Compression of Molecular Dynamics Simulation Data

Shuai Liu

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-65
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-65.html

May 17, 2019

Compression of Molecular Dynamics Simulation Data

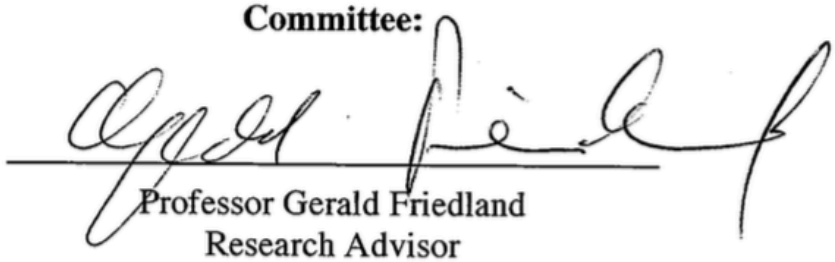
by Shuai Liu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor Gerald Friedland
Research Advisor

Dec 7, 2018
(Date)

* * * * * * *

Professor Kannan Ramchandran
Second Reader

Dec. 10, 2018
(Date)

Compression of Molecular Dynamics Simulation Data

Shuai Liu

Abstract Molecular dynamics (MD) simulation is an approach to explore physical, chemical and biological systems computationally when they cannot be investigated in nature. By modeling the interactions between atoms and/or molecules using Hamiltonian principles, hypotheses about physical, chemical, or biological properties of various systems can be tested. Given that the simulations are done on atomic level, it is easy to generate terabytes of data. MD simulation results therefore not only require large amounts of storage, but also result in very slow to transfer simulation outcomes between computers. In this report, I present work on designing and analyzing algorithms for MD simulation data compression in both the time and space domain. I also present experiments to understand the relationship between compression performance and underlying physical behavior of the simulation. Specifically, I analyze the Shannon entropy of MD simulation data in correlation with different physical parameters.

Acknowledgement

First, I would like to express my deepest gratitude to my research advisor, Professor Gerald Friedland. I am fortunate to have the opportunity to explore this interdisciplinary field with his mentorship. He is a rigorous mentor and scholar. In the meantime, he is also a very nice friend. He provided me tremendous help throughout my projects and my career. Thanks, Gerald!

Second, I would like to thank Professor Kannan Ramchandran and Dr. Alfredo Metere. Kannan gives me a lot of ideas and inspirations throughout many discussions as the second reader, which helps me make a significant improvement of this report. Alfredo is always willing to help me when I have difficulties in physics principles, algorithms and data analysis. Thanks again for your help during my master study in computer science.

Third, I would like to thank the professors and administration staffs from both EECS and chemistry department, who give me generous help. I appreciate their help when I decide to explore the interdisciplinary projects between computer science and physical science. Touching this research area provides me useful technical skills for my future career and gives me inspirations on the future research. The skills I learned during this study also in turn help me develop useful toolkits for chemistry and materials studies, such as machine learning toolkits for automatic materials structure identification, and experimental setup stabilization using machine learning approaches. I appreciate the graduate study platform at UC Berkeley, which allows me to develop the interdisciplinary studies.

Contents

1 Introduction

In this chapter, we discuss the motivation, related theoretical background and the overview of subsequent chapters.

1.1 Theoretical Background

It is challenging to understand complex chemical and biological systems at atomic scale using experimental approaches. Therefore, computational algorithms are designed to investigate the atomic level interactions and dynamics in these systems. In the past a few decades, high performance computer clusters have enabled large scale scientific computing and simulations [1, 2, 3, 4]. Here are several examples to show how simulations can solve real world problems in physics, chemistry and biology:

1. Predict the properties of chemical, biological or material systems [5].
2. Calculate important physical parameters in certain systems [6].
3. Understand the interactions between large bio-molecules [7].

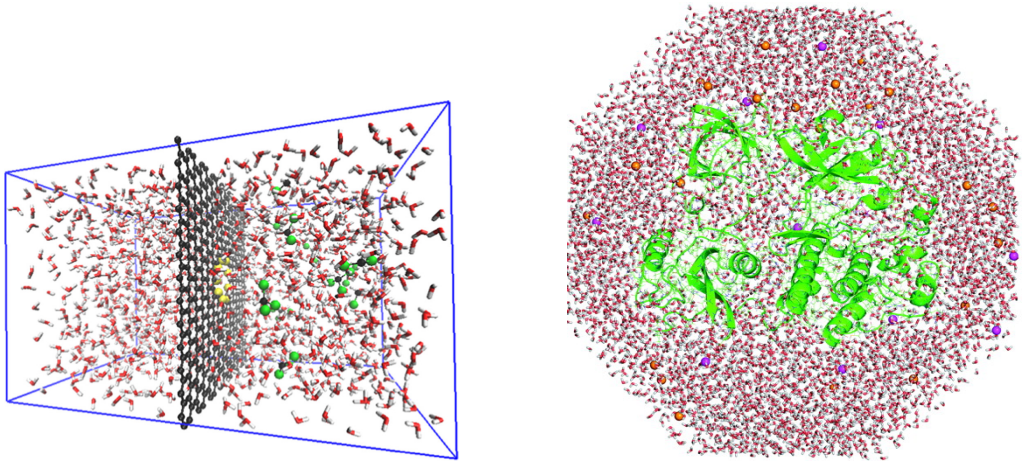


Figure 1.1 Atomic level simulation in materials science and biological systems. The balls in different colors stand for different atoms, such as hydrogen and oxygen. The left figure is reprinted from The Chemical Engineering Science, Volume 127, Jafar Azamat et al., Molecular dynamics simulation of trihalomethanes separation from water by functionalized nanoporous graphene under induced pressure, Pages 285-292, Copyright 2015, with permission from Elsevier. The right figure is reprinted in literature [8] Copyright 2005 National Academy of Sciences.

Figure 1.1 shows two simulated systems: water purification [9] and protein salvation [8]. Specially, molecular dynamics (MD) simulation is commonly used to model and simulate the dynamics of chemical and biological systems at atomic level. By constructing the potential functions between atoms, MD programs can calculate the positions and momentums of atoms as a function of time using the principles of Hamiltonian mechanics. In the MD systems we investigate in this report, Hamiltonian $\mathcal{H}$ is composed of potential energy U and kinetic energy K:

$$\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) = U(\boldsymbol{q}) + K(\boldsymbol{p}) \tag{1}$$

where $\boldsymbol{q}$ is the position vector and $\boldsymbol{p}$ is the momentum vector. The kinetic energy is determined by the momentum $\boldsymbol{p}$ of particles:

$$K(\boldsymbol{p}) = \sum_{i=1}^{N} \frac{||\boldsymbol{p}_i||^2}{2m_i} \tag{2}$$

Potential energy is a function of the position $\boldsymbol{q}$. In the MD simulation systems, the pairwise potential energy is most commonly used:

$$U(\boldsymbol{q}) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N-1} V(\boldsymbol{q}_i, \boldsymbol{q}_j) \tag{3}$$

For example, the Lennard-Jones [10] potential is used to approximate the potential energy between two particles, which is a function of distance r between two particles: $r = ||\boldsymbol{q}_i - \boldsymbol{q}_j||$. The mathematical formula of the Lennard-Jones potential is

$$V(r) = 4\epsilon(\frac{\sigma}{r^{12}} - \frac{\sigma}{r^6}) \tag{4}$$

where ϵ and σ depend on the physical system. The function is plotted as shown in Figure 1.1 when we set $\sigma = 1, \epsilon = \frac{1}{4}$.

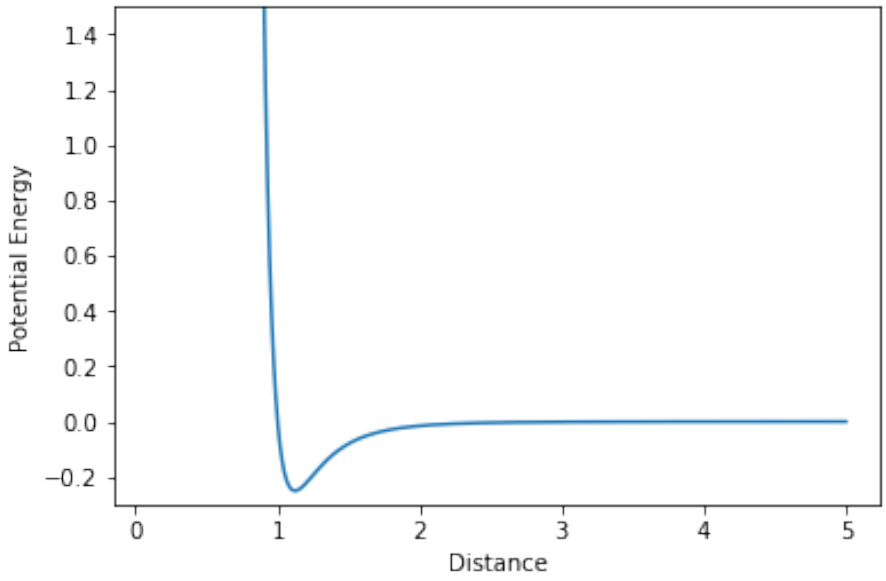


Figure 1.1 Lennard-Jones potential energy as a function of pair-wise distance r.

In each simulation step, forces $\boldsymbol{F}$ is calculated based on the gradient of potential energy

$$\boldsymbol{F} = -\nabla U(\boldsymbol{q}) \tag{5}$$

and then update the velocity and position accordingly. In the time domain, the motion follows Newton's equation:

$$\dot{\boldsymbol{q}} = \frac{\boldsymbol{p}}{m} \tag{6}$$

$$\dot{\boldsymbol{p}} = \boldsymbol{F} \tag{7}$$

MD simulation is to solve these differential equations numerically using symplectic integrator. Figure 1.2 shows a flowchart of MD simulation algorithm.

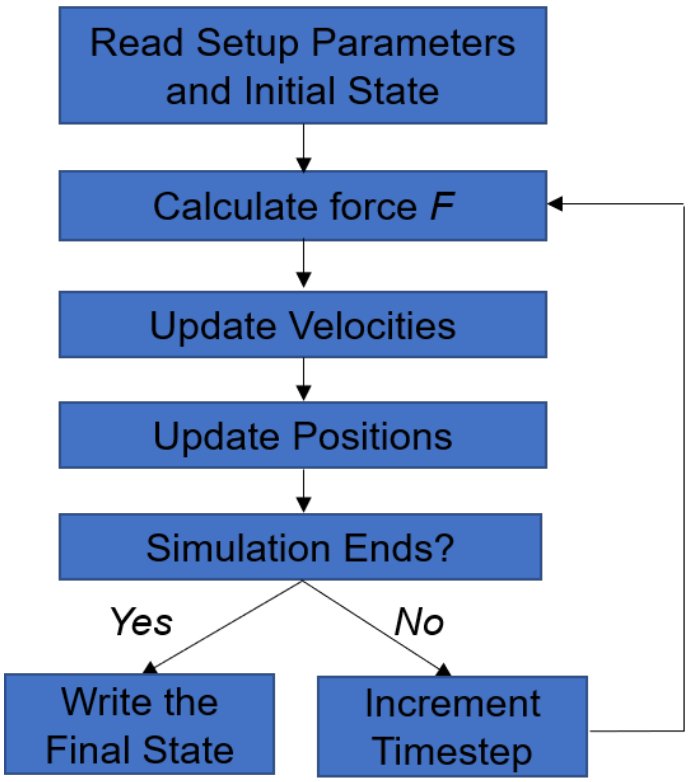


Figure 1.2 MD simulation process.

First, the initial configurations are set, including the initial position, velocity and potential energy function. At each timestep, the simulator will calculate the forces and update the velocities and positions accordingly. In detail, one typical MD simulation algorithm has the following steps:

1. Set the initial position $\boldsymbol{q}_0$, velocity $\boldsymbol{v}_0$ and other parameters, such as time $t = 0$ and timestep $i = 0$.

2. Calculate the force $\boldsymbol{F}$ and acceleration $\boldsymbol{a}$

$$\boldsymbol{F} = -\nabla U(\boldsymbol{q}_i) \tag{8}$$

$$\boldsymbol{a} = \frac{\boldsymbol{F}}{m} \tag{9}$$

3. Update the velocities $\boldsymbol{v}$

$$\boldsymbol{v}_{i+1} = \boldsymbol{v}_i + \boldsymbol{a}\Delta t \tag{10}$$

4. Update the positions $\boldsymbol{q}$

$$\boldsymbol{q}_{i+1} = \boldsymbol{q}_i + \boldsymbol{v}_i \Delta t + \frac{1}{2}\boldsymbol{a}\Delta t^2 \tag{11}$$

5. Go to step 2, $i \leftarrow i + 1$

At each simulation timestep i, the simulator will save the position $\boldsymbol{q}$ of all the atoms as a single frame. The whole MD simulation dataset is composed of the frames ordered by the timesteps.

1.2 Motivation of MD Simulation Data Compression

Given the simulation is performed on molecular level, MD simulator can easily generate large scale datasets. MD simulation expedites the scientific discovery process, but also raises some problems for the data storage and analysis:

1. For data storage, MD simulation datasets can be in TB to PB scale. To address this issue, we design different types of algorithms, such as conventional piece-wise polynomial predictors and novel deep learning methods, to compress the MD simulation data.

2. It is computationally intensive to analyze the whole TB scale datasets. On the contrary, in some datasets, the important events (for example, phase transition) may only happen in some of the frames. In order to capture these events, in this report, we calculate several parameters from an information viewpoint, such as Shannon entropy. We observe that the Shannon entropy drastically changes during the phase transition. These results and methods can be potentially applied to an open but challenging problem: phase transition detection.

In the past, these two problems are usually addressed separately. In fact, these two problems are coherently related: the underlying physics determines the optimal compression performance; the anomaly event happens when the physical behavior drastically changes. In this report, we first start with the empirical analysis of the compression algorithms. Then, we use two studies to investigate the relationship of compression performance and underlying dynamics.

1.3 Overview of Subsequent Chapters

Here is the detailed flow of this report:

1. Chapter 2 gives a brief review about the related work of compression algorithms and data analysis methods.

2. Chapter 3 describes the datasets and experiment setups.

3. Chapter 4 discusses the compression algorithms in time domain.

4. Chapter 5 discusses the compression algorithms in space domain.

5. Chapter 6 gives an analysis of experimental results by comparing the performance of different compression algorithms.

6. Chapter 7 presents two empirical studies to show the relationship of compression performance and the underlying phase behavior.

