Proceedings of Building Simulation 2011:
12th Conference of International Building Performance Simulation Association, Sydney, 14-16 November.

REAL-TIME COMPRESSION OF TIME SERIES BUILDING PERFORMANCE DATA

Rhys Goldstein, Michael Glueck, and Azam Khan
Autodesk Research, 210 King St. East, Toronto, ON, Canada

ABSTRACT

If building performance simulations are to fully benefit from increasing quantities of sensor data, subsets of large datasets must be efficiently extracted at varying levels of detail. A key issue with time series data is that relevant time scales vary by orders of magnitude depending on the desired analysis. To ensure that a subset of a time series is available when needed at an appropriate resolution, lossy compression can be applied in real time as data is acquired. We propose an algorithm that compresses a piecewise constant time series by merging segments within a sliding time window. This procedure tends to preserve prominent edges and spikes. While building control system dashboards and simulation tools often average data over fixed time periods (e.g. hourly averages), the proposed method achieves lower errors for the same compression ratio and provides better support for signal processing, data visualization, and simulation.

INTRODUCTION

The accuracy of a simulation-based building energy analysis depends largely on whether the modeler has convenient access to an adequate amount of sufficiently detailed input data acquired under relevant conditions. The critical importance of simulation input data is leading to increasingly large datasets.

Weather data provides a good example of why large datasets are necessary. First, weather data includes several properties, including air temperature, humidity, direct and diffuse solar irradiance, direct and diffuse solar illuminance, wind velocity and direction, among others. Barnaby and Crawley (2011) indicate that this data must be collected over many years, for numerous cities, and possibly multiple locations within each city due to urban heat island effects. At the same time they call for finer resolutions than the hourly averages typically used for simulation.

An increasing amount of data is being collected and recorded within buildings as well. The recently constructed Y2E2 Building at Stanford University features 2370 HVAC system measurement points, each sampled every minute (Kunz et al., 2009). Detailed sensor networks like these can be used to detect problems in building control systems (Eisenhower et al., 2010) and to improve the accuracy of building energy models (Raftery et al., 2009).

For building performance simulations to fully benefit from increasing quantities of sensor data, advanced data management techniques may be needed to efficiently extract subsets of large datasets. In the case of time series data, relevant time scales vary by orders of magnitude depending on the desired analysis. Sub-minute data may be needed, for example, to investigate brief power quality disturbances. To study annual trends in energy use, however, a low-resolution version of the same time series would likely suffice. Our interest in data compression is not aimed at reducing overall data storage requirements, as we believe raw data should be preserved whenever possible. Instead, we regard compression as a means of supporting the efficient extraction, processing, and exploration of time series data. The idea is to apply a lossy compression algorithm repeatedly to precompute and store copies of every time series at different resolutions. If an original time series is unnecessarily detailed for a particular analysis, a copy at more appropriate resolution will then be readily available.

A simple method for reducing the resolution of a time series is to average the data within consecutive time periods of equal width. But while the resulting fixed-width segments are common in building control system dashboards and simulation tools, they are poor at preserving prominent edges and spikes seen frequently in, for instance, light or electrical data.

We present an algorithm that compresses piecewise constant time series data by maintaining a limited number of segments within a sliding time window of fixed width. If the limit on the number of segments is exceeded, then the most similar pair of adjacent segments is merged. We show how this selective merging procedure preserves prominent edges and spikes, and introduces smaller compression errors for the same compression ratios than the common fixed-width approach. The algorithm can be applied in real time to sensor data, and may also be useful for the output of detailed, long-term simulations.

The compression algorithm is the main focus and contribution of this paper. However, we also hope to inspire other applications of time series represented using constant-valued, variable-width segments with error values. We describe how this way of representing building performance data improves support for signal processing, data visualization, and simulation.

BACKGROUND

Time Series Representation

Figure 1 shows two common techniques for representing a time series, both of which involve sets of consecutive linear segments. The *piecewise linear* representation constrains the endpoints of each segment to coincide with those of neighboring segments (Figure 1a). The *piecewise constant* representation constrains the slope of each segment to zero, but allows discontinuities between adjacent segments (Figure 1b).

(a) Piecewise linear (b) Piecewise constant

Figure 1: Time series representations

This paper focuses on the piecewise constant representation, which we regard as the more flexible of the two. Although the piecewise linear approach seems appropriate for physical properties like temperature that tend to change gradually over time, its tapered edges can be misleading if used to represent dramatic transitions and fluctuations commonly found in light and electrical power data. With enough segments a piecewise constant time series can approximate a continuous quantity, and discrete quantities like numbers of occupants are piecewise constant by nature.

In some cases time series are represented such that all segments must have the same width, while in other cases the segment widths may differ. For example, some sensors produce fixed-width segments by recording values at regular time intervals. Other sensors record a new value only when it differs from the previous by a certain minimum amount. A light sensor of the latter type in an office environment, for example, may record numerous segments during the day when indoor luminance patterns change over time. At night, when lighting conditions tend to remain constant, very few segments would be recorded.

