2015

Pattern Mining and Events Discovery in Molecular Dynamics Simulations Data

Shobhit Sandesh Shakya
Louisiana State University and Agricultural and Mechanical College, shobhit.shakya@gmail.com

PATTERN MINING AND EVENTS DISCOVERY IN MOLECULAR DYNAMICS SIMULATIONS DATA

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical Engineering and Computer Science

by
Shobhit Sandesh Shakya
B.E., Tribhuvan University, 2006
May 2015

To my mom and dad.

Acknowledgements

This work would not have been possible without the efforts of various people, who were with me throughout. I owe my success to all these people.

First, I would like to express my sincere gratitude to my advisor Dr. Jian Zhang for believing in me and giving me his consistent support and guidance throughout my Ph.D. study. His guidance and encouragement have been crucial in shaping not only my dissertation research but also my overall development as a professional researcher.

I would also like to express my sincere gratitude to my advisory committee members: Dr. Bijaya B. Karki, Dr. Jianhua Chen, and Dr. Dirk Vertigan for being in my committee. Their encouragement, cooperation and guidance have been invaluable. Their willingness to evaluate my work is highly appreciated. Also, I would like to thank Dr. Gokarna Sharma and Bidur Bohara for their support and thoughtful inputs that have made my work qualitatively better. I also like to thank all my friends and faculty at Louisiana State University who have made my stay pleasant and successful.

Finally, I would like to thank my family for the support they provided me through my entire life. Your love and support has given me the strength and guidance to pursue my dreams and succeed in my career path.

Table of Contents

List of Tables

List of Figures

Abstract

Molecular dynamics simulation method is widely used to calculate and understand a wide range of properties of materials. A lot of research efforts have been focused on simulation techniques but relatively fewer works are done on methods for analyzing the simulation results. Large-scale simulations usually generate massive amounts of data, which make manual analysis infeasible, particularly when it is necessary to look into the details of the simulation results. In this dissertation, we propose a system that uses computational method to automatically perform analysis of simulation data, which represent atomic position-time series. The system identifies, in an automated fashion, the micro-level events (such as the bond formation/breaking) that are connected to large movements of the atoms, which is considered to be relevant to the diffusion property of the material. The challenge is how to discover such interesting atomic activities which are the key to understanding macro-level (bulk) properties of material. Furthermore, simply mining the structure graph of a material (the graph where the constituent atoms form nodes and the bonds between the atoms form edges) offers little help in this scenario. It is the patterns among the atomic dynamics that may be good candidate for underlying mechanisms. We propose an event-graph model to model the atomic dynamics and propose a graph mining algorithm to discover popular subgraphs in the event graph. We also analyze such patterns in primitive ring mining process and calculate the distributions of primitive rings during large and normal movement of atoms. Because the event graph is a directed acyclic graph, our mining algorithm uses a new graph encoding scheme that is based on topological- sorting. This encoding scheme also ensures that our algorithm enumerates candidate subgraphs without any duplication. Our experiments using simulation data of silica liquid show the effectiveness of the proposed mining system.

Chapter 1
Introduction

1.1 Computer Simulation

Computer Simulation is a widely used tool in operation research and is a key method to evaluate, improve, and optimize many types of processes. The simulation uses an abstract computation model to simulate any physical system. Computer simulations have been extensively used in mathematical modeling of many natural systems in physics (computational physics), astrophysics, chemistry, biology, geoscience, human systems in economics, psychology, social science, engineering and material science [40]. A running system's model is represented as a computer simulation which behaves as the replica of the physical system. It can either used to explore or gain new insights in to new technology or to simulate natural or physical process to study about their behavior.

Computer simulation is the outcome of rapid growth in computer field, following its first large scale deployment during World War II. It was first used to model the process of nuclear detonation. Since then, it is extensively used in scientific community. In this thesis, we study extensively on molecular dynamics simulation data and analyze their microscopic behavior using data mining technique. As far as our knowledge, such kind of analysis has not been done before [64].

1.2 Molecular Dynamics

1.2.1 History

The molecular dynamics method was first introduced by Alder and Wainwright in the late 1950's [4, 8]. They studied the interactions of hard spheres (representing atoms in the system). Many important findings related to the behavior of simple liquids came out from their research. The next crucial finding was in 1964, in which Rahman did the first simulation using a realistic potential for liquid argon [4, 6]. Rahman and Stillinger performed their first molecular dynamics simulation of a realistic system in their simulation of liquid water in 1974 [4, 66]. The first protein simulations was performed in 1977, in which the simulation of the bovine pancreatic trypsin inhibitor (BPTI) was performed [4, 54]. In present context, we find molecular dynamics simulations of solvated proteins, protein-DNA complexes along with the lipid systems. It addresses a variety of issues such as the thermodynamics of ligand binding and the folding of small proteins. Since then the number of simulation techniques has expanded exponentially. Now many specialized techniques for particular problems (quantum mechanical - classical simulations) are being used to study enzymatic reactions in the context of the full protein. Molecular dynamics simulation are also employed in experimental procedures such as X-ray crystallography and NMR structure determination [4].

1.2.2 Molecular Dynamics in Material Science

Molecular dynamics is a form of computer simulation in which atoms and molecules are allowed to interact for a period of time by initially defining a group of properties that represent atomic positions, atomic types and constraints [13].The simulation requires inter-atomic forces which are determined by quantum mechanics, which basically deals with electrons, but can be formulated on interaction between atoms. Once we know about the forces, solving Newton equation will give atomic trajectories, i.e. position of atoms as a function of time:

$$\vec{F_i} = m_i \vec{a_i} \tag{1.1}$$

$$\vec{a} = \frac{d^2\vec{r_i}(t)}{dt^2} \tag{1.2}$$

Here, $\vec{F}$, m, $\vec{a}$ and t represents force, mass, acceleration and time. $r_i(t)$ represents the atomic trajectories, where i ranges from 1 to n, the number of atoms. This atomic trajectory data can be used to calculate and analyze different properties of the system.

Over the last couple of decades, molecular dynamic (MD) simulation has been one of the most common computational practices in science and engineering. The underlying physical model ranges from relatively simple (empirical) pair-wise interatomic potentials to highly sophisticated quantum mechanical formulation (first-principles) [9, 32, 37]. All MD approaches have been adopted in massively parallel environment. Pair potentials simulations have been benchmarked for over a billion of time steps. On the other hand, first-principles simulations despite being computationally very intensive have also exceeded a million of time steps [13]. For a given material system, outputs from these simulations represent various physical (bulk) properties including pressure, volume, energy, temperature, and many derived quantities [16, 59]. More importantly, they include massive data for atomic configurations, which are distributed in 3D space and vary with time. Gaining insight into useful structural and dynamical information contained in the large-scale simulations has always been highly desirable but still poses a tremendous challenge.

Two types of approaches are commonly used to gain insight into massive data generated by MD simulations. First, numerous visualization systems are currently available to display/render atomic configurations in terms of inter-atomic bonds, modular structures (such as polyhedra, clusters), and other forms [13, 16, 37, 52]. Moreover, they support animation and trajectory rendering for a quick navigation through the data. Animation renders the atomic configuration at each time step going through the successive time steps so that one can develop some sense of dynamical behavior. On the other hand, particle trajectories allow a complete representation of the data by rendering positions of all atoms at all-time steps so full information is contained in a single display [15]. Visualization of trajectories and velocity/displacement data can help us assess the nature and extent of atomic movements. Both the animation and trajectory rendering usually work fine

for relatively small-sized data sets. As simulations become larger and longer, frame rates and occlusion become serious issues, and visual inspection becomes less effective. Parallel processing and immersive visualization can be helpful [62, 67]. Second, various analysis methods involve computation of system-wide averaged properties [9, 12, 83]. They include radial distribution functions, bond length and angle distributions, coordination numbers, ring statistics, etc., to understand structural behavior [13, 59]. Similarly, the dynamical analysis involves mean square displacements, bond-event rates, velocity auto-correlation functions, stress auto-correlation functions, etc. Which specific visualization/analysis method works best for a given MD data set depends on the material system simulated and also on its properties being investigated.

Because FPMD simulations generate details about molecular dynamics, not only the simulation data can be used to predict the macro level property of the material with high accuracy, but also the data may contain meaningful information at the atomic level that can help researchers to understand the mechanism leading to a certain property. In particular, by identifying the atomic-level events (change of atomic configuration) relevant to a material property, researchers may be able to create models and explain why the material displays that property. However, in many cases only an extreme small fraction of the molecular dynamics generated by the simulation may be linked to a particular macro level property. And there is an overwhelming amount of data from simulation that are irrelevant. Therefore, it is very important to identify useful structural and dynamical information among the massive data derived from intensive simulations. This is a tremendous challenge in many scientific and engineering areas where simulations are heavily used.

Molecular dynamic simulation is widely used in material science to understand properties of materials. Majority of research are focussed on simulation and visualization techniques but very few research are done on computational methods. Large-scale simulations generate massive amount of data which makes manual analysis of simulation data nearly impossible, particulary when it comes to the detail analysis. This created the necessity of applying data mining techniques to extract meaningful and interesting information from such large scale data in automated fashion.

1.3 Data Mining

As we are growing with different technology in our society, each and every day large amount of data are being generated. We are not only concerned with the space to store such data but extracting interesting and meaningful information from them has been a major issue. The process of extracting such pattern and information from datasets and transforming in to an understandable structure is termed as Data mining or Knowledge Discovery in Database (KDD). It is a non trivial task to find previously unknown and useful information from the data repository. So, it is the extraction of hidden pattern not the data itself from the repository [1, 39].

The manual extraction of information from data has occurred through centuries. The increasing power of computational technology has dramatically increased the amount of data. As the data size have grown, manual analysis has become almost impossible. With ever proliferating computing power and discoveries in mining algorithm such as Clustering, Regression, Support Vector Machine, Graph Mining etc, the bridge between statistical analysis and artificial intelligence has been shortened [1, 60].

With the dramatic increase in data, it's getting tough to not only mining those data but also formalizing the problem statement. Our ability to collect data has exponentially increased. We collect data from different sensors, devices in various format from independent or connected applications. This flooding of data has outrun our capability to store, process and analyze the data sets [23]. In most cases we are unaware of the direction to look for because of the huge magnitude of data. In such case data mining plays an important role in finding useful information. Given a data set, one can generate a hypothesis for a particular problem. Now to prove or disprove the hypothesis, we need data mining techniques, which would confirm the claim. This has made data mining an essential tool in scientific community [1].

As the source of collecting data is increasing, we not only facing the problem for storing them but also getting insight in the data is another challenge. The term "Big Data" was first introduced in 1998 in a Silicon Graphics slide by John Mashey with the title of "Big Data and the Next wave of Infra Stress" [22, 23]. Big data is rapidly expanding in fields like engineering and physical

sciences such as biological and biomedical science. The web page indexed by Google were around 1 million in 1998, went up to 1 billion in 2000 and exceeded 1 trillion by 2008. The rapid expansion of users in the field of social networking, such as Facebook and Twitter has dramattically increased the volume of web data [23]. Moreover data from biological science such as body scan data, data from genome sequence created the need of data mining. This has made data mining extremely popular for studying about the genome structure.

In data mining process, we go through various process. Following are the essential steps in data mining process [24]:

1. Selection
2. Pre-processing
3. Transformation
4. Data-Mining
5. Evaluation

We mainly focus on process 2-5. In Chapter 2, we have discussed on data preprocessing process. We have shown the process of converting the raw data in to the form which is ready for mining. Transformation is another important step in data mining process. We have cleverly transformed the molecular dynamics scattered data in to an event graph model which is foundation for the frequent subgraph mining. Developing noble data mining algorithms is the heart of our research. We have developed three important algorithms in this thesis: 1) Big move detection 2) ToPoMine 3) Primitive ring mining. All three algorithms are used to evaluate the molecular dynamics simulations data and thus producing interesting results for material scientists.

1.4 Motivation and Objectives

The research on mining chemical compounds is dominated by structure mining (representing molecular structure with graph(s) i.e. atom with node and bonds between them with edges) of

the compounds. Thus, they mostly deal with the molecular structure of a substance. Very little research has been done on studying the dynamic nature of such compounds. This makes our research unique and essential in material science field, where researchers study on their dynamic features.

There has been great interest in material science to model and predict macroscopic properties, e.g., strength, viscosity, of a type of material. Such model and prediction hold key to our understanding of many natural phenomena as well as our capability to design new materials. In many cases, the property is determined by microscopic (atomic level) structure and dynamics. Simulation is widely used to study atom dynamics [32]. It generates a large amount of data that detail atomic activities in the material being simulated. Researchers are often too overwhelmed by the amount of data generated by the simulations to discover interesting atom activities relevant to the macro-level properties of a material. However, the interesting activities, i.e., the activities that may form the atomic mechanism underlying a particular material property, are often buried under a lot of other activities that are of no interest. In most cases, due to the large size of the simulation data, it is impossible for scientists to manually go through the data and discover the relevant activities [9, 13, 37]. There is a great potential for data mining to play an important role in this type of discoveries.

Many types of materials studied have regular or semiregular structures. The structure of atoms in the material can be modeled as a graph where the atoms become the nodes and the bonds between atoms become edges. Such atom graph has also been used for modeling large chemical and biological molecules. Graph mining has been applied to these atom graphs for discovering interesting structures in the molecules [16, 21]. However, the atom graph representation represents a static view of the molecular structure and thus is not suitable for modeling atomic dynamics in such type of material. Furthermore, mining the structure graph of a material (the graph where the constituent atoms form nodes and the bonds between the atoms form edges) offers little help in this scenario. It is the patterns among the atomic dynamics that may be good candidate for mechanisms underlying a particular material property. Discovery of such patterns can lead to better models and better predictions of material properties. Rather than static structures, the study of atomic dynamics

in materials focuses on the changes of the structures. In particular, the patterns among the dynamic changes are of importance as the patterns are good candidate mechanisms underlying a particular material property.

Atomic diffusion is an integral feature of any dynamic system. We can relate the diffusion property of any materials to various other properties like viscosity, strength etc. In the following paragraphs we will briefly discuss about the coordination behavior of an atom in a material, which are responsible for the atomic diffusion. To discuss about the pattern of coordination changes in a material and what type of changes contribute in the atomic diffusion, we take an example silica liquid (SiO_2).

In molecular dynamics system atoms are in continuous motion. In the presence of heat they vibrate around its fixed position. This is the innate character of atoms in any material. When we observe a material to microscopic level, we find atom bonded with its neighboring atoms of different species. It is the bond between atoms that held them together and restrict them from escaping their proximity. The bond strength plays a key role in the movement of atoms. However an atom does not remain in stable state all the time. Since, it is a dynamic system, atom constantly goes through bond forming and bond breaking events.

The structure of a given liquid system in defined in terms of atomic coordination which is a fundamental quantity used to characterize the local connectivity. e.g. in silica liquid (SiO_2), we can have four types of coordination (Si-O, O-Si, Si-Si and O-O). Only Si-O and O-Si environments directly involve Si-O bonding and are considered to be important. The mean Si-O and O-Si coordination numbers are close to 4 and 2 respectively. The Si-O coordination consist of mostly four fold species (called tetrahedral states). It is the stable state of Si atom. The O-Si coordination consist of mostly two folds (called bridging oxygen). It is the stable state of O atom. The system is dynamic, therefore the atoms undergo coordination changes with time. When a new bond is formed, a pair of over-coordinated species appears whereas when a existing bond is broken, a pair of under-coordinated species appears. These species disappear if the bond just formed breaks back or the same two atoms form the bond again. These bond change makes the atom to move large

distance or remain in its proximity depending upon the formation of bond with new atom or with the same atom respectively.

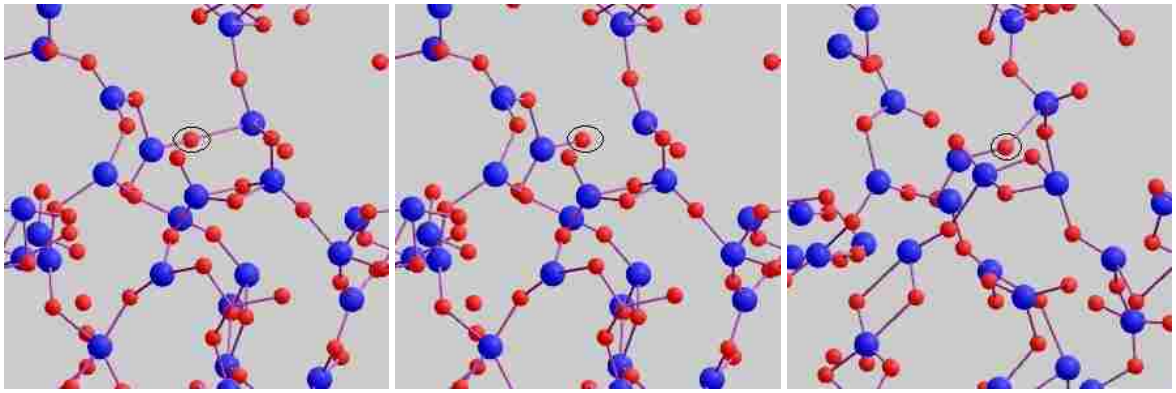

Figure 1.1: Bond forming and breaking with the same atom resulting in small atomic movement. Atom marked with circle is observed for its movement pattern.

In Figure 1.1 we have shown the activity of bond formation of O atom. We marked an O atom and observed its activity. In Figure 1.1 (left) the atom is in stable stage, it is bonded with two different Si atoms. It is a bridging oxygen since it forms bridge of bonds between two Si atoms. Now most of the time this O atom vibrates around its initial position and does not form or break any bond. In Figure 1.1 (middle), we see the marked atom breaking bond with its one of the bonded Si atom. After staying in two fold state for a while, the O atom is again bonded with the same Si atom. This is shown in Figure 1.1 (right). This type of bond breaking and bond forming process does not contribute to the large movement of atoms.

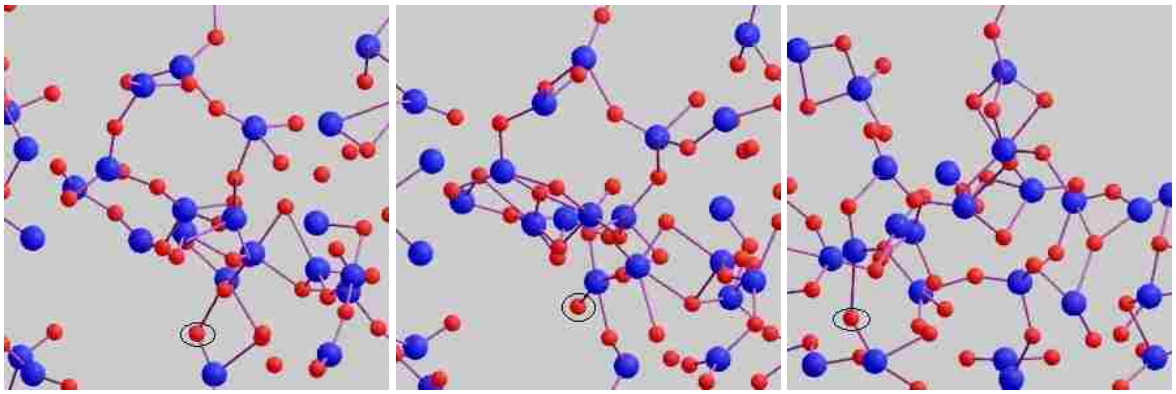

Figure 1.2: Bond forming and breaking with the different atom resulting in large atomic movement. Atom marked with circle is observed for its movement pattern.

In Figure 1.2, we have shown the second case where an O atom breaks a bond and again forms a bond with different Si atom. This is the key reason due to which an atom undergo large atomic movement. In Figure 1.2 (left), we can see the marked O atom in stable state. In Figure 1.2 (middle), it loses a bond with its neighboring Si atom. An atom when in stable state cannot remain in same state for long period of time. It will eventually form a bond with another atom. In Figure 1.2 (right), the same one fold O atom forms a bond with another Si atom. This Si atom is different from the previous neighboring atom of marked O atom. This type of bond formation contributes in significantly larger atomic movement. Our primary objective is to track such type of movement and analyze the pattern of coordination behavior which led to such movement. To achieve this goal we have used different data mining techniques in this research.

To apply any data mining techniques on large scale data sets, it is crucial to filter out unwanted data. It is difficult to locate only useful data. In Chapter 3 we have presented window detection technique [61] which automatically locate only such data which are interesting. In Chapter 4 we discuss in details on applying graph mining algorithm ToPoMine, which mines interesting sequence of events from filtered data.And in Chapter 5 we have presented an efficient algorithm to mine all the primitive rings of a given atom.

The objective behind this dissertation is to study the atomic level properties of material. We focus mainly on data mining techniques to mine microscopic features and relate them with the diffusive behavior of atoms.

1.4.1 Searching for Feature Vectors

The first challenge for us was to look for those atomic features of a material which we can relate to macroscopic property. We came up with three features which can contribute in predicting macroscopic property of a material: 1) Detecting large atomic movement, 2) Mining events (coordination change) and 3) Mining primitive rings.

1.4.2 Detecting Large Atomic Movement

Material Science experts have been detecting the overall displacement of an atom for a given simulation time. However, to detect a time window when an atom is moving a lot over millions of time steps was not easy. We came up with a novel approach to detect those time windows. We discuss this in details in Chapter 3.

1.4.3 Mining Events

An event actually refers to a coordination change of an atom. We propose a event graph which models the coordination change of atoms. We then mine the subgraph to find the interesting sequence of events. Chapter 4 talks in details about this approach.

1.4.4 Mining Primitive Rings

Ring structures of atoms plays a vital role in the physical properties of a material. Counting rings in millions time steps simulation data gets tricky, so in Chapter 5 we came up with an efficient primitive ring mining algorithm.

1.5 Dissertation Organization

This rest of the dissertation is organized as follows: In Chapter 2, we discuss about the preprocessing of data before we start analyzing them, in Chapter 3, we discuss about analyzing molecular dynamics and mining the mechanism of atomic diffusion. In Chapter 4, we discuss about the literature and related work on frequent subgraph mining. We also present our novel subgraph mining technique in the same Chapter and finally in Chapter 5, we discuss about mining the primitive ring structures.

Chapter 2
Data Pre-processing

2.1 Simulation Data

Data used in this study are from parallel first-principles molecular dyamics (FPMD) simulations of silica liquid. The supercell used in the simulations consisted of 72 atoms (24 Si and 48 O atoms), which correspond to a discrete set of 216 (positional) degrees of freedom. Therefore super cell is a system which is cubical in shape and contains all the atoms. The atomic positions in liquid phase are strongly correlated but without any long-range order, and are constanly changing with time. The simulation runs ranged from a couple of hundred thousands of FPMD steps (at high temperature of 4000 K) to little more than one million steps (at the lowest temperature of 2800 K studied). A time step of 3 femtoseconds was used at 2800 K so that the simulation duration slightly exceeded 3 nanoseconds whereas the time step of 1 femtosecond was used at higher temperatures (3000 K, 3500 and 4000 K). Six data sets (two at 2800 K, two at 3000 K, one at 3500 K and one at 4000 K) were generated and used in our analysis. It is important to note that the simulations are finite in both space and time. The supercell length varies from 10.3 to 13.8 whereas the simulation duration ranges from 200 picoseconds to 3.2 nanoseconds.

The snapshot of data set is shown in Figure 2.1. It is in the form of time varying series coordinate.The first row is the total number of atoms in super cell. The second row is the dimension of super cell. The term 'Konfig' refers to the time stamp. Every rows after term 'Konfig' are the coordinates of atoms in 3-D space. There are 72 coordinates between two consecutive 'Konfig'. In Figure 2.2, we can see the image of Si (blue) and O (red) atoms in one time stamp.

```
 72  72   1
 0.1526534E+02  0.1032000E-08  0.1032000E-08  0.1032000E-08  0.1000000E-14
  3000.00000000000
 CAR
Silica_liquid + Water
Konfig=           1
  0.21101795  0.63411968  0.24095908
  0.66588042  0.68759192  0.64335179
  0.70555770  0.33159901  0.85277712
  0.45552926  0.47504759  0.56815863
  0.46097618  0.20471846  0.99159158
  0.13479868  0.08306488  0.60706127
  0.44405439  0.29833631  0.28697836
  0.48434364  0.56931394  0.22986932
  0.27931105  0.29342215  0.74542892
  0.60624401  0.81552352  0.39370914
  0.21584379  0.44986366  0.40510620
  0.29171649  0.87728285  0.39538316
  0.86557044  0.41233217  0.08716319
  0.69431575  0.12179400  0.00635831
  0.93273441  0.69271005  0.80500685
  0.93590230  0.97386273  0.08728985
  0.67282422  0.10464099  0.26557031
  0.02803807  0.93345476  0.80680362
  0.98296988  0.40673939  0.78948549
  0.94354717  0.16842498  0.24525987
  0.94681318  0.66201357  0.10322032
```

Figure 2.1: Raw input file with coordinates of every atoms.

2.2 Radial Distribution Function (RDF)

This study is mainly focussed on mining interesting events. Before we start any mining process we need information about the coordination details of every atoms. From this information we later find out the bond forming and breaking process. This is the fundamental building block in this study to define an event. To generate such coordination information we used AtomViz. To calculate coordination number we need Radial Distribution Function. RDF describes how the density of particles varies from a reference particle in a system. Considering the homogeneous distribution of atoms in space, it gives the probability of finding atoms in a shell dr at the distance r of other atom, taken as a reference point.