7. Chapter 8 draws a conclusion of the report. We also provide the limitations of current work and the outlooks of the future directions.

2 Background and Related Works

Both the physics and signal processing communities have engaged in the process of data processing, analysis and compression. In the physics community, compression algorithms are designed with limitations based on the understanding and conditions of a given physical system. Likewise, in the signal processing community, one main focus is development of mathematical tools and statistical models towards better compression performance. This chapter gives a brief overview of the algorithms developed by both communities.

2.1 Compression Algorithm

Compression algorithms usually contain two modules: the encoder and decoder. The encoder transforms the original data into a compressed format. The decoder reverses this process: it transforms the compressed data back to its original format. There are two types of compression algorithms [11]: lossless and lossy compression algorithms. Lossless compression algorithms, such as entropy-based compression [12], can fully recover the original data through this encoding-decoding process. In this case, the compression performance can be evaluated by the number of bits required per sample. The lower bound of the lossless compression algorithm can be determined by the Shannon entropy.

Lossy compression algorithms are designed to store the data more efficiently while losing redundant or detailed information, which means that some original information cannot be fully reconstructed once the data is compressed. In this case, in order to evaluate the compression performance, the reconstruction error is measured as a function of compression rate.

2.2 MD Simulation Data Compression

MD production runs usually consist of several hundred millions to several trillions timesteps on system configurations of at least 10^4 particles, up to several billions, and occasionally even a few trillion particles. In the real world, the positions and velocities are real numbers. However, in order to perform the numerical calculations and store the data using binary coding, the data must be quantized and stored as double precision floating point numbers. From this viewpoint, there is always an error when the positions and velocities are quantized.

To efficiently store these large-scale MD simulation datasets, many compression algorithms have been proposed in the past a few decades. Most of them focus on the compression of MD simulation data in time domain, which is to compress the trajectory of single atoms at different timesteps.

One naive strategy is differential encoding [13]. Rather than saving the position q_i of each atom at each single timestep i, the differences between two consecutive frames are calculated and stored. At the beginning of the simulation, the initial value q_0 will be stored. At each timestep t, the position differences with the last frame $(t-1)$: $q_t - q_{t-1}$ are calculated and stored. This algorithm is suitable for the datasets where the differences are small and *ideally* constrained in some intervals $q_{i+1} - q_i \in [A, B]$. With these properties, the data can be quantized and stored efficiently. The disadvantage of differential encoding is that the quantization error may propagate as a function of time.

Recently, Jan Huwald et al [14] proposed a lossy compression algorithm using linear predictors to compress MD simulation data. The idea was to search piece-wise linear predictors that could bound the residual errors uniformly within a certain threshold ϵ. By setting the absolute error threshold to 10^{-1}, they obtained 1 bit per sample compression performance on certain datasets.

Other lossy compression algorithms have also been proposed [15, 16] by assuming certain aspects of the simulation data, such as sparsity or low rank properties. For example, the data may be represented by a few parameters in the frequency domain if it has certain periodicities; if the simulation data has low rank approximation, it can be projected to a low dimension subspace by calculating the principle components. However, these algorithms require certain assumptions on the datasets, which may not be applied to general MD simulation compression.

2.3 Inspiration from Multimedia Compression

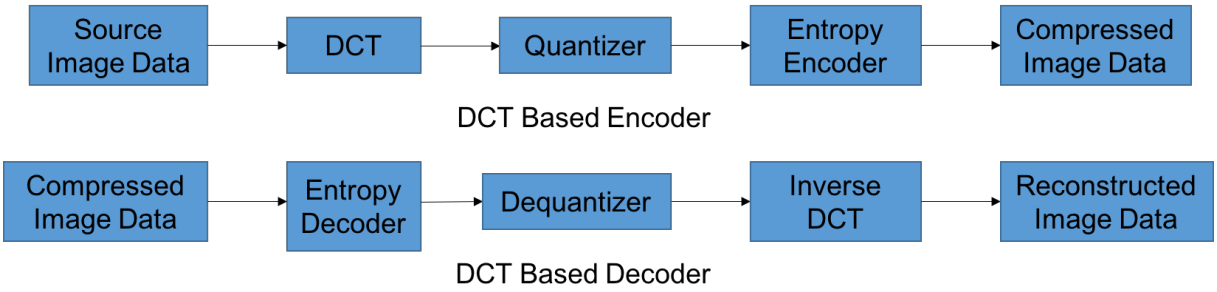


Figure 2.1 DCT based compression algorithm. The figure is adapted from literature [17].

Both MD simulation datasets and video datasets are composed of frames. In the computer science field, many multimedia compression algorithms have been developed for image and video datasets. Discrete cosine transform (DCT) based compression (scheme shown in Figure 2.1), for example, JPEG [17], is a type of well-established standard compression algorithms. DCT transforms the data from real space to discrete cosine space. For example, JPEG is based on block-wise DCT followed by quantization and entropy coding steps. Another type of compression algorithm is to design the filter using local spatial correlations explicitly, such as Paeth filter based compression algorithm [18]. These algorithms have been applied to many real-world image and video datasets.

2.4 Outlook of The Report

In this report, we start with time series modeling to analyze the MD simulation datasets in the time domain. Then, we present a general framework for MD simulation data compression using the piece-wise polynomial predictors. Moreover, we utilize the deep learning based methods, such as autoencoder [19] and LSTM gers1999learning, on this MD simulation data compression. In the space domain, we assign the data points onto the 3D cubic grid using a greedy approach. Then, Paeth filter based compression algorithm can be applied on these structured datasets. Moreover, the compression performance is dependent on the underlying physics. We will illustrate the relationship of compression performance and underlying phase behavior using two examples.

3 Dataset and Experiment Setup

In this chapter, we give a detailed description on the datasets we will use in this report. In the following chapters, the empirical evaluation of the compression algorithms are based on these two datasets.

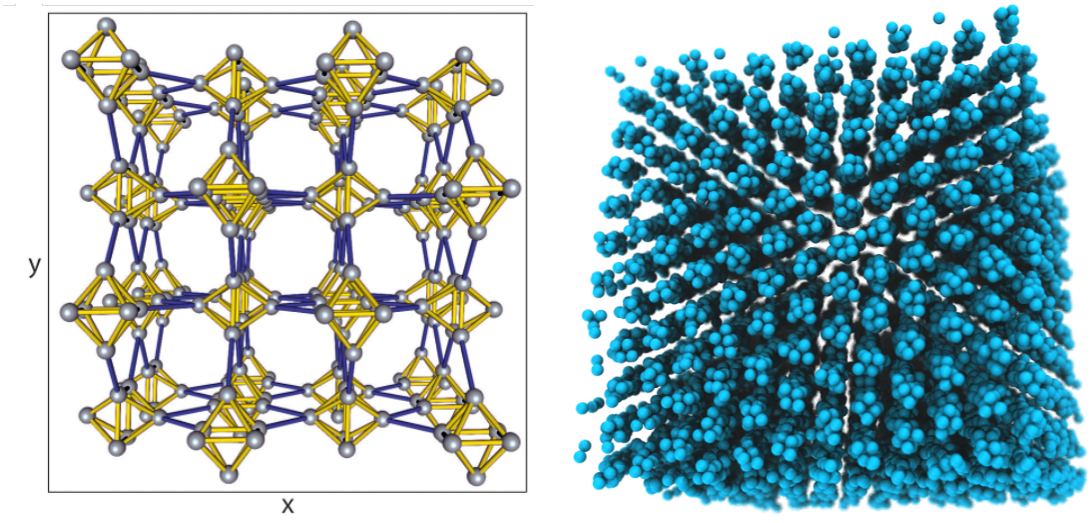
3.1 Datasets

In this report, we test our algorithms on two datasets:

1. MOFCOOL dataset: formation of the metal organic framework crystals from the liquid [20].

2. LIQCRY dataset: formation of the liquid crystals from the liquid phase. [21].

In both datasets, there are 16,384 particles. Each particle has three dimensions: x, y, z coordinates. The first dataset has 6,000 timesteps and the second dataset has 192,000 timesteps.



MOFCOOL Dataset LIQCRY Dataset

Figure 3.1 Visualization of the physical systems. The MOFCOOL dataset represents a type of system containing rigid framework structures. The LIQCRY dataset represents the liquid crystal formation process. The left figure is reprinted Reprinted from The Journal of Chemical Physics, 141, Alfredo Metere et al., Formation of a new archetypal Metal-Organic Framework from a simple monatomic liquid, 234503, Copyright 2014, with permission from with permission from AIP Publishing LLC.

In the MOFCOOL dataset, there are two different phases. The system is in the liquid phase during the first 1,000 timesteps. There is a phase transition from the liquid phase to the solid phase from timestep 1,000 to timestep 1,500. From timestep 1,500 to timestep 6,000, the physical system is in the solid phase. The LIQCRY dataset is similar: during the first 120,000 timesteps, the system is in the liquid phase. There is a phase transition from the liquid phase to the liquid crystal phase between timestep 120,000 and timestep 125,000. From timestep 125,000 to timestep 192,000, the system is in the liquid crystal phase. Figure 3.1 shows the visualization of these physical systems. The motivation of choosing these two datasets is that their phase transition behaviors are different: the phase transition from the liquid phase to the liquid crystal phase is more subtle than the phase transition from the liquid phase to the solid phase. By testing on both datasets, we claim that these compression algorithms are applicable for various MD simulation datasets. Also, we will compare the compression results to illustrate how the underlying physics and the compression performance are correlated.

3.2 Evaluation of Algorithms in Subsequent Chapters

In the following chapters, we will evaluate the performance of compression algorithms using these two datasets. The compression algorithms are evaluated by the number of bits needed per sample as a function of mean absolute error and/or absolute error threshold. In detail:

1. In chapter 4, we focus on the compression of single particle trajectories at different timesteps.

2. In chapter 5, we focus on the MD simulation data compression in a single timestep.

3. In chapter 6, we will compare the performance of different compression algorithms on these two datasets.

4. In chapter 7, we investigate how the underlying phase behavior influences the compression performance using two case studies.

4 Compression Algorithms in Time Domain

In this chapter, we design algorithms to compress the trajectories of single particles. We start with conventional time series analysis to understand the patterns in the trajectories. Then, we design several compression algorithms for MD simulation dataset: piece-wise polynomial based compression algorithms and deep learning based compression algorithms.

4.1 Time Series Modeling

Before the discussion of compression algorithms, we start with traditional time series modeling on the MD simulation datasets. Here, we examine the MOFCOOL dataset as an example. To simplify the problem, we only consider the trajectory of a single atom at different timesteps $i = 1, ..., n$. There are several traditional time series models, such as autoregression (AR), moving average (MA) and autoregressive integrated moving average (ARIMA). The mathematical formula are:

$$\text{AR}(p) \text{ Model: } X_t = \sum_{i=1}^{p} \alpha_i X_{t-i} + \epsilon_t \tag{12}$$

$$\text{MA}(q) \text{ Model: } X_t = \mu + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t \tag{13}$$

$$\text{ARIMA}(p, q) \text{ Model: } X_t = \sum_{i=1}^{p} \alpha_i X_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t \tag{14}$$

where X_i, ϵ_i are data and white noise at timestep i, respectively. α and θ are parameters of the model. To determine the order p, q, we calculate the autocorrelation function (ACF) and partial autocorrelation function (PACF) of position q as a function of time shown in Figure 4.1.

$$\text{ACF: } \gamma(h) = \frac{Cov(X_t, X_{t-h})}{Var(X_t)} \tag{15}$$

$$\text{PACF: } \alpha(h) = \frac{Cov(X_t, X_{t-h}|X_{t-1}, ..., X_{t-h+1})}{\sqrt{Var(X_t|X_{t-1}, ..., X_{t-h+1})Var(X_{t-h}|X_{t-1}, ..., X_{t-h+1})}} \tag{16}$$

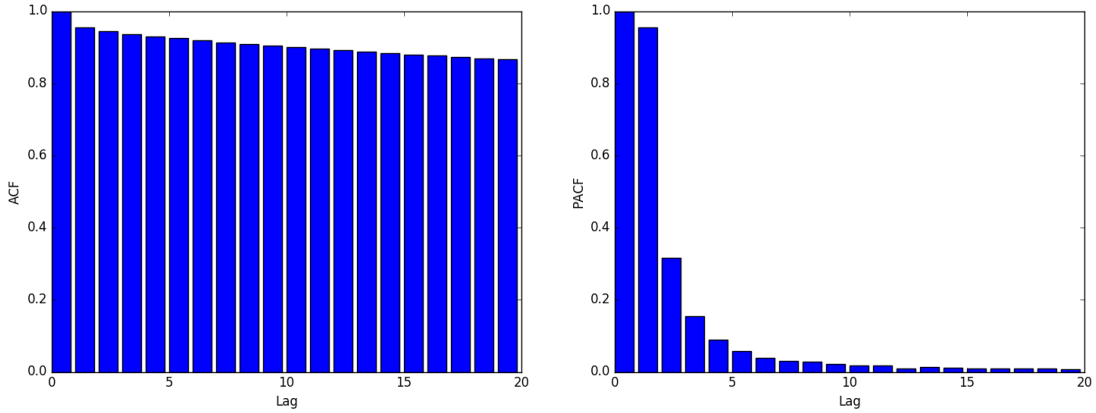


Figure 4.1 Autocorrelation function (ACF) and partial autocorrelation function (PACF) of position q in MOFCOOL dataset.

In the time domain, the MOFCOOL dataset (and some general MD simulation datasets) has the following properties:

1. From ACF, we observe that the positions are highly correlated. AR could be a good candidate to model the data in time domain.

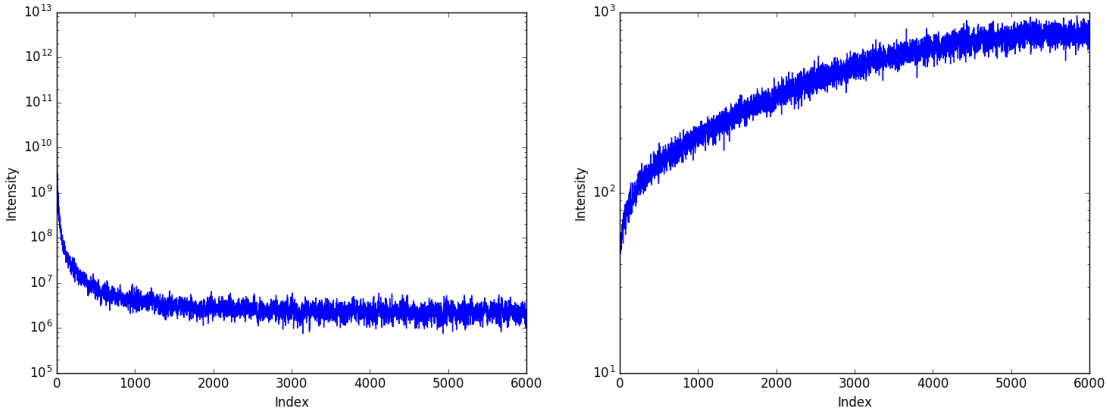
2. From PACF, the components from lag 1 to 5 are significant (p value< 0.05), which indicates that the order $p = 5$ in the AR model. The current position could be approximated by a linear combination of the positions in last five timesteps.

