\lambda^-, \lambda_{\text{mid}}]$.

After k iterations, we get λ with the accuracy 2^{-k} , so a few iterations are sufficient. Once we are done, the values b_k corresponding to the final λ_{mid} are returned as the desired bitrates.

The only remaining question is how to determine the dependence $D_k(b)$. In principle, we can try, for each layer k , all possible bitrates b , and thus get an empirical dependence $D_k(b)$.

We have shown, that this dependence can be described by the following analytical formula: $A_1 \cdot (b - b_0)^\alpha$ for all the values b until a certain threshold b_0 , and $A_2 \cdot 2^{-b}$ for $b \geq b_0$. This model is a slight modification of a model from [6]. So, instead of trying all possible values b , it is sufficient to try a few to determine the values of the parameters A_i , b_0 , and α corresponding to the given layer.

Results. To test our algorithm, we applied it to 3-D meteorological data: temperature T, pressure P, the components U, V, and W of the wind speed vector, and the waver vapor missing ratio WV.

For meteorological data, the resulting compression indeed leads to a much smaller L^∞ error bound Δ_{new} than the L^∞ error bound Δ_{MSE} corresponding to the bitrate allocation that optimizes the MSE error. The ratio $\Delta_{\text{new}}/\Delta_{\text{MSE}}$ decreases from 0.7 for $b_0 = 0.1$ to 0.5 for $b_0 = 0.5$ to ≤ 0.1 for $b_0 \geq 1$.

Acknowledgments. This work was partially supported by NSF Grant CDA-9522207, the ARO grants DAA H04-95-1-0494 and DAA D19-99-1-0012, by the Texas Instruments Grant, and by NASA under cooperative agreement NCC5-209. This work was partly performed during O. Kosheleva's visit to Brazil; this visit was sponsored by the Brazilian funding agency CTINFO/CNPq.

The authors are thankful to Sergio Cabrera for his guidance and help, and to the anonymous referees for valuable suggestions.

References

1. Cabrera, S.D.: Three-Dimensional Compression of Mesoscale Meteorological Data based on JPEG2000, Battlespace Digitization and Network-Centric Warfare II, Proc. SPIE **4741** (2002) 239–250.
2. Kosheleva, O.M.: Task-Specific Metrics and Optimized Rate Allocation Applied to Part 2 of JPEG2000 and 3-D Meteorological Data, Ph.D. Dissertation, University of Texas at El Paso, 2003.

3. Kosheleva, O.M., Usevitch, B., Cabrera, S., Vidal, E.Jr.: MSE Optimal Bit Allocation in the Application of JPEG2000 Part 2 to Meteorological Data, Proceedings of the 2004 IEEE Data Compression Conference DCC'2004, Snowbird, Utah, March 23–25, 2004, p. 546.
4. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis, Springer, London, 2001.
5. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: Computational Complexity and Feasibility of Data Processing and Interval Computations, Kluwer, Dordrecht, 1997.
6. Mallat, S., Falzon, F.: Analysis of low bit rate image transform coding, IEEE Trans. Signal Proc. **46** (1998) 1027–1042.
7. Moore, R.E.: Methods and Applications of Interval Analysis. SIAM, Philadelphia, 1979.
8. Taubman, D.S., Marcellin, M.W.: JPEG2000 Image Compression Fundamentals, Standards and Practice, Kluwer, Boston, Dordrecht, London, 2002.
9. Vavasis, S.A.: Nonlinear Optimization: Complexity Issues, Oxford University Press, New York, 1991.