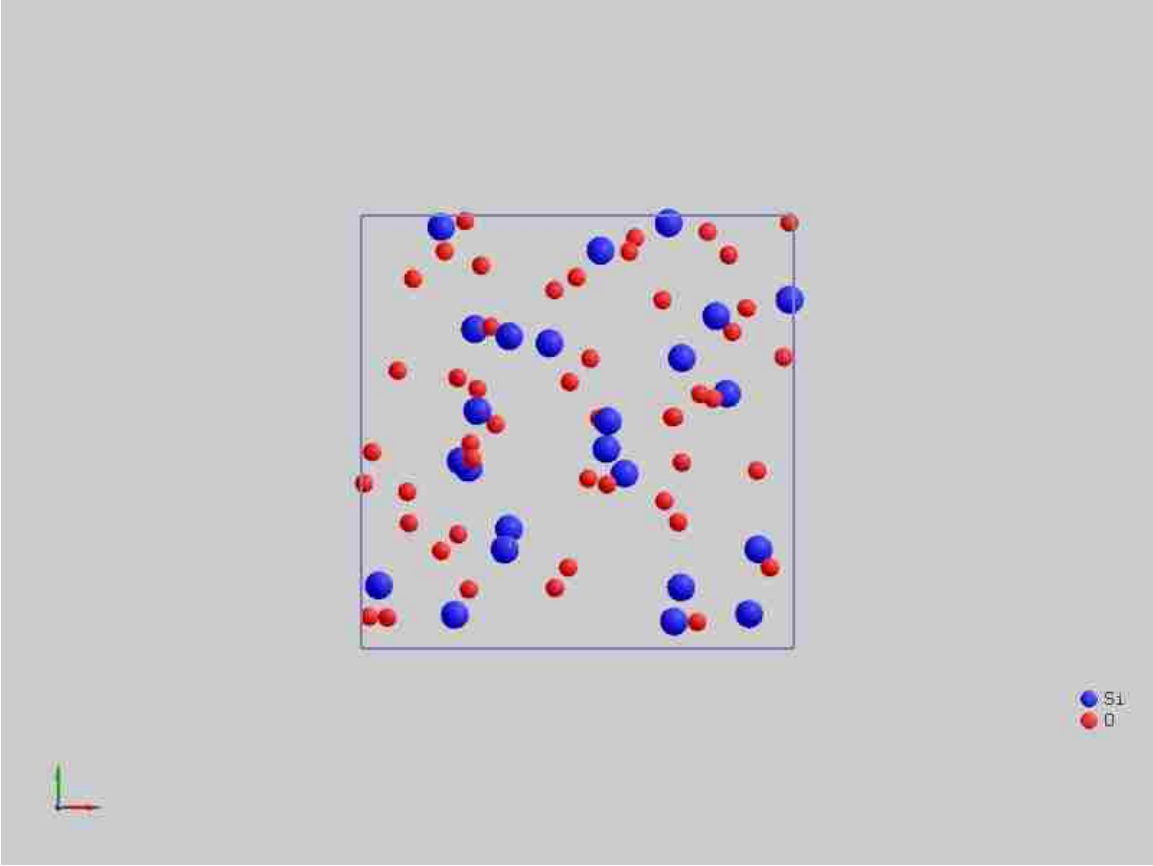

Figure 2.2: Snapshot of Si and O atoms in one time stamp.

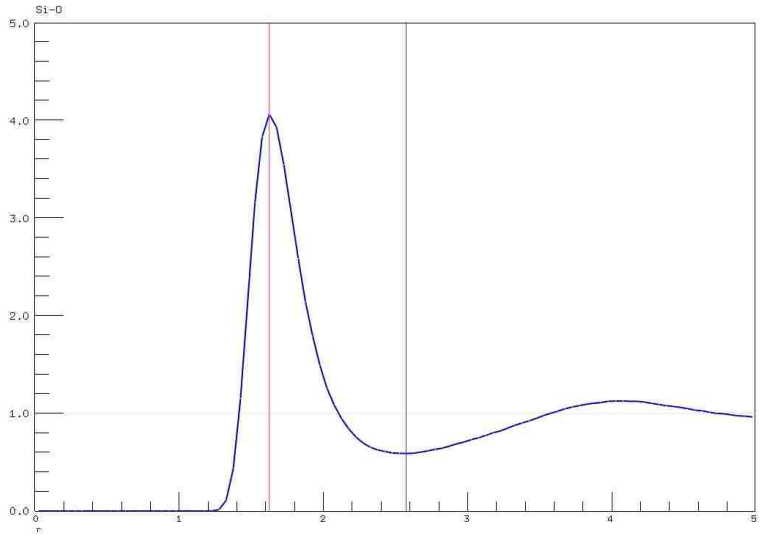

Figure 2.3: Radial Distribution Frequency (RDF) calculation between Si-O. X-axis is the distance in angstrom and Y-axis is the probability of finding atoms.

Definition 1 *(Radial Distribution Function) Let $g(r)$ be the RDF of a system, then for any particle i in the system, there are $g(r)$ number of atoms per volume at a radial distance r from the atom i [14].*

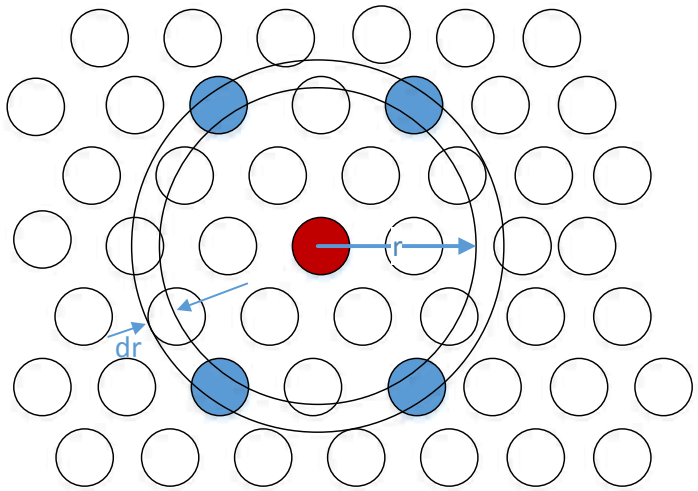

Figure 2.4: Space discretization for RDF calculation.

Figure 2.4 shows the presence of atoms which falls in the region dr. By changing r and counting the number of atoms in region dr average radial distribution is calculated. In figure 2.3 we have shown the average radial distribution of Si-O and O-Si. At the peak the probability of finding atom is maximum. We denote it by r_{min}. Any atom closer than r_{min} will not from any bond because of repulsive force between the atoms. The first vertical line in the figure represents r_{min}. The second vertical line represents r_{cut}. Any atom beyond this distance can not be bonded. Therefore any atom in between r_{min} and r_{cut} forms coordination bond with the reference atom. Figure 2.5 is the image of bond formation between Si and O atoms.

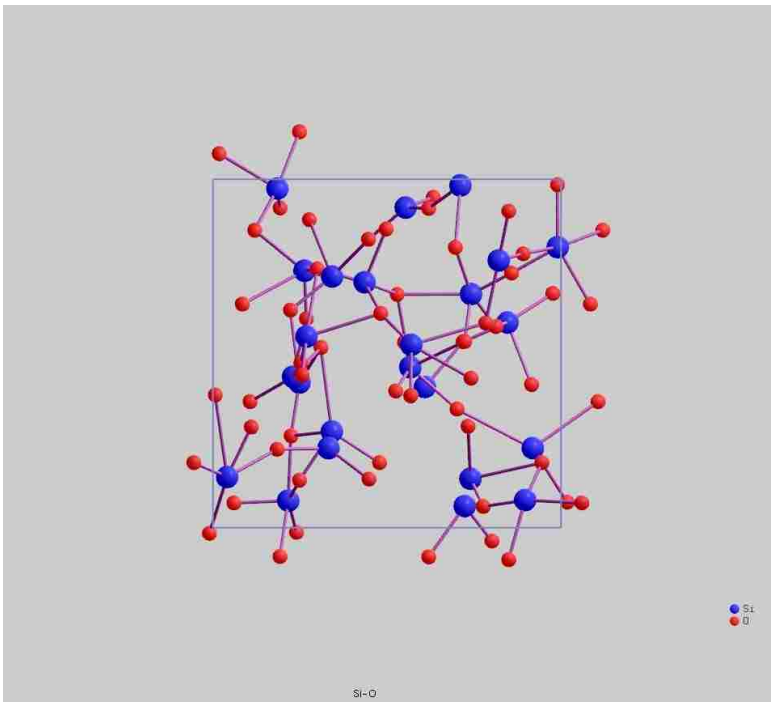

Figure 2.5: Bond formation between Si and O atoms.

2.3 Removing Periodicity

The atomic position in the silica liquid changes with time. But the atoms inside super cell do not skip the box. As we can see in the Figure 2.6 there exists periodicity in the coordinates of atoms. When an atom goes out of the super cell in one direction, it re enters from opposite direction. To do any further analysis on the data, we remove periodic behavior of data set by unfolding the coordinates. We assume that the maximum distance an atom can travel in a single time stamp is half of the super cell.

$$d_{max} = L/2 \tag{2.1}$$

where L is the length of super cell.

Since all the coordinates of the input data are normalized we only compare the distance travel by an atom with 0.5.

Below the unfolding technique is discussed in detail.

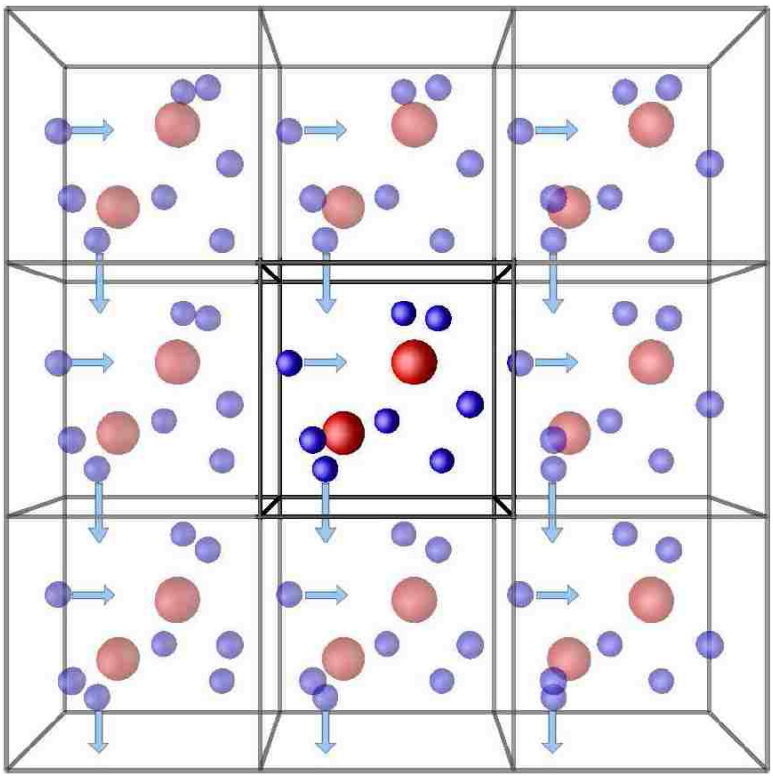

Figure 2.6: Representation of idea of periodicity boundary condition [2]

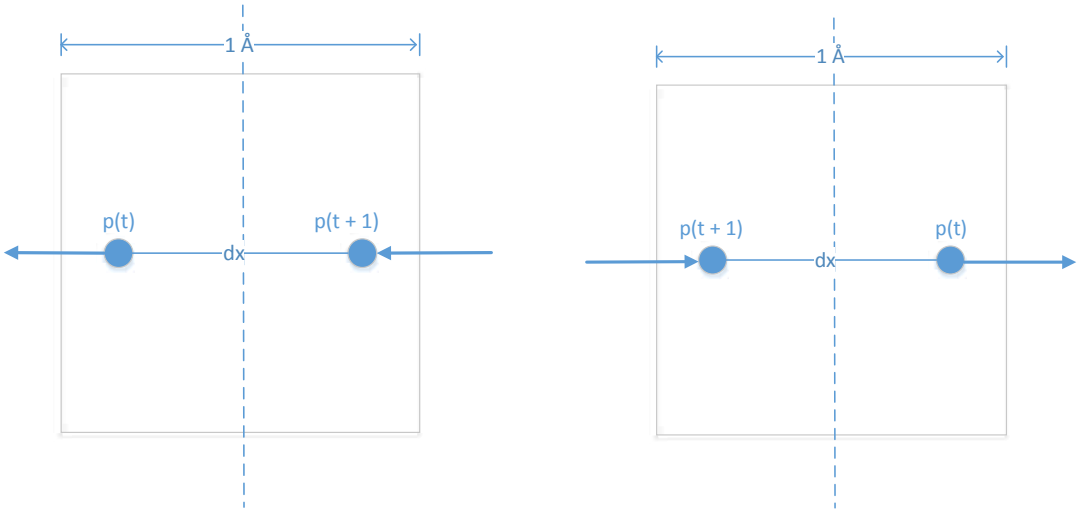

Figure 2.7: Top: Atom leaving from left of and entering from right of super cell. In this case dx is positive. Bottom: Atom leaving from right and entering from left of super cell. In this case dx is negative.

Let the position of an atom o at time t be $p(t) = (x_1, y_1, z_1)$ and at $(t + 1)$ be $p(t + 1) = (x_2, y_2, z_2)$. Now let dx be the change in x coordinate.

We calculate dx as

$$dx = x_2 - x_1 \tag{2.2}$$

Similarly,

$$dy = y_2 - y_1 \tag{2.3}$$

$$dz = z_2 - z_1 \tag{2.4}$$

Figure 2.7 (left) shows the position of atom o at time t and $(t + 1)$. Let's only consider the movement of atom in x direction i.e. dx. Since we are only considering x coordinate, we can represent position $p(t)$ by x_1 and $p(t+1)$ by x_2. To move from x_1 to x_2, atom o can take two paths. One is going to the right direction and other is exiting the super cell from left and then entering from right. The direction of movement is determined by calculating dx.

We will have two conditions:

i) If $dx > 0.5$ then the new unfolded coordinate will be $x_2 = x_1 + dx - 1$.

ii) If $dx < -0.5$ then $x_2 = x_1 + dx + 1$.

The second condition can be visualized in Figure 2.7 (right). Here the position of x_1 and x_2 is interchanged. Similarly, we do the same with dy and dz.

i) If $dy > 0.5$ then, $y_2 = y_1 + dy - 1$.

ii) If $dy < -0.5$ then, $y_2 = y_1 + dy + 1$.

and,

i) If $dz > 0.5$ then, $z_2 = z_1 + dz - 1$.

ii) If $dz < -0.5$ then, $z_2 = z_1 + dz + 1$.

After unfolding the coordinates of each atom we can now plot its displacement with respect to the initial position as a function of time. We define it as $g(t)$. It is the Euclidean distance between a reference point and a point at time t.

$$g(t) = |p(t) - p(t_0)|^2 \tag{2.5}$$

where $p(t)$ is the position coordinate at time t and $p(t_0)$ is the position coordinate at initial reference time t_0

Figure 2.8 (top) shows an example of this plot.

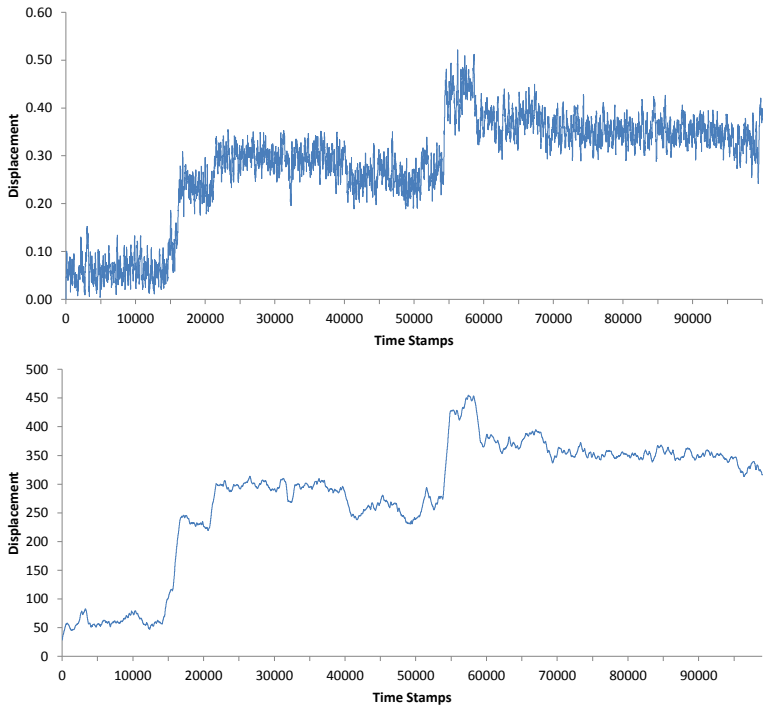

Figure 2.8: Top: The displacement (g) versus time (t) plot. Bottom: Smoothing of the curve on top. X axis is the time stamps and Y axis is the displacement from initial position

2.4 Data Smoothing

Data smoothing is an essential process in statistics and image processing field. This process is mainly concerned in developing approximate function to capture meaningful pattern in the data. It helps to leave out noise and fine scale pattern from the data. There are various techniques to perform such action such as:

- Additive smoothing
- Butterworth filter
- Digital filter

- Kalman filter
- Kernel smoother
- Laplacian smoothing

As you can see in Figure 2.8, an atom constantly undergoes random oscillations. A large movement is a motion whose magnitude of displacement is significantly greater than the random moves. We are only concerned with such movement. The plot has a lot of jitters which we have to remove before doing any further analysis. We used convolution, which is often used in signal processing, as filters for noise removal. It is the process of multiplying one signal by shifted version of other signal resulting the average of their product. Convolution is a useful process because it accurately describes some effects that occur widely in scientific measurements.

2.4.1 Smoothing Algorithm

We used *moving average* which is the most common algorithm in statistical analysis and computer vision. The "sliding-average smooth" with a rectangular function is the simplest and very effective method of smoothing a signal. This method replaces each point in the signal with the average of "x" adjacent points, where "x" is a positive integer called the "smooth width" [56]. For example, for a 3-points smooth (x = 3):

$$S_i = \frac{Y_{i-1} + Y_i + Y_{i+1}}{x} \tag{2.6}$$

Figure 2.8 (bottom) shows the $g(t)$ after smoothing. In chapter 3, we have discussed in detail about this smoothing process.

Chapter 3
Detecting Extreme Atomic Motion in Simulation Data

3.1 Introduction

In this chapter, we propose a useful approach to analyze a given position-time series produced by parallel molecular dynamics simulation for understanding the dynamical behavior of the material system under consideration. In particular, our interest is to explore atomic diffusion, that is, a process by which a material results in the net transport of constituent atoms. While our analysis can be applicable to any type of materials, here we target a liquid (dense) phase. Atomic diffusion in a solid phase is generally too slow to be captured in currently achievable simulation time-scale with any MD approach even at high temperatures close to the melting point. We use silica (SiO_2) liquid as an example in this study [30, 59]. Silica is one of the most abundant materials of our planet and is important both technological and geological viewpoints. Unlike other liquids, silica liquid is extremely viscous (i.e., highly immobile) so it is interesting to explore how the constituent (Si and O) atoms diffuse or move in this system. Moreover, it was simulated using first-principles molecular dynamics method, which is generally expected to predict physical (macroscopic-scale) properties with high accuracy so the data may contain meaningful information at the microscopic (atomic) level. In particular, our aim is to identify atomic-level events (i.e., changes in atomic configurations occurring through bond formation and breakage) relevant to a material property such as diffusion. It is likely that such events are confined in both space and time involving few atoms and extending over short time intervals. This means that only a relatively small fraction

of the simulation data may actually be worth examining to reveal underlying mechanisms. Our objective is to mine those time windows where such interesting events occurs so that we could analyze the pattern of atomic coordination changes which led to the large atomic diffusion.

The proposed MD simulation data analysis scheme consists of two stages: First, it examines the movement pattern of the constituent atoms, which can involve continuous motion (flowing) or discrete movements (hopping). Using ideas from signal processing, we design an algorithm to automatically detect large movements in the simulation output that contains both the large movements and the regular random movements of the atoms. The algorithm also identifies the time intervals in which the large movements happened if there are any. Second, our scheme describes the liquid system in terms of basic structural units (coordination states representing local atomic connectivity or bonding) and examines how they evolve with time. It relates the detected large movements to the coordination changes the corresponding atoms undergo. Such coordination change may be part of the atomic level mechanisms that determine the diffusion property of the material. Our results show that the proposed analysis scheme is effective in identifying the large movements and the coordination changes that are associated with the movements. It may also help atomistic visualization by pinpointing the time intervals when interesting events likely to occur in the simulation so that we do not need to look through millions of time steps - only corresponding subsets of data may be visualized.

3.2 Material and Methods

3.2.1 MD Data Analysis Scheme

Our scheme to analyze the dynamical behavior of a simulated material system involves two main stages: automatic detection of large movements and identification of the related coordination changes. The main challenge arises because such interesting events are relatively rare and short-lived, and an individual atom may undergo large movements only a few times over long duration. Therefore, it is not feasible to manually examine the entire simulation, which can be arbitarily long

consisting of over a million of time steps. We design algorithms to automate this process, which enables us to extract useful information from a massive position-time series.

3.2.2 Detecting Large Atomic Movements

All atoms contained in the supercell are not anticipated to move by large amounts even in a liquid phase. It is likely that some atoms do not change their positions significantly during a finite time interval or even over the entire simulation time span. This would perhaps be the case for all (or most) atoms in a solid phase. However, many atoms in a simulated liquid system do undergo large positional changes depending on its temperature and composition. Our main interest here is in characterzing the behavior of highly mobile atoms. When an atom undergoes a large displacement, it may do so in two ways: 1) The atom moves gradually, making small step each time. Such incremental steps add up over a long time period to give a large net movement. 2) The atom makes a few large jumps that account for almost all the distance travelled by it. Each such jump occurs over a very short time interval. The movement pattern of individual atoms can thus be a continuous (forward) motion or a set of abrupt jumps or a combination of both. We assume that these movements significantly contribute to the diffusion process. On the other hand, the atoms showing an oscillating motion (back and forth) may remain confined in a local neighborhood for long time, and as such, they are considered to be irrelevant.

We define $g(t)$ to be the displacement of a constituent atom with respect to its position in the reference (initial) configuration as a function of time. Figure 3.1 plots the displacement against time to display two types of motions. Our analysis mainly focuses on the atomic motion consisting of large jumps, which is a hopping like mechanism. We call them the large movement or jump. Such motion can be dominating at low temperatures. Indeed, silica liquid at 2800 K shows that 82 % of large displacements result from jump like motions occuring over short time intervals, with 18 % being incremental forward motion. As one can see in Figure 3.1, an atom constantly undergoes random (oscillating) moves. A large movement is a motion whose magnitude of displacement is

significantly greater than the random moves. (One large movement is shown on the left plot of Figure 3.1).

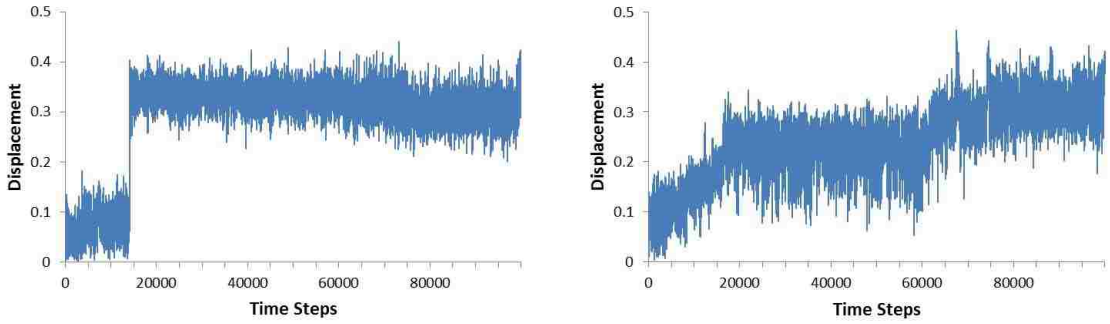

Figure 3.1: The displacement (g) versus time (t) plots. Two type of motions are shown: jump-like motion (left) and incremental forward motion (right).

Smoothing Using Convolution

In mathematics, convolution can formally be defined as operation between two functions, just as multiplication, addition, and integration. Addition takes two numbers and produces a third number, whereas convolution takes two signals and generates a third signal. Convolution is used in many fields of mathematics, such as probability and statistics. In linear systems, convolution can be used to describe the relationship between three signals: the input signal, the impulse response, and the output signal [65].

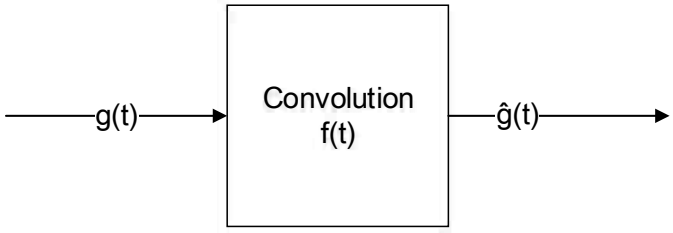

Figure 3.2: Input signal $g(t)$ convolved with window function f(t) to produce $\hat{g}(t)$. $g(t) * f(t) = \hat{g}(t)$

The g-t plot has a lot of jitters associated with oscillating motion of atom. In order to detect large movements, our first step is to smooth the function $g(t)$. From a signal processing point of

view, we consider $g(t)$ (with or without large movement) as a signal that is perturbed by some high frequency noise (the jitters). The goal of the smoothing step is to remove these noises. In signal processing, convolution is often used in such situation as filters for noise removal. The convolution of the function $g(t)$ with a window function f results in a transformed function $\hat{g}(t)$:

$$\hat{g}(t) = \int_{-\infty}^{\infty} f(x)g(t-x)dx \tag{3.1}$$

We smooth $g(t)$ using a square window function, i.e.

$$f(x) = \begin{cases} \frac{1}{a}, & \text{if } -\frac{1}{2a} << x << \frac{1}{2a} \quad (3.2a) \\ 0, & \text{otherwise} \quad (3.2b) \end{cases}$$

Figure 3.3 shows the results of the smoothing on the two curves in Figure 3.1.

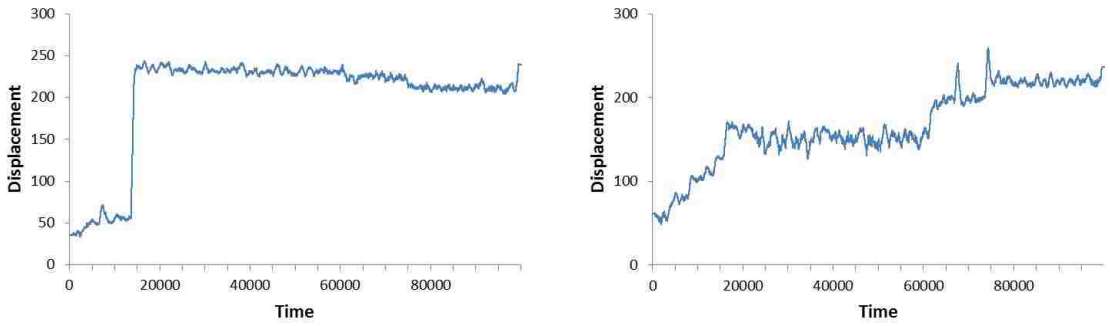

Figure 3.3: Smoothing for the two curves in Figure 1. Left figure corresponds to large movement (jump like motion) and right to forward incremental motion.