In physics, Newtonian mechanics also describes the movement of particles as a function of time. The positions and velocities of particles can be approximated as:

$$q_{i+1} = q_i + v_i \Delta t + \frac{1}{2} a_i \Delta t^2 \tag{17}$$

$$v_{i+1} = v_i + a_i \Delta t \tag{18}$$

where q_i, v_i, a_i are the position, velocity, acceleration of the atom at timestep i. An important note is that these two equations only provide a finite approximation, because the physical parameters q, v, a are changing continuously. This quadratic form is also the motivation for us to investigate the piece-wise polynomial based compression algorithms. We will discuss the details in section 4.2.



Position Data Velocity Data

Figure 4.2 Position and velocity data in discrete cosine space by DCT transformation.

We also analyze the data in discrete cosine domain by taking DCT transformation. The DCT transformation is

$$\text{DCT:} \quad f_m = \sum_{k=0}^{n-1} x_k \cos[\frac{m\pi}{n}(k+\frac{1}{2})] \tag{19}$$

where f and x are the data in cosine space and physical space, respectively. DCT transformations of position and velocity data are shown in Figure 4.2. From the analysis, we summarize the properties of the MOFCOOL dataset (and some general MD simulation datasets):

1. For the velocity data, the spectrum has strong intensity in high frequency regime. On the contrary, for position data, the frequency spectrum has high intensity in low frequency regime.

2. In contrast to the datasets in literature [14, 15], the spectrum of position data has a long and flat tail. Thus, in the MOFCOOL dataset (and in many general MD simulation datasets), the frequency distribution is not sparse.

4.2 Piece-wise Polynomial based Predictors

In Hamiltonian mechanics, the positions and velocities are real values. There is always a quantization step when we perform numerical calculations and store them as finite pre-

cision numbers. MD simulation data may require a certain error threshold. There are several reports [14, 22] of MD simulation data compression based on this error threshold.

Under this error threshold setting, the compression problem can be reduced to a search problem: find the predictor function f such that the residual error is uniformly bounded by the threshold ϵ given by

$$\max_{i=1...n} |f(t_i) - q_i| \leq \epsilon \tag{20}$$

Ideally, function f has the following properties:

1. The function can be expressed by a small number of parameters, which is essential for decent compression performance.

2. The predictors are mathematically and computationally easy to compute.

3. The models are ideally interpretable and consistent with the physics nature.

Piece-wise polynomial functions can be fully parameterized by the coefficients of the polynomials, which is a good candidate. Mathematically, we iteratively search the polynomial functions that can fit the data points $q_1, ..., q_{i+1}$ with a uniform error threshold ϵ. If the parameter space is not empty, then we try $q_1, ..., q_{i+2}$. Otherwise, we save the current function for $q_1, ..., q_i$ (ending at timestep i) and start another function from q_{i+1}. Also, the end timesteps need to be stored. By solving the linear programming (LP) with incrementally added constraints, we can easily solve this problem and transform the position data from physical space to polynomial coefficient space. Algorithm 4.1 provides the details of the encoding and decoding algorithms. We use $a_{j,k}$ to denote the kth coefficient of jth polynomial function.

Algorithm 4.1 Piece-wise Polynomial Compression

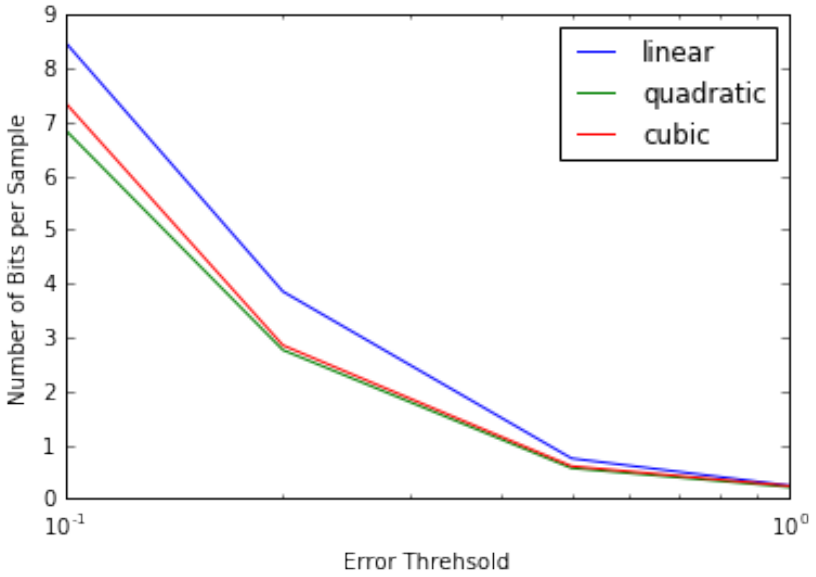
procedure ENCODING($q_1, ..., q_n, d, \epsilon$) ▷ d: order of polynomial
  Start with initial parameter space: $\mathcal{A}^0 = \mathbb{R}^{d+1}$
  Index of the first polynomial function: $j = 1$
  The ending timestep: $e_0 = 0$
  for $i = 1, ..., n$ **do**
    Add $|\sum_{k=0}^{d} a_{j,k}(i - e_{j-1})^k - q_i| \leq \epsilon$ as additional constraint $\forall \boldsymbol{a}_j \in \mathcal{A}^i \subseteq \mathcal{A}^{i-1}$
    if $\mathcal{A}^i$ is empty **then**
      Save an arbitrary predictor $\boldsymbol{a}_j^* \in \mathcal{A}^{i-1}$ at ending timestep $e_j = i - 1$
      Restart another function with index $j \leftarrow j + 1$
      Find $\mathcal{A}^i \subseteq \mathbb{R}^{d+1}$ such that $\forall \boldsymbol{a}_j \in \mathcal{A}^i$: $|\sum_{k=0}^{d} a_{j,k}(i - e_{j-1})^k - q_i| \leq \epsilon$
  Save the parameters of the final predictor $\boldsymbol{a}_m^*$ and $e_m = n$
  return predictors $\boldsymbol{a}_1^*, ..., \boldsymbol{a}_m^*$ and ending timesteps $e_1, ..., e_m$

procedure DECODING($\boldsymbol{a}_1^*, ..., \boldsymbol{a}_m^*, e_1, ..., e_m$)
  Start with the first predictor $\boldsymbol{a}_j^*$ with $j = 1$
  The valid time window for $\boldsymbol{a}_j^*$ is between e_{j-1} to e_j ($e_0 = 0$)
  for $i = 1, ..., n$ **do**
    if $e_j < i$ **then**
      Get the next predictor $\boldsymbol{a}_{j+1}^*$, $j \leftarrow j + 1$
    Reconstruct the data by $\hat{q}_i = \sum_{k=0}^{d} a_{j,k}^*(i - e_{j-1})^k$
  return reconstructed data $\hat{q}_1, ..., \hat{q}_n$

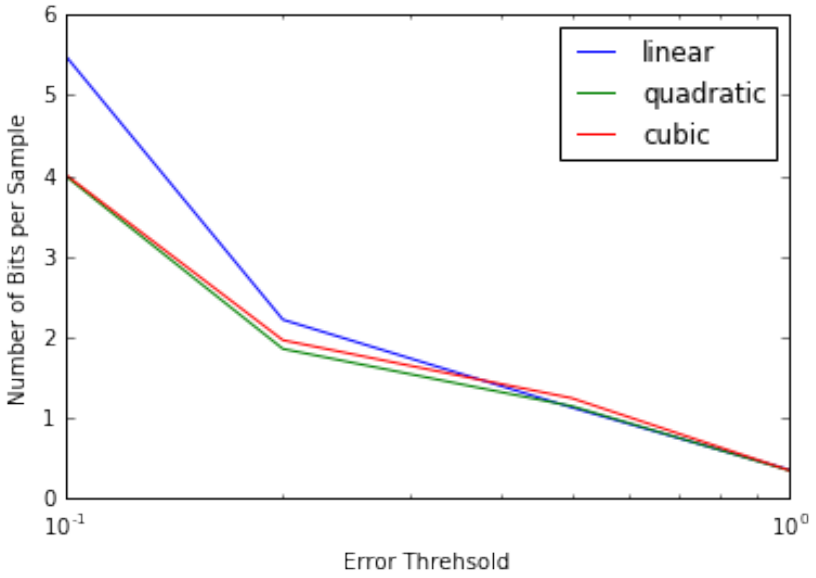
We test algorithm 4.1 on MOFCOOL and LIQCRY datasets. The results are plotted in Figure 4.3. We evaluate the compression performance by calculating the number of bits needed per sample (NBPS) at certain error threshold:

$$\text{NBPS} = \frac{32 \times N_{parameters}}{3 \times N_{particles}} \tag{21}$$

The factor 32 is based on the storage of coefficients using 32 bits single precision floating point number. The factor 3 is to normalize over 3 different directions.



MOFCOOL dataset



LIQCRY dataset

Figure 4.3 Number of bits per sample (NBPS) using piece-wise polynomial based compression algorithm (Algorithm 4.1) under different error thresholds. The algorithm is tested on both MOFCOOL and LIQCRY datasets.

In previous reports [22], the authors calculated the predictor from scratch at each timestep, which was computationally intensive. Also, they did not perform systematic

study on the compression performance with different polynomial order d. For example, Jan Huwald et al. proposed compression algorithms based on linear predictors where the first order approximations were calculated. However, based on Newton's 2^{nd} law, the second order terms also carry the information of the positions. In our algorithm, we extend the original linear functions to quadratic and cubic functions. By adding second and third order information, we get better compression performance. Interestingly, when the second order information is added, the compression performance is significantly improved (20%-23% NBPS decrease in some error thresholds). However, when the third order information is added, the performance is not improved significantly (the NBPS even increases in some cases). Another interesting observation is that when we increase the error threshold to 1, the performances are almost the same. There is a trade-off on choosing the order d of polynomial function. On one side, polynomial functions with lower order d can possibly underfit the data, which in turn requires more segments. However, a higher order polynomial function is prone to overfitting. Moreover, the time complexity and the number of parameters per function are also scaled with d. Therefore, the optimal polynomial order needs to be determined empirically. For these two datasets, piece-wise quadratic function ($d = 2$) achieves the best compression performance.

Piece-wise polynomial based compression algorithms are efficient and scalable. The encoding process solves the linear programming with incrementally added constraints. The total number of constraints is N_pT, where N_p is the number of particles and T is the number of timesteps. The total number of coefficients is $N_f(d+1)$, where N_f is the number of polynomial functions and d is the order. The complexity of the decoding process is $O(dNT)$. Moreover, this algorithm can be easily implemented in a parallel way, because it treats the trajectory of each single particle independently.

From the physics point of view, this algorithm is to the find polynomial function to approximate the position in each segment. Thus, we can easily approximate the velocity and acceleration in a continuous fashion (except the discontinuous points). The velocity v, acceleration a and the force F can be approximated respectively:

$$v = \dot{q}(t), a = \ddot{q}(t), F = m\ddot{q}(t) \tag{22}$$

where $q(t)$ is the piece-wise polynomial function.

4.3 MD Simulation Data Compression Using Deep Learning Models

In the previous section, we examine the performance of the piece-wise polynomial based compression algorithm. A more complex function class for compression is neural network. Recently, many research areas have shown interests in deep neural networks [23]. Vanilla neural network is composed of multi-layer perceptron (MLP) with nonlinear activation functions. There are several variants of deep neural networks, which are designed for certain applications. For example, convolutional neural networks (CNNs) are designed to capture the local correlations in image and video data [24, 25]. Recurrent neural networks (RNNs) are designed for sequential data modeling, such as time series modeling [26] and semantic analysis [27]. In the scope of this report, we choose autoencoder and long short term memory (LSTM) for MD simulation data compression task.

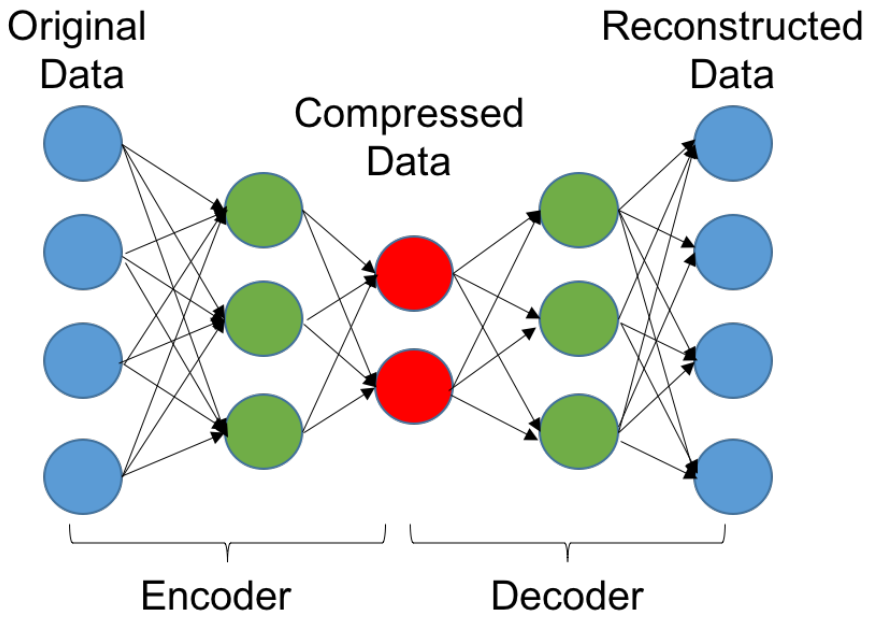


Figure 4.4 Architecture of autoencoder.