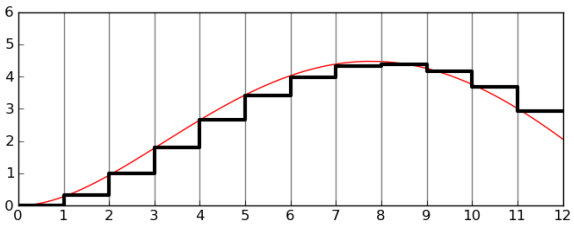
Time series output by simulations can also have fixed-width or variable-width segments. Consider the simple model below, in which $T(t)$ and $\dot{T}(t)$ represent temperature and its rate of change at time t.

$$\begin{cases} \dot{T}(t) = 0.5 \cdot \left(5 \cdot \sin\left(\frac{\pi \cdot t}{12}\right) - T(t)\right) \\ T(0) = 0 \end{cases}$$

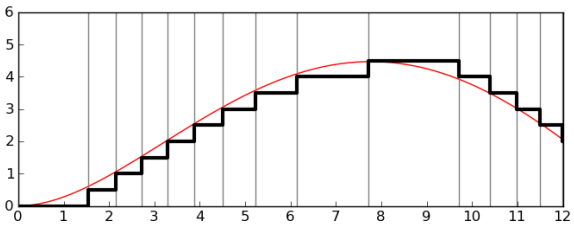
The temperature is approximated by repeatedly evaluating the following, with $\dot{T}_{approx}$ being some suitable approximation for $\dot{T}$.

$$T(t + \Delta t) = T(t) + \dot{T}_{approx} \cdot \Delta t$$

The time step Δt is arbitrary. It is common practice to use the same time step throughout a simulation, as done to produce the time series with fixed-width segments shown in Figure 2a. Here $\dot{T}_{approx}$ was obtained with the Modified Euler method, which is described in a similar context in dos Santos and Mendes (2004). The approximation converges on the smooth red line as the time step is reduced.



(a) Fixed-width segments



(b) Variable-width segments

Figure 2: Time series produced by simulation

If the state of a model changes at fixed time steps, one is said to be using *discrete time simulation*. Another option is to vary the time steps: to shorten them for improved efficiency when data is changing rapidly, while lengthening them for computational efficiency when data is changing slowly. Figure 2b shows another time series produced by a Modified Euler temperature simulation. In this case the segments vary in width, as the time step was repeatedly calculated according to the following formula. Here a, b, and c are arbitrary constants.

$$\Delta t = max\left(a, min\left(b, \left|\frac{c}{\dot{T}_{approx}}\right|\right)\right)$$

Variable time steps are a key feature of *discrete event simulation*, an approach that is often used to integrate models with different time advancement schemes.

Time Series Compression

Given a time series y, the task is to produce a new time series $\hat{y}$ while striving to minimize both the memory required to store $\hat{y}$ and some measure δ of the average compression error. We measure this error over the time period $t_a \leq t < t_b$ using (1).

$$\delta = \sqrt{\int_{t_a}^{t_b} (y(t) - \hat{y}(t))^2 dt} \qquad (1)$$

Relevant work on time series compression can be found in the field of data mining. Morbitzer (2003)

and others have investigated applications of data mining to building performance simulation, though in this community there has been little focus on general time series compression techniques. Our impression is that, if a time series of building performance data contains too many segments for a particular analysis, it is typically averaged within consecutive time periods of equal width. This data reduction technique, which we refer to as *Fixed-Width Compression*, is expressed formally in (2).

$$\hat{y}(t) = \frac{1}{\Delta t} \cdot \int_{i \cdot \Delta t}^{(i+1) \cdot \Delta t} y(\tau) d\tau \qquad \text{where} \quad i \cdot \Delta t \leq t < (i+1) \cdot \Delta t,\ i \in \mathbb{N} \tag{2}$$

In the data mining community there are many compression methods developed for various time series representations. Our interest is in relatively simple algorithms for which y and $\hat{y}$ are both piecewise constant with variable-width segments. Lazaridis and Mehrotra (2003) present one such technique, called *Poor Man's Compression* (PMC), which constrains the output time series to be within an arbitrary tolerance h of the original. The algorithm repeatedly merges the two most recently acquired segments so long as $|y(t) - \hat{y}(t)| \leq h$ remains true for all t. Choosing an appropriate value for h may be difficult in the case of building performance data due to the wide range of possible sensor types and makes.

Keogh et al. (2001a) describe a method that involves "converting the problem into a wavelet compression problem, for which there are well known optimal solutions, then converting the solution back [to a piecewise constant representation]." But because it operates on only recently acquired segments, PMC seems easier to apply in real time as data is acquired.