From now on, we use $\hat{g}(t)$ instead of $g(t)$ in our analysis. A large movement corresponds to a time window in which the displacement of an atom changes at a higher rate, i.e., the derivative $\hat{g'}(t)$ is of large values. To detect the large movements, we used a threshold T . If in a continuous time window, $\hat{g'}(t) \geq T$, then the corresponding atom exhibits a large movement in that time window. Figure 3.4 plots both $g(t)$ and $\hat{g'}(t)$ for both large movement, i.e., jump-like motion (top) and forward incremental motion (down). The red line shows a properly-chosen threshold. The large movement displayed in the plot of $g(t)$ (green) corresponds to the window in which the $\hat{g'}(t)$

values (blue) are above the threshold in the plot. As we can see our window detection technique identifies only the large movement and excludes the forward incremental motion. We focus on the large movement (jump-like) motion because they are dominant type of motion for the larger displacement.

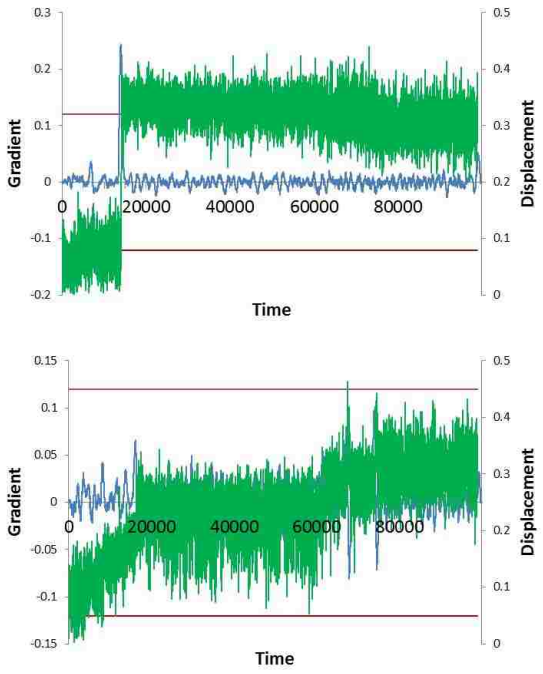

Figure 3.4: Green line is the displacement curve g(t) with y-axis on the right, blue line is gradient curve $\hat{g}(t)$ with y-axis on left and red line is the threshold T (with T = 0.12). For each curve x-axis is the simulation time.

The key to accurate detection is to find a proper threshold. The goal is to distinguish the jumps from the regular oscillation. For this purpose, we used two different approaches two calculate threshold values.

Mixture of Gaussian Model

A Gaussian mixture model(GMM) is a probabilistic model that combines various single gaussian models with unknown parameters and gives the best abstract estimate of those distributions [3]. Gaussian mixture models can also be seen as a k-means clustering. A GMM acts as a hybrid of

different uni-modal Gaussian models by using a discrete set of Gaussian functions, each with their own mean and covariance matrix. It improves the modeling capability [58].

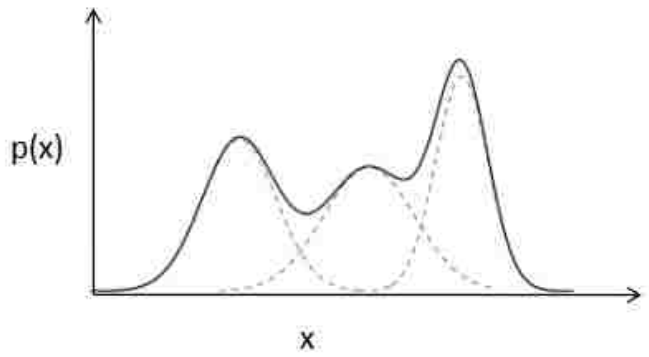

Figure 3.5: Mixture of Gaussian model

Here, we examine the distribution of the peak $\hat{g}'(t)$ values. As we have observed, during the simulation, an atom undergoes small random movements many times. And for some atoms, it may also undergo a few large movements. All movements lead to local maximums of $\hat{g}'(t)$ (showing as bumps in the plot of $\hat{g}'(t)$). Generally speaking, random movements do not travel large distance in a short period of time. Therefore, the local maximums of $\hat{g}'(t)$ due to random movements often have small values. If there is no large movement, the distribution of the local maximums will concentrate around small values and go to zero quickly once the value goes large.

When there are both random (oscillating) movements and large movements, the distribution of the local maxima becomes a double-mode distribution (i.e., two bell-shaped distributions summed together, the peaks of the two distributions may not be of the same height). Most local maximums (corresponding to random move) concentrate around small values (peak 1 of the distribution). A few local maximums (corresponding to the large movements) concentrate around large values (peak 2 of the distribution). One can separate the random movements from the large movements by a threshold whose value lies in between the two modes of the distribution.

In reality, we observe that $\hat{g}'(t)$ has many tiny peaks. Rather than considering all the local maximums, we further divide the time into windows, then examine each window and find the

value of the highest peak in that window. A collection of peak values are obtained in this way. We plot the distribution of these peak values in Figure 3.6. We observe that the frequency counts decreases dramatically when the value of the peaks increases. However, at a certain point, such fast decreasing trend is interrupted and inverted (the counts increase temporarily) while the value of the peaks goes large.

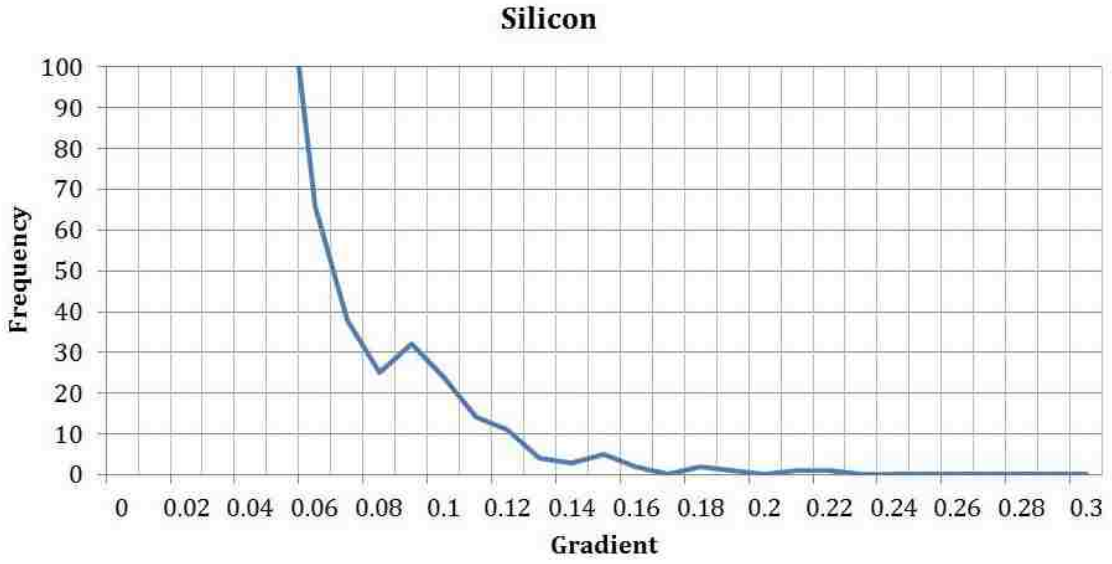

Figure 3.6: Histogram of peak values zoomed in to the portion where the decreasing trend of the curve is inverted.

We use a mixture of Gaussian model [50, 57] to model the combined distribution of the random move and the large movement, i.e., the probability of local maximum values, x, are defined to be:

$$P(x) = \sum_{z} P(x|z)P(z) \tag{3.3}$$

where z takes two values (Let $z = 0$ corresponds to the random moves and $z = 1$ the large movements respectively). $P(x|z)$ is a Gaussian distribution $N(\mu_z, \sigma_z^2)$ whose mean μ_z and variance σ_z^2 depend on the value of z. After fitting the probability density to the local maximum values, we obtain $N(\mu_0, \sigma_0^2)$ and $N(\mu_1, \sigma_1^2)$. The threshold is then set to be the value between μ_0 and μ_1 where the two distributions have the same densities. Once the threshold is obtained, the windows of large movements are detected as the windows in which the $\hat{g}'(t)$ values are above the threshold.

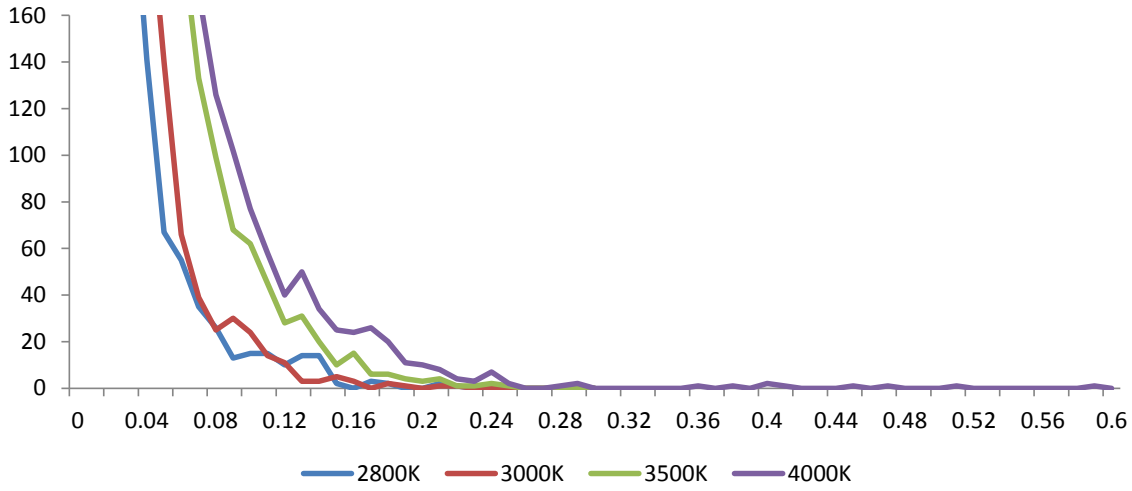

Figure 3.7: Peak-value histogram plot for four different temperatures. The plot shows the effect of temperature on the threshold values.

Figure 3.7 plots the distributions of the peak values of the speed of the moves (slop of the smoothed displacement curve). As discussed before, the regular random movements and the large movements concentrate on different peak values. On the distribution plot, the peak values of the large movements manifest as a bump on the rapid declining side of the distribution curve corresponding to the peak values of the regular random moves. Figure 3.7 shows that when simulation temperature increases, the peak values of both the large movements and the regular random moves shift higher. The regular random moves under high temperature give peak values larger than that of the large movements under lower temperature. This agrees with our observation in Figure 3.7 that a regular random move under 4000K may have the same or higher level of magnitude than a large movement under 2800K. The threshold for detecting large movements, therefore, is temperature dependent.

Weighted Average

We consider the derivative of the displacement (i.e. $\hat{g}'(t)$). Both moves lead to local maximum (local peak) of the derivative. The values of the derivative at these local maximums vary but in general the peak values corresponding to the regular oscillations are smaller than those corresponding to

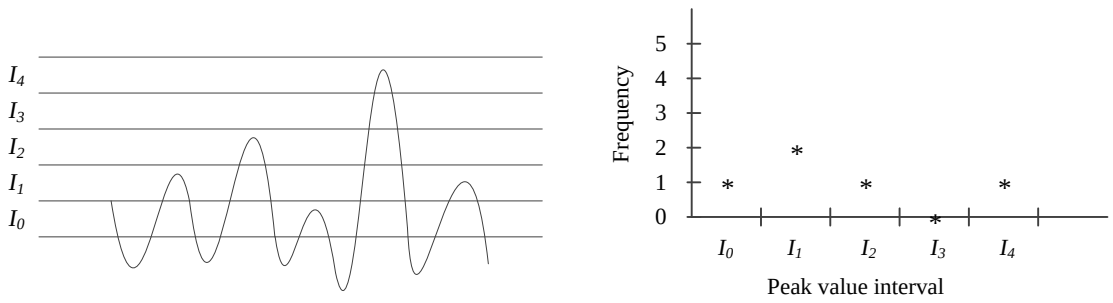

Figure 3.8: Synthetic example showing the counting of peaks (local maximums). Top: an example curve of derivative v.s. time is shown. The horizontal lines indicate 5 intervals. There is 1 peak in interval I_0 ($f_{I_0} = 1$), 2 peaks in I_1 ($f_{I_1} = 2$), 1 peak in I_2 ($f_{I_2} = 1$), 0 in I_3 ($f_{I_3} = 0$) and 1 in I_4 ($f_{I_4} = 1$). Bottom: plot of data points derived from peak counts. We simplify each interval into its middle value v and obtain a collection of data points $\{(v, f)\}$.

the large jumps. To distinguish the two types of move, we construct a model to represent the distribution of the (local) peak values caused by the regular oscillation. We then establish the threshold that separates the two types of move at the place where significantly more peaks are observed than what the model says for the regular oscillations. Those extra number of peaks are considered to come from the large jumps. To further simplify the matter, we view all the peaks above the threshold as caused by the large jumps because above the threshold, the number of peaks from regular oscillation is negligible.

In order to construct the model, our program counts the number of local peaks (local maximums) on the derivative ($\hat{g}'(t)$)-against-time curve for different ranges of peak values. Figure 3.8 gives a toy example on how the counting is conducted. We divide ($\hat{g}'(t)$)-against-time curve by the collection of k small, equal-length intervals $\{I_0, I_1, ..., I_k\}$. For each interval, the program counts the number of the peaks (local maximums of $\hat{g}'(t)$) whose value falls in that interval. Because the intervals are small, we represent each interval by its middle value. This gives us a set of points $\{(v, f)\}$ where v is the value representing the interval and f is the the frequency, i.e., number of peaks who reach that value. (Technically speaking, the peaks fall in the interval. For simplicity, we say that they reach the value representing that interval.) These points provide an approximation to the distribution of the peak values (the right side of Figure 3.8).

We model the trend in these data points (i.e. the relation between the value and the number of peaks reaching that value) using an exponential function:

$$\tilde{f}(v) = e^{Av+B} \tag{3.4}$$

where A and B are two model parameters. The calculated data points in the low peak value regime actually follow nice exponential trend at both 2800 K and 4000 K whereas the data at high peak values systematically differ from this trend.

We know that local maximums with very small magnitude come from regular oscillations. And similarly those with very large magnitude come from large jumps. To discover the trends in the distribution of the peak values caused by the regular oscillations, we determine the parameters (A and B) by fitting the model to the data points ((v, f) pairs) that have very small v (e.g., $v < 0.05$ at 2800 K). Those data points are considered to correspond to the regular oscillations. Figure 3.9 plots the data points (v, f) and the exponential model $\tilde{f}$. The blue data points (circles) have small v and are used to fit the exponential function (red line). The green data points (triangles) are not used for model fitting. We observe that following the trend (red line) displayed by the peaks caused by regular oscillation, the number of peaks of such type decreases quickly to zero when v increases. However, in some intervals, the actual count of peaks (some green points) can be much larger than what the trend indicates. This is because these actual counts include the peaks caused by large jumps. The more the count deviates from the trend, the more large jumps are presented. Our threshold for determining large jump is calculated following this view. Let τ be the threshold and Θ be the collection of data points that are not used in model fitting. We calculate the threshold as:

$$\tau = \frac{\sum_{(v,f)\in\Theta} v \cdot |(\tilde{f}(v) - f)|}{\sum_{(v,f)\in\Theta} |(\tilde{f}(v) - f)|} \tag{3.5}$$

We assume that the right most blue data point is the lower bound of threshold value. We use Equation 3.5 which is the weighted average of green data points (triangles) with respect to the fitting function, to find the optimal threshold (τ). Once the threshold is obtained, the windows of

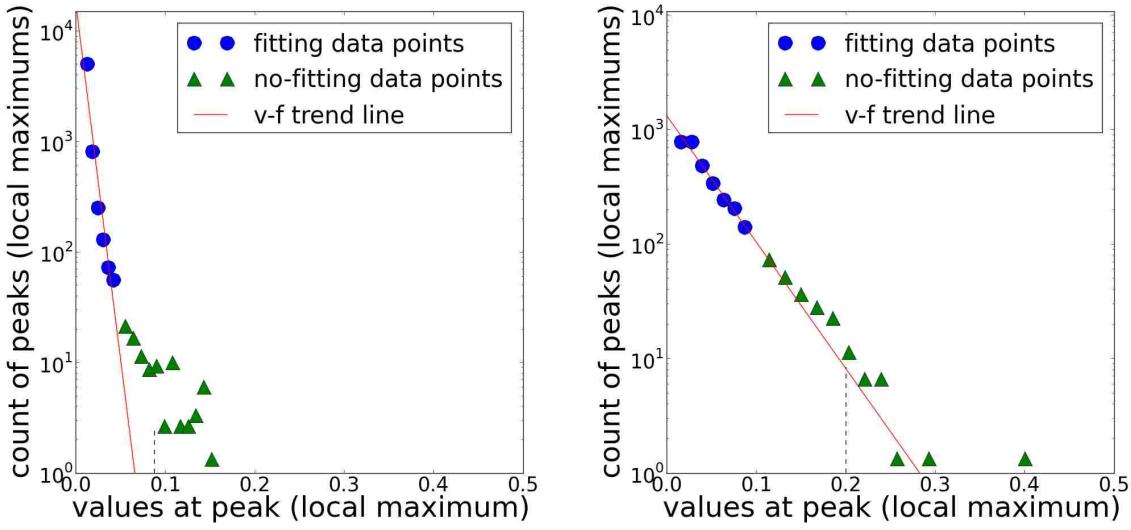

Figure 3.9: Distribution of the values at local maximum at two temperatures: 2800 K (a) and 4000 K (b). The circle (blue) and triangles (green) are count of peaks that fall in the corresponding intervals. The red curve is an exponential function that models the trend displayed by the circles, i.e., the distribution of the peaks caused by regular oscillation. Vertical black dotted line on X-axis represents the τ value. Note that the Y-axis is log-scaled.

large jumps are detected as the windows in which the $\hat{g}'(t)$ values are above the threshold. We then capture the time window where atoms shows large jumps using this threshold value.

Algorithm Description

Algorithm 1 gives the pseudo code for calculating threshold and detecting those time windows where atoms shows hopping motion. We start by loading raw data in D which contains the co-ordinates of all the atoms. For each atom, we calculate their displacement $g(t)$ and then gradient $\hat{g}'(t)$. We then count the local maxima which are the peak count in $\hat{g}'(t)$ and then stores them in P. P stores the peak count of all the atoms. After counting all the peaks we fit them with equation 3.4. We only use those peaks which have high frequency to fit equation 3.4. Now with those data points $\hat{P}$ which are not used in fitting, we calculate threshold τ using equation 3.5. Equation 3.5 is the weighted average of data points in $\hat{P}$ with respect to function 3.4. Once we have the threshold,

we can use it to find all big moved time windows W of all atoms. We later use W to count the coordination number of all the atoms which shows hopping motion during those time windows.

Algorithm 1 Threshold_Calculation

$D \leftarrow$ load data with coordinates of atoms

$C \leftarrow$ calculate RDF and generate coordination number of atoms

for each atom **do**

$g(t) \leftarrow$ calculate displacement from origin

$\hat{g}(t) \leftarrow$ perform smoothing on $g(t)$ using convolution function from equation 3.1

$\hat{g'}(t) \leftarrow$ calculate gradient of data $\hat{g}(t)$

$P \leftarrow$ calculate peak count on $\hat{g}(t)$-against time

end for

fit the points in P using equation 3.4

$\hat{P} \leftarrow$ points which are not used to fit equation 3.4

calculate threshold τ using points in $\hat{P}$ and equation 3.5

for each atom **do**

$W \leftarrow$ use τ to calculate big move time windows

end for

3.2.3 Coordination Changes Responsible for Large Movements

We define the structure of a given liquid system in terms of atomic coordination, which is a fundamental quantity used to characterize the local connectivity and can be calculated by counting the atoms of species β that lie within a sphere centered at atom of species α and of radius defined by the corresponding r_{min} value. For the silica liquid, we can have four types of coordination (Si-O, O-Si, Si-Si, and O-O). Only Si-O and O-Si environments directly involve Si-O bonding and are considered here. The mean Si-O and O-Si coordination numbers are close to 4 and 2, respectively. We can decompose each coordination environment into a variety of coordination species, which

unlike mean coordination takes an integer value. At each time step, we have complete information about the coordination states of all constituent atoms:

$$C^{i}_{\alpha\beta}(t) = |\{1 \leq j \leq N : d(i,j) \leq r^{\alpha\beta}_{min} \wedge type(j) = \beta\}| \text{for i} = 1.....N_{\alpha} \tag{3.6}$$

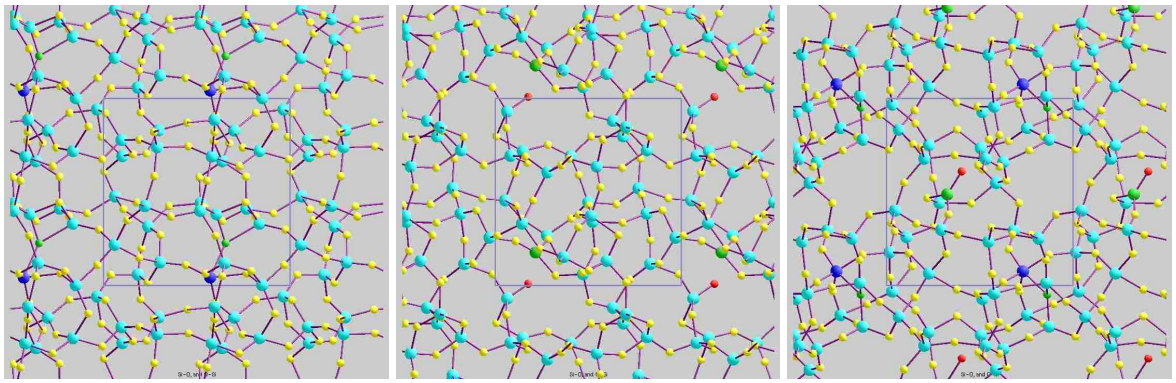

Figure 3.10: Three possible coordination configurations of Si and O atoms are shown. Large spheres are Si atoms and small are O atoms, the simulation box is also marked. The coordination states are color coded: 1(red), 2(yellow), 3(green), 4 (cyan) and 5 (blue). Left: five fold coordinated Si and three fold coordinated O atoms. Center: three fold coordinated Si and one fold coordinated O atoms. Right: three and five fold coordinated Si atoms, and one and three fold coordinated O atoms represented by their respective colors.

For the silica liquid considered in this study, the Si-O coordination consists of mostly four-fold species (called tetrahedral states) with five- and three-fold species also present. Similarly, the O-Si coordination consists of mostly two-fold (called bridging oxygen) with singly coordinated (called non-bridging oxygen) and three-coordinated species. The system is dynamic and the atoms undergo coordination changes with time. When a single bond event (formation or breakage of bond) occurs, a pair of new coordination states is formed. When a new bond is formed, a pair of over-coordinated species appears whereas when an existing bond is broken, a pair of under-coordinated species appears. These species disappear if the bond just formed breaks back or the same two atoms form the bond again.

Suppose that silica liquid assumes a perfectly tetrahedral network structure at some point of time. In other words, all Si atoms are in four-fold coordination state and all O atoms are in two-fold state. In such an environment, two types of coordination changes (events) are possible for Si

as well as O. A given Si atom changes from four-coordination state (4) to five-coordination state (5) and then returns back to four-coordination state (4). This sequence is referred to as 454. If the intermediate state is three-coordination state (3), the sequence is referred to as 434. Similarly, for O atoms, the coordination changes from two-fold state (2) to three-fold state (3) and then back to two-fold state (2). This sequence is referred to as 232. The other sequence in which the intermediate state is two-fold is referred to as 212.

For each sequence of coordination changes, more than two atoms may be involved. Once a pair of odd coordination species is formed, a different bond may break (in the cases of 454 and 232) or form (in the cases of 434 and 212) so that one of the original coordination states continues to exist and one additional state appears. More such coordination annihilation and formation events can occur in subsequent time steps until a perfect tetrahedral network is restored. This process results in a substantial atomic reconfiguration or rearrangement so that the participating Si and/or O atoms are likely to make substantial movements. In particular, odd coordination species (may be considered as coordination defects) appear to serve as the transition states for atomic self-diffusion. We may also view this process as migration of coordination species thereby causing the associated Si/O atoms to move from one coordination shell to another.

How often the Si or O atoms undergo such changes in their coordination state is thus expected to be relevant for the dynamical behavior. It is not clear which sequences of coordination changes are responsible for the diffusion. Not all coordination changes are relevant. For instance, a bond may break and then (the same bond) forms, resulting in a coordination-change event, which will not affect the atomic reconfiguration. Even if a different bond is formed, large atomic movement may not necessarily happen. Out of a large number of coordination changes that occur during the simulation, only a small fraction may actual contribute to the diffusion. Our goal is to automatically identify and quantify these coordination changes.

For every time window of large atomic movement we detect, we search for different sequences of coordination changes the atom under consideration undergoes and count the number of each sequence. The 454 and 343 sequences are considered for Si movement window whereas the 232

and 212 sequences are considered for O movement window. A time window can have more than one occurrence of coordination events. By doing this, we expect to answer the question: can large atomic jumps be described in terms of the coordination changes? In other words, can we identify the coordination changes, which are responsible for atomic diffusion?

3.3 Results and Discussion

3.3.1 Atomic Movement Patterns

Figure 3.11 presents a few examples of atom displacements under temperatures 2800K (three examples on the left) and 4000K (three examples on the right). A large movement is a displacement of magnitude larger than that of the regular random movements. The middle panel on the left shows a typical large movement and the bottom panel shows a typical atom without large movement. When temperature increases, the magnitude of the regular random movements also increases. A regular random movement under 4000K may have the same or higher level of magnitude as a large movement under 2800K. Under 4000K, A movement needs to display much higher magnitude to be considered as a large movement. (Note that in the plot, the scale of the displacement (Y axis) is different on the left from the ones on the right. Many movements on the right have higher magnitude than that on the left.)

Table 3.1: Percentage of hopping motion, which resulted in large diffusion.

