Floating-point numbers in Sprintz

Zihan Yu, Rober Wang

1 INTRODUCTION

1.1 Use Case / Motivation

We live in an exciting age. Technology is advancing faster than ever and with the recent developments in computing power and storage, the path has been paved for a new era: the internet of things. Over the recent decade, our society and daily lives have become increasingly intertwined with technology and automated devices. From our cars to our iphones, more and more everyday objects are producing valuable data that can be extracted and analyzed. With this recent in□ux in the sheer amount of raw data, researchers from top universities and private corporations are seeing breakthroughs in the □elds of deep learning, AI, and machine learning. However, before any novel neural network learning algorithms can be implemented, at the root of this new era, lies one fundamental question: How can we store and compress data in the most e□cient way possible?

1.2 Current State of the Art

When it comes to the internet of things, most of the data that is created takes the form of integer or □oating point values. Currently, there are quite a few state of the art developments that can e□ectively handle the compression of integers such as delta-of-delta encoding, zigzag encoding, or a combination of assorted methods such as Sprintz. Also, delta-of-delta encoding works especially well for data that do not change signi□cantly in value between sequential data points. This makes it a perfect integer compression technique for most IoT data. In terms of lossless □oating point compression, however, the only well known method is Facebook's Gorilla. As one of the only e□ective ways of compressing □oating-point values, Gorilla has been widely adopted by most of today's technology □rms. One of the reasons for Gorilla's success is that the compression scheme works directly on the □oating point values. There is no need for a pre-processing stage or any sort of quantization. As a result, the compression is lossless. The way it works is that given time-series data , Gorilla applies a standard delta of delta compression to the timestamp values and then a separate technique for the □oating-point values. For the □oating-points, the algorithm begins by storing the value completely uncompressed in memory. Then for all subsequent values, Gorilla applies the xor operation along with few additional compression techniques. Using this algorithm, Facebook claims that they achieved compression ratios of more than 12x, from 16 byte values down to 1.37 bytes. While this may seem impressive, it's important to note that the tests in which this compression ratio was achieved, were all done on Facebook speci□c data.

1.3 Intuition for Approach

Recognizing the current gap between the deluge of □oating point time series data (i.e. stock market data) and the lack of □oating point compression techniques, Atlas and I decided to tackle this issue by creating a modi□ed version of Sprintz. In its unmodi□ed state, Sprintz is a time series compression algorithm designed for integer compression. While it can also handle □oating point values, it does so by preprocessing them through a lossy quantization step which converts the □oats to integers. For our research experiment, we wanted to □nd a way to compress □oats losslessly using Sprintz. To achieve this goal, however, we would need to devise a new method of quantization as Sprintz's current method of quantization loses information when it rounds o□ the □oats.

1.4 Contribution

We came up with two ways of modifying Sprintz that allowed us to perform lossless □oating point compression and decompression. The intuition behind our approach is based on the IEEE 754 □oating point representation which involves a sign bit, exponent bits, and a mantissa. Rather than using Sprintz quantization technique, which involves taking the □oor of the value, we devised a two-dimensional integer representation for each □oat value. By breaking down each □oat in such a way, we were able to avoid the rounding process that the original Sprint's quantization technique used. Two dimensional integers allowed us to achieve lossless compression of □oats using Sprintz, but it did come at a slight space overhead.

2 IMPLEMENTATION

2.1 Overview

The algorithm consists of two parts: Lossless □oating point conversion and Sprintz. Sprintz is a time series compression algorithm designed to improve data analysis on the internet of things (IoT) devices, which often have less memory space and lower bit width. Although Sprintz shows a higher compression ratio for 8 and 16-bits data, it only supports integer operation. In order for Sprintz to process □oating-point numbers, we need to represent □oating points using integers before compression.

2.2 Floating Point Conversion

We explored two ways to convert □oating-point numbers into integers:

- Split the □oat in the middle and break it into two parts. Then interpret each part into an integer that contains exactly the same bits.
- Split the □oat into sign bit, mantissa and exponent using the IEEE-754 standard. Then store the sign bit and exponent into one integer and the mantissa into another.

For our algorithm, we only apply it on 16-bits □oating-point numbers, which can be split into two 8-bits integers using method 1, or two 16-bits integers using method 2. The □gures below show how 16-bits □oat is interpreted into integers using the above methods.