The *Sliding Window And Bottom-up* method (SWAB) from Keogh et al. (2001b) is similar to PMC in that it operates on recently acquired segments. Instead of merging only the two most recent segments, the algorithm searches all pairs of adjacent segments acquired within a sliding time window. If any pairs can be merged without violating an arbitrary maximum error level, the pair associated with the least additional error is merged. The selective merging procedure is compelling and can be applied in real time. However, SWAB operates on piecewise linear time series.

PROPOSED ALGORITHM

Overview

Our method involves the application of a *compression buffer* that contains a limited number of segments and spans a limited length of time. The algorithm is similar to SWAB in that recently acquired segments enter a buffer where they are selectively merged with adjacent samples. Unlike SWAB, both y and $\hat{y}$ are piecewise constant. The input time series may have fixed-width or variable-width segments, but in either case the output segments will generally vary in width.

The algorithm requires two parameters: the buffer size m and the target resolution Δt_{res}. If all input segments are considerably shorter than Δt_{res}, then the output segments will have an average width of approximately Δt_{res}. The output resolution is constrained such that there will never be more output segments than input segments in any time window, and there will never be more than $m+1$ output segments completely contained in a time window of width $m \cdot \Delta t_{res}$.

Unlike many preexisting compression algorithms, ours does not constrain the quality of the output time series. Recall that SWAB limits the compression error associated with each output segment, while PMC restricts the distance between all corresponding input and output segments. These *quality guarantees* conform with the vision that, given an original time series, a single compressed time series is produced and used for all subsequent operations. Our vision is different. We intend our compression algorithm to be used multiple times with the same time series but different values of Δt_{res}. The result would be several archived copies of the original time series, each at a different resolution. When data is extracted for an analysis, one selects the resolution that best balances the need for precision with technological limitations related to processing power or bandwidth.

For our data management strategy, it is more practical for the compression process to restrict resolution than error. However, one must still be able to determine whether an analysis was compromised by excessive data compression. We therefore formulate the algorithm such that a compression error is stored with each segment of the output time series.

Segment Merging Calculations

We store a time series as a sequence of datapoints $[t_i, y_i, \delta_i]$, where t_i is the start time of the i^{th} segment, y_i is its value, and δ_i is its error value. For the i^{th} segment to be a *complete segment*, the $(i+1)^{\text{th}}$ datapoint must also be available to provide the end time t_{i+1}. Note that a time series can be expressed as a function: $y(t) = y_i$, with i chosen such that $t_i \leq t < t_{i+1}$.

As mentioned, our algorithm compresses a time series by repeatedly merging adjacent segments. Merging the i^{th} and $(i+1)^{\text{th}}$ segments means replacing the i^{th} and $(i+1)^{\text{th}}$ datapoints with a new one, $[\hat{t}_i, \hat{y}_i, \hat{\delta}_i]$. This merged datapoint is given by (3) below, where $\Delta t_i = t_{i+1} - t_i$ and ${\rho_i}^2 = {y_i}^2 + {\delta_i}^2$.

$$\begin{aligned} \hat{t}_i &= t_i \\ \hat{y}_i &= \frac{\Delta t_i \cdot y_i + \Delta t_{i+1} \cdot y_{i+1}}{\Delta t_i + \Delta t_{i+1}} \\ \hat{\delta}_i &= \sqrt{\frac{\Delta t_i \cdot {\rho_i}^2 + \Delta t_{i+1} \cdot {\rho_{i+1}}^2}{\Delta t_i + \Delta t_{i+1}} - \hat{y}_i^2} \end{aligned} \tag{3}$$