First, we hypothesize that the trajectory of atoms can be expressed in a lower dimension latent space. With this hypothesis, Ardita Shkurti et al. proposed the PCA based MD simulation data compression toolkits [15]. However, the MD simulation data is not linear. Here, we propose to apply the autoencoder as a learnable non-linear projection. Figure 4.4 shows the architecture of autoencoder [19]. The autoencoder has the same input and output size (blue nodes in Figure 4.4). The middle layer (red nodes in Figure 4.4) is the

compressed data. The encoder is to map the original data from $\mathbb{R}^d$ to lower dimension space $\mathbb{R}^{k_0}$. The decoder is to approximate the reverse mapping to reconstruct the data. The encoder and decoder are both neural networks. Mathematically, we are searching

$$w_1^*, w_2^* = \underset{w_1, w_2}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} ||g(f(x_i; w_1); w_2) - x_i||_1 \tag{23}$$

where f and g are encoder and decoder networks; w_1 and w_2 are their weights, respectively. These weights are optimized through back-propagation using first order optimization methods. It is uncommon to use maximum absolute error threshold as a constraint in deep learning model fitting. To quantitatively compare the compression performance of the autoencoder with the piece-wise polynomial predictors, we set the objective function to be mean absolute error as eq. 25.

Here, we perform experiments on the MOFCOOL and LIQCRY datasets. The structure of the autoencoder is the same as Figure 4.4. The input and output are both the trajectories of atoms in one direction, which has 6,000 dimension. In order to compare the results in these two datasets with the same input dimension, we divide the LIQCRY dataset into segments with size 6,000. Both the encoder and the decoder have a single hidden layer with constant size 1,000 (green nodes in Figure 4.4). Then, we tune the dimension of middle layer k_0 from 1 to 1,000. The mean absolute errors are calculated as a function of middle layer size (Figure 4.5).

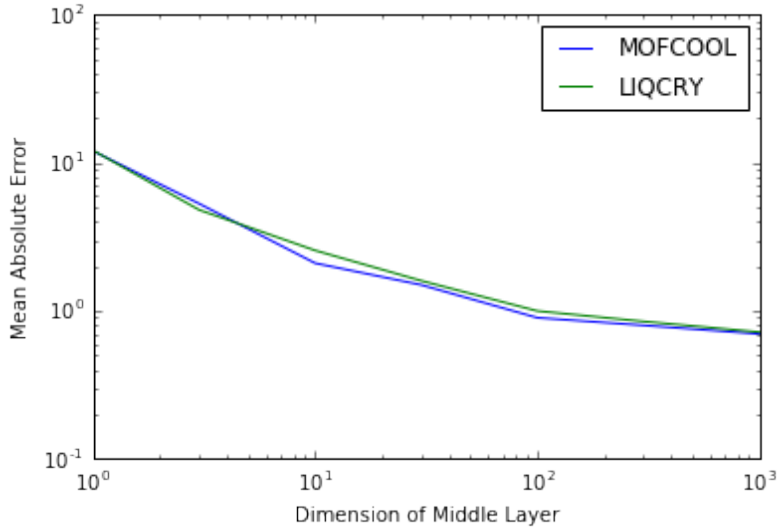


Figure 4.5 Mean absolute errors with different middle layer size.

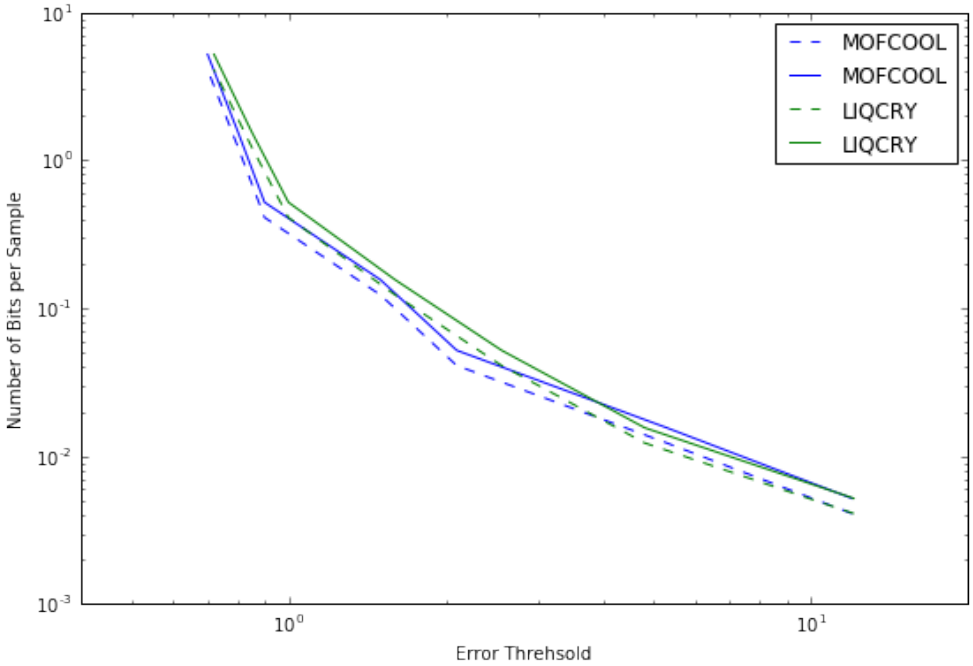


Figure 4.6 Compression performance of autoencoder based algorithm on MOFCOOL and LIQCRY datasets. The mean absolute errors are calculated at different compression rates. The dash lines are calculated only based on the storage of the compressed results in latent space, whereas the solid lines also take the storage of decoder weights into consideration.

As expected, by increasing the dimension of the latent space, the reconstruction error decreases. The compressed data in latent space and the weights of the decoder can be stored as single precision floating point numbers. Figure 4.6 shows the quantitative results of mean absolute errors at different compression rates. In comparison to the results of piece-wise polynomial models (Figure 4.3), the autoencoder based compression algorithm does not outperform the piece-wise polynomial based algorithm. The hypothesized reasons are

1. The autoencoder does not consider how the data follows Newton's 2^{nd} Law.
2. The overhead: the weights of the decoder needs to be saved. However, if a single autoencoder can be generalized to many datasets, the overhead can be ignored in comparison to the storage of the datasets.

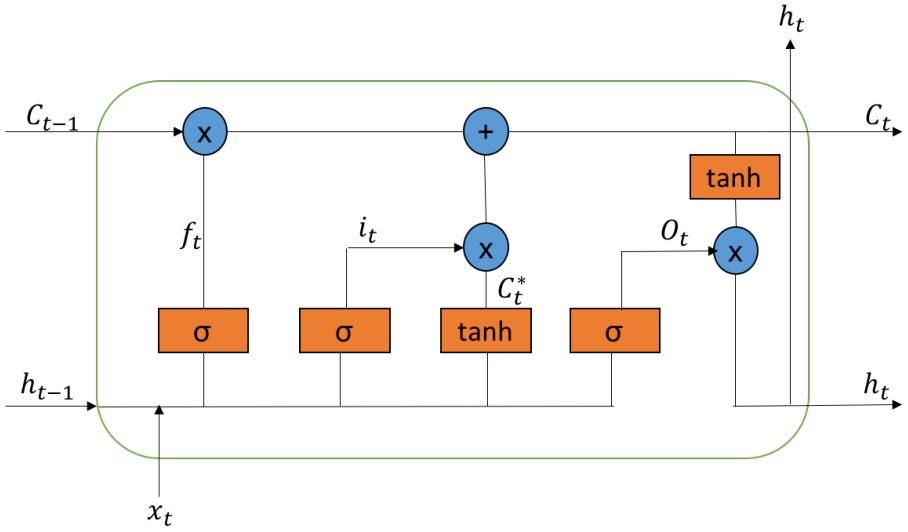


Figure 4.7 Architecture of LSTM. The σ is the sigmoid function. The x_t and h_t are input and output at time t, respectively. i_t, f_t and O_t are the input gate, forget gate and output gate, respectively.

As we discussed in the previous section, one of issues in the autoencoder model is that it does not explicitly model the time dependency in the trajectory. To address this issue, we use a recurrent neural network to model the trajectory in this section. A well-established example of recurrent neural network is the Long Short-Term Memory (LSTM) model [28]. The architecture of LSTM is shown in Figure 4.7. There are several modules in LSTM:

1. Inputs: 1) the input in the current timestep x_t and 2) the information flow from last timestep C_{t-1}, h_{t-1}.

2. Gating functions: the sigmoid functions as multipliers.

3. Outputs: 1) the output in current timestep h_t, 2) the information flow to next step h_t, C_t, 3) the final output of neural network.

Mathematically, the functions in Figure 4.7 are

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{24}$$

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{25}$$

$$C_t^* = \tanh(W_C * [h_{t-1}, x_t] + b_C) \tag{26}$$

$$C_t = f_t * C_{t-1} + i_t * C_t^* \tag{27}$$

$$O_t = \sigma(W_O * [h_{t-1}, x_t] + b_O) \tag{28}$$

$$h_t = O_t * \tanh(C_t) \tag{29}$$

where W are weight matrices and b are the bias vectors. By combining these modules, LSTM can get, filter and process the input of the current timestep (including the information passed from last timestep) and feed them into the next timestep.

We implement LSTM for the MD simulation data compression task (Algorithm 4.2). The basic idea is to generate prediction of the current position based on the position data in previous timesteps using LSTM. Then, the residual will be stored by quantization. In comparison to the algorithms we discussed in the previous sections, the residual error *must* be saved in order to recover the data in future timesteps.

Algorithm 4.2 LSTM For MD Data Compression

procedure ENCODE($q_1, ..., q_n, p$) ▷ p: the sequence length of LSTM model
    Train LSTM model to predict the current position using last p timesteps
    Store the full data from timestep 1 to timestep p: $q_1, ..., q_p$
    for $i = 1, ..., n - p$ **do**
        Predict $\hat{q}_{i+p}$ using LSTM model with the data in last p timesteps $q_i, ..., q_{i+p-1}$.
        Store the residual $r_{i+p} = q_{i+p} - \hat{q}_{i+p}$ by quantization and entropy encoding
    return $q_1, ..., q_p, r_{p+1}, ..., r_n$, LSTM model

procedure DECODE($q_1, ..., q_p, r_{p+1}, ..., r_n$, model)
    for $i = 1, ..., n - p$ **do**
        Estimated $\hat{q}_{i+p}$ from $q_i, ..., q_{i+p-1}$ using LSTM model
        Reconstruct the data $q_{i+p} = \hat{q}_{i+p} + r_{i+p}$.
    return Reconstructed data $q_1, ..., q_n$

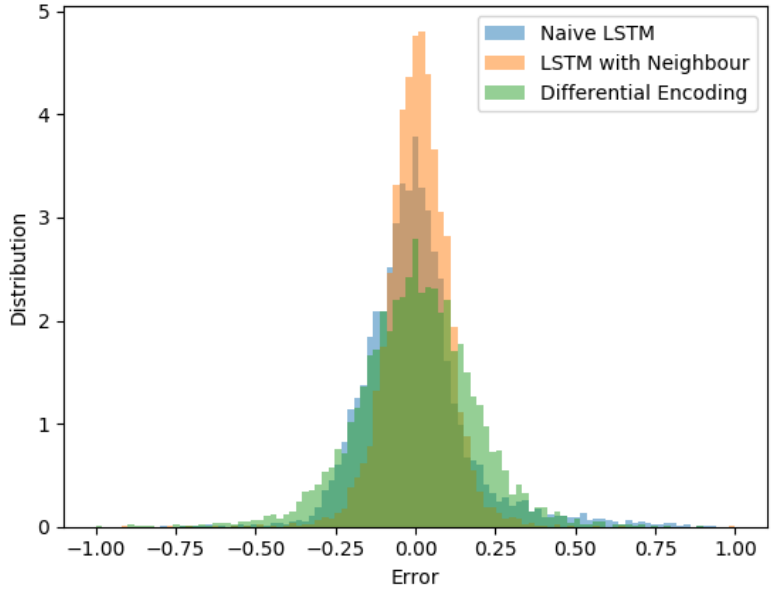


Figure 4.8 Distribution of residual error r in the MOFCOOL dataset.

In comparison to naive differential encoding (residual $r = q_i - q_{i-1}$), LSTM model gives a smaller residual error (result shown in Figure 4.8).

To further improve the LSTM model, we take the neighbour particles into consideration.

In the MOFCOOL and LIQCRY datasets, there exists a cutoff distance T in the potential energy function shown in Figure 4.9. An equivalent assumption is: there is no interaction between two particles when their distance is greater than the cutoff T.

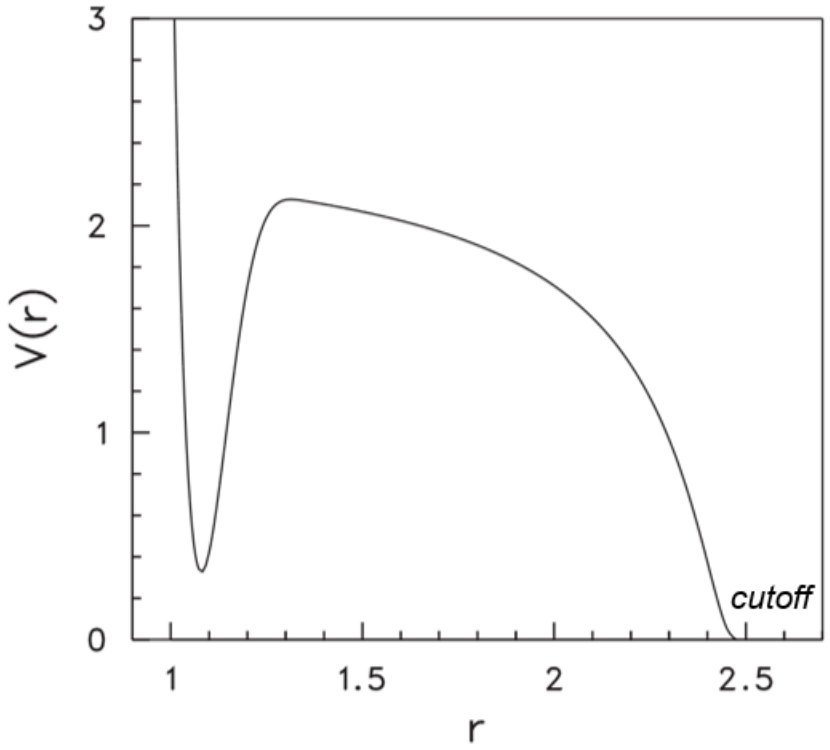


Figure 4.9 Potential energy $V(r)$ as a function of Euclidean distance r. The figure is reprinted Reprinted from The Journal of Chemical Physics, 141, Alfredo Metere et al., Formation of a new archetypal Metal-Organic Framework from a simple monatomic liquid, 234503, Copyright 2014, with permission from AIP Publishing LLC.