2.3 Sprintz Architecture

Sprintz is a time series compression algorithm which is the state-of-the-art for lossless compression multivariate time series on low

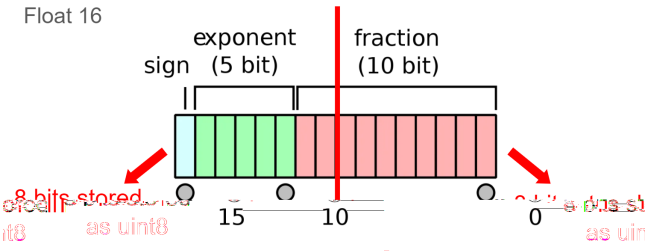

Figure 1: Figure 1. Split a 16-bits □oating-point number into two 8-bits unsigned integers using method 1

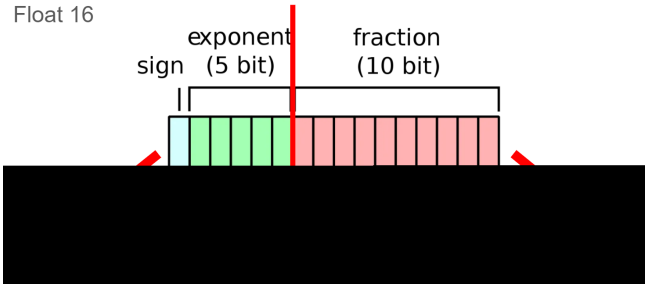

Figure 2: Figure 2. Split a 16-bits □oating-point into sign bit, exponent and mantissa with method 2

memory and low bit-width IoT devices. The Sprintz algorithm consists of the following components:

- `Forecasting` Sprintz trains a forecaster to predict each sample based on previous values. In encodes the di□erence between the next value and the predicted value [1].
- `Bit packing` Sprintz bit pack the data and includes a header to help unpack [1].
- `Run-length encoding` Sprintz waits for a block in which some bit is non-zero and then writes out the number of zeros, instead of output the whole block of zeros [1].
- `Entropy coding` This is equivalent to Hu□man encoding [1].

The entire Sprintz pipeline can be as follows: The dataset -> delta encoding -> zigzag encoding -> (run-length encoding) -> bit packing. Depending on the number of dimensions of the data, the result can either be column-major for low dimensions, or row-major for high dimensions. The pipeline is shown in □gure 3 below.

3 EXPERIMENTS

3.1 Datasets

For the experiment, we used UCR Time Series Archive [2]., a repository of 85 univariate time-series datasets from a range of domains. The UCR dataset is widely used to benchmark time series algorithms. Because some data from UCR is short, the experiment concatenates all the data into a single long time series. To mitigate arti□cial jumps in value from the end of one time series to the

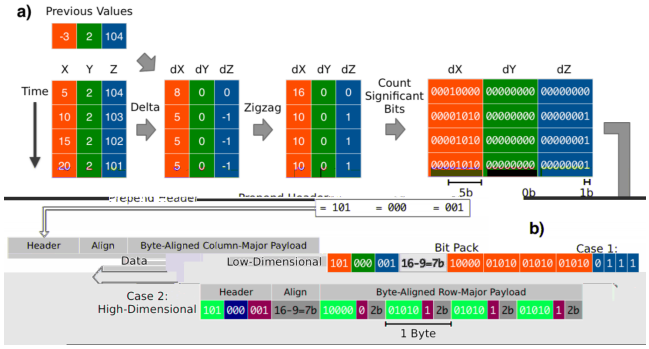

Figure 3: Figure 3. Sprintz algorithm pipeline [1]

beginning of the next one, the program linearly interpolates the data and normalize all the data.

3.2 Comparisons and observations

The experiment compares the performance of the following four Sprintz variations with a set of other state-of-the-art compression algorithms:

- `SprintzFIRE + Huff` The full Sprintz pipeline
- `SprintzDelta + Huff` Replace the forecasting component with delta encoding.
- `SprintzFIRE` The Sprintz pipeline without Hu□man encoding
- `SprintzDelta` Replace the forecasting component with delta encoding without Hu□man encoding.