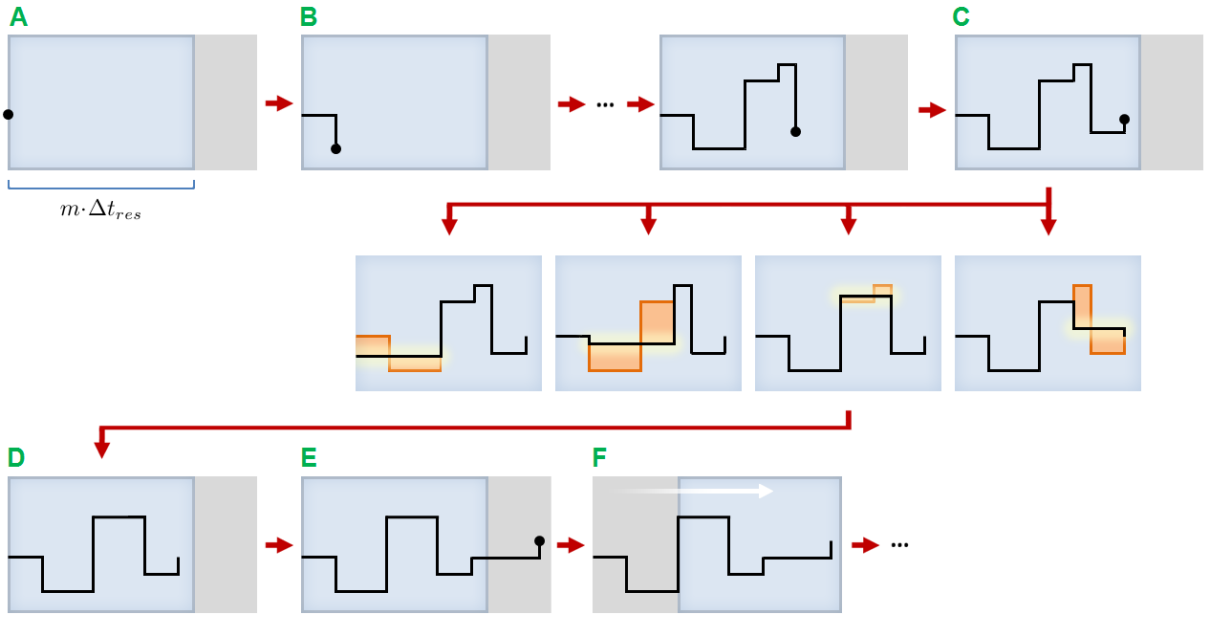


Figure 3: Example of real time compression. Segments shown on a blue background are in the compression buffer. Transition A – B shows the arrival of a datapoints within the buffer's time window, C – D illustrates the selective merging of segments in a full buffer, and E – F shows the arrival of a datapoint beyond the end of the time window.

The compression procedure involves applying (3) to multiple pairs of segments before deciding which pair to merge. The chosen pair is the one that gives the lowest cost C, which depends on both the error values and the widths of the segments.

$$C = (\Delta t_i + \Delta t_{i+1}) \cdot \hat{\delta}_i - (\Delta t_i \cdot \delta_i + \Delta t_{i+1} \cdot \delta_{i+1})$$

One may reduce memory requirements by excluding δ_i from all datapoints in an original time series, since there we assume $\delta_i = 0$. Whenever two segments with different values are merged, however, the resulting error must be recorded as it influences subsequent merging operations. A pair of adjacent segments with large error values is more likely to be merged than an otherwise congruent pair with smaller error values. It might be beneficial to assign measurement errors or simulation-estimated uncertainty values to δ_i in an original time series, but we leave that idea for future research.

Compression Procedure

Starting with an empty compression buffer, newly acquired datapoints are added in real time while observing two constraints. The first constraint is that the number of complete segments in the compression buffer cannot exceed m. If a new datapoint completes the $(m+1)^{\text{th}}$ segment within the buffer, then two segments must be merged to bring the total back down to m. The second constraint is that the combined duration of all segments in the buffer cannot exceed $m \cdot \Delta t_{res}$. If a new datapoint breaks this condition, one or more of the oldest segments are released from the buffer and become part of the output time series.

Figure 3 illustrates the procedure with a size-4 compression buffer. The first datapoint is added to the buffer at stage A. At stage B the second datapoint arrives and completes the first segment. The process continues until stage C, when the buffer gains a fifth segment completely contained in the $m \cdot \Delta t_{res}$ time window. Because $m = 4$, a pair must now be merged. As shown, there are four pairs of adjacent segments to consider. In this case the third pair yields the smallest value of C, and is therefore selected as the least costly to merge. At stage D this pair has been merged, and only four segments remain in the buffer.

When the next datapoint arrives at stage E, it completes a segment that extends beyond the $m \cdot \Delta t_{res}$ time window. In this case, in order to keep the buffer's combined segment width under $m \cdot \Delta t_{res}$, the two oldest segments must be released. As shown in stage F, the buffer slides forward to encompass the new datapoint. The two released segments have become part of the output time series, and may no longer be altered.

There are two basic ways to use the algorithm to produce multiple copies of a time series at different resolutions. The first option is to apply multiple compression buffers directly to the original time series with different values of Δt_{res}. The second option is to apply the algorithm once to the original time series, then repeatedly to its own output with successively coarser target resolutions (i.e. larger values of Δt_{res}). Both strategies seem reasonable. The second option requires less computation but increases latency, as a segment can enter a compression buffer only after it exits the preceding one. We used the first option for all results shown in this paper.

Results

Qualitative Observations

Raw sensor data was acquired with a maximum sample rate of 2 Hz, where each datapoint was recorded only if its value differed by a certain minimum amount from that of the preceding datapoint. This produced piecewise constant time series data with variable-width segments.

In Figure 4, roughly two hours of light sensor data is shown in its original form and after applying compression buffers with two different target resolutions. Clearly information is lost as the resolution decreases. Note how the compressed segments contract in places to better represent prominent features.