Based on the cutoff T, we can define the neighbour of atom i as:

$$\mathcal{N}_i = \{j : r_{ij} \leq T\} \tag{30}$$

Then, the potential energy can be simplified as

$$U(\boldsymbol{q}) = \sum_i \sum_{j \in \mathcal{N}_i, j>i} V(\boldsymbol{q}_i, \boldsymbol{q}_j) \tag{31}$$

Therefore, the positions of neighbour particles may carry the information of the potential energy function. We test the optimized LSTM model, which contains the coordinates

of 9-nearest neighbour atoms as extra input. With this extra information, the prediction performance is improved, which is evident by a narrower distribution of residual r (Figure 4.8). We also compare the performance of LSTM with the traditional time series models as a function of input sequence length shown in Figure 4.10.

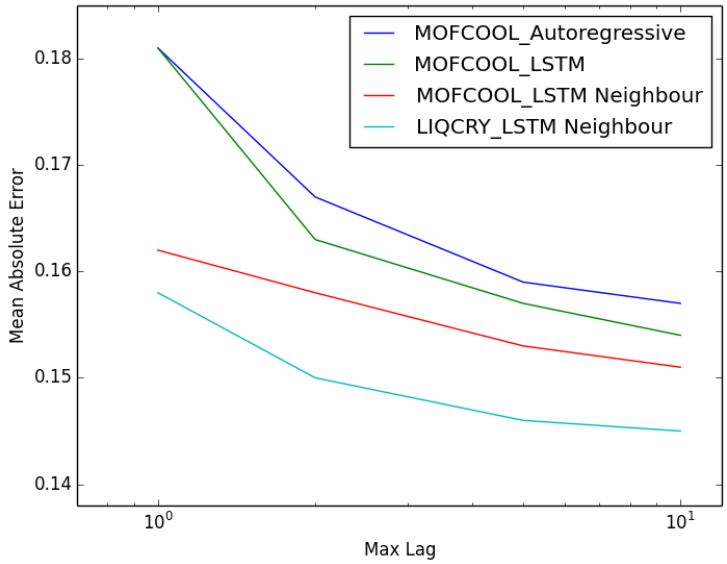


Figure 4.10 Mean absolute error as a function of input sequence length (lag) using different models.

From the result, increasing the sequence length does not improve the prediction performance significantly. This is consistent to the result of our basic time series modeling in section 4.1. In the time domain, the dependency can be mostly represented by the information in the last 5 timesteps.

To quantitatively evaluate the compression performance of algorithm 4.2, we quantize the residuals r to 2^n bins uniformly (in order to represent data using n bits). Then, the quantized data are passed through the Huffman encoder. The compression performance is shown in Figure 4.11. From the result, we clearly observe that the reconstruction error becomes very large when we quantize the data to a very few number of bits. The large error (10^3) occurs because the error propagates as a function of time. In this algorithm, the current position prediction depends on the noisy reconstructed data in previous timesteps. Thus, the reconstruction error in previous timesteps will propagate to current and future

timesteps. Since the trajectories in the LIQCRY dataset are longer, the propagation error is more significant. In comparison to the algorithms discussed in the previous sections, such as piece-wise polynomial based compression algorithm and autoencoder based compression algorithm, LSTM based compression algorithm usually requires more bits to store the data under the same mean absolute error.

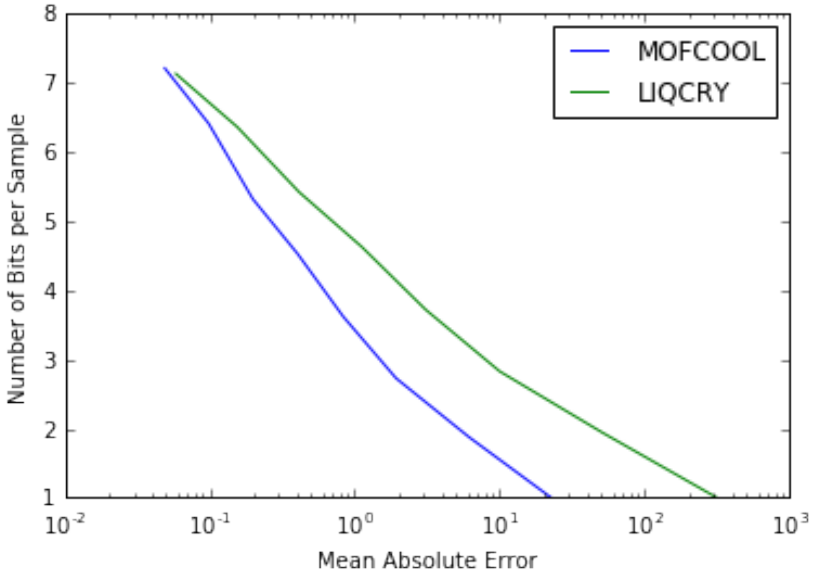


Figure 4.11 Number of bits needed per sample under different mean absolute error using LSTM based compression algorithm. The compression performance is evaluated on both MOFCOOL and LIQCRY datasets.

4.4 Summary

In this chapter, we implemented two types of models to compress the MD simulation data in time domain: piece-wise polynomial functions and deep learning based models. The piece-wise polynomial function is more consistent with Newton's law, which also gives better compression performance than deep learning based models. Using the piece-wise polynomial function-based compression algorithm, we can achieve 1 bit per sample compression performance with Angstrom level error.

5 Compression Algorithms in Space Domain

In chapter 4, we focus on MD simulation data compression in the time domain. In this chapter, we design the compression algorithms in the space domain: compression of a single frame. MD simulation datasets are similar to the video datasets: both of them are composed of frames at different timesteps. For video datasets, each frame is an image, which is a well-indexed data format. Most of the compression algorithms for images and videos are based on spatial correlation (with neighbour pixels). However, in the space domain, the MD simulation data is not well-ordered: there is no correlation between its index i and position q. The conventional multimedia compression algorithms cannot be directly applied to the MD simulation datasets. In this chapter, we start with a greedy approach to index the MD simulation data in the 3D space. Then, we apply a 3D-Paeth filter based compression algorithm as an example to illustrate how multimedia compression algorithms can be modified to compress MD simulation data.

5.1 Data Rearrangement

We propose a greedy algorithm to index the particles (Algorithm 5.1). We initialize a 3D cubic grid with size $d = \lceil \sqrt[3]{n} \rceil$ and assign the particles to their nearest grids. The distance metric is defined as the distance between the particle and the center of the cube. We also generate an indicator grid in case that some cubes are not filled.

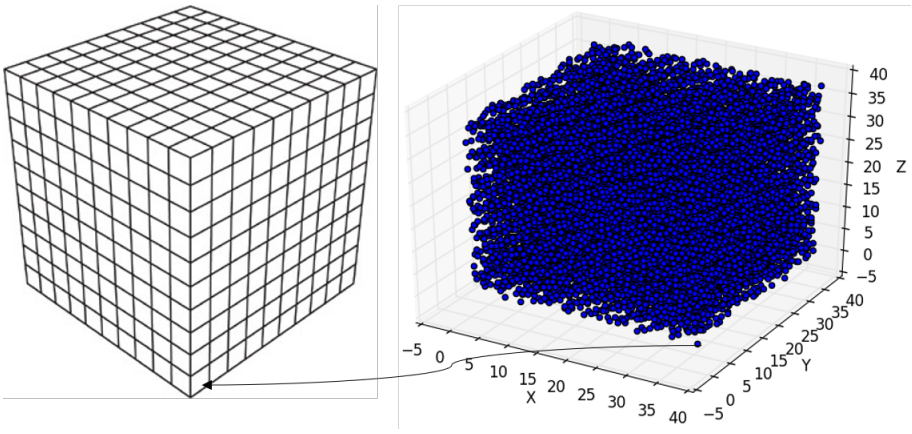


Figure 5.1 Schematic drawing of indexing algorithm. We greedily assign particles into nearest cubes.

Algorithm 5.1 Greedy Data Indexing

procedure INDEX($A_1, ..., A_n$) ▷ A: atom
  Initialize the 3D Grid $\mathcal{G}$ with bounding box size l_{bb}.
  A list $\mathcal{U}$ of particles that have been assigned (initially empty).
  $d = \lceil \sqrt[3]{n} \rceil, l = \frac{l_{bb}}{d}$
  for $i, j, k = 0, ..., d-1$ **do**
    Calculate the coordinate (x, y, z) of the center of the grid:

$$x = (i + \frac{1}{2})l, \quad y = (j + \frac{1}{2})l, \quad z = (k + \frac{1}{2})l \tag{32}$$

    Find the nearest atom A^* that is not in $\mathcal{U}$:

$$A^* = \underset{A \notin \mathcal{U}}{\operatorname{argmin}} (A_x - x)^2 + (A_y - y)^2 + (A_z - z)^2 \tag{33}$$

    where A_x, A_y, A_z are the x, y, z coordinate of particle A, respectively
    Put A^* in to the current grid position $\mathcal{G}_{ijk} = A^*$
    Append A^* to $\mathcal{U}$
  return $\mathcal{G}$

To visualize the result, we start the experiment by assigning the particles into a 2D grid, where only x, y coordinates are considered. The first frame in the MOFCOOL dataset is indexed and visualized in Figure 5.2. There are several observations:

1. In the x, y directions, the greedy algorithm shows promising indexing result. The indices of the atoms in the x, y directions are approximately linear with their positions q_x, q_y. The artifacts (misindexed samples) only occur on the edges.

2. The velocity does not have correlation with its coordinate. The distribution of velocity is still random.

3. On the unindexed (z) direction, the positions are still random.

To avoid the third issue, in the following sections, the data are indexed in the 3D grid. After greedy data indexing, many multimedia and machine learning algorithms can be applied to the MD simulation dataset. For example,

1. By the spatial locality, multimedia compression algorithms, such as Paeth filter based compression algorithm, can be implemented for the MD simulation data compression.

2. The neighbours of an atom can be easily tracked.

3. Machine learning algorithms on the structured 3D grids, such as 3D convolutional neural networks, can be applied to the MD simulation datasets.

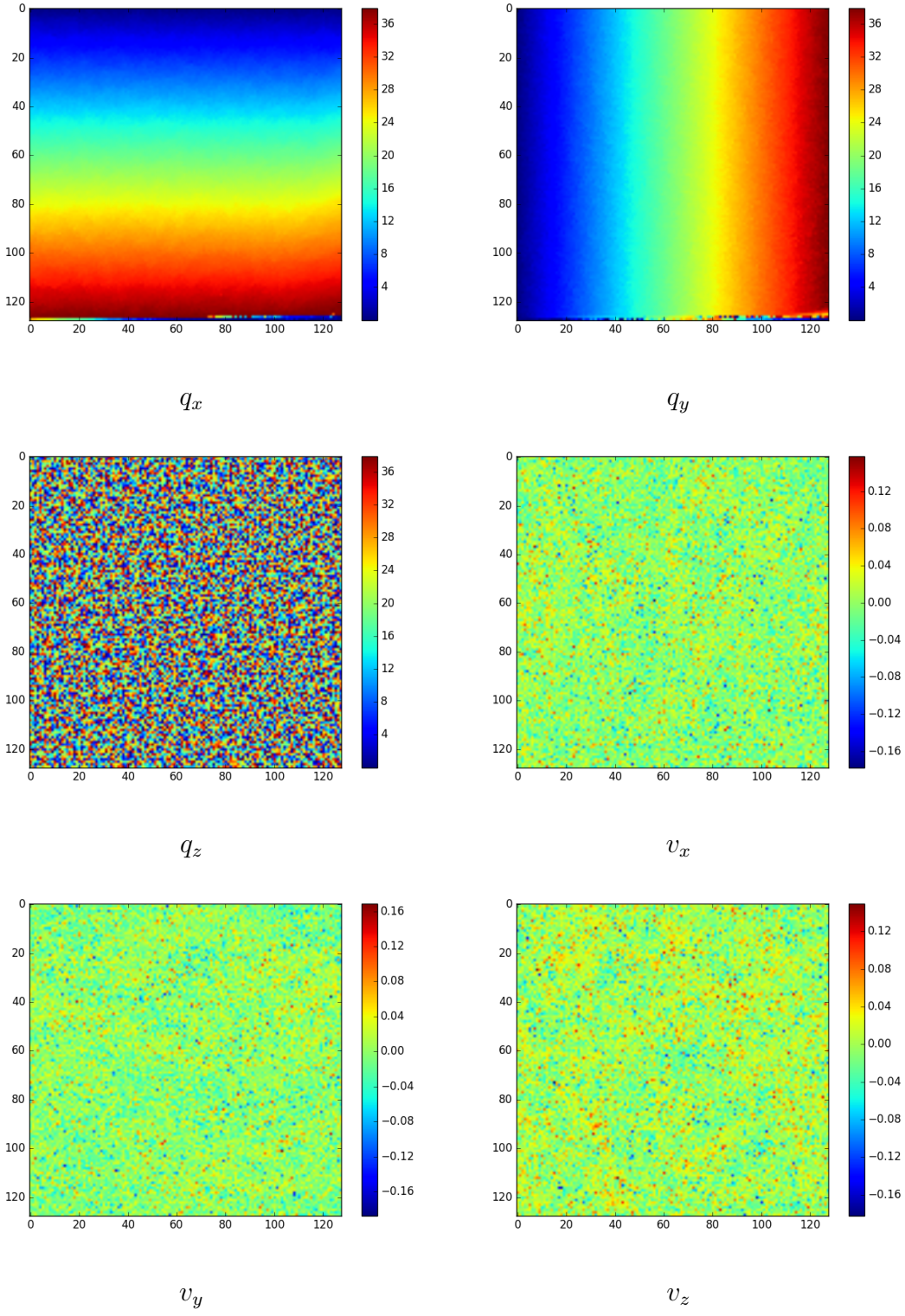


Figure 5.2 Visualizations of position and velocity data after greedy indexing on the 2D grid. For this indexing task, only x and y coordinates are considered.

5.2 Paeth Filter and Quantization

The Paeth filter [29] based portable network graphics (PNG) compression algorithm is a multimedia compression algorithm designed for image. In the indexed MD simulation dataset, the position $\boldsymbol{q}$ (a vector that has x, y, z components) and its index i, j, k are correlated. We extend the Paeth filter into 3D case. The set of neighbour $\mathcal{N}$ here is defined as $\{\boldsymbol{q}_{i-1,j,k}, \boldsymbol{q}_{i,j-1,k}, \boldsymbol{q}_{i,j,k-1}\}$. The boundary is padded with 0.