	2800K	3000K	3500K	4000K	5000K	6000K
hopping motion (%)	82	91.6	100	100	100	100

As discussed earlier, we define two types of movements: flowing, in which small steps add up over a long time to give a large net displacement; and hopping in which almost all the distance traveled is accounted for by a few large jumps. Table 3.1 shows the percentage of hopping movements (among both flowing and hopping) occurred in 6 datasets. (We count only the flowing and the hopping movements because they account for significant displacements of the atoms. All atoms undergo random small movements that do not lead to significant displacement and are ignored in

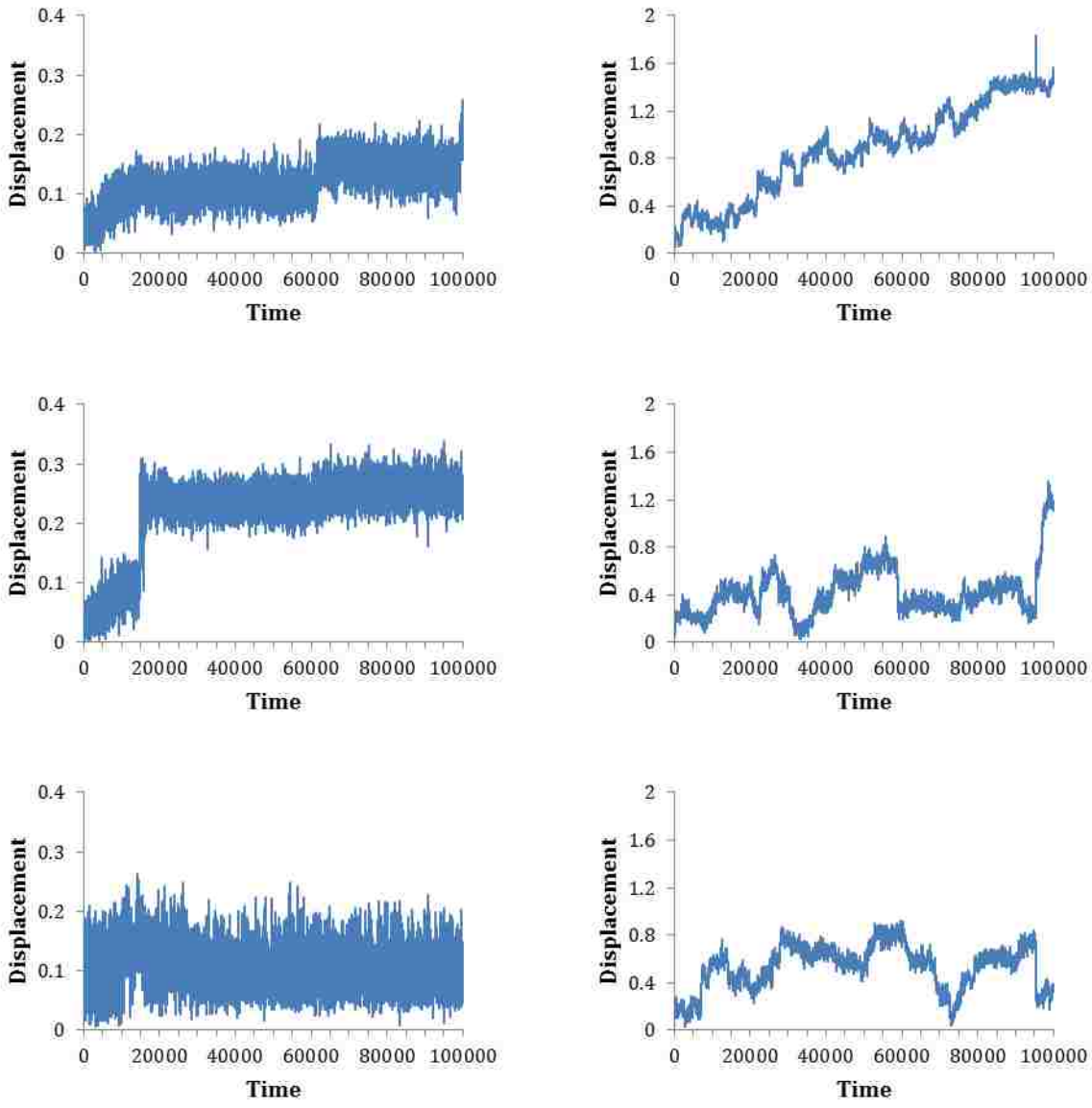

Figure 3.11: Three different types of movement for temperatures 2800K (left) and 4000K (right). As the degree of atomic movement increases with temperature, the maximum limit of displacement (X-axis) at 4000K is higher than 2800K .

this study.) Clearly hopping movement is the main source of displacement and therefore, may be the main contributor to diffusion. Hence we focused on the hopping movement.

Table 3.2: Threshold values for large movements of silicon and oxygen atoms for different temperatures.

	2800K	3000K	3500K	4000K
Si	0.08	0.1	0.11	0.2
O	0.09	0.13	0.15	0.2

3.3.2 Effect of Smoothing

Figure 3.12 shows the effect of applying smoothing with different parameters. The top row of the figure plots the displacements of an atom in the simulations under temperatures 2800K, 3000K and 4000K. The second, third and last rows show these curves after applying the smoothing method described in the previous section, under different choices of the parameter a (which defines the window size) in Equation 3.2a. When a takes a small value (second row), there is not enough smoothing. When a takes a large value (bottom row), the resulting curve becomes much smoother but the large movement also becomes less clear, i.e., the slope of the jump is no longer steep. The value for a we used in smoothing is 1000 time steps (middle row). Although there are still small perturbation, the curve is much smoother than the original displacement plot. At the same time, the large movements are preserved and become more prominent after the smoothing.

3.3.3 Threshold for Detection of Large Movements

Table 3.2 lists the thresholds determined by our program and used for the detection of large jumps at different temperatures. As expected, with increasing temperature, threshold value also increases. And at the same temperature, the threshold for the oxygen atoms tends to be somewhat higher than that for the silicon atoms. This is because oxygen atoms have greater mobility and undergo regular oscillations with larger magnitudes than those of the silicon atoms.

3.3.4 Coordination States and Changes

The mean Si-O and O-Si coordination numbers are close to 4 and 2, respectively, and they show slight monotonic increase with increasing temperature. Since both mean Si-O and O-Si coordi-

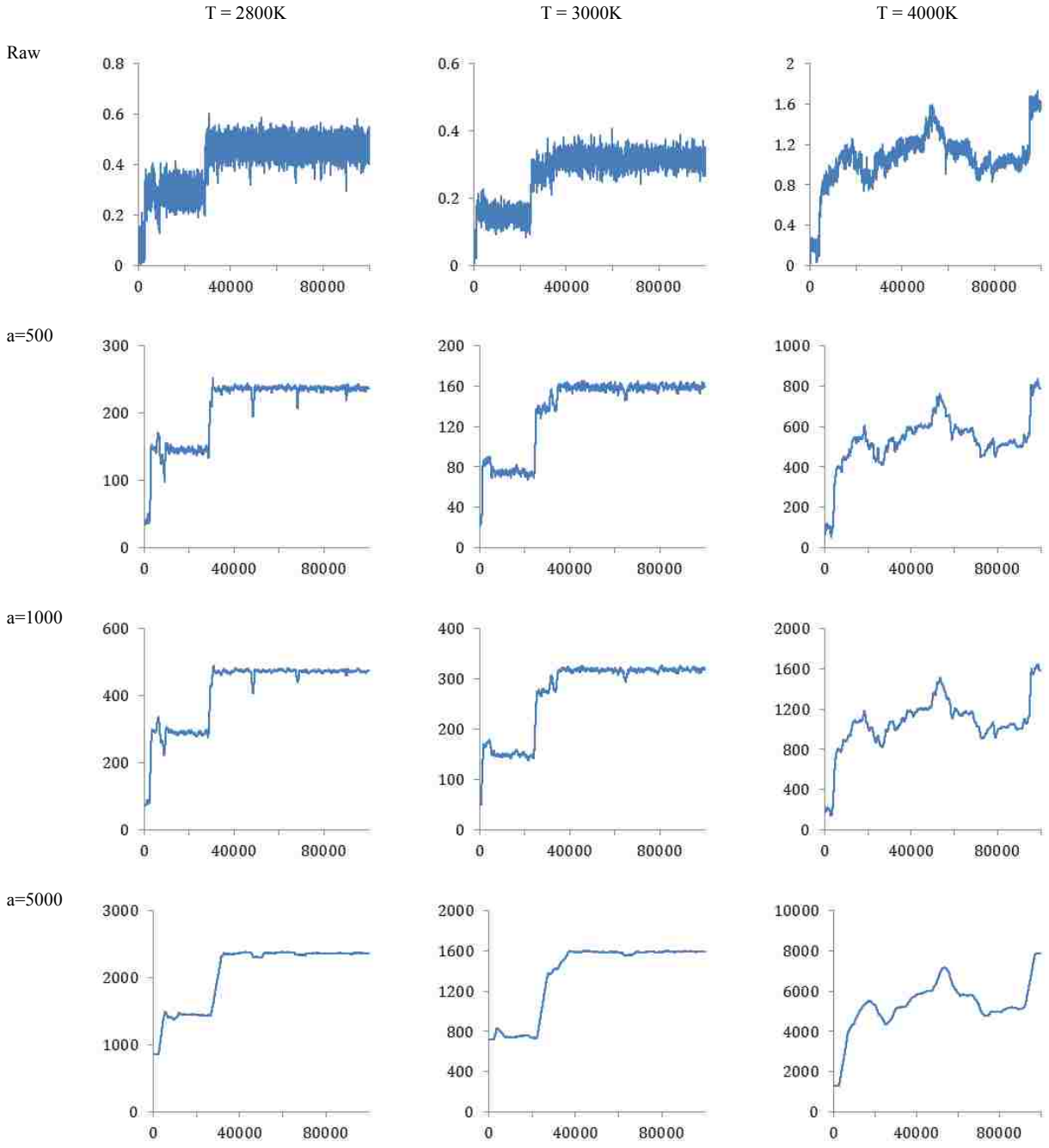

Figure 3.12: The effect of window size on smoothing for three different temperatures with time on X-axis and displacement on Y-axis. When a takes a small value (second row), there is not enough smoothing. When a takes a large value (bottom row), $\hat{g}(t)$ becomes much smoother but the large jump also becomes less clear, i.e., the slope of the jump is no longer steep. The value for a we used in the smoothing is 1000 time steps (middle row).

Table 3.3: Counts of Coordination changes over entire simulation time for different temperatures.

		2800K	3000K	3500K	4000K	5000K	6000K
Si	454	593	585	1941	3988	8478	11603
	434	180	228	919	1966	6127	10190
O	232	533	597	2025	4354	11022	17311
	212	175	219	889	1919	7258	14014

nation numbers differ from the ideal numbers of 4 and 2, respectively, the liquid structure cannot be a perfectly tetrahedral environment containing only bridging oxygen. The other coordination species (structural units) must be present in the liquid. At 2800 K, the Si-O coordination contains three- and five-fold species in addition to the dominant four-fold species (tetrahedra) whereas the O-Si coordination contains single- and three-fold species in addition to the dominant two-fold species (bridging oxygen). These odd species that are present in small amounts can be considered as coordination defects in an otherwise ideal tetrahedral environment. Temperature though it affects the mean coordination very little does influence the coordination distribution. With increasing temperature, the proportions of tetrahedra and BO decrease whereas the proportions of the odd species increase. The distribution becomes wider and more asymmetric extending more to higher coordination on heating with appearance of more defects at the cost of four-fold Si and two-fold O species. The silica liquid has 98.8% tetrahedra at 2800 K, which decreases to 85.7% at 4000 K. Even singly coordinated and octahedral Si atoms appear. Similarly, free O (not bonded to any Si) and four-fold coordinated O atoms are present. At 2800 K, the total amount of non-tetrahedral defects is 1.21%. This means that in the supercell consisting of 24 Si atoms, on average one non-tetrahedral defect will appear at an interval of 3.5 time steps during the simulation. About 70% snapshots should be free of any defect. In the reality, a defect survives over multiple time steps, and multiple defects co-exist at the same time. More snapshots are thus expected to be defect free.

Coordination change happens frequently during the simulation. Most times, coordination change has little impact on the atoms. A bond may break and some time later the same bond may reform. Such change does not lead to large atom movements. Even when an old bond breaks and a new bond forms, large atom movement may not happen. Given the premise that atom move-

Table 3.4: Counts of coordination changes within time windows of large movements for different temperatures.

		2800K	3000K	3500K	4000K	5000K	6000K
Si	454	3	3	21	107	1278	3495
	434	4	7	23	84	1089	3254
O	232	6	18	72	376	1926	4190
	212	6	18	84	250	1582	3705

ments, particularly the large movements are connected to the diffusive property of the material, it is important to exam the coordination changes that occurred together with large movements. In Table 3.3 and 3.4, we show the count of different types of coordination changes at different simulation temperature. Table 3.3 gives the counts of coordination changes during the whole simulation and Table 3.4 gives the counts for those that happened while the atom undergoes large movement. In both cases, the number of coordination change increases while the temperature of the simulation increases.

Consider the counts of "454" and "434" changes the Si atoms undergo in the time windows of large Si movements. At 2800 K, the numbers of "454" and "434" changes associated with such windows are 3 and 4, respectively. The numbers are much smaller than the total counts of these coordination changes. This means that tiny fractions of the total coordination changes actually cause the Si atoms to move by large amounts. It is interesting to note that the ratio of relevant "454" and "434" changes is 0.85 in time windows of large movements, compared to the high ratio of 3.3 of the total "454" and "434" changes (at temperature 2800K). The ratio in windows of large movements is much lower than the total ratio and this is so for all temperatures. This means that the "434" changes despite their low abundances are more involved in large atomic movements.

Finally, we observe that although for most large movements, the atom that moved also displayed coordination changes, there are a few large movements in which the atom moved does not experience coordination changes. At 2850 K, 90% large movement windows show coordination changes, most of the "454" and the "434" types. We further observed that for those windows in which O atom has moved significantly without undergoing coordination changes, one of its neigh-

boring Si atoms also shows large movement and goes through some coordination change. Hence some large movements may be a result of the neighbor's movement rather than the coordination change.

3.4 Conclusion

In this chapter, we have proposed a system to automate the detection of large atomic jumps in molecular dynamics simulations. The underlying algorithm employs methods from signal processing and uses a regression model to distinguish different types of motions. Automated analysis enables us to process massive amounts of simulation data to identify atomic activities that are relevant to material properties but are elusive to manual analysis due to the sheer size of the data need to be processed. Once the time intervals of such large movements are detected, the detail analysis of atomic structural changes (such as bonding and coordination) can be performed. While our automated analysis scheme is demonstrated for post processing of simulation outputs, it would be interesting to do in-situ analysis, which is particularly desirable for very large atomic systems. Performing data analysis while running simulations means that we do not need to save all simulation data. For instance, only time windows in which potentially interesting atomic events occur may be saved for further detailed analysis. In a real - time analysis scenario, one may run the simulation for a certain time to obtain a sample of data and use this sample to estimate appropriate mode parameters. Once the model is set up, the program can perform in-situ analysis. Moreover, our approach is scalable and suitable for parallel implementation. The detection of large atomic movements deals with atoms individually whereas the computation of structural changes as done here in terms of coordination states processes small groups of atoms which lie within finite space cutoff.

We have applied our automated analysis method to study the dynamical behavior of silica liquid using first-principles MD simulation data sets consisting of up to 1.3 millions steps. The detection of large jumps allows us to investigate the bond (coordination) changes that are more closely connected with atomic movements. For silicon atoms there are more 454-type coordination

changes (i.e. atom goes from a four-fold state to a five-fold state, and then comes back to the four-fold state) than 434-type changes in the overall simulation, but during large jumps there are almost equal amounts of 434 and 454-type coordination changes. Our analysis suggests that only a small fraction of coordination changes contribute to large atomic movements and the 434-type change may be more relevant than the 454-type. Similar observations were made with the oxygen atoms. Our automated analysis system can play an important role in obtaining such observations and insights, particularly, for the liquid phase at relatively low temperatures. The detection of large atomic movements makes a lot of sense for dynamical systems like silica liquid studied here. The dynamics in solid phases is generally extremely slow to capture any such large atomic jumps during finite MD simulations that are currently feasible. However, defects and disorder in crystals cause local structural distortions and they tend to enhance local mobility. Our approach is expected to be applicable to such cases with appropriate thresholds on atomic displacements, bond length distortions, bond-angle distortions, etc. defined relative to the corresponding system-wide mean values.

The detection of large movements allows us to investigate the bond changes that are more closely connected with atomic movements. We observe that, although for silicon atoms, there are more ”454”-type coordination changes than ”434”-type changes in the overall simulation, during large movements, there are equal amount of ”434” and ”454”-type changes. This suggests that only a small fraction of coordination changes contribute to atomic move and the ”434”-type change may be more important to atomic move than the ”454”-type. Similar observation is made with the oxygen atoms. These observations can provide insights for deriving models of atomic diffusion. Our automated analysis system can play an important role in obtaining such observations and insights.

Chapter 4
Graph Mining and Pattern Discovery in Atom Dynamics

4.1 Inroduction

In this chapter, we address the problem of discovering patterns in atomic dynamics for material study. We propose an event graph to model the atomic activities. Different from atom graph, the nodes in the event graph represent an event in the atomic dynamics. The edges in the graph represent dependency relationships among the events. Patterns in the atomic dynamics can then be mined as patterns in the event graph. This provides a tool for material scientists to identify interesting connections of atomic activities that may be useful for modeling and predicting material properties.

In many cases, event graph is a directed acyclic graph (DAG) and the interesting patterns are frequent subgraphs of the event graph. We propose a mining algorithm for discovering frequent subgraphs in DAGs. The foundation of our algorithm is a canonical graph encoding that uses a numbering scheme based on a modified version of the topological sort. Frequent subgraph mining is a popular topic in data mining and many algorithms have been proposed [19, 35, 47, 49, 68, 73, 74, 77]. A large fraction of the existing algorithms uses a certain numbering scheme to build a unique encoding of graphs that is resistant to isomorphism. For example, gSpan [73] uses a DFS-based numbering. We choose topological sort based numbering because in the underlying graph, the edges are used to model dependency relationships. Topological sort is a natural choice in such setting as it respects the dependencies. To the best of our knowledge, our algorithm is the

first to utilize topological sort in the numbering scheme for subgraph mining. We further extend the topological ordering technique such that, we can have total order of all nodes in a given graph.

We also take advantage of the topological sort in generating candidate subgraphs during the mining process. Our algorithm follows the approach of subgraph-growth in candidate generation. However, the generation process is limited to a growth tree that has no duplication. The growth-tree is designed using the topological-sort-based encoding. We show that each possible candidate is explored once and only once on the growth tree. Hence the number of subgraphs that are examined by our algorithm to see whether they are frequent is no more than that of any other algorithms based on the subgraph-growth approach. We implemented and tested our algorithms on simulations of the material silica (SiO_2). The results show the potential of our mining technique for material-science discoveries.

The main contributions of our work are:

- We proposed an event graph model for modeling atomic dynamics in material simulations
- We considered subgraph mining in DAGs and proposed a mining algorithm based on topological sorting. We show that there is no duplication in the candidate subgraphs examined by our algorithm when searching for frequent subgraphs.

4.2 Event-Graph Model for Atom Dynamics

We describe our event-graph model in this section. We remark that the event-graph model and the graph-mining algorithm proposed are both applicable to simulation data of different types of materials and different types of atomic activities. However, in this section, we will focus on the Silica (SiO_2) liquid system [32] and examine the activities of bond forming and breaking between the silicon and the oxygen atoms in the system.

Many materials studied by material scientists have regular or semi-regular structures. For example, under a wide range of temperatures and pressures, the atoms (silicons and oxygens) in silica form a tetrahedral structure. Each silicon atom is bonded with 4 oxygen atoms and each oxygen

atom is bonded with 2 silicon atoms. Because the atoms undergo constant movements, this regular structure is not always hold. Over time, bonds may form and may break. For a period of time, a silicon atom may change its bond state from bonding with 4 oxygen atoms to 3. An oxygen atom may change its bond state from 2 to 1. (We name these states by the number of bonds that the atom has. If a silicon bonds with 4 oxygen atoms, we say it is in state 4.) Change of bond state happens to all the atoms over time. And there are multiple possible bond state for both silicon and oxygen, particularly when the material is under high temperature or high pressure.

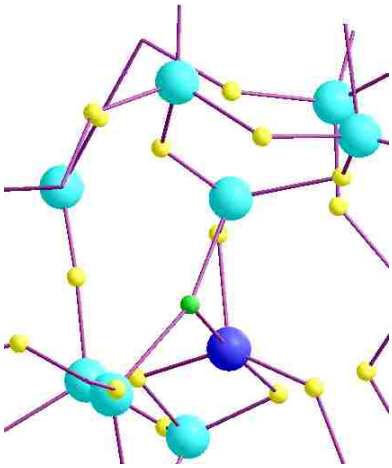

Figure 4.1: A snapshot of the atoms and their bonds in the simulation of SiO_2 system. Silicon atoms are represented by large spheres and the oxygen atoms are represented by small spheres. Silicon atoms in regular bond state (bonding with 4 oxygen atoms) are colored in cyan and those in non-regular bond state are colored blue. (There is one silicon atom in state 5.) Oxygen atoms in regular bond state (bonding with 2 silicon atoms) are colored in yellow and those in non-regular bond state are colored green. (There is one oxygen atom in state 3.)

Figure 4.1 shows an example structure of the silicon and the oxygen atoms in the material SiO_2. Silicon atoms are represented by large spheres and the oxygen atoms are represented by small spheres. Silicon in regular bond state (bonding with 4 oxygen atoms) are colored in cyan and those in non-regular bond state are colored blue. Oxygen atoms in regular bond state (bonding with 2 silicon atoms) are colored in yellow and those in non-regular bond state are colored green. Most silicon and oxygen atoms are in regular state and form tetrahedral structures. There is one silicon atom in state 5 and one oxygen atom in state 3.

Atomic dynamics involving change of bond state are studied intensively by material scientists. This is because in many cases, the micro (atomic) level structure and activities determine the macro-level properties of the materials. Often times, it is not an isolated activity (e.g., a single bond state change) but a series of related activities that form the underlying mechanism determining a particular material property.

For example, in silica, the atoms in the tetrahedral structure (state 4 silicon and state 2 oxygen) are difficult to move. A state 3 (or in some cases state 5) silicon atom is more possible to move a relatively large distance. The viscosity of the silica are thus related to atoms in non-regular bond state. Observe that change of bond state involves two atoms. If a silicon atom goes from state 4 to 3, it looses one of its neighboring oxygen atom. Correspondingly that oxygen atom also looses one silicon neighbor and therefore changes its state. There are multiple possible changes of state this oxygen atom can experience. One is that it goes from state 3 to state 2 and the other is it goes from state 2 to state 1. If the oxygen atom goes from 3 to 2, there must be a change of state (2 to 3) in the past that led this atom to be in state 3. We say that the later change of state (3 to 2) is dependent on the early change of state (2 to 3) as without the early one, the later one is not possible. (There is a small technical detail in the definition of the dependency here. From a material science point of view, changes that involve forming and then breaking the same bond are of little interest. For two changes of state to be considered as dependent, we require that the bond formed in one change of state be different from the bond broken in the other.) There can be more dependencies. The early change of state (2 to 3) itself involves a silicon atom. This change of state may be dependent on some other changes of state by that silicon atom in the past. In general, there may be a set of interdependent changes of states that lead to situations in which the atoms are more likely to move and hence contribute to the viscosity of the material. The patterns among such interdependent state changes are thus of interest for material scientists when constructing models that can explain or predict such property of silica liquid.

We use an event graph to model these atomic dynamics.An event graph can be represented as $G = (V, E)$, where *V is a set of nodes $E \subseteq V \times V$ is a set of edges*. Each node in V correspond

to an event and the edges in E correspond to the dependency relationship between the events. (We abuse names and use the term "node" and "event" interchangeably when describing event graphs.) The edges of the graph are directed and the direction goes from an event to the other that it depends on. The number of incoming and outgoing edges varies from problem to problem. In general, a node in an event graph could have multiple incoming and outgoing edges. For the bond events, there are at most two incoming edges and at most two outgoing edges. This is so because each event involves only two atoms and the dependency is based on each atom's current and previous bond state. However, the event-graph model itself does not impose such limitation. The graph model can be applied to other events in atomic dynamics where different numbers of incoming and outgoing edges may arise.

The event graph has some properties:

1. The graph is a directed acyclic graph (DAG). Because the dependency relationship between two events are sequential in time (x dependent on y means x happens after y), the directed graph is acyclic.

2. There is an order, based on the types of the events, among the out neighbors of each event.

An event in the event graph describes an occasion of bond forming or bond breaking. It involves two atoms (a silicon and a oxygen in silica). Let $S = \{s_1, s_2, s_3, ...\}$ be the collection of silicon atoms and $O = \{o_1, o_2, o_3, ...\}$ the collection of oxygen atoms. We denote by C_s the set of possible changes of state that a silicon atom can undergo and C_o the possible changes an oxygen can undergo. Note that we can impose an order $\prec_{cs}$ on the elements in C_s ($\prec_{co}$ for C_o). Let $\alpha_1 = (i_1, i_2)$ and $\alpha_2 = (j_1, j_2)$ be two state changes such that $\{\alpha_1, \alpha_2\} \subseteq C_s$ then we can order them as $\alpha_1 \prec_{cs} \alpha_2$ if one of the following holds

i) $i_1 < j_1$ ii) $i_1 = j_1$ and $i_2 < j_2$

Suppose for silicon atoms, the set of changes of state we are interested in are {(3 to 4), (4 to 3), (4 to 5), (5 to 4)}. They can be ordered by comparing the state before the change and if the states are equal, the state after the change (e.g., (3 to 4) $\prec_{cs}$ (4 to 3) and (4 to 3) $\prec_{cs}$ (4 to 5)).

An event is a five-tuple (s, o, cs, co, t) where $s \in S$ and $o \in O$ are the two atoms. $cs \in C_s$ and $co \in C_o$ are the changes of bond state happened to the two atoms in the event at time t. Although each node in the graph can be uniquely identified by the two atoms involved in the event plus the time when the event happened, what is more important to material scientist is the type of the nodes: it is more interesting to find patterns such as a silicon atom goes to state 3 and then back to state 4 rather than identifying whether it's silicon number 6 or silicon number 8 that goes through such change of state. Therefore, we label each node in the graph by the type of the corresponding event. The type of an event is a two-tuple $(cs, co) \in C_s \times C_o$. This gives the complete node set of the event graph

$V = \{e : \forall e \in \{(C_s, C_o) \in C_s \times C_o\}\}$

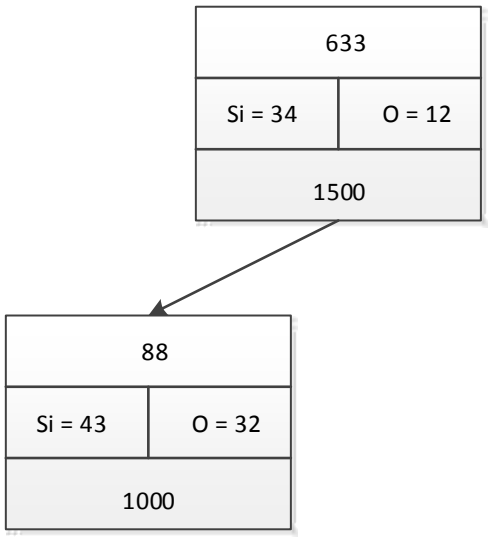

Figure 4.2: Detail structure of event (node).