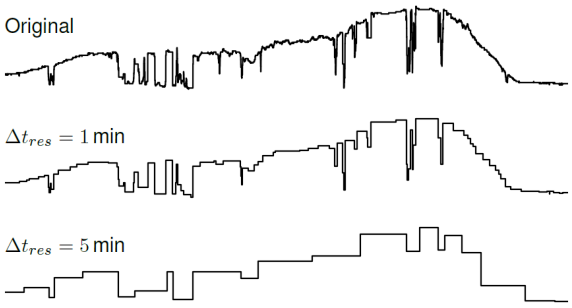


Figure 4: Light sensor data compressed with a size-12 buffer at 1- and 5-minute target resolutions

A section of the same dataset is shown expanded in Figure 5. Four different compression schemes are applied to the time series, but the target resolution is the same in each case. The first compressed time series is produced with Fixed-Width Compression (FWC), the usual means of reducing building performance data. The remaining time series are produced with size-m compression buffers, abbreviated CB-m.

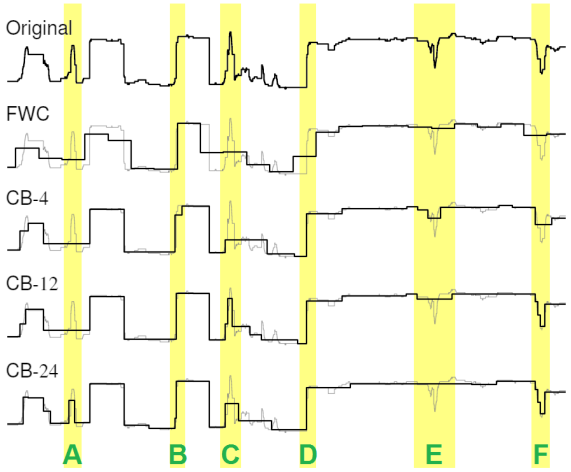


Figure 5: Light sensor data compressed with a 1-minute target resolution using different schemes

Looking at Figure 5, the prominent rising edge labeled B is apparently preserved by the simple FWC algorithm. This turns out to be incidental; the edge is preserved only because it happens to coincide with the boundary between one fixed-width output segment and the next. By contrast, the edge labeled D coincides with the middle of an output segment, and is consequently misrepresented in the compressed time series. Note that these two edges and others emerge from all three compression buffers intact.

With FWC, spikes in the original data that are narrower than Δt_{res} are flattened and widened to some extent. Depending on the situation, a compression buffer may preserve such spikes. Looking again at Figure 5, the spikes labeled C and F are considerably flattened after passing through the size-4 compression buffer, yet well represented when the buffer size is increased to 12. This makes sense because, as the segments that compose a spike pass through a compression buffer, they will survive unaltered only if there are other segments in the buffer that can be merged instead. When the buffer size is increased to 24, spike A is preserved as well, though this may have come at the expense of one or two segments near spike C. Note that spike E is essentially neglected by all four compression schemes. The 1-minute target resolution may be too coarse preserve this feature.

Analysis of Compression Ratios and Errors

From visual observations we gain an intuitive sense that the application of a compression buffer outperforms FWC, and that a buffer with a maximum size of a dozen or more segments produces better results than a buffer limited to a few segments. To test this perception quantitatively, we applied FWC and compression buffers of different sizes to eight different time series. Each time series was acquired over a 4-day period by a different sensor, and compressed with the following target resolutions: 1, 5, 15, and 30 seconds; 1, 5, 15, and 30 minutes; and 1 hour. We recorded the resulting compression ratios and associated compression errors. The *compression ratio* is the number of output segments divided by the number of input segments. The compression error δ was defined in (1), but because we store error values with each output segment, we are able to use the more convenient expression in (4). Here i, t_i, and δ_i pertain to datapoints of the output time series $\hat{y}$. The input time series y is no longer needed to compute the error.

$$\delta = \sqrt{\frac{1}{t_b - t_a} \cdot \sum_{i=a}^{b-1} (t_{i+1} - t_i) \cdot {\delta_i}^2} \qquad (4)$$

Figure 6 shows a scatter plot of the compression ratios and errors obtained using data from four temperature sensors. As expected, relatively small compression ratios are associated with relatively high errors. For each compression scheme (i.e. FWC, CB-4, CB-12, or CB-24), the points in the scatter plot are quite spread out. This indicates that the errors one encounters for a given compression ratio vary significantly from one