First, we define an average value $\bar{\boldsymbol{q}}$. There are multiple ways to define the average value in 3D. For example, Schmitt et al. did a grid search to find an optimal combination [30]. Here, we use one simple combination:

$$\bar{\boldsymbol{q}} = \frac{2}{3}[\boldsymbol{q}_{i-1,j,k} + \boldsymbol{q}_{i,j-1,k} + \boldsymbol{q}_{i,j,k-1}] - \boldsymbol{q}_{i-1,j-1,k-1} \tag{34}$$

Then, in the neighbour $\mathcal{N}$, we find the value $\tilde{\boldsymbol{q}}$ that is closest to $\bar{\boldsymbol{q}}$:

$$\tilde{\boldsymbol{q}} = \underset{\boldsymbol{q} \in \mathcal{N}}{\operatorname{argmin}} ||\boldsymbol{q} - \bar{\boldsymbol{q}}||_1 \tag{35}$$

Finally, $\tilde{\boldsymbol{q}}$ is subtracted from the original data $\boldsymbol{q}_{i,j,k}$. The residual $\hat{\boldsymbol{q}}_{i,j,k}$ is saved:

$$\hat{\boldsymbol{q}}_{i,j,k} = \boldsymbol{q}_{i,j,k} - \tilde{\boldsymbol{q}} \tag{36}$$

We apply this process to the indexed MD simulation dataset by looping over indices i, j, k. To recover the data, we reverse this process in the same order: find the neighbour $\tilde{\boldsymbol{q}}$ that is closest to $\bar{\boldsymbol{q}}$ and add it back to the residual $\hat{\boldsymbol{q}}_{i,j,k}$:

$$\boldsymbol{q}_{i,j,k} = \hat{\boldsymbol{q}}_{i,j,k} + \tilde{\boldsymbol{q}} \tag{37}$$

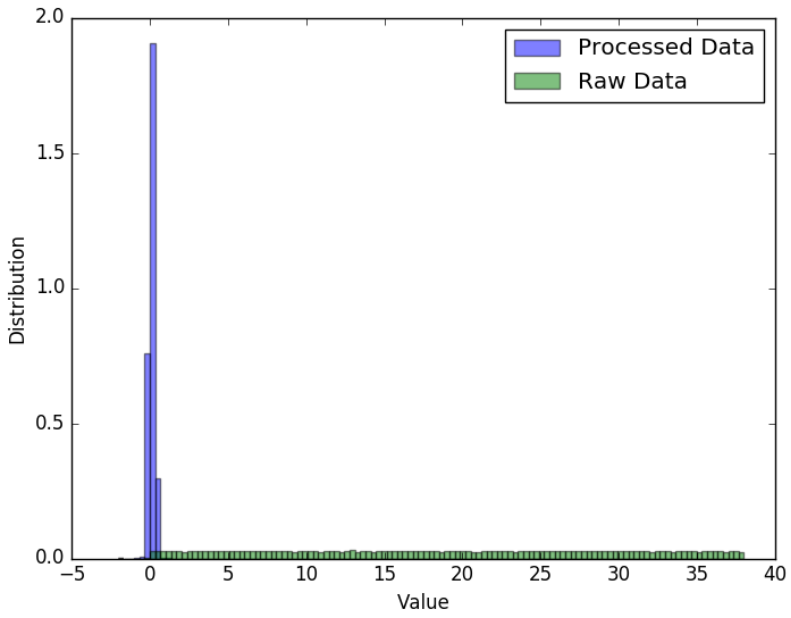
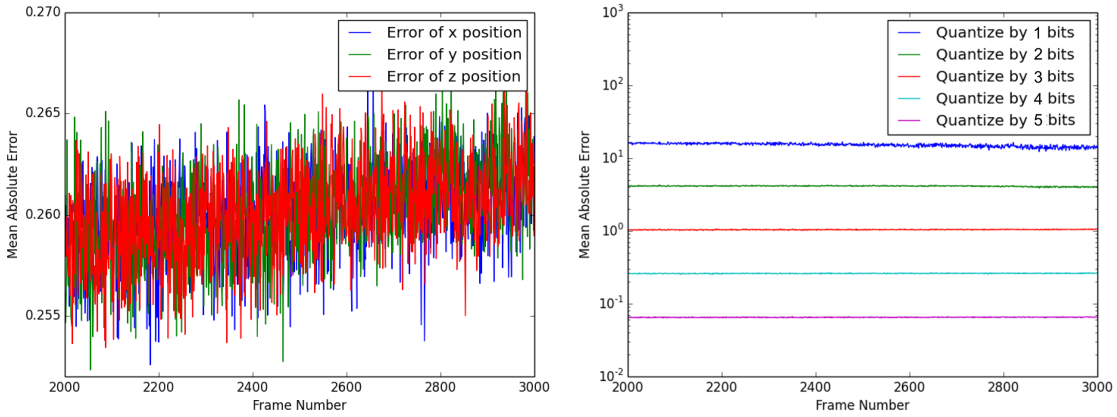


Figure 5.3 Distribution of raw data q and the residual $\hat{q}$ in MOFCOOL dataset.



(a) Error on different directions (b) Error with different quantizations

Figure 5.4 Mean absolute error under different conditions: (a) on different directions by quantizing the data into 4 bits (2^4 bins), (b) quantized by different number of bits and average over 3 directions. The errors are calculated based on timestep 2000-3000 in the MOFCOOL dataset.

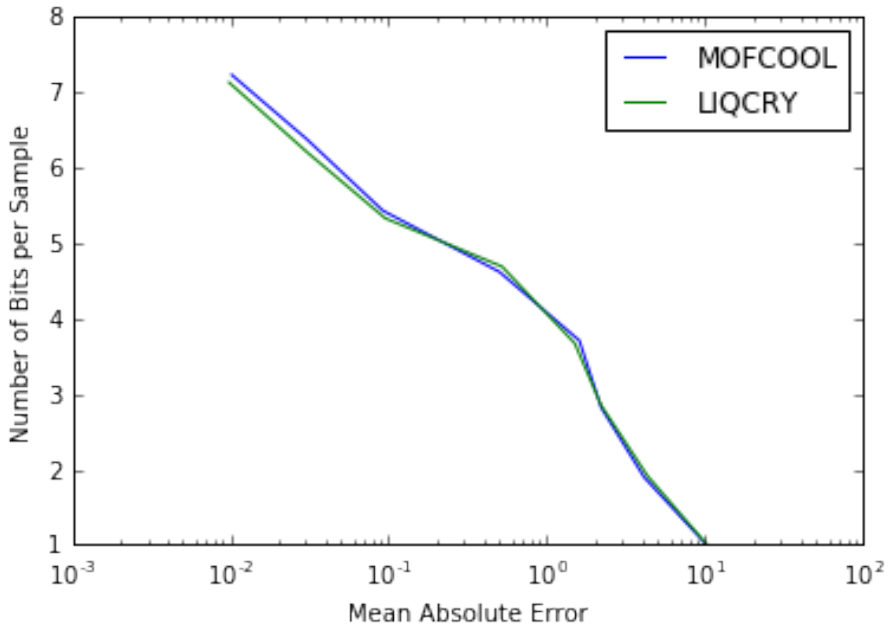


Figure 5.5 Number of bits per sample under different mean absolute errors. We evaluate the compression performance of Paeth filter based compression algorithm on *indexed* MOFCOOL and LIQCRY datasets.

We first index all the particles in the MOFCOOL and LIQCRY datasets following Algorithm 5.1. Then, we apply the 3D-Paeth filter based compression algorithms on the indexed datasets. Figure 5.3 shows the distribution of the residuals $\hat{\boldsymbol{q}}_{i,j,k}$ and the raw data $\boldsymbol{q}_{i,j,k}$ (the vectors are flatted to 1D scalars). The residuals have a narrower distribution (within boundary [-1, 2]) in comparison to the raw data. This indicates that the residuals could be saved using fewer bits under the same quantization precision. We quantize the data uniformly between [-1, 2] with 2^n bins (in order to represent them using n bits). We further apply entropy encoding on the quantized data through the Huffman encoder. Figure 5.4 shows the reconstruction errors of the 3D-Paeth filter based compression algorithm on different directions, and under different quantizations in timestep 2000-3000. 3D-Paeth filter based compression algorithm achieves similar performance along different directions, which overcomes the drawbacks in 2D-based indexing method (the randomness of q_z in Figure 5.2). We evaluate the algorithm by calculating the mean absolute error under different compression rates. The results are shown in Figure 5.5. Using data indexing and the 3D-Paeth filter, we can compress the data using only 1/8 of original storage with the mean absolute error at 0.01 Angstrom level.

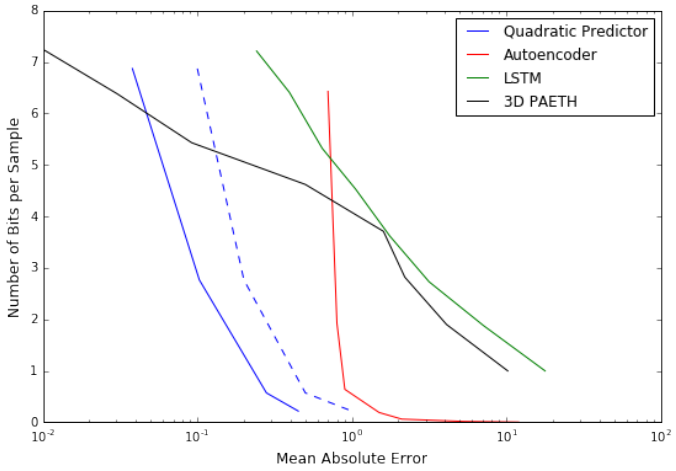
5.3 Summary

In this chapter, we design a greedy indexing method for the MD simulation datasets. The indexing method provides a solution to order the MD simulation data based on their position values. Then we compress the indexed data using the 3D-Paeth filter based compression algorithm. Multimedia compression algorithms can take advantage of the spatial correlation in the indexed data, which achieves promising compression performance.

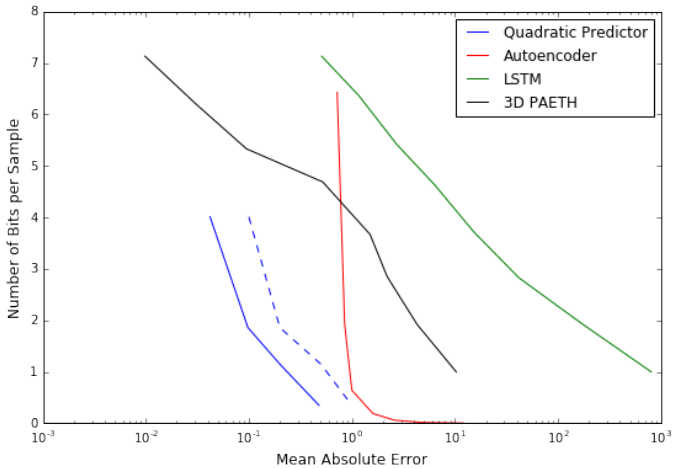
6 Comparison of Compression Algorithms

In this chapter, we will make a comparison of the compression algorithms we investigated in Chapter 4 and Chapter 5.

6.1 Comparison of Different Compression Algorithms



MOFCOOL Dataset



LIQCRY Dataset

Figure 6.1 Compression performance of different compression algorithms on the MOFCOOL and LIQCRY datasets. We evaluate the algorithms by calculating the number of bits per sample (NBPS) under different mean absolute error. The dash line is the error threshold for the quadratic predictor (details discussed in chapter 4.2).

We plot the results in Figure 6.1. Here are several remarks:

1. The piece-wise polynomial function-based algorithm can compress the data to less than 1 bit per sample, with Angstrom level absolute error. However, LSTM and Paeth filter based compression algorithms cannot break 1-bit limit, because at least one bit is required to represent each residual.

2. Quadratic predictor provides good compression performance in a wide range of mean absolute error.

3. Paeth filter based compression algorithm provides the best compression performance in the low error regime.

Table 6.1 Summary of Different Compression Algorithms

Algorithm	Principle	Pros
Piece-wise Polynomial	Linear Programming	Low NBPS
Autoencoder	Neural Network	Easy to Implement
LSTM	Neural Network	Small Model
3D-PAETH	Spatial Correlation	Fast, High Accuracy

Table 6.1 gives a brief summary of different compression algorithms. One can choose the suitable algorithm to achieve good storage-precision balance. In detail:

1. The piece-wise polynomial based compression algorithm transforms the data from physical space (positions) to polynomial space (coefficients of polynomials). The highlight of this algorithm is that it can break 1-bit per sample limit. This algorithm takes the assumption that the trajectory segments can be approximated by polynomial functions, which is consistent with the underlying physics: the prediction step in MD simulation is quadratic (Newton's equation). Mathematically, the coefficients are calculated by formulating linear programming with incremental constraints. However, the time complexity of linear programming cannot be ignored. Another limitation is that, this work is based on a manually determined error threshold (L^∞ norm of the error) ϵ. This setting is reasonable when certain precision is required in specific MD simulation dataset, which has been studied in many previous literature (details in Chapter 4.2). However, in most of the lossy compression settings (for other data formats such as image), the error is usually based on the average over large number of samples, rather than a hard threshold for all the samples.

2. Some deep learning methods can be modified for MD simulation data compression. Deep neural networks can express and fit nonlinear functions, which can possibly capture the dependencies and correlations in the time domain. Moreover, these models can be implemented easily using the deep learning packages, such as Tensorflow and Pytorch. There are also several disadvantages of the deep learning based approaches. First, the optimization of neural network is a non-convex problem and thus the performance is not guaranteed. Second, the decoder is not free: the weights of the neural networks need to be stored to reconstruct the data, which is an storage overhead.

3. The 3D-Paeth filter based compression algorithm utilizes the spatial locality of the indexed data *explicitly*: the residual is calculated as the difference with its neighbour's value. The advantage of this algorithm is that the arithmetic calculation is simple. The disadvantage is that it requires data preprocessing, which is an overhead.

4. In LSTM and 3D Paeth filter based compression algorithms, the *residuals* must be saved. On the contrary, in piece-wise polynomial based compression algorithm, the *coefficients* must be saved.

6.2 Future Directions

There are some possible improvements on these compression algorithms:

1. The encoding process of the piece-wise polynomial based compression algorithm is solved by linear programming with incremental constraints. The time cost is considerable when the dataset is large. One possible future direction is to parallel this algorithm and make it scalable.