Figure 4.2 is a subgraph, which shows the detail information of an event. Each event consist of three rows of information. Top row is for unique identifier of the event. As we have discussed earlier an event consists of two atoms which go under bond change, the middle row is for those two atoms. $Si = 34$ and $O = 12$ denotes that Si atom went through (3 to 4) bond change and O atom went through (1 to 2) bond change. The last row is the time stamp at which the corresponding event occurred. In the figure node 633 is the source node and it is dependent on the node 88. In

another word node 88 is causing the node 633 to happen. The figure also says that the Si atom went from 4 to 3 coordination state at time stamp 1000 and then same Si atom went from 3 to 4 coordination state at time stamp 1500.

Note that in general, if both P and Q are ordered set, then an order can be established on the set of the tuples in $P \times Q$. Given the orders for C_s and C_o. A complete order on the types of event can be obtained. We denote by $x \prec_t y$ if the the type x is before the type y in the order.

The atom dynamics concerning bond changes can be fully captured by the event-graph model. Once an event graph is constructed for the atom dynamics of a particular material simulation, the interdependent changes of bond state are the subgraphs in the event graph. We are interested in the frequent subgraphs because they are the candidates for material scientists to exam in order to identify atomic level mechanisms for predicting material properties. Two subgraphs will be counted as two copies of the same subgraph if they are isomorphic, i.e., the types of events in the two subgraphs are the same and the dependency relations among the events are the same too.

Often time, material scientist are interested in the patterns of atom dynamics that lead to a particular type of events. Our subgraph mining focuses on this type of problems and considers only subgraphs that has a single source node and the node is of a specified event type. In an event graph, every node is causally dependent with each other. If there is a path from node a to node b then we say b is causing a. We only consider such subgraphs where all nodes are causing the source node. We call such subgraph a Restricted Dependency Subgraph (RDS)

Definition 2 Restricted Dependency Subgraph (RDS)*: A RDS is a single-source directed acyclic graph. Let v_0 be the only source node. For every node in the subgraph, there is a path from v_0 to that node in the subgraph. Let $e(a \rightarrow \{n\} \rightarrow b)$ where $n = \{\phi\} \cup \{V\}$, gives the path from node a to node b going through n different nodes. Then following condition is true in RDS*
$\forall i : \exists e(v_0 \rightarrow n \rightarrow b_i)$

Problem Definition: Given an event graph of atom dynamics, an event type t, and a threshold T, find all the RDS whose source node is of the type t and whose frequency is above the threshold.

Formally, let $\sigma(g, t)$ is the occurrence frequency of RDS $g \in G$ with source node of type t. Now Frequent RDS Mining is to find every subgraph g, such that $\sigma(g, t) \geq T$.

4.3 Frequent Subgraph Mining

Extraction of useful information from data is known as data mining. With the increasing rate and complexity of data, there is urgency in the development of data mining algorithms in big data. Furthermore most of the data we find today are structured in nature. Hence, there exists a need to develop tools which could mine such structural database. A graph, as a general data structure can represent wide spectrum of data and also the complicated relations among them [5, 41].

Graphs are becoming important in modelling information which is making graph mining equally essential research area within the domain of data mining. The field of study is focussed on mining frequent subgraphs within the graph data sets. The research is mainly concentrated on: (i) optimal way of generating candidate subgraphs (without generating duplicates) and (ii) best way to identify the interesting frequent subgraphs in a computationally efficient way [5, 41].

Few popular research domain on graph mining are listed below:

- Frequent subgraph mining [20]
- Correlated graph pattern mining [43]
- Optimal graph pattern mining [72]
- Approximate graph pattern mining [44]
- Graph pattern summarization [71]
- Graph classification [36]
- Graph clustering [25]
- Graph indexing [63]

- Graph searching [75]
- Graph kernels [28]
- Link mining [17]
- Web structure mining [46]
- Work-flow mining [27]
- Biological network mining [34]

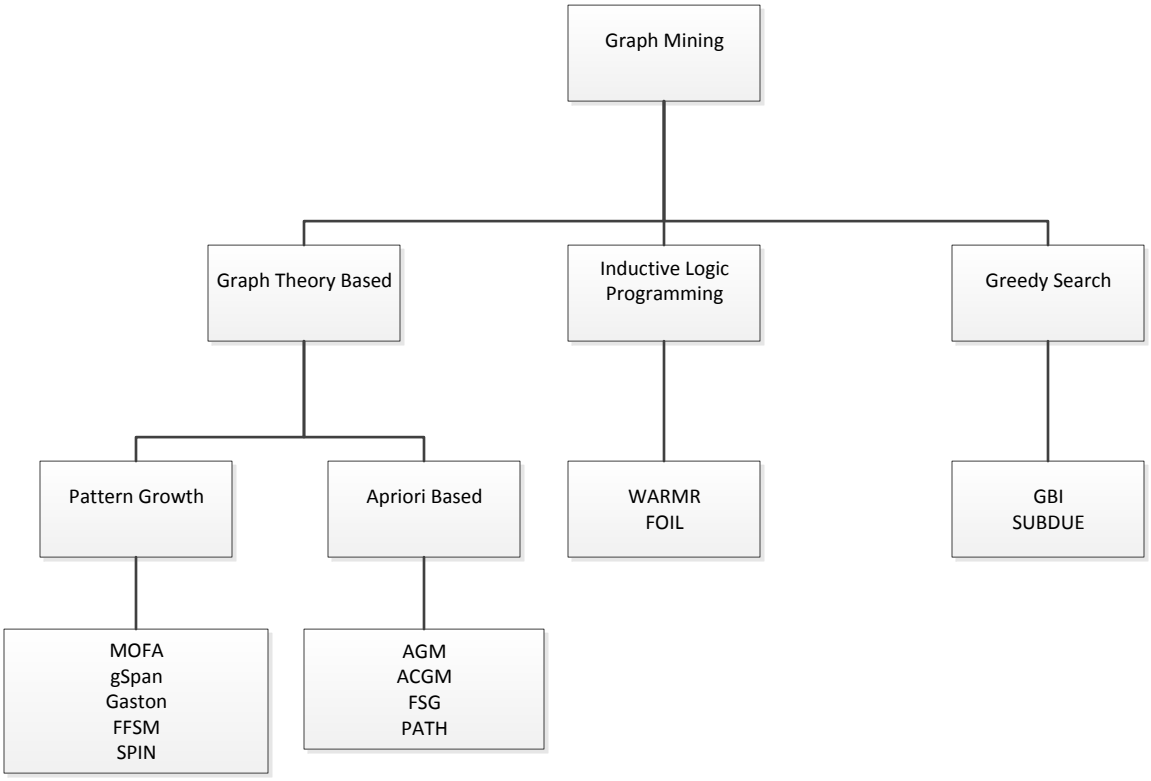

Figure 4.3: Categorization of graph mining algorithms.

Frequent Subgraph Mining (FSM) is the integral part of graph mining. In FSM we find all the frequent subgraphs from a given data set, whose occurrence counts are above a specified threshold. The basic idea behind FSM is the candidate subgraphs generation in either a depth first or breadth

first manner, and then identify if the candidate subgraphs are frequent enough to be considered interesting. The three main research issues in FSM are thus how to efficiently and effectively (i) generate the candidate frequent subgraphs and (ii) solve the graph isomorphic problem iii) counting the frequency of subgraphs which are above threshold. Ideal candidate subgraph generation requires that the generation of duplicates are avoided. Various encoding techniques are used to identify isomorphic graphs followed by effective frequency counting of subgraphs. FSM, in many respects, can be viewed as an extension of Frequent Itemset Mining (FIM) [7]. Therefore many proposed techniques in FIM is applied in FSM. For example apriori based algorithms. Figure 4.3 shows the categorization of graph mining algorithm [5, 41].

4.3.1 Formalism of Frequent Subgraph Mining

Frequent subgraph mining techniques can be categorized into two categories: (i) Apriori-based approaches, and (ii) pattern growth-based approaches. Apriori based share similar characteristics with Association Rule Mining (ASM) [7]. The search is based on Breadth First Search (BFS) to explore subgraphs in given database. It starts with the graph of small size and proceeds in bottom-up manner. Therefore, before considering subgraphs of (k + 1) size, this approach has to first consider all subgraphs of size k. Here different subgraphs of size k are combined to generate the candidate subgraphs of size (k + 1). Consequently, large number of candidate subgraphs are generated. Therefore to avoid the overhead of apriori algorithm pattern growth approach algorithms have been developed. The pattern-growth approach extends the frequent subgraph by adding one edge in every possible position [51]. The distinction between two approaches is shown in Figure 4.4.

Canonical Representation

The most common way of representing a graph structure is by employing and adjacency matrix or adjacency list. In an adjacency matrix row i and column j represent the vertex of graph and the intersection of i and j represents edge(s) connecting v_i and v_j. The value of intersection $< i, j >$ indicates the number of edges between v_i and v_j. Although, the adjacency matrices is the most

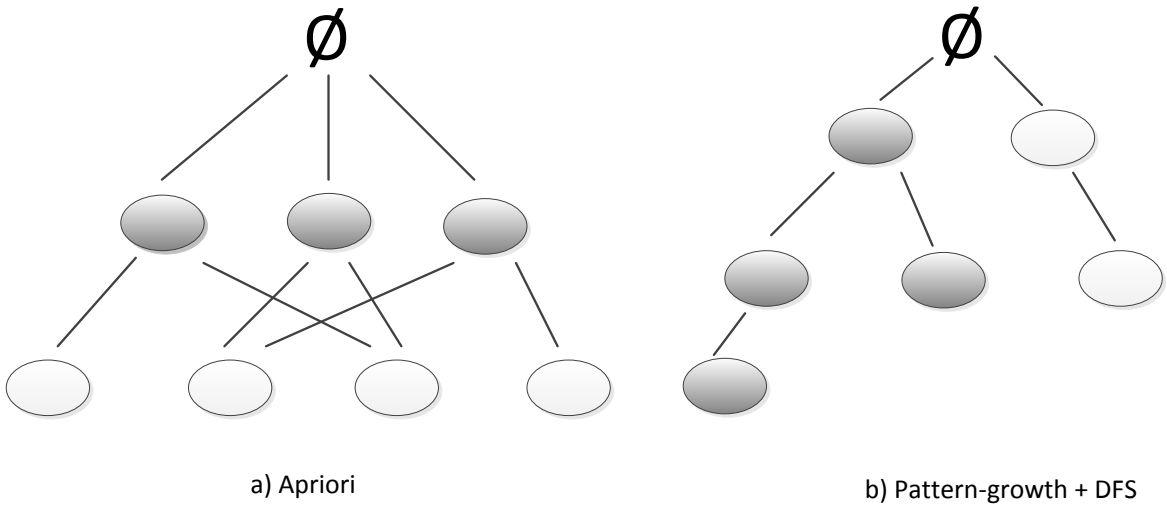

Figure 4.4: Types of search space: apriori based and pattern growth based

popular way of representing a graph, it does not helps in isomorphism detection since a graph can be represented in many different ways depending on how vertexes are enumerated [70]. To identify isomorphic graph is it necessary to adopt a consistent labeling strategy that ensures no two isomorphic graph are labeled in different ways. This is known as canonical labeling strategy [41].

Below are listed few canonical labeling strategies:

- Minimum DFS Code (M-DFSC).
- Canonical Adjacency Matrix (CAM).
- DFS Label Sequence (DFS-LS).
- Depth-Label Sequence (DLS).
- Breadth-First Canonical String (BFCS).
- Depth-First Canonical String (DFCS).
- Canonical Representation of Free Trees.

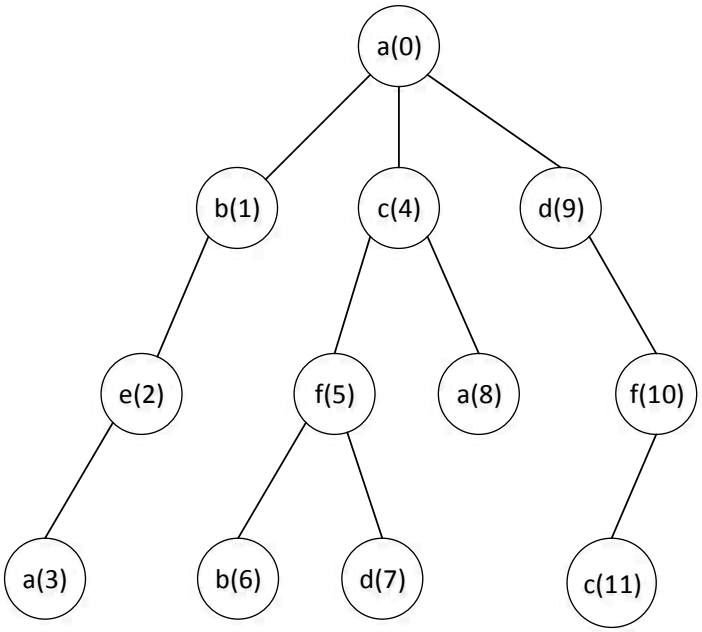

Figure 4.5: A labeled ordered tree T

Minimum DFS Code (M-DFSC) There are various way of DFS encoding. In every technique each vertex is given a unique identifier generated from a DFS traversal. Each edge of the graph in the DFS code is represented by a 5-tuple: (i, j, l_i, l_e, l_j), where i and j are the vertex, l_i and l_j are the labels of vertex and l_e is the label for the edge. Using DFS lexicographic order, the MDFSC of a graph g can be represented as the canonical labeling of g [73].

Given a graph G, $Z(G) = Code(G, T) \mid \forall T$, T is a DFS tree for G, based of DFS lexicographic order, the minimum one, $min(Z(G))$ is called the Minimum DFS code of G. It is also the canonical label of G [41, 73].

Canonical Adjacency Matrix (CAM) For a given adjacency matrix M of a graph g, encoding of M can be generated by concatenating the lower (or upper) triangular entries of M, including entries on the diagonal. The different permutation of the set of vertices corresponds to different adjacency matrix, therefore CAM of g is defined as the maximal or minimal encoding [38, 41].

DFS Label Sequence (DFS-LS) During a DFS traversal of labeled ordered tree T, the labels of $\forall vi \in V$ are added to a string S. Whenever backtracking occurs, a unique symbol is added to S. Those symbols can be -1 or $ or / [81]. The DFS-LS code for the example tree given in Figure 4.5 is represented as {abea$$$cf b$d$$a$$df c$$$} [41].

Depth-Label Sequence (DLS) During a DFS traversal of labeled ordered tree T, depth-label pairs consisting the depth and label $\forall vi \in V$, $(d(v_i), l(v_i))$, are added to a string S. Now T can be represented as the depth-label sequence which is given by $S = \{(d(v_1), l(v_1)), \ldots, (d(v_k), l(v_k))\}$ [10]. The DLS code for the example tree given in Figure 4.5 is given as (0, a), (1, b), (2, e), (3, a), (1, c), (2, f), (3, b), (3, d), (2, a), (1, d), (2, f), (3, c) [41].

Breadth-First Canonical String (BFCS) Given a labeled ordered tree T, every vertex label is added to astring during a BFS traversal of T. A $ symbol is used to partition the siblings of a node and a # represents the end of string encoding [18]. The BFCS of T is the lexicographically minimal of these encoding. Therefore, the BFS string encoding of tree in Figure 4.5 is given as abcde$f a$f$a$bd$$c#.

Depth-First Canonical String (DFCS) It is similar to BFCS but using DFS. Then minimal encoding of labeled ordered tree T, is then the DFCS of T [18]. The DFS encoding of tree in Figure 4.5 is given as abea$$$cfb$d$$a$$dfc$$$# [41].

Canonical Representation of Free Trees Trees with no root nodes are free trees. In this representation, a unique vertex is selected as the root to construct a free tree. At first all leaf vertexes and their incident edges are recursively removed until a single vertex is left. The remaining vertex is called the center. A rooted unordered tree is obtained with the center as the root [41].

Candidate Generation

Candidate generation is an essential phase in FSM. The primary focus on candidate generation research is to be complete and generate less redundant candidate subgraphs as much as possible. Many frequent-subgraph mining algorithms run in iterations or use recursions. During each iteration or recursion, it generates candidate subgraphs and then counts the frequencies of these candidates. Only those whose frequency is above the threshold are used in candidate generation for the next iteration. There are two approaches to generate candidate subgraphs: merging and growth. In merging, two frequent subgraphs are merged to generate candidates. The growth approach starts with empty graph and generate candidate by grow the graph one edge at each step [41].

Below are listed few strategies used in candidate generation [41]:

- Level-wise join.
- Rightmost path extension.
- Extension and join.
- Equivalence class based extension.
- Right-and-left tree join.

Level-wise join The level-wise join strategy was introduced by Kuramochi and Karypis (2001) [47]. Here, two frequent k subgraphs which share the same (k - 1) subgraph, are merged to generate a (k + 1) subgraph candidate. The common (k - 1) subgraph is the core for these two frequent k subgraphs. The main problem with this strategy is that one k subgraph can have at most k different (k - 1) subgraphs. Moreover, the joining operation produces many redundant candidates. Which makes this approach highly inefficient. Later they solved this issue by limiting the (k - 1) subgraphs to the two (k - 1) subgraphs with the smallest and the second smallest canonical labels. By using this join operation, the number of redundant candidates generated was significantly reduced [41].

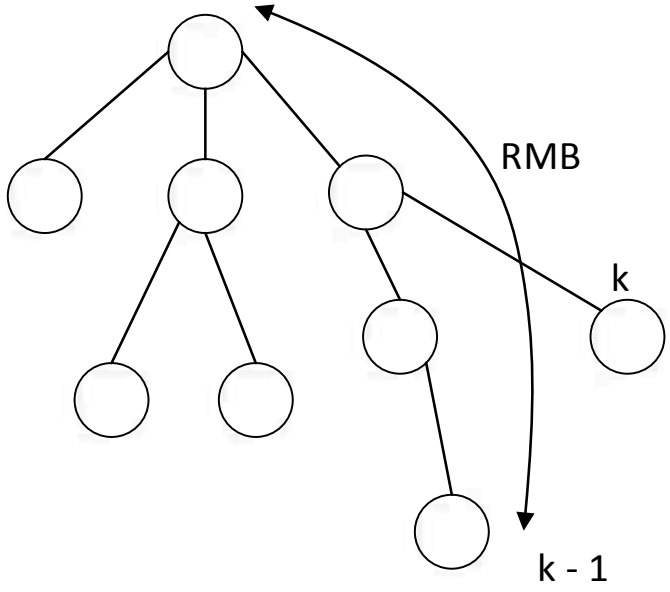

Figure 4.6: Right most branch expansion.

Rightmost Path Extension Right most path extension is the most popular candidate generation strategy. It generates (k + 1) subtrees by adding an edge to the Right Most Branch(RMB) of k subtrees. In figure 4.6 RMB denotes the right most branch, which is the path from root ϕ to the right most leaf (k-1). In tree generated by using Right Most Path extension, each node is a subtree pattern. A node S is connected with another node T if and only if T is added to the RMB of S. Every 1-subtree is the right most expansion of root ϕ and every (k + 1)-subtree is the right most expansion of k subtrees. Therefore all subtree patterns in the lattice can be enumerated by either Breadth First Search of Depth First Search. The enumeration tree (enumeration DAG) formed by right most path expansion is sometimes used to illustrate how a set of pattern is completely enumerated in search problem [41]. The enumeration DAG have been used extensively in Association Rule Mining and many Subtree Mining algorithm.

Extension and join The extension and join strategy was first introduced in [35]. It uses a BFCS representation; where a leaf at the bottom level of a BFCF tree is referred as a leg [41]. For a node V_n in an enumeration tree, if the height of the BFCF tree corresponding to V_n is assumed to be h

the all child nodes of V_n can be generated by either of the following two operations:

1. Extension Operation: By adding a new edge at the bottom level of the BFCF tree which produces a new BFCF with height h + 1.

2. Join Operation: By joining V_n with one of its sibling nodes, which yields a new BFCF with height h.

Equivalence Class Based Extension Equivalence class based extension was first used in [82] and is founded on a DFS-LS representation for trees. In this approach, a (k + 1)-subtree is generated by joining two frequent k-subtrees of same equivalence class. Two k-subtrees T_1, T_2 are in the same prefix equivalence class if they share the same encoding up to the (k - 1)-th vertex. Each member of the class can be represented as a two tuple (l, p), where l is the k-th vertex label and p is the depth-first position of the k-th vertexs parent. It is proved that all (k + 1)-subtrees with the prefix of size (k - 1) can be generated by merging each pair of members of the same equivalent class [41, 82].

Right-and-left Tree Join The right-and-left tree join strategy was proposed by Hido and Kawano (2005). It basically uses the rightmost leaf and leftmost leaf of the tree to generate candidates in a BFS manner [41].

Graph Isomorphism Detection

Graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ are isomorphic if there is an bijective function F from V_G to V_H such that for all nodes u and v in V_G ,

$$(u, v) \in E_G \quad \text{if} \quad \text{and} \quad \text{only} \quad \text{if} \quad (F(u), F(v)) \in E_H \tag{4.1}$$

Informally, if two graphs G and H are isomorphic then the nodes of one graph can be rearranged (without deleting or adding any edges), so that two graphs are identical, without considering labels on the node. Graph isomorphism problem fall in NP problem. No NP-completeness

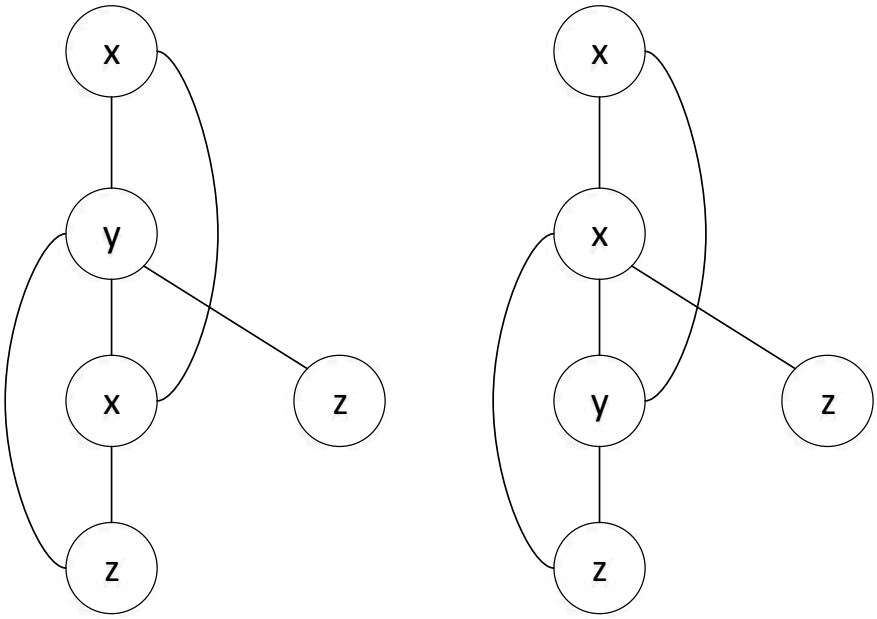

Figure 4.7: Example of two isomorphic graphs.

proof is known yet. Every algorithm known yet are exponential. If two graphs are isomorphic they must have,

- the same number of vertices
- the same number of edges
- the same degrees for corresponding vertices
- the same number of connected components
- the same number of loops
- the same number of parallel edges.

Also, if both graphs are connected/not connected and if one graph has a pair of connected vertices then other graph must have the corresponding pair of vertices connected for them to be isomorphic graphs. In general it is easier to prove any two graphs are not isomorphic. Therefore if

we could fail one of above mentioned properties then the given graph won't be isomorphic. Subgraph isomorphism detection is fundamental to FSM. There are significant number of algorithms developed which are all focussed on reducing computational overhead associated with subgraph isomorphism detection. In figure 4.7 we can see two graphs which are exactly the same but have different orientation of vertices and edges. They are isomorphic graphs. Subgraph isomorphism detection techniques are broadly categorized in to: exact matching and error tolerant matching. Most FSM algorithms use exact matching techniques. Ullman and Nauty are popular exact matching algorithms [41].

Frequency Counting

There are two different problem in FSM : i) *graph transaction based FSM* ii) *single graph based FSM* [41]. In graph transaction based FSM input data consists of collection of graphs. A subgraph g is considered to be frequent if it's count is greater than certain threshold. The count of subgraph is usually referred as *support*. The support of g is counted using two different approach, either *transaction based counting* or *occurrence based counting*. Transaction based counting can only be used in graph transaction based FSM where as occurrence based counting can used in both graph transaction and single graph based FSM, although it is typically used in single graph based FSM [41].

In transaction-based counting the support is calculated by counting the occurrence of g per transaction. We only count one per transaction regardless of whether g occurs one or more than once per graph transaction. In a given database $G = \{G_1, G_2, , G_T\}$ consisting of a set of graph transactions, and a support threshold $\sigma(0 < \sigma \leq 1)$; then the set of graph transactions where a subgraph g occurs is defined by $\delta_G(g) = Gi|g \subseteq G_i$. Thus, the support of g is defined as:

$$sup_G(g) =\mid \delta_G(g) \mid /T \tag{4.2}$$

where $\mid \delta_G(g) \mid$ is the cardinality of $\delta_G(g)$ and $T =\mid G \mid$. Thus, if $sup_G(g) \geq \sigma$ then g is considered as frequent subgraph. In occurrence we simply count the frequency of g in G [41].

4.3.2 Frequent Subgraph Mining Algorithms

Frequent subgraph mining (FGM) algorithm has several application in chemical informatics and biological network analysis. Since subgraph isomorphism is a NP-complete problem, a significant amount of research has been directed to effective candidate generation strategy. The method employed for the candidate generation is the most distinguishing factor of all the FGM algorithms. For the discussion purpose, in this section FGM algorithms are classified in to i) general purpose and ii) pattern dependent FGM. The difference between the two approach is that in the later case the nature of application domain drives the nature of subgraph we mine. They are specialized and limited in nature. [41].

General Purpose Frequent Subgraph Mining

In this section we will discuss the general purpose frequent subgraph mining in terms of inexact and exact matching.

Inexact FGM Inexact FGM employs an approximate approach to compare the similarity between two graphs. Any two graphs do not need to be entirely same to contribute to the support count. Inexact FGM algorithms do not guarantee to be complete but bring some efficiency gain during computation. There are only few examples of inexact FGM. However, the most frequently quoted inexact FGM algorithm in literature is SUBDUE [45]. SUBDUE uses minimum description length method to compress the graph data and beam search method to restrict the search space. It shows good results in image analysis and CAD circuit analysis. However, the scalability of the algorithm is an issue i.e. the runtime of algorithm increases exponentially with the graph size [41].