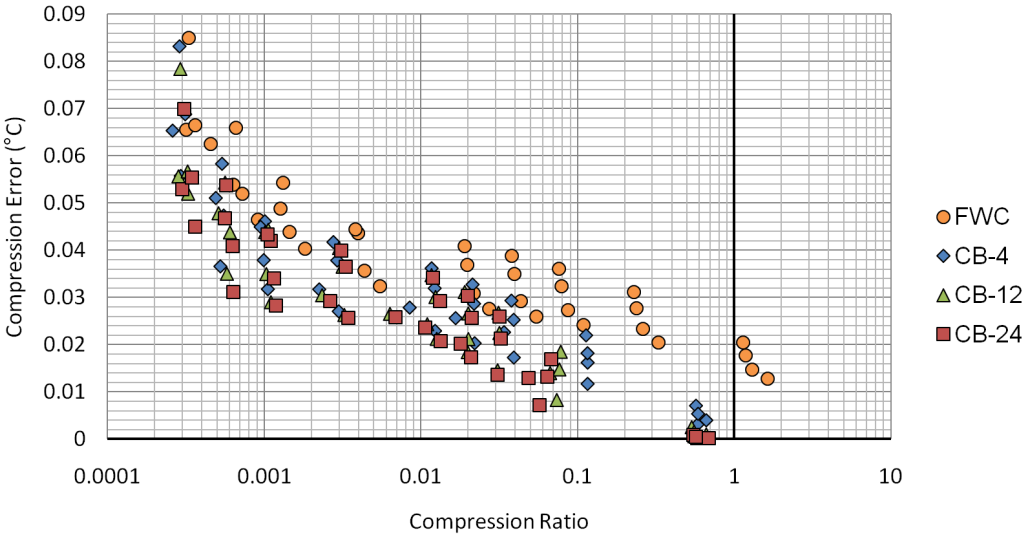


Figure 6: Temperature data compression statistics.

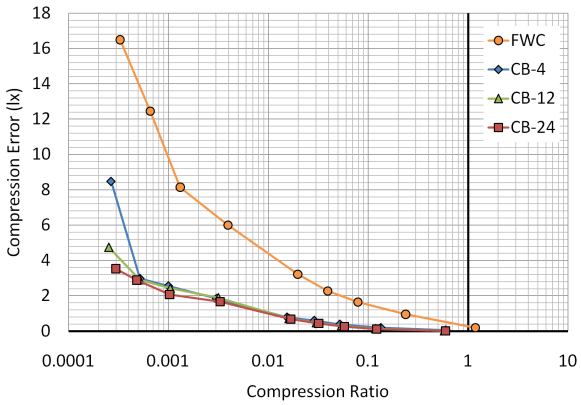
time series to the next, even if both are collected using the same type of sensor. Despite large deviations, the errors produced by FWC are noticeably higher than those produced by the compression buffers, particularly as compression ratios approach 1. There also appears to be a slight tendency for compression errors to decrease with increasing buffer size.

Because a compression buffer can transform a time series only by merging segments, the number of segments cannot increase and the compression ratio cannot exceed 1. Furthermore, as reflected in Figure 6, the compression error will converge on zero as the compression ratio approaches 1. With FWC, by contrast, the number of output segments can exceed the number of input segments. Care must to taken with FWC to avoid the case of the rightmost points of Figure 6, where an excessively fine target resolution increases the size of a time series and still introduces error.

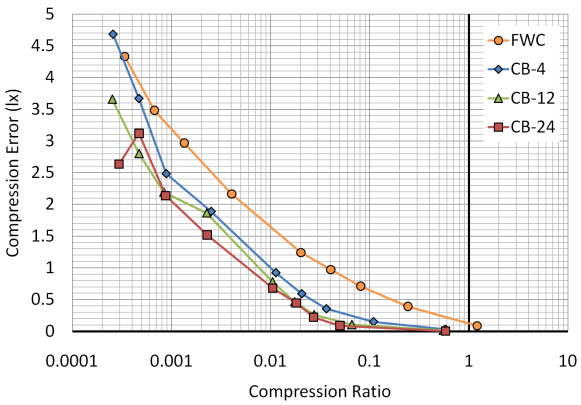
For the light and electrical current data, the results for each sensor are shown in a separate scatter plot. For light sensor 1, in Figure 7a, the errors produced by all three compression buffers are roughly four times smaller than those produced by FWC over a wide range of compression ratios. For light sensor 2, in Figure 7b, the compression buffers still appear to outperform FWC, but the relative difference is only appreciable for compression ratios greater than 0.01.

On the right-hand side of Figure 7b, and in a few other cases as well, the size-12 and size-24 compression buffers yield noticeably smaller errors than the size-4 buffer. This is consistent with our qualitative observations, as larger compression buffers appear more likely to preserve spikes. Somewhat surprisingly, there are many cases in which all three compression buffers produce essentially the same error. It is possible that while certain spikes are prominent in appearance, they may be too small in width to have a noticeable effect on our quantitative error measurements.

We note that for the two leftmost points of CB-24 in Figure 7b, the compression error unexpectedly increases with target resolution. The reason this is possible is that, given a finer target resolution, the compression buffer operates over a shorter time window and may miss opportunities to better "distribute" the output segments.