2. For deep learning based methods, if the neural networks can be generalized to many different datasets, the overhead of saving the decoder can be ignored. In the future, different neural network architectures and numerical tricks can be tested to make the neural network more generalizable.

3. In recent years, several neural network architectures are proposed for the point clouds, such as pointCNN [31]. These algorithms can possibly improve the current compression and analysis methods in the space domain.

4. In LSTM and 3D Paeth filter based compression algorithms, we implement the naive quantization with fixed step size. In the future, the quantization method can be optimized to achieve better compression performance.

6.3 Summary

In this chapter, we make comparisons of different compression algorithms based on analysis of the empirical results. Among these algorithms, piece-wise quadratic function provides the best performance on the MOFCOOL and LIQCRY datasets. Moreover, we provide future outlooks on the possible improvements of these compression algorithms. We expect that the compression performance can be further optimized by exploring these possible directions.

7 Compression Performance and Underlying Physics

Compression performance depends on both the efficiency of compression algorithm and the underlying physics of the MD simulation system. In chapter 6, we discussed the performance of different compression algorithms. In this chapter, we will discuss the relationship of compression performance and underlying physics. As we discussed in chapter 3, there are phase transitions in both MOFCOOL and LIQCRY datasets. We will illustrate how the phase behavior affects the compression performance.

7.1 Study I: Performance of Piece-wise Polynomial Based Compression Algorithm

In this section, we study how the underlying phase behavior affects the compression performance of piece-wise polynomial based algorithms.

7.1.1 What is a Phase Transition?

The concept of "phase transition" is originally from physics, which describes the process of phases of matter changing from one to another,, such as solid phase to liquid phase, or liquid phase to gas phase. Theoretically, phase transitions happen when the derivatives of the partition function have discontinuities or divergences [32]. Phase transitions can be observed either experimentally or computationally by measuring and calculating physical parameters [33]. In terms of signal processing, a "phase transition" is generalized to many physical and social systems. For example, Pin-Yu Chen et al. discussed about the phase transition of community detectability [34, 35] as a function of edge connection probability. Applied to physics, Akinori Tanaka et al. proposed to implement neural networks to detect the phase transition in the Ising model [36]. We show two examples of phase transitions in Figure 7.1. Moreover, as described in chapter 3, first order phase transitions are present in both the MOFCOOL and LIQCRY datasets. In this section, we illustrate how the compression performance is related to the underlying first-order phase transition.

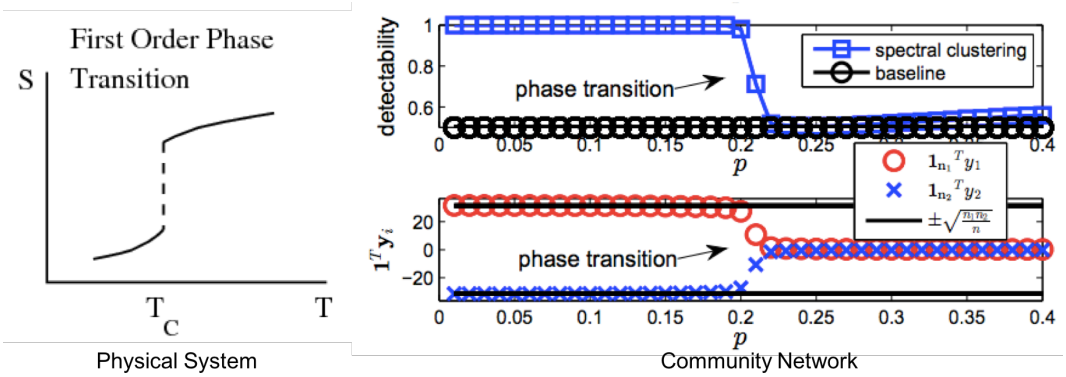


Figure 7.1 Phase transitions in physics and signal processing. In the physical system, the discontinuity of entropy presents during the first order phase transition. In the community detection system, there is a drastic drop of detectability as a function of edge connection probabilities, which is also described as "phase transition" in literature [35]. The right figure is from literature [35] copyright © 2015 IEEE.

7.1.2 Experimental result

Here, we provide an empirical result to show how the compression performance and the underlying physics are related. In order to investigate the compression performance before and after the phase transition, we divide the MOFCOOL dataset into 6 batches. Each batch has 1,000 timesteps. In the first batch (first 1,000 timesteps), the system is in the liquid phase. The phase transition happens in the second batch (timestep 1,000-2,000). In all the other batches, the system is in the solid phase. The piece-wise polynomial based compression algorithm is tested on each batch and the results are shown in Figure 7.2.

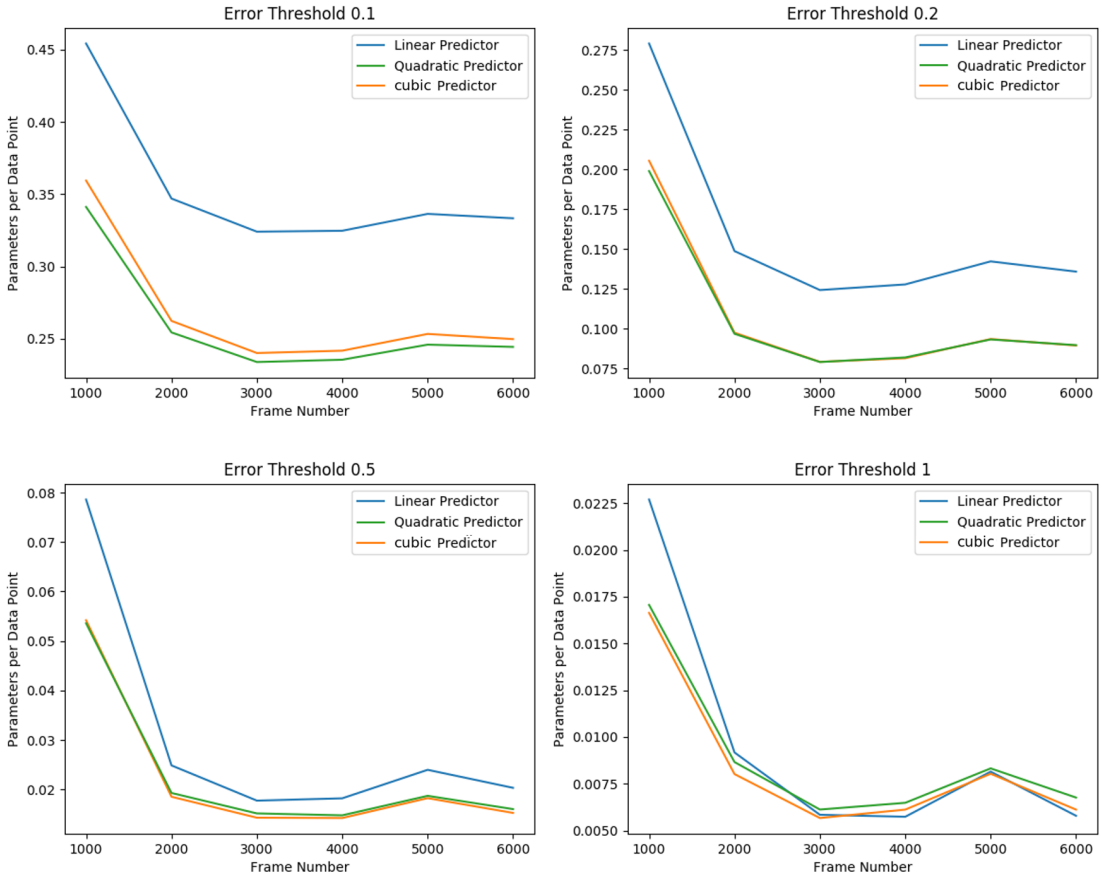


Figure 7.2 Number of coefficients per sample using piece-wise polynomial based compression algorithms on the MOFCOOL dataset. The data is divided to batches with size 1,000. The result is calculated on each batch.

Empirically under the same error threshold, more coefficients are required when the underlying system is in the liquid phase (first 1000 frames). This example proves that the compression performance is strongly correlated with the underlying physics. In the piece-wise polynomial based compression algorithm, the number of coefficients is determined by the number of segments (details in Chapter 4.2). Intuitively, if the first order and second order derivatives (which are corresponding to velocity and acceleration) are large and changing drastically, the trajectory will be difficult to be approximated by simple functions, and in turn more segments are required. Physically, these dynamics can be related to the energy transfer rate in the MD simulation system. In the future, we will develop theory to estimate this energy transfer rate to study the relationship of compression performance and underlying physics quantitatively. An outlook of this theory will be

provided in chapter 8.

7.2 Study II: Information Entropy of Quantized Data

7.2.1 Gibbs Entropy and Shannon Entropy

There are many good resources discussing the relationship of Gibbs entropy and Shannon entropy [32, 37]. Here, we only provide a short discussion. In physics, the Gibbs entropy is

$$S_G = -k_B \sum_{i=1}^{n} p_i \log p_i \tag{38}$$

where p_i is the probability that state i occurs in the energy fluctuation, and k_B is Boltzmann constant. In information theory, Shannon [38] also defined the information entropy H.

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{39}$$

where p is the probability mass function of the discrete distribution. The formula of Shannon entropy and Gibbs entropy are closely related. The differences are:

1. In physics, the definition of Gibbs entropy contains Boltzmann constant. Thus, its unit is $J \cdot K^{-1}$. In information theory, Shannon entropy is usually measured by bit as unit.

2. The bases are different: Gibbs entropy takes the natural log, whereas Shannon entropy takes $\log_2$ (to be measured as bits in binary coding).

There exists a bijection between Gibbs entropy and Shannon entropy, since two definitions differ by a constant multiplier.

7.2.2 Experimental result

On one hand, the entropy of a physical system is discontinuous during the first order phase transition [39]. On the other hand, Shannon entropy provides the lower bound of entropy based compression. Therefore, the compression performance and the phase behavior should be related. Here, we calculate the Shannon entropy of the quantized velocity data:

1. In the real world, the velocities are real numbers. We first quantize the velocity data using n bits by assigning the velocity v into $m = 2^n$ bins uniformly in the interval $[v_{min}, v_{max}]$.

2. The index is calculated by

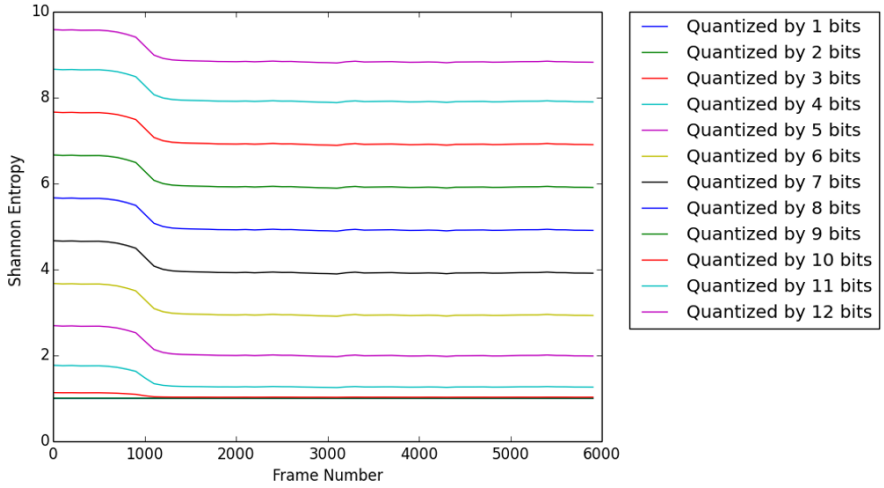
$$Index = [\frac{(v - v_{min})2^n}{(v_{max} - v_{min})}] \tag{40}$$

   where n is the number of bits to represent the data. Then, the empirical distribution $\hat{p}$ is calculated by normalization.

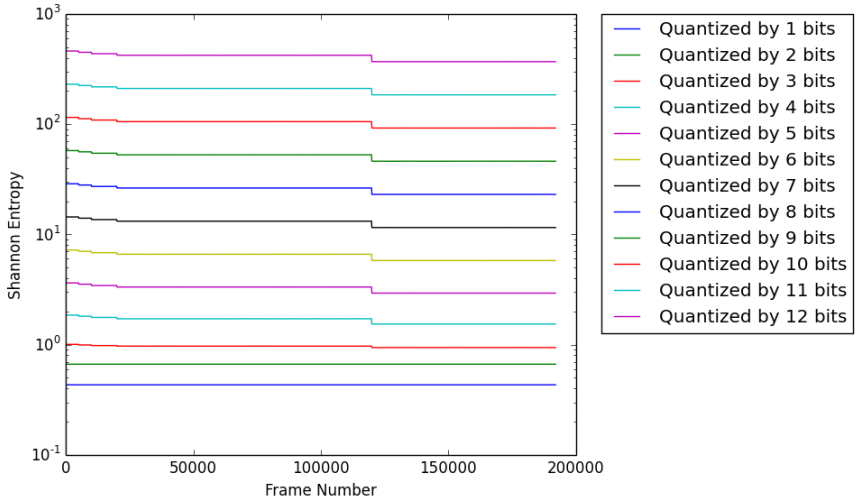
3. Calculate the Shannon entropy H of empirical distribution $\hat{p}$.

$$H(\hat{p}) = -\sum_{i=1}^{m} \hat{p}_i \log_2 \hat{p}_i \tag{41}$$

In these two datasets, we have $v_{min} = -2$ and $v_{max} = 2$, which can cover all the velocity values. From a physics viewpoint, this estimate is also related to the kinetic energy (because it measures the spreadness of the velocity data). The entropy calculation under different timesteps are shown in Figure 7.3.



MOFCOOL Dataset



LIQCRY Dataset

Figure 7.3 Shannon entropy of quantized data under different quantizations.

From the experimental result, we observe that the Shannon entropy drops drastically during the phase transition. In the MOFCOOL dataset, the Shannon entropy drops between frame 1000 and frame 1500, which is related to the phase transition from the liquid phase to the solid phase. In the LIQCRY dataset, the Shannon entropy drops between frame 120,000 and frame 125,000, which can be attributed to the phase transition from the liquid phase to the liquid crystal phase. This trend is generally consistent over dif-

ferent numbers of quantization bins, except when the data are quantized and represented by less than 3 bits. This Shannon entropy change can be also applied to detect the phase transition during the MD simulation as an on-fly analysis. In the future, we will develop a more general phase transition detection framework. The outlook will be presented in chapter 8.