The comparison results are shown in the □gures below:

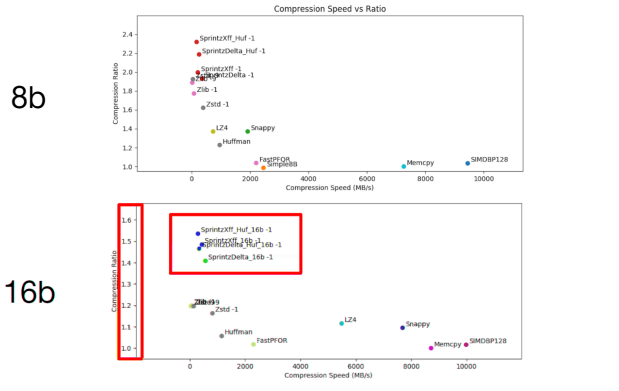

Figure 4: Figure 4. Experiment result using the Sprintz's original method of quantizing □oat into integer and feed the quantized data into Sprintz

There are two key observations comparing the experiment results:

- The lossless conversion from □oating-point to integers yields a better compression ratio than the original lossy quantization method. The new methods achieve compression ratios

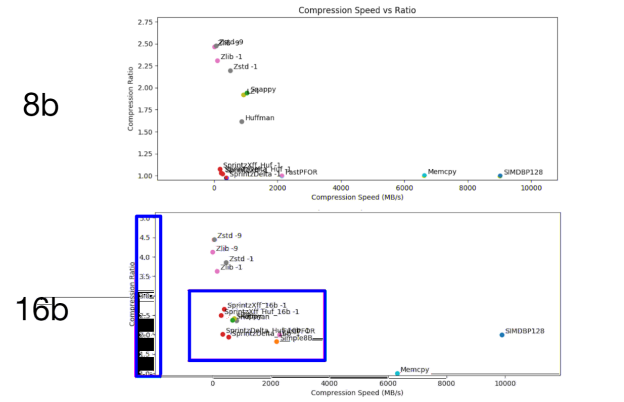

Figure 5: Figure 5. Experiment result using method 1 from above, which splits a oating-point number right in the middle and interpret each part in an unsigned integer

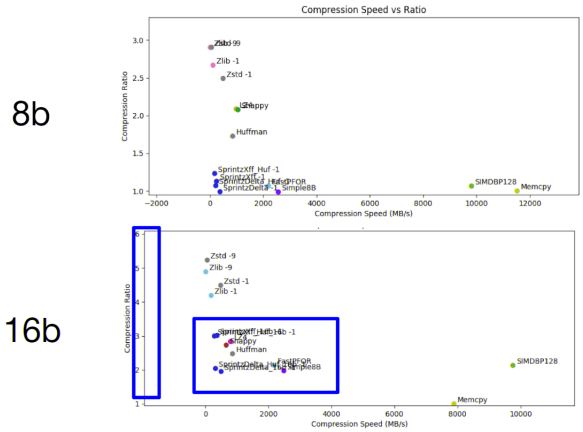

Figure 6: Figure 6. Experiment result using method 2 from above, which divide a oating-point number into sign bit, exponent and mantissa, and interpret them into two integers

in a range from 2 - 3, compared with the original method of 1.4 - 1.6.

- Sprintz doesn't perform so well with the lossless data comparing with the other compression algorithms. This is because Sprintz isn't as e ective when the time series has large, abrupt changes and few variables. The mantissa part from the lossless conversion could cause the interpreted integer values to uctuate drastically, therefore Sprintz cannot handle the data so well as the others.

4 CONCLUSION

In the end, while we managed to successfully implement the lossless compression scheme, it wasn't very e ective. Sprintz was built for integer compression, and when it comes to oating point data points, it's oftentimes acceptable to use lossy compression techniques. In many instances, precise data point values are not as signi cant as larger trends in the data. In terms of future work, it would be interesting to explore some of the other novel ideas in oating point compression. So far, Gorilla is one of the only major players in the eld, and we think it's time for that to change.

REFERENCES

[1] John Guttag Davis Blalock, Samuel Madden. 2018. Sprintz: Time Series Compression for the Internet of Things. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 3 (2018), Article 93.

[2] B. Hu N. Begum A. Bagnall A. Mueen Y. Chen, E. Keogh and G. Batista. 2015. The UCR time series classi cation archive. *www.cs.ucr.edu/ eamonn/time$_s$eries$_d$ata*(2015).