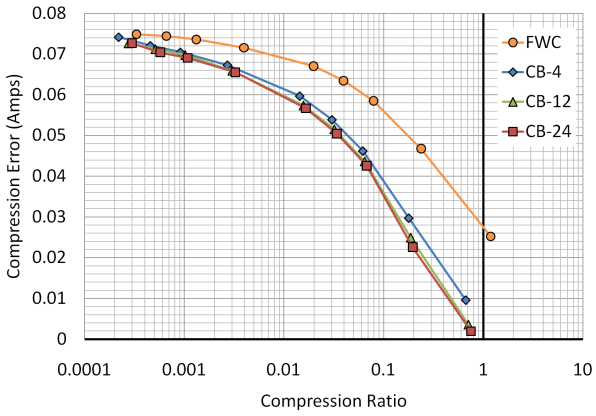
(a) Light sensor 1



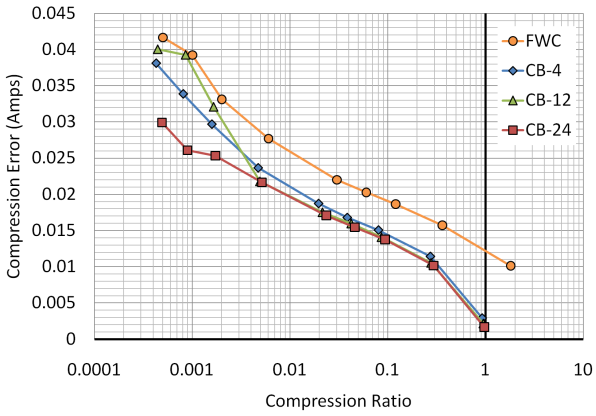
(b) Light sensor 2

Figure 7: Light sensor data compression statistics

The electrical current sensors were set up to monitor personal computers in an office. As shown in Figure 8, the compression buffers again outperform FWC, particularly for compression ratios greater than 0.1.



(a) Current sensor 1



(b) Current sensor 2

Figure 8: Electrical data compression statistics

FWC has one advantage not represented in the scatter plots: the individual segment start times t_i need not be stored in the output time series. Assuming one stores only the value y_i of each segment, and assuming all scalar values require the same storage space, the FWC compression ratios should all be reduced by a factor of 2. With this correction the compression buffers would still outperform FWC for light sensor 1, but in certain other cases FWC would appear more competitive. The assumptions that lead to this 2-fold correction are questionable, however. We argue that each output segment should contain an error value δ_i, which implies a 3:2 correction. If one were to store the minimum and maximum input values as well, there would be less benefit still in discarding the start times.

APPLICATIONS

Application to Signal Processing and Visualization

Our approach to time series compression supports further signal processing and visualization in a number of ways. First, by precomputing and storing copies of a time series at different resolutions, one supports interactive graphs like that of the well known web-based application *Google Finance*. With this type of visualization tool, the resolution of the visible data changes automatically as the user manipulates the time period of interest. Second, time series with variable-width segments may be processed with fewer computations, as few segments are needed for relatively flat regions. Third, the error values stored in the compressed segments provide useful information.

Here we demonstrate how a common signal processing routine, a low-pass filter, can be applied to a compressed time series produced by the proposed algorithm. Filtering is typically formulated as a convolution using a *window function*. We assume that a window function f is defined only between -1 and 1, and normalized such that $\int_{-1}^{1} f(x)dx = 1$. With variable-width segments, we require the corresponding cumulative function F, where $F(x) = \int_{-1}^{x} f(u)du$. Below are f and F for the Hann function, a window function frequently used by low-pass filters.

$$f(x) = \frac{1 + cos(\pi \cdot x)}{2}, \; -1 < x < 1$$

$$F(x) = \frac{1}{2} \cdot \left(1 + x + \frac{sin(\pi \cdot x)}{\pi}\right), \; -1 < x < 1$$

With errors values stored in the compressed segments, we can produce not only a filtered signal z, but also an approximation s of the signal's variability. When z and s are evaluated at time t, the window function is centered on t and scaled such that its half-width in the time domain matches the parameter d. Both z and s are defined below, with ${\rho_i}^2 = {y_i}^2 + {\delta_i}^2$ and $\Delta{\rho_i}^2 = {\rho_{i+1}}^2 - {\rho_i}^2$. The integers a and b are chosen such that $t_a \leq (t - d) < t_{a+1}$ and $t_b \leq (t + d) < t_{b+1}$.

$$z(t) = y_b - \sum_{i=a}^{b-1} (y_{i+1} - y_i) \cdot F\left(\frac{t_{i+1} - t}{d}\right)$$

$$s(t) = \sqrt{{\rho_b}^2 - \sum_{i=a}^{b-1} \Delta{\rho_i}^2 \cdot F\left(\frac{t_{i+1} - t}{d}\right) - z(t)^2}$$

Figure 9 shows roughly five hours of electrical data, the same data compressed with a size-12 buffer at a 2-minute target resolution, and the result of applying a low-pass filter to both the original and compressed time series. The filter uses a Hann window with a 5-minute half-width. In the CB-12 plot, the region bounded by $y_i \pm \delta_i$ is shaded. For the filtered data, the shaded regions are bounded by $z(t) \pm s(t)$.