7.3 Summary

In some previous studies, such as [14] and chapter 4-5 in this report, the compression performance was evaluated by a limited number of datasets due to the limitation of computational power, since the MD simulation datasets are usually very large. Thus, some of the previous studies are agnostic to the underlying physical systems. In this chapter, we provide two experimental examples to illustrate that the compression performance and the underlying physics are closely related. In the future work, we will focus more on the study of this relation by developing theoretical tools. Also, we aim to develop a phase transition detection framework using these tools.

8 Conclusion and Outlook

In this report, we design several compression algorithms for MD simulation datasets. We also provide an empirical analysis to understand the relationship between the compression performance and the underlying physical behavior. In this chapter, we summarize the results in the previous chapters and give an outlook to the future.

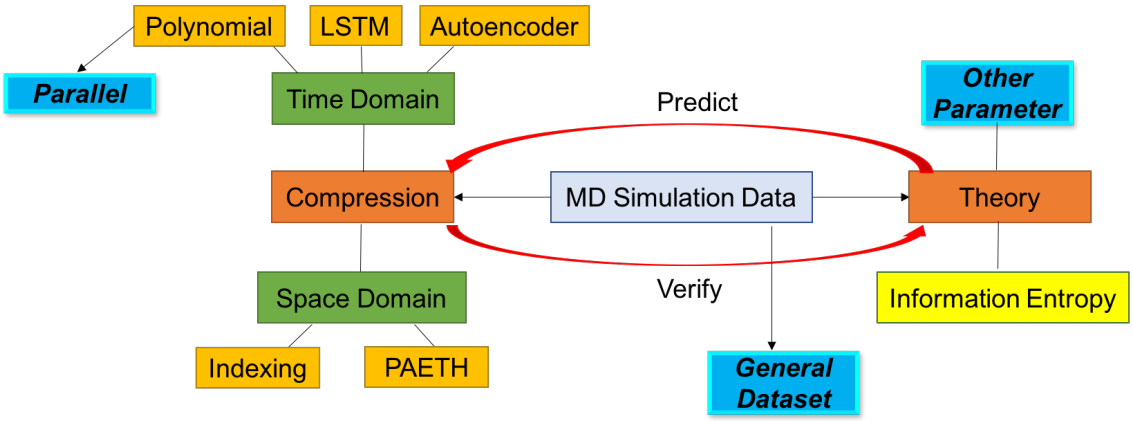


Figure 8.1 Summary of the report. This report provides an empirical analysis of the compression algorithms. We also give a brief discussion from the information theory viewpoint by calculating the information entropy in different physical phases. Some future directions are given in the blue rectangles.

8.1 Conclusions

In this report, we develop the compression algorithms and analyze their performances empirically (Figure 8.1). In the time domain, we develop piece-wise polynomial based and deep learning based compression algorithms. Piece-wise polynomial based compression algorithms give better compression performance. Using piece-wise quadratic predictor, we can achieve 1 bit per sample compression performance with the error at Angstrom level.

In the space domain, we first provide a greedy approach to index the position data. After the preprocessing, we apply the Paeth filter based compression algorithm as an example to illustrate how the multimedia compression algorithms can be utilized on the MD simulation datasets. This method gives the best performance in small absolute error regime.

We can obtain 0.01 Angstrom level error with using about 1/10 to 1/8 of the original storage.

The compression performance is dependent on both the algorithm efficiency and the underlying physics. In chapter 6, we make a systematical comparison of different compression algorithms. In chapter 7, we calculate and compare 1) the Shannon entropy of quantized velocity data and 2) the empirical compression performance of piece-wise polynomial predictors in different physical phases, such as solid, liquid and liquid crystal phases. This study shows that the underlying physics takes an important role on the compression performance. In the future, we will build theoretical tools to study this relationship quantitatively.

8.2 Future Outlook

In chapter 6, we summarized the future directions of compression algorithms. Here, we give an outlook to the ongoing and future works on the theoretical side.

In chapter 7, we observe that the Shannon entropy drastically changes during the phase transition. Based on the relationship of Shannon entropy and Gibbs entropy, we aim to build a general phase transition detection framework. Other than Shannon entropy, we hope to find other parameters to describe the underlying physics quantitatively and to detect the phase transition. For example, we aim to estimate the energy transfer rate between the kinetic energy and the potential energy in the MD simulation systems.

The Hamiltonian in some MD systems can be formulated as eq. 1 in chapter 1.1. The energy transfer rate between the kinetic energy $K(\boldsymbol{p})$ and the potential energy $U(\boldsymbol{q})$ is calculated by

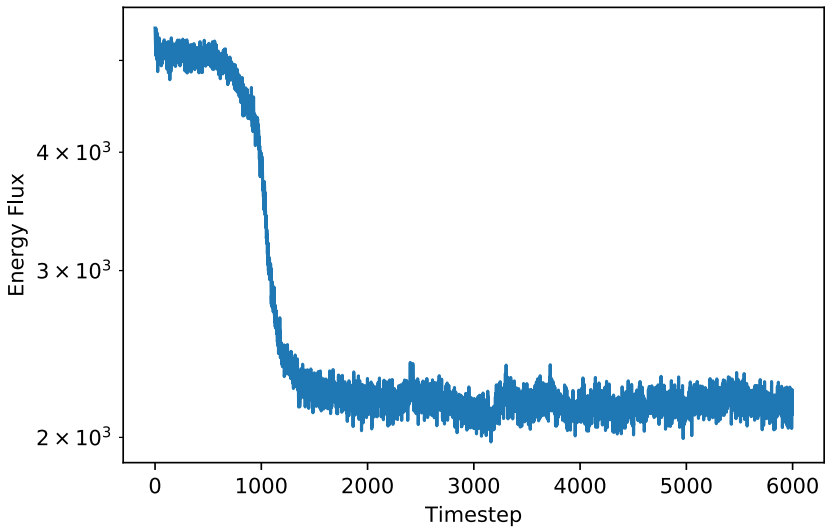
$$\frac{dK(\boldsymbol{p})}{dt} = \dot{\boldsymbol{p}} \cdot \frac{\boldsymbol{p}}{m} = \dot{\boldsymbol{p}} \cdot \dot{\boldsymbol{q}} \tag{42}$$

$$\frac{dU(\boldsymbol{q})}{dt} = \dot{\boldsymbol{q}} \cdot \frac{dU(\boldsymbol{q})}{d\boldsymbol{q}} = -\dot{\boldsymbol{q}} \cdot \boldsymbol{F} = -\dot{\boldsymbol{q}} \cdot \dot{\boldsymbol{p}} \tag{43}$$

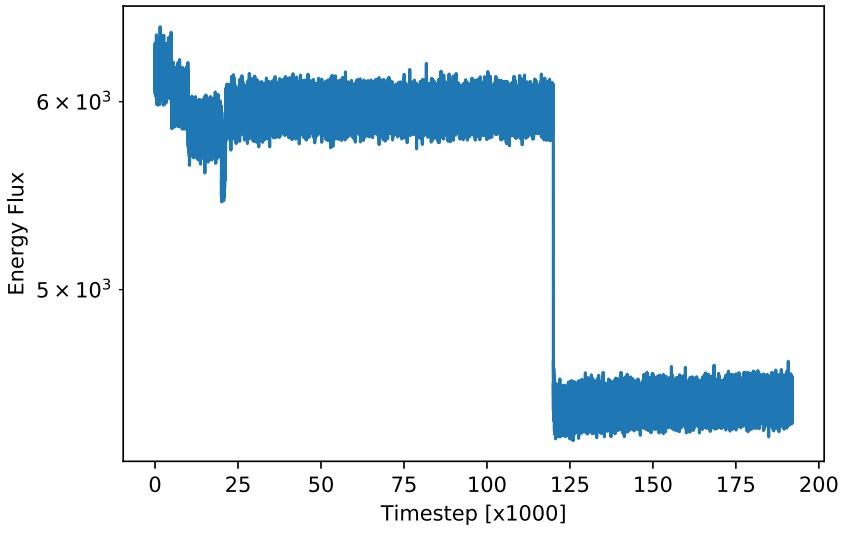
Here, we define $|\dot{\boldsymbol{p}} \cdot \dot{\boldsymbol{q}}|$ as the energy transfer rate S

$$S = |\dot{\boldsymbol{p}} \cdot \dot{\boldsymbol{q}}| = |\frac{d\boldsymbol{q}}{dt} \cdot \frac{d\boldsymbol{p}}{dt}| = |\boldsymbol{F} \cdot \boldsymbol{v}| \tag{44}$$

We take the absolute since the transfer rate *value* is calculated. The energy transfer rate of MOFCOOL and LIQCRY datasets at different timesteps are calculated and shown in Figure 8.2.



MOFCOOL Dataset



LIQCRY Dataset

Figure 8.2 Energy transfer rate between kinetic and potential energy at different timesteps.

These two curves are similar. In detail, in MOFCOOL dataset, the system is in liquid phase at the first 1000 frames, which has the higher energy transfer rate. In the later frames, when the system is in solid state, the energy transfer rate drops. For LIQCRY dataset, during the first 120,000 frames, the system is in liquid phase, which has higher energy transfer rate. The energy transfer rate drops during the phase transition. Given the observation that this parameter drastically changes during phase transition, in the future, we aim to develop a phase transition detection tool using this parameter. Moreover, the energy flux reflects the magnitude of velocity and force. These physical parameters are related to the first and second order derivatives of position data, which may potentially explain the performance of polynomial compression algorithm under different phase behavior (chapter 7.1).

References

[1] Giovanni Ciccotti, Mauro Ferrario, Christof Schuette, et al. Molecular dynamics simulation. *Entropy*, 16:233, 2014.

[2] Wm G Hoover. Molecular dynamics. In *Molecular Dynamics*, volume 258, 1986.

[3] Martin Karplus and J Andrew McCammon. Molecular dynamics simulations of biomolecules. *Nature Structural and Molecular Biology*, 9(9):646, 2002.

[4] DC Rapaport. Molecular dynamics simulation. *Computing in Science & Engineering*, 1(1):70–71, 1999.

[5] Anthony K Rappe and William A Goddard III. Charge equilibration for molecular dynamics simulations. *The Journal of Physical Chemistry*, 95(8):3358–3363, 1991.

[6] Shūichi Nosé. A molecular dynamics method for simulations in the canonical ensemble. *Molecular physics*, 52(2):255–268, 1984.

[7] R Tycko, G Dabbagh, RM Fleming, RC Haddon, AV Makhija, and SM Zahurak. Molecular dynamics and the phase transition in solid c 60. *Physical review letters*, 67(14):1886, 1991.

[8] M. Karplus and J. Kuriyan. Molecular dynamics and protein function. *Proceedings of the National Academy of Sciences*, 102(19):6679–6685, 2005.

[9] Jafar Azamat, Alireza Khataee, and Sang Woo Joo. Molecular dynamics simulation of trihalomethanes separation from water by functionalized nanoporous graphene under induced pressure. *Chemical Engineering Science*, 127:285 – 292, 2015.

[10] John Edward Jones. On the determination of molecular fields.—ii. from the equation of state of a gas. *Proc. R. Soc. Lond. A*, 106(738):463–477, 1924.

[11] Anil K Jain. Image data compression: A review. *Proceedings of the IEEE*, 69(3):349–389, 1981.

[12] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[13] Khalid Sayood. *Introduction to data compression*. Morgan Kaufmann, 2017.

[14] Jan Huwald, Stephan Richter, Bashar Ibrahim, and Peter Dittrich. Compressing molecular dynamics trajectories: Breaking the one-bit-per-sample barrier. *Journal of computational chemistry*, 37(20):1897–1906, 2016.

[15] Anand Kumar, Xingquan Zhu, Yi-Cheng Tu, and Sagar Pandit. Compression in molecular simulation datasets. In *International Conference on Intelligent Science and Big Data Engineering*, pages 22–29. Springer, 2013.

[16] Ardita Shkurti, Ramon Goni, Pau Andrio, Elena Breitmoser, Iain Bethune, Modesto Orozco, and Charles A Laughton. pypcazip: A pca-based toolkit for compression and analysis of molecular simulation data. *SoftwareX*, 5:44–50, 2016.

[17] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[18] Alan W Paeth. Image file compression made easy. In *Graphics Gems II*, pages 93–100. Elsevier, 1991.

[19] Alireza Makhzani and Brendan Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

[20] Alfredo Metere, Peter Oleynikov, Mikhail Dzugutov, and Michael O'Keeffe. Formation of a new archetypal metal-organic framework from a simple monatomic liquid. *The Journal of Chemical Physics*, 141(23):234503, 2014.

[21] Alfredo Metere, Sten Sarman, Tomas Oppelstrup, and Mikhail Dzugutov. Formation of a columnar liquid crystal in a simple one-component system of particles. *Soft matter*, 11(23):4606–4613, 2015.

[22] H Ohtani, K Hagita, AM Ito, T Kato, T Saitoh, and T Takeda. Irreversible data compression concepts with polynomial fitting in time-order of particle trajectory for visualization of huge particle system. In *Journal of Physics: Conference Series*, volume 454, page 012078. IOP Publishing, 2013.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[24] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[26] Ken-ichi Kamijo and Tetsuji Tanigawa. Stock price pattern recognition-a recurrent neural network approach. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 215–221. IEEE, 1990.

[27] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[28] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[29] Alan W Paeth. A fast algorithm for general raster rotation. In *Graphics Interface*, volume 86, 1986.

[30] R Schmitt and P Fritz. Lossless compression of computer tomography point clouds. In *Advanced Mathematical And Computational Tools In Metrology And Testing: AMCTM VIII*, pages 291–297. World Scientific, 2009.

[31] Yangyan Li, Rui Bu, Mingchao Sun, and Baoquan Chen. Pointcnn. *arXiv preprint arXiv:1801.07791*, 2018.

[32] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[33] T Schneider and E Stoll. Molecular-dynamics study of a three-dimensional one-component model for distortive phase transitions. *Physical Review B*, 17(3):1302, 1978.

[34] H Eugene Stanley. *Phase transitions and critical phenomena*. Clarendon Press, Oxford, 1971.

[35] Pin-Yu Chen and Alfred O Hero. Phase transitions in spectral community detection. *IEEE Transactions on Signal Processing*, 63(16):4339–4347, 2015.

[36] Akinori Tanaka and Akio Tomiya. Detection of phase transition via convolutional neural networks. *Journal of the Physical Society of Japan*, 86(6):063001, 2017.

[37] Leon Brillouin. *Science and information theory*. Courier Corporation, 2013.

[38] Claude E Shannon. A note on the concept of entropy. *Bell System Tech. J*, 27(3):379–423, 1948.

[39] Herbert B Callen. Thermodynamics and an introduction to thermostatistics, 1998.