Exact FGM Exact FGM algorithms are more specific and common than inexact FGM algorithms. They can be applied to both graph transaction based and single graph based mining. FGM algorithm are complete and guaranteed to find all the frequent subgraphs, however in the expense

of efficiency of algorithm. Such algorithm performs efficiently only on sparse graph data sets. Due to this completeness restriction, they undergo expensive subgraph isomorphism comparison [41].

Exact FGM algorithms can be broadly categorized in terms of traversing strategy: i) BFS based and ii) DFS based. BFS based algorithms are more efficient because it allows the pruning of infrequent subgraphs in early stage of FGM process, however it costs high I/O and memory usage. In Association Rule Mining algorithm, such as Apriori [38], BFS strategy is used. Here (k + 1) subgraph can only be frequent if its immediate parent k subgraph is frequent. BFS make sure that complete set of candidates of size k is processed before moving to (k + 1) candidates [41].

FGM algorithm that employs DFS strategy, uses less memory as they traverse the lattice in DFS manner [41]. gSpan is arguably the most cited FGM algorithm which uses DFS strategy. It uses canonical representation, M-DFSC to uniquely represent each subgraph. The algorithm uses DFS lexicographic ordering. It uses right most path extension method to construct a tree like lattice structure where each node is a frequent subgraph. The (k + 1) subgraphs in the tree are generated by one edge expansion from the k-th level of the tree. The search tree is traversed in DFS manner and all subgraphs without minimal DFS codes are pruned [41].

Pattern Dependent Frequent Subgraph Mining

Mostly users are interested in a certain specific type of patterns i.e. some subset of the set of all frequent subgraphs. Such special patterns are mined according to their topology. Pattern dependent FGM algorithms can be categorized according to the nature of pattern of subgraphs which we are mining: i) relational patterns ii) maximal and closed pattern and iii) cliques [41].

Relational Pattern Mining Relational graphs are widely used in modeling large scale networks i.e. biological or social networks. Relation pattern mining has three important features which differentiate it from general purpose FGM algorithms: i) vertex has distinct labels, ii) very large graph data sets and iii) graph has certain connectivity constraints. Thus relational graph mining is focussed on identifying all frequent sub graph with certain connectivity constraint [41, 76].

CLOSECUT is one of the most popular relational pattern mining algorithm. It is directed at mining closed frequent subgraphs with connectivity constraints. CLOSECUT uses pattern growth approach to integrate connectivity constraints and uses graph condensation and decomposition techniques [41, 76].

Mining Maximal and Closed Patterns The frequency of frequent subgraphs increases exponentially with the size of the graph. For a frequent k-graph, the number of its frequent subgraph can scale up to 2^k. Therefore, maximal and closed FGM approaches have been proposed to constraint the number of candidate frequent subgraphs [41]. Let MFS denotes the set of maximal frequent subgraphs, CFS the set of closed frequent subgraphs and FS the set of all frequent subgraphs in the graph database then MFS $\subseteq$ CFS $\subseteq$ FS.

Let $MFS = g \mid g \in FS \wedge \neg(\exists h \in FS \wedge g \subset h)$.

The objective of maximal frequent subgraph mining is to find all graph patterns that belong to MFS. Two example of maximal FGM algorithms are SPIN and MARGIN.

Let $CFS = g \mid g \in FS \wedge \neg(\exists h \in FS \wedge g \subset h \wedge sup(g) = sup(h))$.

The aim of closed frequent subgraph mining is to find all patterns that belong to CFS. CLOSECUT and SPLAT are two examples of closed FGM algorithms [41].

Mining Cliques Clique is an important concept in graph theory. It is defines as a graph where every vertex is adjacent to every other vertices. Recently it has been found that discovering frequent cliques has several application in domain such as communication, finance and bio-informatics. CLAN [69] is a FGM algorithm which mines cliques. It is directed at mining frequent closed cliques from large dense graph database. The algorithm uses the properties of the clique to mine the frequent structure or sub clique using isomorphism testing by employing canonical representation of a clique [41, 69].

In this frequent subgraph mining section, we have presented the overview on frequent subgraph mining process which are most commonly referred to in the literature. With reference to the literature, various mining strategies have been presented with respect to different kind of graphs.

We explained in brief about the three most important area in frequent subgraph mining: i) encoding strategy ii) candidate generation strategy and iii) approach to frequency counting [41].

4.4 Topological-Sorting Based Frequent Subgraph Mining

4.4.1 Topological Ordering

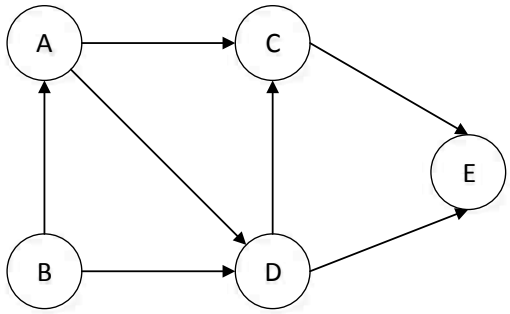

Figure 4.8: A digraph example for topological sort.

Candidate generation and graph isomorphism are the two main aspects of every sub graph mining algorithms. Here we have incorporated topological sort based encoding to address those problems. We call it topological ordering. Lets first discuss about topological sorting in brief before moving forward with topological ordering.

Topological sort is a process of assigning a linear ordering to the vertices of a Directed Acyclic Graph (DAG) so that if there is an ar from vertex i to vertex j, the i appears before j in the linear ordering.

A topological sorting of Figure 4.8 can be B, A, D, C, E. There could be several topological sorts for a given DAG. A topological ordering is possible if and only if the graph has no directed cycles. Every DAG has at least one topological ordering. Topological sorting problem can be solved in linear time. Algorithm 2 gives the pseudocode for topological sorting.

Algorithm 2 Topological_Sort

$L \leftarrow$ Empty list that will contain the sorted elements

$S \leftarrow$ Set of all nodes with no incoming edge

while S is not empty **do**

  remove a node n from S

  add n to tail of L

  for each node m with an edge e from n to m **do**

    remove edge e from the graph

    if m has no other incoming edges **then**

      insert m into S

    end if

  end for

end while

if graph has edges **then**

  return error (graph has at least one cycle)

else

  return L (a topologically sorted order)

end if

As we have discussed eariler we are mining a specific type of subgraph Restricted Dependency Subgraph (RDS) and the graph we are mining on is a directed acyclic graph. Topological encoding has two major advantages over any encoding techniques in this scenario.

1. Avoiding the generation of duplicate candidate subgraphs:

   We developed a canonical encoding sequence TSCode which avoids the generation of duplicate candidates subgraphs. This process is explained in details in section Canonical Encoding of Directed Acyclic Graphs Using Topological Sorting

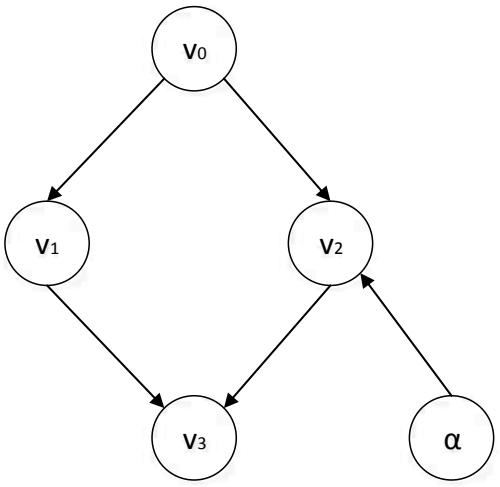

Figure 4.9: Example graph which is not a RDS.

2. Restricting the search space to only generate Restricted Dependency Subgraph (RDS):
   As per the definition of RDS, there should be a path from source node v_0 to every other node in the subgraph. If we generate any candidate as shown in Figure 4.9, such candidate should be avoided since the node α is not contributing to source node v_0. and also there is no path from v_0 to α. The TS-growth path of figure is v_0, v_1, v_2, α but according to the TSCode, node α should come before v_2. Therefore this will avoid the generation of such subgraphs.

4.4.2 Canonical Encoding of Directed Acyclic Graphs Using Topological Sorting

Our encoding of graphs is based on topological sort. A DAG defines a partial order and topological sort can extend the partial order into a total order (numbering). We choose topological sort because the graphs we consider models dependency relationship and numbering by topological sort respects such relationship. While growing subgraph as discussed in section B, we don't grow the subgraphs which are not RDS. Topological ordering automatically eliminates such subgraphs where a node has no contribution to the root. Figure 4.10 shows the numbering of an example graph by BFS, DFS, and Topological Sorting.

To obtain the encoding of a graph, we first use topological sort to give each node in the graph a unique ID number. (We call such ID the Topological Sorted ID – TSID.) The topological sort

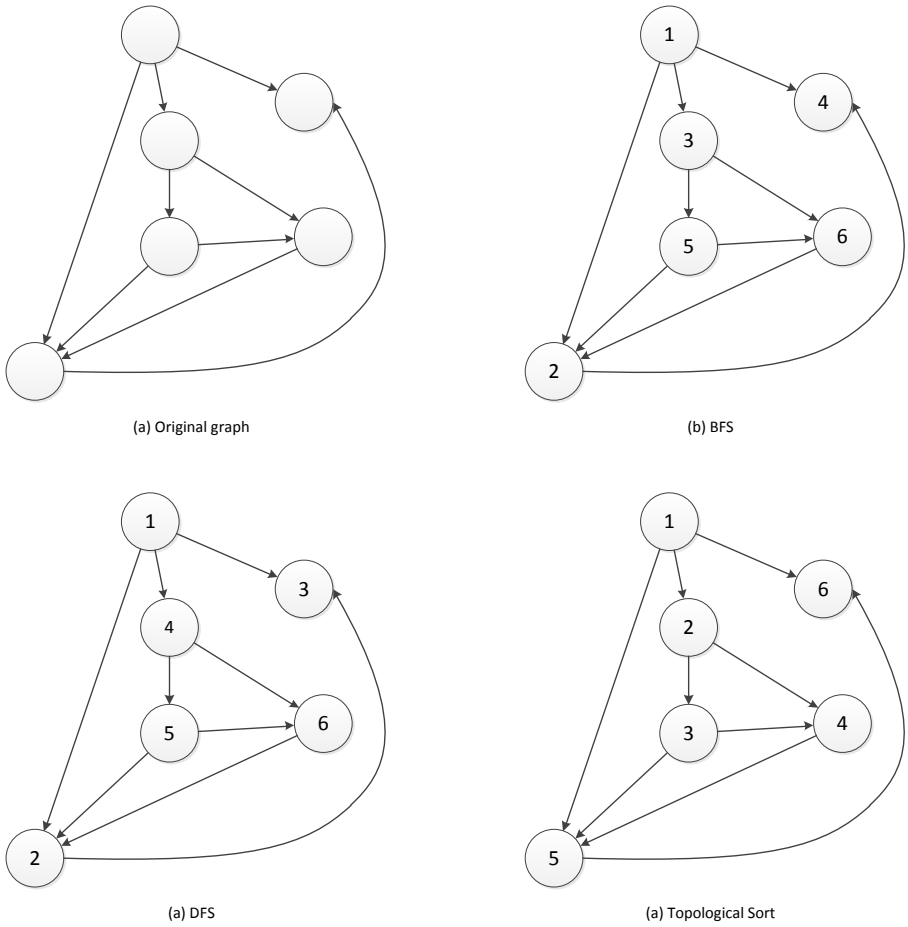

Figure 4.10: Numbering by BFS, DFS, and Topological Sorting. Numbering by topological agrees better with the dependencies between the nodes.

employed by our encoding is a modified version of the standard algorithm. When extending a DAG into a total order, the standard topological sort does not ensure unique numbering, i.e., two isomorphic graph may lead to two different numbering because of the difference in the representations of the graphs. Our modification imposes an order to ensure an unique numbering when multiple nodes are free to be numbered.

Algorithm 3 gives the detailed algorithm for assigning the TSIDs. It takes as input an RDS graph with the source v_0 and outputs the TSIDs of the nodes in the graph.

Algorithm 3 Topo_Numbering

$k \leftarrow 0$

add v_0 to the priorityqueue S

while S is not empty **do**

remove the first element n of S

give n the ID number k

$k \leftarrow k + 1$

for each node m with an edge e from n to m **do**

remove edge e from the graph

if m has no other incoming edges **then**

insert m into S

sort elements in S according to the ordering specified in Eq. 4.3

end if

end for

end while

Suppose n and m are two nodes. Let $IN(x)$ be the in_neighborhood of x. Assume nodes in $IN(n)$ and $IN(m)$ are already numbered. Let $min(V)$ be the minimum ID number among the nodes in V. Let $type(x)$ be the type of the node. We define the following order on the nodes:

$$\begin{aligned} &n \prec_n m \text{ if } min(IN(n)) < min(IN(m)) \\ &n \prec_n m \text{ if } min(IN(n)) = min(IN(m)),\ type(n) \prec_t type(m) \end{aligned} \tag{4.3}$$

In figure 4.11 lets assume type(v_1) < type(v_2) then by following the standard topological ordering rule, we can order the nodes from figure 4.11 in four different ways: $v_0v_1v_2v_3v_4$, $v_0v_2v_1v_3v_4$, $v_0v_1v_2v_4v_3$, $v_0v_2v_1v_4v_3$. This is because topological ordering does not provide total ordering. When we apply our modified topo ordering to the same graph we get a single and complete ordering of the nodes i.e. $v_0v_1v_2v_3v_4$.

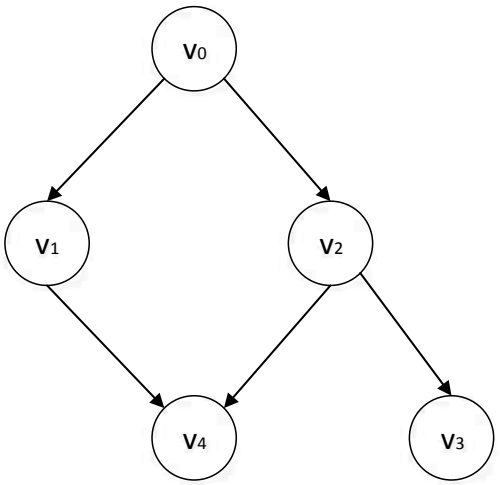

Figure 4.11: Example graph to demonstrate topological ordering.

After obtaining the TSID for each node in the graph, we encode each edge as a 4-tuple (i, j, ti, tj) where i and j are the TSID of the two end nodes. The edge is directed. i is the node where the edge points to and j is the node where the edge points from. And ti and tj are the types of the two nodes.

The whole graph is encoded by the the sequence of the edge codes, each corresponds to an edge in the graph. Our canonical encoding is then the sorted sequence of the edge codes. Intuitively, to sort the edges (codes), we compare the TSID of the head of the edges (the node pointed to by the edge) first. The edges that go into a node with larger TSID always appears behind edges that go into a node with smaller TSID. In particular, the edges going into the node with the largest TSID are at the end of the sequence. If two edges has the same head (going into the same node), they are compared by the TSIDs of their tails (the nodes they going out from). If the TSIDs of both the head and the tail are the same, the types of the nodes are compared. Formally, let x and y be two edge codes, we say

$$x \prec_{ec} y \text{ if } x[i] \prec y[i] \text{ and } \forall j < i,\ x[i] = y[j] \tag{4.4}$$

We call the canonical code for a graph the TSCode of the graph.

Depth First Search Encoding (DFSCode) and Topological Encoding (TSCode)

In this section we will discuss about the difference between Depth First Search Encoding and Topological Encoding techniques. DFSCode encoding is one of the mostly used encoding technique in frequent subgraph mining area. It is similar to TSCode encoding technique but differs in the way graph is traversed. DFSCode encoding scheme uses DFS to provide the unique identifier to the nodes. Lets take an example of graph in figure 4.11. In the figure lets assume the name of node is also the type of corresponding nodes. With DFS encoding, then ordering of nodes will be $v_0v_1v_4v_2v_3$. We can assign serial number for this ordering as v_0 - 0, v_1 - 1, v_4 - 2, v_2 - 3, v_3 - 4. After getting this order we can encode each edges of the graph as a 4-tuple (i, j, ti, tj) same as in TSCode. Now with the help of equation 4.4 we can sort these edge encoding as: $(0, 1, v_0, v_1)$, $(0, 3, v_0, v_2)$, $(1, 2, v_1, v_4)$, $(3, 2, v_2, v_4)$, $(3, 4, v_2, v_3)$.

Now using topological ordering as we have discussed earlier, we can order the nodes of same graph as $v_0v_1v_2v_3v_4$. The serial number assigned to these nodes will be v_0 - 0, v_1 - 1, v_2 - 2, v_3 - 3, v_4 - 4. Using the same equation 4.4 we can order the edge encoding as: $(0, 1, v_0, v_1)$, $(0, 2, v_0, v_2)$, $(1, 4, v_1, v_4)$, $(2, 3, v_2, v_3)$, $(2, 4, v_2, v_4)$.

In terms of encoding process, we can't much difference but as we discussed in section Topological-Sorting Based Frequent Subgraph Mining, TSCode has much more advantages in candidate generation process.

Theorem 4.4.1 *Two single-source directed acyclic graphs are isomorphic iff the two graphs have the same TSCode.*

Proof. Consider two isomorphic single-source directed acyclic graphs. After apply algorithm 3, the corresponding nodes in the two graphs will be numbered the same. Then the corresponding edges in the two graphs will have the same encoding. This lead to the same TSCode for the two graphs.

Conversely, suppose two graphs have the same TSCode. We relabel the nodes in the two graphs according to their TSID in the TSCode respectively. Because the TSCode specifies the edges in

the two graphs, the same TSCode specifies the same set of edges for the two graphs. Therefore, they are isomorphic. □

4.4.3 Mining with Duplication-Free Subgraph Growth

Many frequent-subgraph mining algorithms run in iterations or use recursions. During each iteration or recursion, it generates candidate subgraphs and then counts the frequencies of these candidates. Only those whose frequency is above the threshold are used in candidate generation for the next iteration. There are two approaches to generate candidate subgraphs: merging and growth. In merging, two frequent subgraphs are merged to generate candidates. The growth approach starts with empty graph and generate candidate by grow the graph one edge at each step.

The exploration of the candidate subgraphs performed by the growth approach can be illustrated by a growth tree. The tree is rooted at the empty graph. The childrens of the root are the subgraphs that contain only one edge. Each children in turn gives rise to their own children, which are subgraphs of two edges derived by adding one edge to their parents. Such growth continues until each path of growth reach the full graph. The growth tree is directed and if a subgraph G' is the result of growing another subgraph G by an edge, G is the direct predecessor of G' and G' is the direct successor of G.

Figure 4.12 shows an example (incomplete) growth tree. The nodes in the growth tree are subgraphs of the original graph. If a graph G' is derived by growing one edge from the graph G, there is an edge in the growing tree pointing from G to G'. A growth path for generating a candidate subgraph is the path in the growth tree that goes from the root of the tree to the node that corresponding to that subgraph. For example, in the figure, $1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 10$ is a growth path that lead to the full original graph and $1 \rightarrow 2 \rightarrow 4 \rightarrow 6$ is a growth path leading to the subgraph at node 6.

The problem with the growth approach is that the same subgraph can be reached by different path of growth. The growth tree has duplication nodes, i.e., the nodes that correspond to the same subgraph. Clearly, a mining algorithm that explores all the nodes in the tree will waste a lot of

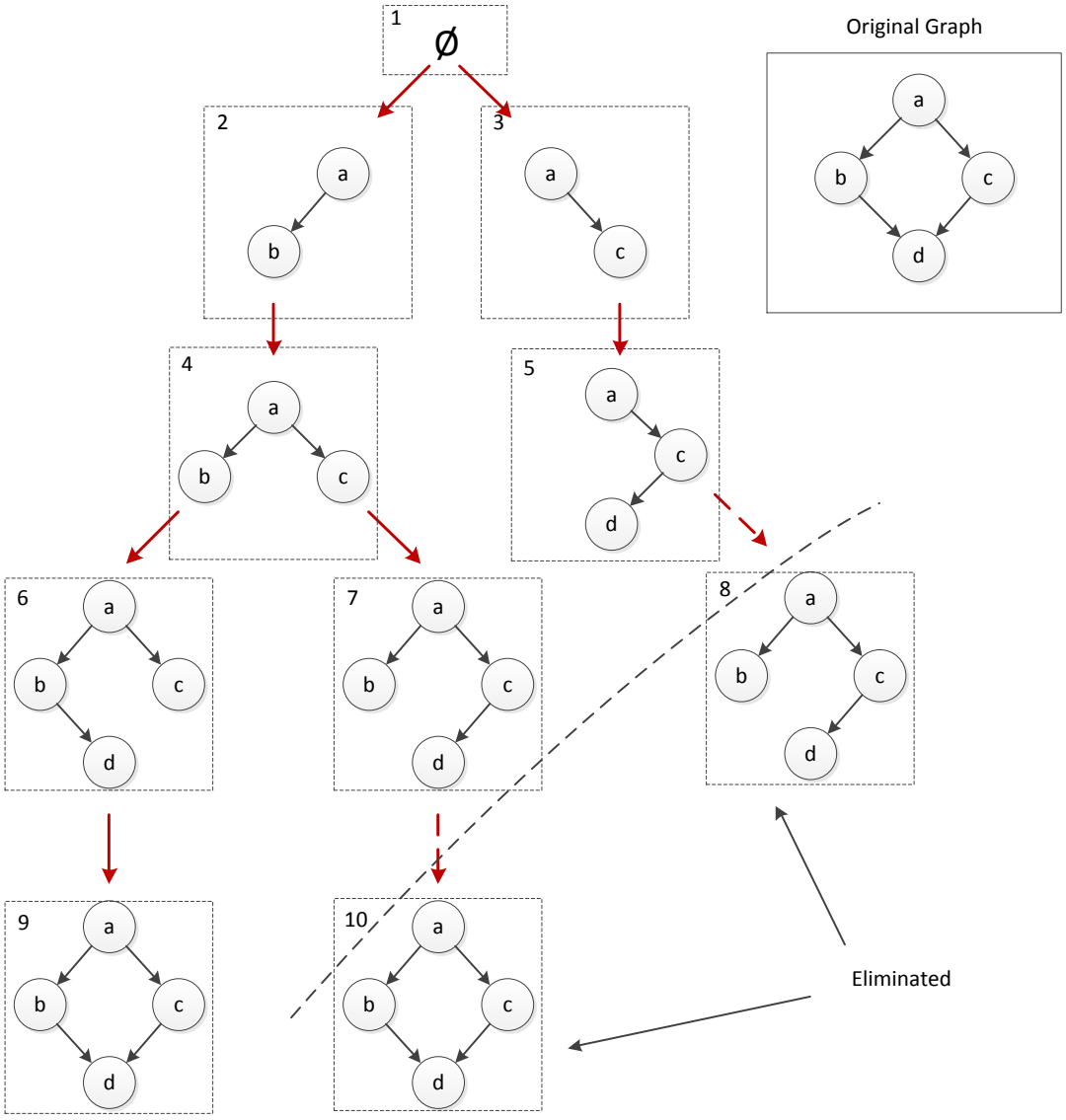

Figure 4.12: An incomplete growth tree. The nodes in the growth tree are subgraphs of the original graph. If there a graph G' is derived by growing one edge to the graph G, there is an edge in the growing tree pointing from G to G'. There are duplications in the tree, i.e., nodes corresponding to same graphs (e.g., nodes 7 and 8). A efficient mining algorithm is required to ignore the duplications. When explore the growth tree, our algorithm eliminate the two duplicate nodes as indicated here.

work. And a good mining algorithm that uses the growth approach needs a strategy which can ensure that it generates candidate only for the part of the growth tree that is not a duplication.

Our mining algorithm uses the growth approach. It explores the growth tree and generates candidate following paths we call the TS-growth path. The TS-growth path ensures that there is no duplication among the candidates generated by our algorithm.

Definition 3 TS-growth path*: A TS-growth path is a grow path such that for every node G_s and its direct successor G_{s+1} on the path (G_s grows into G_{s+1} by adding an edge e), if we calculate the TSCode of G_{s+1} and a path-derived code for G_{s+1}, the two codes are the same. The path-derived code for G_{s+1} is obtained by appending, to the end of the TSCode of G_s, a path-derived encoding of the edge e. The path-derived encoding of e is constructed as follows: If G_s and G_{s+1} have the same number of nodes, the path-derived code of e is constructed in a regular way where the ends of e use their TSIDs from G_s. If edge e adds a new node, the end of e that is in G_s still uses its TSID from G_s in the construction of the path-derived edge code and the other end (the new node) uses a path-derived ID whose value is the number of nodes in G_s plus one.*

As an example, we consider the nodes 5 and 8 in Figure 4.12 The growth path from the root to node 5 is a TS-growth path. However, extending it to node 8 will not result in a TS-growth path. After adding the edge (a, b), to calculate the path-derived code for the edge, the vertex a uses its TSID from the subgraph in node 5 (the value is 1) and the vertex b get a path-derived ID whose value will be 4 (the number of vertices in the subgraph in node 5 plus 1, 3+1 = 4). On the other hand, if we run Algorithm 3 on the subgraph in node 8, the vertex b will receive a TSID 2. Since the TSID of vertex b is different from its path-derived ID, the TSCode of the subgraph in node 8 is different from its path-derived code too. Therefore, extending the growth-path from node 5 to node 8 will not result in a TS-growth path. Our algorithm, which follows only TS-growth path, will not visit node 8.

The detail of our subgraph mining algorithm is given in Algorithm 4. Let G be the event graph and R be the set of nodes in the event graph that are of a specified type and T a frequency threshold. The algorithm finds all the frequent RDS subgraphs and collect them in the set F.

Algorithm 4 ToPo_Mining

for $v \in R$ **do**
  for each edge e that goes from v and has occurred more than T times **do**
    create a subgraph of edge e and add it to the set F
  end for
end for
for each subgraph $G_s \in F$ **do**
  for each edge $e \in G$ from n to m and $n \in G_s$ but $e \notin G_s$ **do**
    create a subgraph G_{se} including G_s and e
    calculate the path-derived code of G_{se} following the process given in Definition 3.
    calculate the TSCode of G_{se}
    if the TSCode of G_{se} is the same as its path derived code **then**
      if the occurrence count of G_{se} is above T **then**
        $F \leftarrow F \cup G_{se}$
      end if
    end if
  end for
end for

Theorem 4.4.2 *Algorithm 4 generates candidate for each RDS subgraph once and only once.*

Proof. Because Algorithm 4 grows subgraphs following the TS-growth path. To prove the theorem, we only need to show that for any RDS subgraph, there is one and only one TS-growth path leading to S.