The filtered data in Figure 9 can be sampled to approximate the average power consumption and noise level of a personal computer near a certain time t. Directly sampling the original or compressed time series is less informative, since t may coincide with an anomalous spike in the data. Note that it is far more efficient to

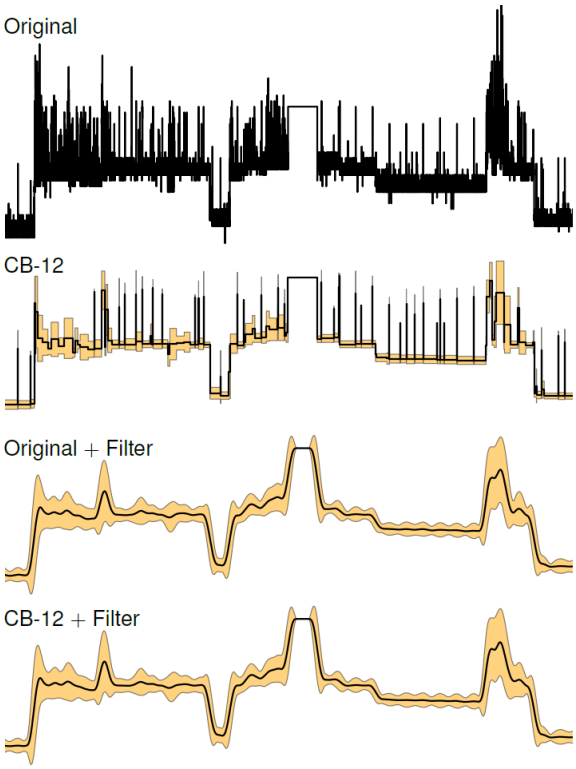


Figure 9: Electrical data with low-pass filtering applied to the original and compressed time series

filter the compressed data than the original data, as in this case the latter contains roughly a hundred times more datapoints. Yet because the filter parameter d is somewhat greater than Δt_{res}, applying the filter to the compressed time series produces essentially the same result as applying it to the original.

Application to Building Performance Simulation

The proposed compression algorithm is appropriate for building performance data because it can be applied to a wide range of sensor types without the risk of selecting inappropriate quality guarantees, and without the risk of increasing the number of segments in an unexpectedly sparse time series. A database would record a separate time series for each sensor and each target resolution. Subsets of various time series would then be extracted from the database at requested resolutions to serve as simulation input data.

For a typical discrete time simulation, compressed input data would be requested at a resolution somewhat finer than the simulation time step. Although the use of variable-width segments would reduce bandwidth requirements between the database and the simulation platform, the data would have to be converted into a fixed-width representation before entering the simulation. For a discrete event simulation, as applied to building performance in Zimmerman (2001), variable-width segments could be input directly. Simulations of either type should be able to input the compression errors, and use them to help estimate uncertainty.

Future building energy models may become so detailed that real time compression is required for simulation output data as well as input data. A trend towards longer simulation periods has been been observed by Morbitzer (2003), who also highlights the important information provided by short-term fluctuations found in long-term simulation results.

REFERENCES

Barnaby, C. S. and Crawley, D. B. 2011. Weather data for building performance simulation. In Hensen, J. L. M. and Lamberts, R., editors, *Building Performance Simulation for Design and Operation*, pages 37–55. Spon Press.

dos Santos, G. H. and Mendes, N. 2004. Analysis of numerical methods and simulation time step effects on the prediction of building thermal performance. *Applied Thermal Engineering*, 24(8–9):1129–1142.

Eisenhower, B., Maile, T., Fischer, M., and Mezic, I. 2010. Decomposing Building System Data for Model Validation and Analysis Using the Koopman Operator. In *Proceedings of the National IBPSA-USA Conference*, New York, USA.

Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S. 2001a. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Santa Barbara, USA.

Keogh, E., Chu, S., Hart, D., and Pazzani, M. 2001b. An Online Algorithm for Segmenting Time Series. In *Proceedings of the IEEE International Conference on Data Mining*, San Jose, USA.

Kunz, J., Maile, T., and Bazjanac, V. 2009. Summary of the Energy Analysis of the First year of the Stanford Jerry Yang & Akiko Yamazaki Environment & Energy (Y2E2) Building. Technical report, CIFE, Stanford University.

Lazaridis, I. and Mehrotra, S. 2003. Capturing Sensor-Generated Time Series with Quality Guarantees. In *Proceedings of the International Conference on Data Engineering*, Bangalore, India.

Morbitzer, C. 2003. *Towards the Integration of Simulation into the Building Design Process*. PhD thesis, University of Strathclyde, Glasgow, Scotland.

Raftery, P., Keane1and, M., and Costa, A. 2009. Calibration of a Detailed Simulation Model to Energy Monitoring System Data: A Methodology and Case Study. In *Proceedings of the International IBPSA Conference*, Glasglow, Scotland.

Zimmerman, G. 2001. A New Approach to Building Simulation Based on Communicating Objects. In *Proceedings of the International IBPSA Conference*, Rio de Janeiro, Brazil.