Clearly, there is a TS-growth path leading to each RDS subgraph of size one edge. Assume all the RDS subgraph of size k edges has a TS-growth path. We show that every RDS subgraph of size $k+1$ has a TS-growth path. Let G_{k+1} be such an RDS subgraph and $c = \{ec_1, ec_2, \ldots ec_{k+1}\}$ be the TSCode of G_{k+1} where ec_i is the code for the edges in G_{k+1}. Consider the graph G_k derived

by remove the edge whose edge code corresponds to ec_{k+1}. G_k has k edges and therefore there is a TS-growth path P from the root of the growth tree to G_k. Furthermore, the TSCode of G_k is exactly $\{ec_1, ec_2, \ldots ec_k\}$. By appending ec_{k+1} to the TSCode of G_k, we obtain the TSCode of G_{k+1}. Following the definition of TS-growth path, the path P plus G_k growing into G_{k+1} is a TS-growth path from the root to G_{k+1}.

We now consider the uniqueness of the TS-growth path. Suppose there are two nodes in the growth tree G_s and $G_{s'}$ that are isomorphic. Then the TSCode of G_s is the same as the TSCode of $G_{s'}$. Assume there is a TS-growth path P going from the root to G_s and there is another path P' going to $G_{s'}$. We examine the nodes on the two paths backwards starting from G_s and $G_{s'}$ one at a time. If all the corresponding node on the two paths have the same TSCode, then the two paths are exactly the same. For the two paths to be different, there is at least one node on P that has different TSCode than the corresponding one on P'. Let G_d be the first node on P (coming for G_s) that has a different TSCode than that of the corresponding node $G_{d'}$ on P'. G_d and $G_{d'}$ are different subgraphs. But the direct successor of G_d is isomorphic to the direct successor of $G_{d'}$ and the two have the same TSCode. Recall by definition, the TSCode of the direct successor on a TS-growth path is obtained by appending the edge code to the TSCode of the direct predecessor. Since the two direct predecessors have different TSCodes, it is not possible to obtain the same TSCode by appending edge code. □

There are two implications from Theorem 4.4.2. First, it shows that Algorithm 4 correctly mines frequent subgraphs as it explores all the RDS subgraphs and count their frequencies. The algorithm does not omit any RDS subgraphs in its exploration. Second, the amount of exploration on the growth tree performed by Algorithm 4 is no more than any other growth-based mining algorithm. since it does not explore any of the repeated nodes on the growth tree.

4.5 Experiment Results

We used data from parallel first-principles molecular dyamics (FPMD) simulations of silica liquid to test our algorithm. The supercell used in the simulations consisted of 72 atoms. The atomic positions in liquid phase are strongly correlated but without any long-range order, and are constanly changing with time. Data from simulations for 3 different temperatures (3000K, 3500K and 4000K) are considered. The simulation runs ranged from a couple of hundred thousands of FPMD steps (at high temperature) to little more than one million steps (at the low temperature). The time step of 1 femtosecond was used in the simulation. Note that the simulations are discrete in both space and time. All experiments of ToPoMine are done on an Intel(R) Core(TM) i7 @2.80 GHz PC with 4 GB of main memory running 64 bit Windows 7.

For simulation data of each temperature, an event graph was constructed. We ran our mining algorithm on the event graph. A subset of events of the same type were chosen as the starting points for growing the subgraphs. We varied the settings to mine subgraphs starting from different type of events, subgraphs of different size, and used different threshold in defining frequent subgraphs. Figure 4.13 shows some subgraph discovered from the SiO_2 simulation data. Each node in a graph represents a bond event. We label the node by the type of the corresponding event. For example, the label [5432] means a silicon atom changes bond state from 5 to 4 and the corresponding oxygen atom changes bond state from 3 to 2. If an event x is dependent on another event y, x is drew above y. If the dependency is due to a silicon atom, the edge going to y is an arrow pointing down and to the left. If the dependency is due to an oxygen atom. the edge going to y is an arrow pointing down and to the right. As expected, we observe the trend that bigger graphs have smaller frequent. And the event graphs are not always a tree (e.g., graph (e) is not a directed tree.).

The base version of our algorithm takes a threshold T and mines all subgraphs that are above the threshold. We modified our algorithm to discover the most frequent subgraphs for different subgraph sizes and different simulation temperatures. Figure 4.14 gives such frequent subgraphs. The rows corresponds to different graph sizes and within each row, we list the most frequent subgraph for simulation under different temperatures. For example, graph (i) is the most frequent

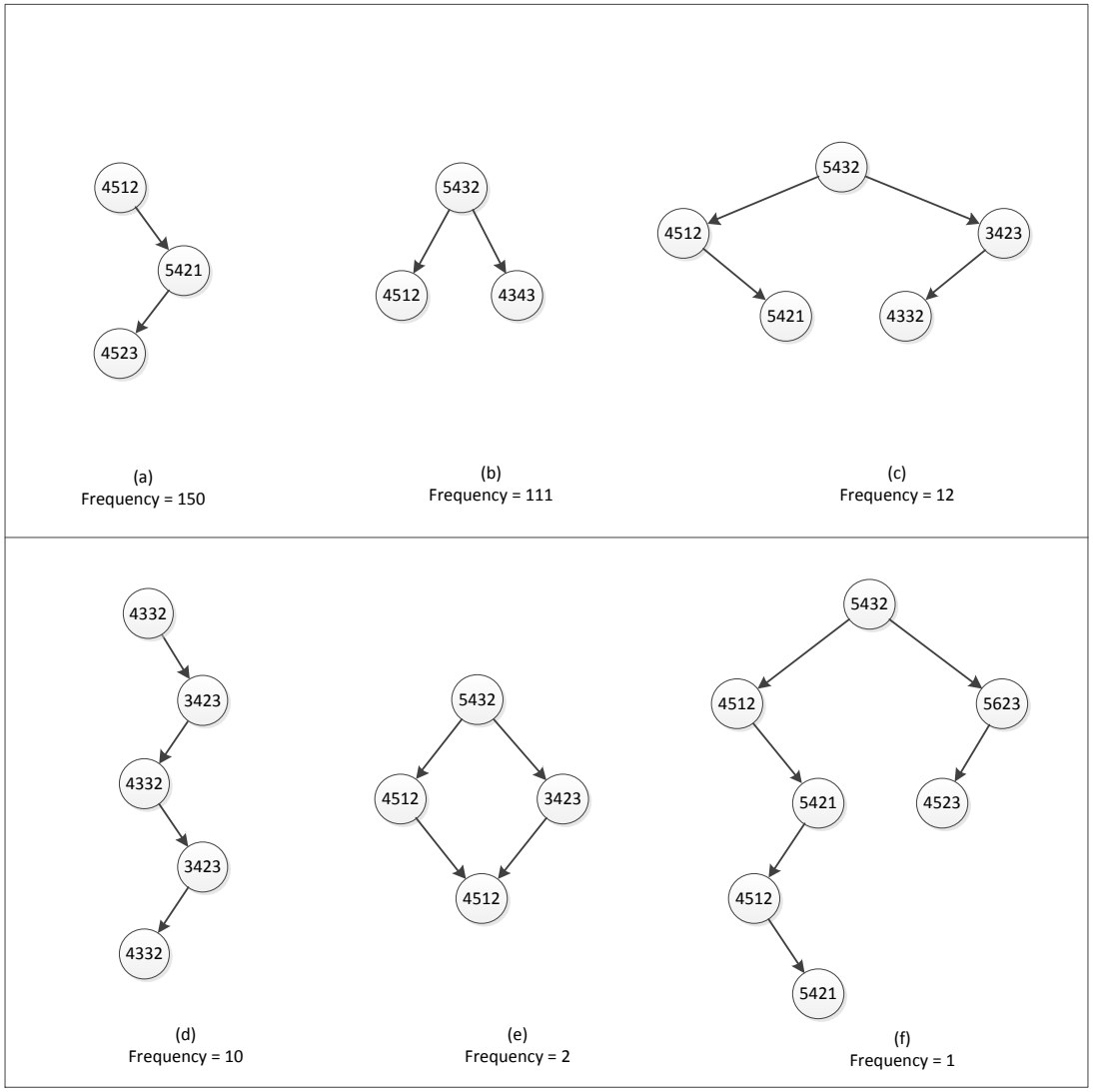

Figure 4.13: A few example subgraphs discovered in the SiO_2 atom dynamics from simulation using our algorithm. Each node represents a bond event and is labeled by the type of the event. For example, the label [5432] means a silicon atom changes bond state from 5 to 4 and the corresponding oxygen atom changes bond state from 3 to 2. If an event x is dependent on another event y, x is drew above y. If the dependency is due to a silicon atom, the edge going to y is an arrow pointing down and to the left. If the dependency is due to an oxygen atom. the edge going to y is an arrow pointing down and to the right. We observe the trend that bigger graphs have smaller frequent.

subgraph of size 4 (edges) in the event graph for simulation at temperature 4000K. There are 90 subgraphs of this type in the event graph. Note that the number of counts for the subgraphs increases along with the simulation temperature. This is expected because with higher temperature,

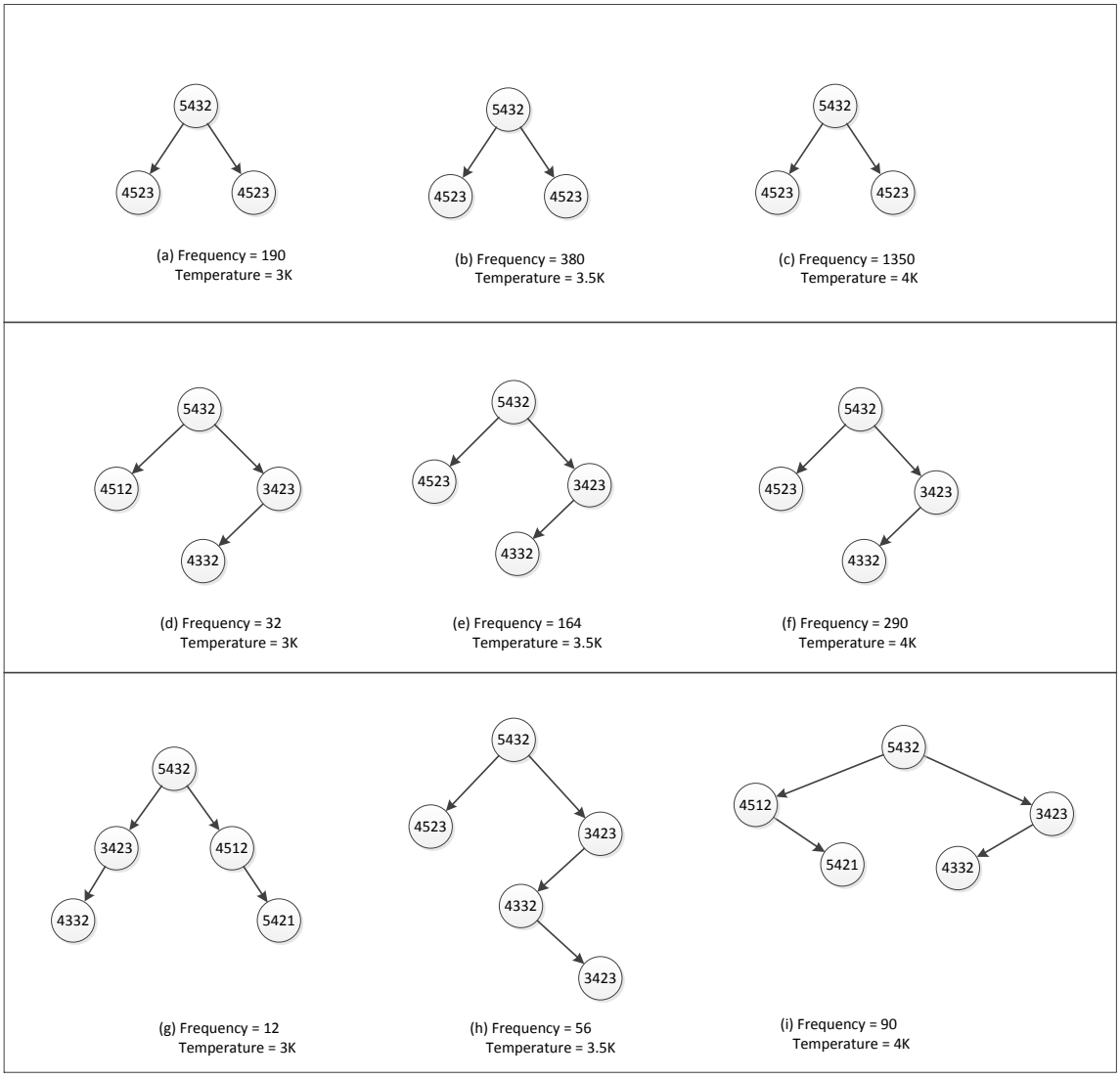

Figure 4.14: Frequent subgraphs mined by our algorithm. The rows corresponds to different graph sizes and within each row, we list the most frequent subgraph for simulation under different temperatures. For example, graph (a) is the most frequent subgraph of size 2 (edges) in the event graph for simulation at temperature 3000K. There are 190 subgraphs of this type in the event graph that covers 1 million times steps of simulation. Note that the number of counts for the subgraphs increases along with the simulation temperature. We also observe that when only small size subgraphs are considered, the most frequent subgraph is the same graph for all simulation temperatures. When larger subgraphs are considered, the most frequent subgraph is different for different temperatures.

there are more bond events and hence the count of the frequent subgraphs increase. We also observe that when only small size subgraphs are considered, the most frequent subgraph is the same for all simulation temperatures. When larger subgraphs are considered, the most frequent subgraph is

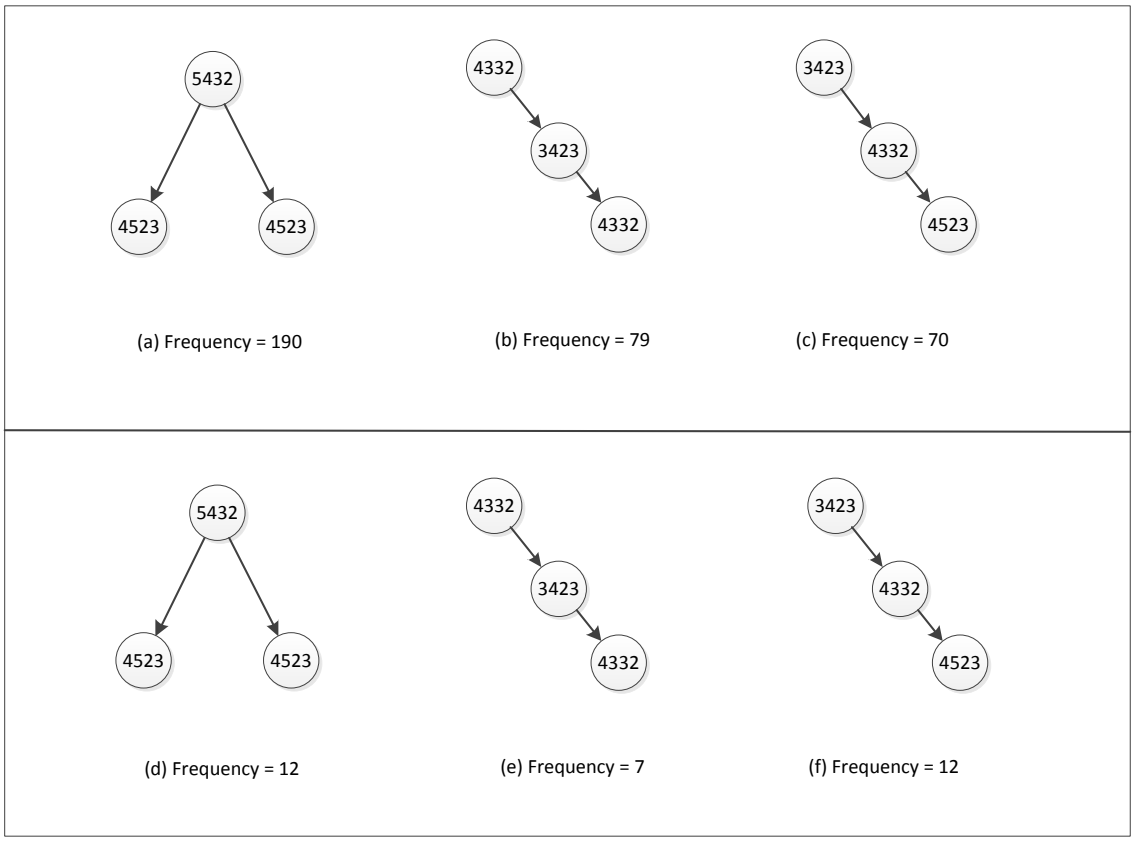

Figure 4.15: Frequent subgraphs mined by ToPoMine during large atomic movements and regular movements. Figure (a), (b) and (c) are for the regular movements and Figure (d), (e) and (f) are for large movements. Subgraphs in each column are of same type.

different for different temperatures. This is so for the particular simulations used in our experiment. More simulation data are needed to verify whether this is a general trend.

Provided the atom movements, particularly large atomic displacements are connected to the diffusive property of the materials. But all atoms contained in supercell do not show large movement even in liquid phase. Very few atoms show such behavior. When an atom undergoes large displacement, we categorize them as two types of motion i) *continuous flowing motion*: atom moves gradually making small step each time. ii) *discrete hopping motion*: atom makes few large jumps that accounts for almost the total distance travelled by it. We found that about 80% of large displacement is caused by *discrete hopping motion*. We mined all the atoms that undergoes *discrete hopping motion* and the time windows where they showed such movements with the help of algorithm presented in previous chapter. We denote such set of atoms by $A = \{a_1, a_2, a_3, ...\}$ and

each $a \in A$ has its own set of big move time windows denoted by $t = \{t_1, t_2, t_3, ...\}$. We then mined all RDS, in those big moved time windows whose frequency were above threshold value.

For the material scientists, mining the types of events which are responsible for large atomic displacements is of great interest. Here by combining ToPoMine and large movement mining algorithm, we move one step ahead. We not only mine the type of events but the chain of events that lead to the root event which caused large atomic displacement. These chain of events are the subgraphs which are associated with such movements. While mining such subgraphs we only take those events as a root which have atoms from set A. We denote such set of event as $V_l \subseteq V$. This gives us two different set of events: i) entire set of events in graph $G(V, E)$. i.e V and the set of all subgraphs rooted at V be g. ii) set of events responsible for *discrete hopping motion* i.e V_l and the set of all subgraphs rooted at V_l be g_l . We mined these two different set of subgraphs and recorded the top 10% frequent subgraphs. We did not find any different types of subgraphs in these two sets. But the interesting result to notice is that the percentage of some subgraphs in g_l with respect to same type of graph present in g was considerably higher. This shows the contribution of some subgraphs for *discrete hopping motion* is higher than others. Figure 4.15 shows some examples of subgraphs which are associated with such movements. Figure 4.15 (a), (b) and (c) are frequent subgraphs for regular movement and (d), (e) and (f) are the frequent subgraphs for large movements. Both are at temperature 3000K. Here we have shown three different types of subgraphs: type (a)-(d), type (b)-(e) and type (c)-(f). This results shows that about 14.6% subgraphs of type (c)-(f) are contributing for the large movements where as only 5.9% and 8.13% subgraphs of type (a)-(d) and type (b)-(e) respectively are contributing for large moves. Hence the type (c)-(f) is the major cause for event type 3423.

In figure 4.16, we have shown the performance analysis of ToPoMine. Figure 4.16 (a) and 4.16 (b) are the plots for time and space vs number of nodes in graph respectively. With the increasing graph size, the runtime increases linearly. It is an inherent result since the running time of the algorithm depends on frequency and size of subgraphs grown from the root nodes. They both are the features of data sets. So it is the features that makes the runtime to increase linearly not

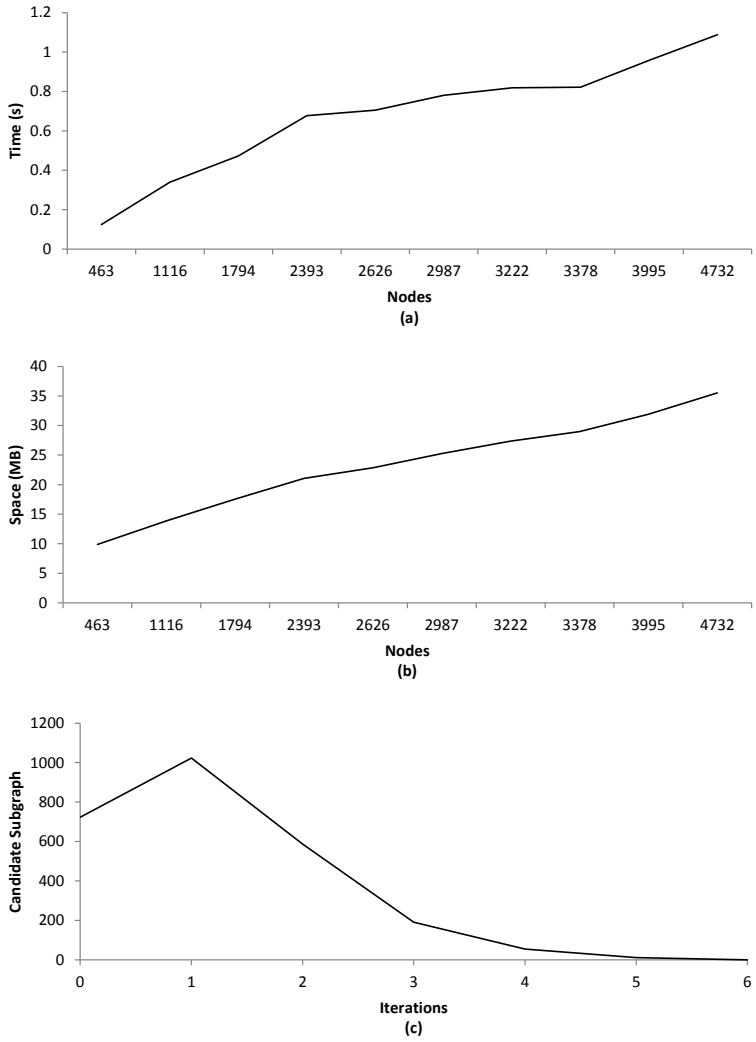

Figure 4.16: Performance analysis of ToPoMine. In both Figure (a) and (b) X-axis is the number of nodes in given event graph. Y-axis in Figure (a) is the running time and in Figure (b) it is the space occupied by the program in main memory. In Figure (c) X-axis is the number of iterations and Y-axis is the number of candidate subgraphs generated in each iteration.

the data size. We also tried to understand why ToPoMine is efficient for event graphs. After analyzing the structure of data and discovered frequent subgraphs, we find that it has lots of tree-like structures. Therefore it reduces the search space more efficiently. Figure 4.16 (c) shows the result for number of candidate subgraphs generated in each iteration. At iteration 0 (i.e. before running ToPoMine), we select nodes of specific type. In next iteration when we add an edge to the initial nodes, multiple one edge subgraphs is generated from a single node. That is why in first iteration number of candidate subgraphs is increased and then start to decrease exponentially.

With every iteration the size of subgraph is increased by one edge and since the number of frequent subgraphs is sensitive to the size of subgraph we are mining, we see the exponential decrease in number of candidate subgraphs in every subsequent iterations.

4.6 Related Work

Frequent subgraph mining is a popular topic in data mining and has been considered by a lot of work. Many algorithms have be proposed, including GSpan [73], Gaston [55], TSP [33], MOFA [16] as a few examples. In most cases, the difference between different algorithms lies on designs to perform the following two tasks: 1) how to encode the graphs in a unique way that is resistant to isomorphism and 2) how to enumerate the candidate subgraphs that may be of high frequency.

A popular approach to encode the graph is to use the deepth-first search (DFS) based ordering [10, 73, 80]). Our algorithm uses an encoding based on topological sorting. Topological sorting is a natural choice for the problem considered in this chapter because the graphs are derived from dependency relationships among events. Particularly when a dependency relationship also means ordering in time, topological sorting assigns closer numbers to events closer in time.

There are two main approaches for enumerating candidate subgraphs: merging(e.g., FFSM [35], GREW [48], HSIGRAM [49]) and growth (e.g., GSpan [73], TSP [33], MOFA [16]). In both approaches, duplication elimination is a key to efficient enumeration. Our algorithms differs from the existing ones by employ an enumeration based TS-growth paths. This enumeration is duplication free.

A few work has considered subgraph mining in directed graphs. mSpan [53] analyzed a financial network by modeling it as a directed weighted graph. FP-growth is used to obtain a unique graph encoding. We consider a different type of directed graphs and application scenario in this chapter. We also use a different encoding scheme that is suitable to our application scenario.

Finally, several work has explored the application of frequent subgraph mining for chemical and biological discoveries involving large molecules[16, 55]. We consider a different problem in

this chapter where not the structure of but the dynamics of the atoms are important. Our event model is different from the atom graphs used previously for modeling molecules.

4.7 Conclusion

We consider the problem of pattern discovery in atom dynamics from material simulations in this chapter. For many types of material under study, atomic level activities determines macroscopic properties of the material. Therefore, discover the patterns in atom dynamics that may be the mechanism that decides material properties is a key to better models of materials and better prediction of their properties.

An event graph is proposed to model the atomic dynamics as directed acyclic graph. We further proposed an algorithm to mine the frequent subgraphs in the event graph. The mining is based on an encoding using topological-sorting. The encoding is unique with respect to isomorphism and our algorithm enumerates candidate subgraphs in a duplication-free fashion. Experiment results on data from simulation of SiO_2 system shows the potential of our mining algorithm.

Chapter 5
Primitive Ring Mining

5.1 Introduction

Chemical bonds are widely used to model the structure of solid and liquid materials. Topological analysis of bond network is extremely important to understand the physical and biological properties. For such kind of analysis, structure of a material is modeled as topological network or a graph i.e. atoms are considered as vertices and bonds between them as edges. Graph based algorithm such as SUBDUE [45] and MolFea [29] use such abstraction to study the structure of materials [84].

The automation of experimental techniques in biology and chemistry has led to the generation of enormous amount of data. But the generation of database is only the first step towards the better understanding of underlying characteristics of biological and chemical compounds [29]. The popularity of graph mining has escalated in recent years due to the increasing complexity of computer simulated chemical network. For such structural analysis, efficient graph mining algorithm is essential. Large graph data sets are commonly found in Molecular Dynamics (MD) simulation data. Many graph algorithms have been used to analyze MD data set. Ring analysis is one such analysis which are used to characterize topological order of the materials [84].

Silica compounds are the most abundant substance on earth. Various forms silica form the constituents of many natural elements. There has been many research in crystalline and amorphous of silica compounds to study about their physical structures. In this chapter we will discuss on the study of silica liquid [79].

Close path (ring) mining generates a great interest in data mining community [74, 78].It has been revealed that the primitive ring structures plays key role on properties like light absorption and photoluminescence [85]. To the best of our knowledge, primitive ring structure has never been used to explain the dynamic behavior of a compound. We address this problem by developing a relation between primitive rings and viscosity. But viscosity is not directly related with the primitive ring structures. As viscosity is inversely proportional to the diffusivity of a compound, we find if there is any relation between diffusivity and ring structures.

Counting the rings in millions time steps simulation data gets tricky. There has been a lot of research on this area [42, 74, 78, 83, 85]. Most of the current algorithms require computation of all-pair shortest path matrix for all the neighbors of interested node [26, 31]. The size of this matrix could reach giga-elements in size as the size of ring reaches near 30 [78]. Another ring analysis approach called shortest-path analysis [11, 83] has been mostly used to mine ring structure. To find all the primitive rings of size $2L$ for a single node using shortest path analysis, we have to check all the shortest paths of size L. Which makes this approach highly inefficient.

In this chapter we propose an efficient algorithm to mine the primitive rings. To find larger rings of size L, we transfer the information from smaller rings so that we have to do less comparison among the shortest paths of size L. We implemented and tested our algorithm on simulation data of silica (SiO_2). In a silica compound, two Si atoms are connected with each other via an O atom. While building a graph, we only take Si atom as a vertex and O atom as an edge between Si atoms. In our case all the edges have equal weight. We take advantage of Breadth First Search (BFS), which behaves as Dijkstra's algorithm for not weighted graph.

The main contributions of our work are:

- Mining primitive ring in efficient way.
- Mining primitive ring structure when the diffusivity of atoms are higher.

5.2 Preliminary Concepts

A topological network is formed by two sets of elements: nodes (vertices) and edges (links). Edges are the connected links between any two nodes. Any structure consisting of atoms and bonds can be represented as a topological network. A path in a network is a sequentially connected nodes and edges without overlapping and a ring is simply a close path or a cycle. Ring r with n nodes is called an n-member ring and size(r) gives the number of nodes in ring r. Two nodes are connected by only one edge and is denoted by e. e.g. an edge between two nodes s and t is denoted as e_{st}.

Definition 4 *A primitive ring is a ring, if there is no shorter path between any two points in a ring than path of the ring itself. For example, If in Figure 5.1 if path* $a > c > b$*, then* $a - 1 - b - 2 - a$ *and* $1 - b - 2 - c - 1$ *are the primitive rings [78].*

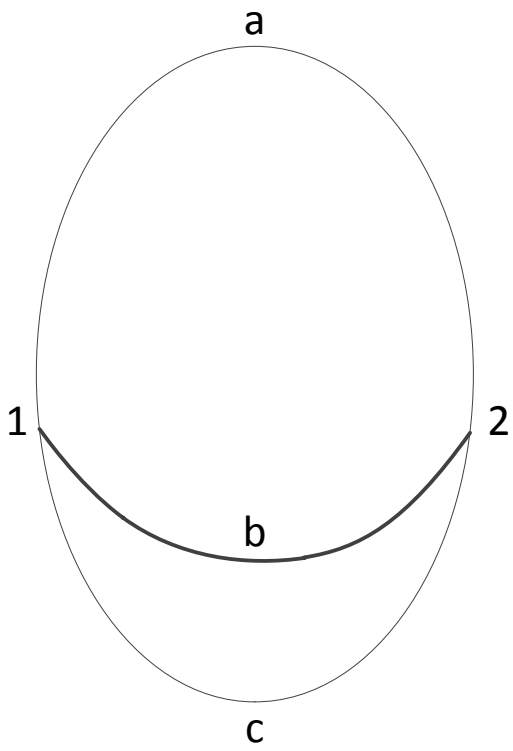

Figure 5.1: Primitive ring example

Definition 5 *Level of a node* v *is the shortest distance of* v *from a source node* S*.*

Definition 6 *If a ring r can be formed by joining any two nodes in ring r' and $size(r) \leq size(r')$, then r is the subring of r', denoted by $r \subseteq r'$ (proper subring, if $r \subset r'$) and r' is the super-ring of r. For each ring r, $V(r)$ is the vertex set of r.*

We define P_s as a shortest path from a source node S to any node s. Here, P_s is a set of nodes from S to node s in a sequential order. i.e. $P_s = \{S, n_1, n_2, ..., s\}$. $P(S, s)$ is a set of such shortest paths from S to s.

Similarly, we define $P_t = \{S, p_1, p_2, ..., t\}$, $P(S, t)$ = set of P_t.

Definition 7 *(RING DISTANCE) Ring distance between any two nodes s and t is the minimum distance between them along the ring. i.e. $RD(s, t) = min\{[|P_s| + |P_t|], L - [|P_s| + |P_t|]\}$, where L is size of the ring.*

Definition 8 *(Tail node) Tail node of a path is the end node of that path.*

Definition 9 *(GOOD PAIR) A good pair of paths are two potential candidate paths which could form a primitive ring. e.g. For good pair paths P_s and P_t, if we connect s and t with an edge they will form a primitive ring. Now for paths P_s and P_t to be a good pair, they have to satisfy the following conditions:*

1. $\forall m \in P_s \cap \forall n \in P_t = \phi$.

2. $\forall m \in P_s$, $\forall n \in P_t$, $SD(m, n)$ = $RD(m, n)$.

where, $m, n \in V(g)$ and $SD(m, n)$ is the shortest distance between m and n in the original graph.

Definition 10 *(Lookup Table) If a pair of $p1$ and $p2$ form a primitive ring then we create a table for the root node. This lookup table stores the information that $p1$ and $p2$ has formed a primitive ring. We will add the tail node of $p1$ and $p2$ in the lookup table. This information helps to determine if a given pair are good pair or not in later stages.*

5.3 Problem Statement

Given a graph $G(V, E)$ with vertex set $V(G)$, edge set $E(G)$, ring size L and a node o, we find complete primitive ring set R of node $o \in V(G)$ of $size \leq L$.

5.4 Tree rooted at Single Node

The main idea behind this approach is when you draw several paths from a single source node, you will find a ring if the paths intersect with each other. However, the rings found by this analysis are not all primitive. Path of a node is always with respect to the source node and there can be multiple shortest paths to a particular node. The set of such shortest path is denoted by $P(S, v)$, where S is the source node and v is any node in the graph. Any path $p \in P(S, v)$ is denoted by the series of node from S to v. In Figure 5.2, let path p_1 is $a - b - d$ and path p_2 is $a - b - e$. For these two pair to form primitive ring , they should be good pair. Therefore we first need to check if the node d and e are connected by an edge and then check if they are good pair.

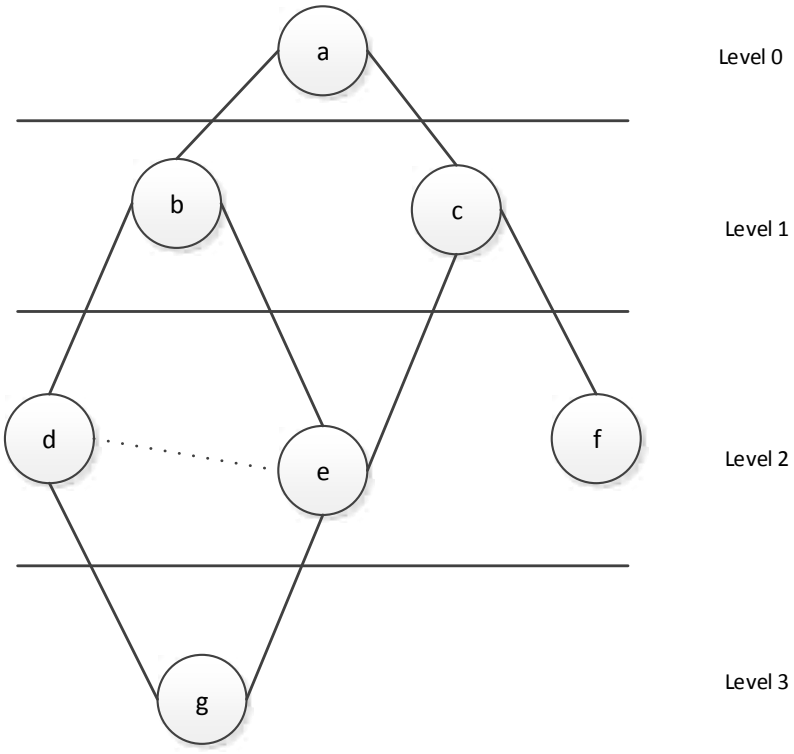

Figure 5.2: Solid lines are the paths visited while doing Breadth First Search (BFS). Dotted lines indicate that it has found visited node which confirms that there is a cycle.

Lemma 5.4.1 *If a primitive ring r is formed at level k, then no subring of r will be formed in any level, $l > k$.*

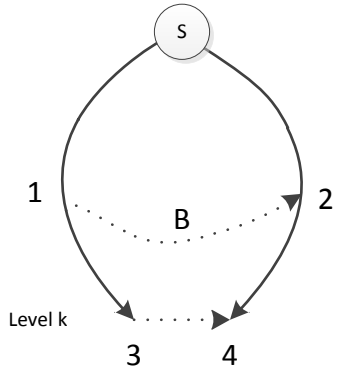

Figure 5.3: S is the source node. 1,2,3,4 are any node on the paths S-1-3 and S-2-3. 3 and 4 are the leaf nodes at level k and dotted line between them is one edge link. B is any arbitrary path from 1 to 2.

Proof. Every node in ring r is in level less than or equal to k. Until k, since r is primitive, the distance between any two nodes in ring r is the shortest path distance along the path of the ring itself. In Figure 5.3, let $S-1-3-4-2-S$ be a primitive ring at level k. At level $l > k$, we found a path B between two arbitrary nodes 1 (on path $S-1-3$) and 2 (on path $S-2-4$). For $S-1-B-2-S$, to be subring of $S-1-3-4-2-S$, $distance(S-1-B-2-S) < distance(S-1-3-4-2-S)$. Since the path B is formed in level m, at least one node in path B should be from m which makes $distance(S-1-B-2-S) > distance(S-1-3-4-2-S)$. Hence this makes $S-1-B-2-S$ not subring of $S-1-3-4-2-S$.

□

5.5 Algorithm Description

This algorithm initially starts by finding all shortest paths from a single source node S to all other nodes at each level by applying Breadth First Search (BFS). BFS forms a tree which is denoted by $T(S)$. Note that a node v of $T(S)$ can have multiple shortest paths from node S denoted by set $P(S,v)$. In each level of $T(S)$, for all v, we check if it is linked with any node v' of same level. If we find a one edge link from v to v', we first check if any path in $P(S,v)$ and $P(S,v')$ are good pair. If they qualify from the good pair test, we form a ring. In case of even ring we check if v

and v' are connected by a node at one level greater than that of v and v'. For all remaining part of algorithm description we will take odd ring as an example.

In Lemma 5.4.1, we have explained that if two paths p_1 and p_2 form primitive ring at level k then any path which are the extension of p_1 and p_2 will not form any primitive rings with each other at level greater than k. Therefore if we know the pair of shortest paths at level k which have formed primitive ring(s) then we use this information and reduce the number of comparison among shortest paths in level $l > k$.

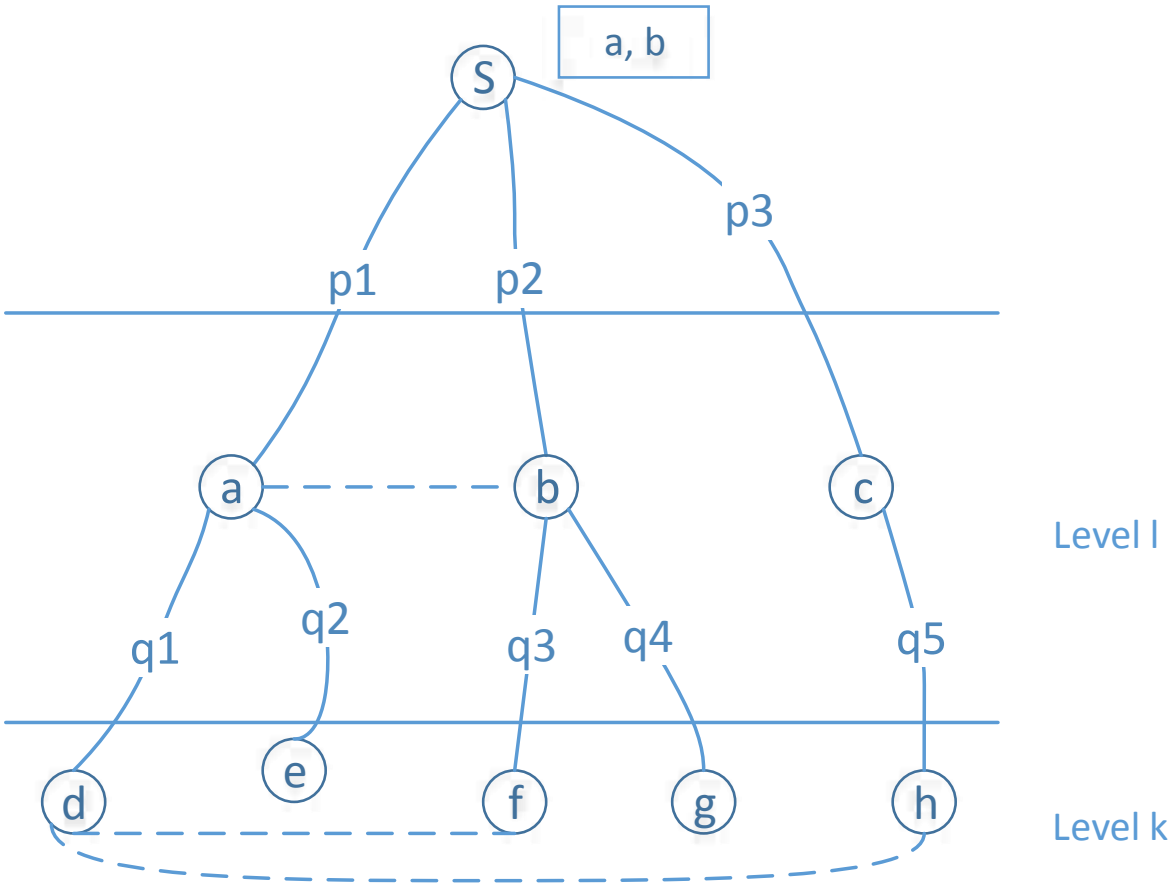

Figure 5.4: Representation of search space for primitive ring mining. Lines with arrow head are pointer to the particular nodes. Dotted lines are one edge connection. Rectangular boxes are tables which stores the information of paths that have formed primitive rings.

In Figure 5.4, let node a and b are connected by an edge. Now we check if $p1$ and $p2$ are good pair and form a ring. Note that a and b can have multiple shortest paths from S. If $p1$ and $p2$ form a ring we store the information of $p1$ and $p2$ in lookup table. The lookup table stores the tail node of $p1$ and $p2$ as a pair. Now for every pair of paths, we check if any of the node in the path has entry in lookup table. As shown in Figure 5.4, d, e, f, g, h are nodes at level $k > l$. According to Figure 5.4 there is a link between d and f. Therefore we check if path $p1 + q1$ and $p2 + q3$ form a primitive ring. If any node $n1 \in \{p1 + q1\}$ and any node $n2 \in \{p2 + q3\}$ have entry in lookup table as a pair then they will not form a primitive ring. As a result of which we don't have to go through primitive ring test. In look up table we find the entry of a and b. They are the nodes in $p1 + q1$ and $p2 + q3$ respectively. Therefore, $p1 + q1$ and $p2 + q3$ do not qualify for primitive ring test. Apart from d and f, we also have one edge connection between d and h. Since, none of the nodes in $p1 + q1$ and $p2 + q3$ have entry in lookup table as a pair, they become the candidate for primitive ring test.

5.5.1 Primitive Ring Test

To test if a given ring r is a primitive ring is straight forward. For every pair of nodes in r, we check if there exists a path shorter than the ring distance. If we find such path then the ring is not primitive. To implement this we check if given two pairs are good pairs or not.

Every time we find two nodes of same level connected by an edge there will be a ring but we have to make it sure first if it is primitive or not. According to Lemma 5.4.1, only two cases are possible: 1) discovering a primitive ring 2) discovering a super ring. Once a primitive is formed by any two paths $p1$ and $p2$, they will never form another primitive ring again which is proven by Lemma 5.4.1. Therefore, we eliminate large number of candidate paths for primitive ring checking.

Algorithm 1 is the pseudo-code for primitive ring testing. We give $path1$ and $path2$ as input, we then form a ring r by concatenating those two pairs. It goes through every pair of nodes of two paths and check if there exist any shortest distance.

Algorithm 1: Good_Pair

Input: $path1, path2, L$
Output: $boolean$

```
if path1 and path2 share any node other than source node then
    return false
Create a ring r from path1 and path2
for node n1 in r do
    for node n2 in r do
        Find shortest distance from n1 to n2
        if SD < RD then
            return false
return true
```

Algorithm 2: Build_Shortest_Path

Input: l,node[]
Output: shortest_paths[]
Assumption: Q is a special queue which separates nodes of different levels with level separator.

```
lvl ← 0
Create a Q
Q.enqueue(node[])
while Q is not empty ∧ lvl ≤ l do
    while Q.peek! = lvl_separator do
        v ← Q.dequeue
        for adj_n ∈ v.adjacentnodes do
            if adj_n is not visited then
                adj_n.visited ← true
                Q.enqueue(adj_n)
            if v.level < adj_n.level then
                for path in shortest_path[node, v] do
                    shortest_path[node, adj_n] ← path + adj_n
    if all nodes of single level are processed then
        Q.enqueue(lvl_separator)
        lvl++
```

Algorithm 3: Primitive_Ring_Mining

```
   Input: L,node
   Output: ring_count[]
1  for l = 0 to (L-1)/2 do
2      Build_Shortest_Path(l, all nodes of level l)
3      for path1 in shortest_path do
4          for path2 in shortest_path do
5              if (∀ node1 ∈ path1, ∀ node2 ∈ path2) pair not in lookup table then
6                  if Primitive_Ring_Test(path1,path2) then
7                      form_ring with path1 and path2
8                      ring_count[l] ← ring
9                      add path1.tailnode and path2.tailnode pair in lookup table
```

Algorithm 2 is basically about finding shortest paths from a source node. We have used BFS approach to find all the shortest paths of size l. Here shortest_path[] list is a global data structure which is shared by both Build_Shortest_Path and Primitive_Ring_Mining methods.

In algorithm 3, we have given the pseudo-code of primitive ring mining. It starts with building a shortest_path table. It contains the shortest path information of every node up to level $\leq (L-1)/2$ (for odd ring) from a source node. For even ring it is up to $L/2$. At each level, it does good pair check for all pair of paths (line 5) . If it finds any good pair, it goes further for primitive ring test (line 6). When a good pair passes primitive ring test, it forms a primitive ring and stores it in $ring\_count[]$ list (line 8). $ring\_count[]$ stores primitive rings of size $\leq L$ for a given node. After forming a ring it stores necessary information in lookup tables of root node (line 9).

5.6 Experiment and Results

We used data from parallel first-principles molecular dynamics (FPMD) simulations of silica liquid to test our algorithm. The supercell used in the simulation consisted of 72 atoms. From which we generated coordination file which had coordination information of all atoms. We used AtomViz to generate this file. From coordination file, we only selected Si atoms. Each Si atom is bonded with another Si atom via O atom. Here nodes are Si atoms and bond between Si atoms are edges.

Since Si atoms are bonded through O atoms, we considered that bond via O as an edge. If there are n time stamps, then there will be n number of graphs. So the input will be the set of graphs, $G = G_1, G_2, ..., G_n$.

We used the simulation data of 100,000 time steps. Therefore, for each time step, we created one graph. Our main goal for this experiment was to analyze the distribution of different size of rings during normal and big moved time windows [61]. We first captured all the atoms which showed large displacement using windows detection technique as explained in Chapter 3. For each big moved atom, we mined the time windows where it showed big movement. Let atom a shows large movement and $W = \{w_1, w_2, w_3\}$ is the big moved time windows of a. Each $w \in W$ contains series of time steps. So for each time window we will have $|w|$ number of graphs. In each such graphs we mine the primitive rings of different size. We did this for every big moved atoms and took the average of ring count.

Let $A = \{a_1, a_2, ...a_m\}$ be the set of big moved atoms and $W_{a_i} = \{W^1_{a_i}, W^2_{a_i}, ...W^n_{a_i}\}$ be the set of big moved time windows for each atom $a_i \in A$ where $i = \{1, 2, 3...m\}$. Now we calculate the count of ring of size r_{size} as:

$$F(r_{size}) = \sum_{a_i \in A} \frac{\sum\limits_{W^j_{a_i} \in W_{a_i}} N_{a_i, W^j_{a_i}}(r_{size})}{|W_{a_i}|} \tag{5.1}$$

Here, $F(r_{size})$ is the total average count of ring of size r_{size}. $N_{a_i, W^j_{a_i}}(r_{size})$ is the count of ring of size r_{size} of atom a_i during time window $W^j_{a_i}$ where $j = \{1, 2, ...n\}$. For this experiment we counted 3, 4, 5, 6, 7, 8 and 9 member rings.

Figure 5.5 shows the distribution of rings for normal time steps and big moved time windows. We found similar distribution for both cases. The ring count gets to peak value at ring size 6 and again starts to decrease.

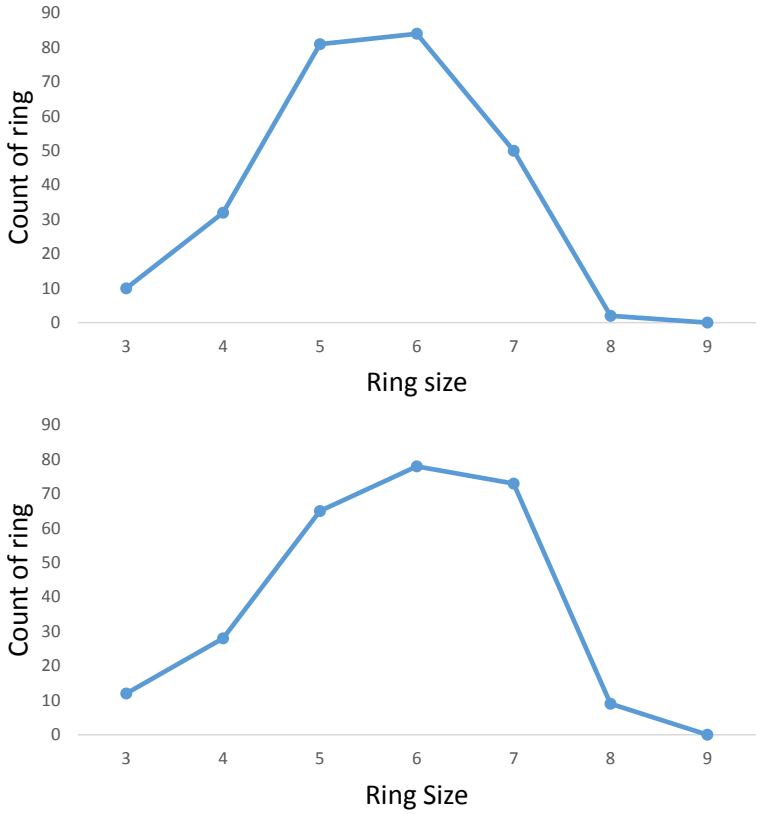

Figure 5.5: Top: Ring distribution for big moved time windows. Bottom: Ring distribution for normal move time windows.

ANOVA Test

To check if the two distribution of rings are similar, we did ANOVA test. ANOVA test is used to determine if there is significance difference between the means of two of more unrelated groups. It also tests whether two groups comes from population with equal variance. Specifically, it tests the null hypothesis:

$$H_0 = \sigma_1^2 = \sigma_2^2 \tag{5.2}$$

From the calculation, we get F(1, 10) = 0.002. Therefore, the F test indicates that there is not enough evidence to reject the null hypothesis that the two batch variances are equal at the 0.05 significance level. Hence we can say that the two distribution of rings are from the similar distributions.

Performance Analysis

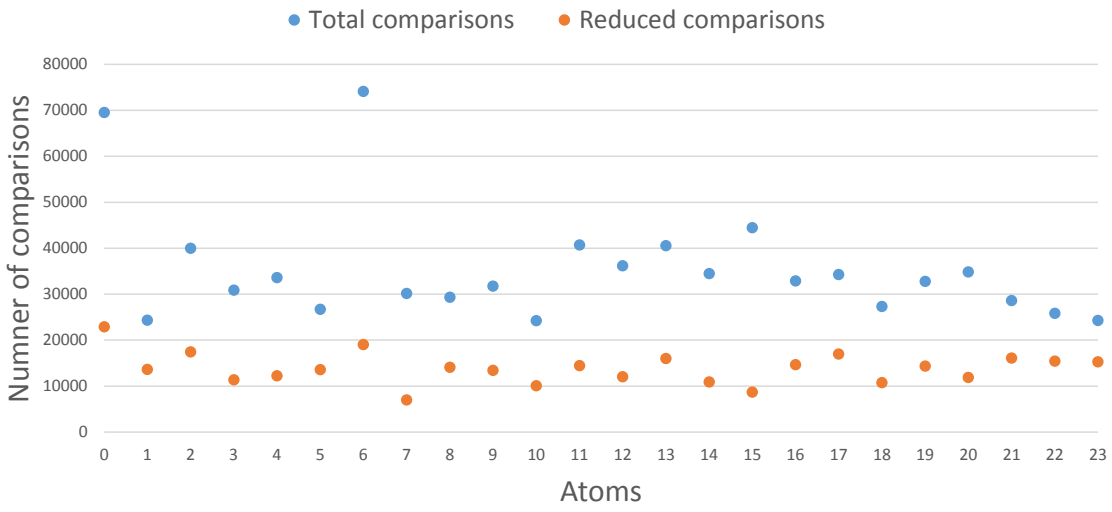

Figure 5.6: Number of comparisons made for each atoms. Y-axis is the count of comparisons and X-axis is the atom ID of 24 Si atoms.

We claim that our algorithm reduces the number of comparisons to be made for the primitive ring test by significant magnitude. To test this, we chose 500 random graph samples and ran our algorithm and for all 23 atoms. Here, choosing 500 random graph samples is equivalent to choosing 500 random time stamps. We ran both brute force and our algorithm on same set of graphs. In Figure 5.6 blue dots are the number of comparisons made by brute force algorithm for each atom in 500 graphs and green dots are reduced number comparisons made from our algorithm in same setup. The figure shows that most of the time the number of comparisons we have to make for primitive ring test is reduced by around 50%. This is a big gain in terms of performance comparing to the existing ring counting algorithm. In Figure 5.7, with the help of our heuristic, we can see the total primitive ring test have been reduced by more than 50%.

5.7 Discussion and Conclusion

The new algorithm has enabled primitive ring analysis for several thousand time steps of data. We studied the distribution of primitive rings for both big moved time windows and normal time

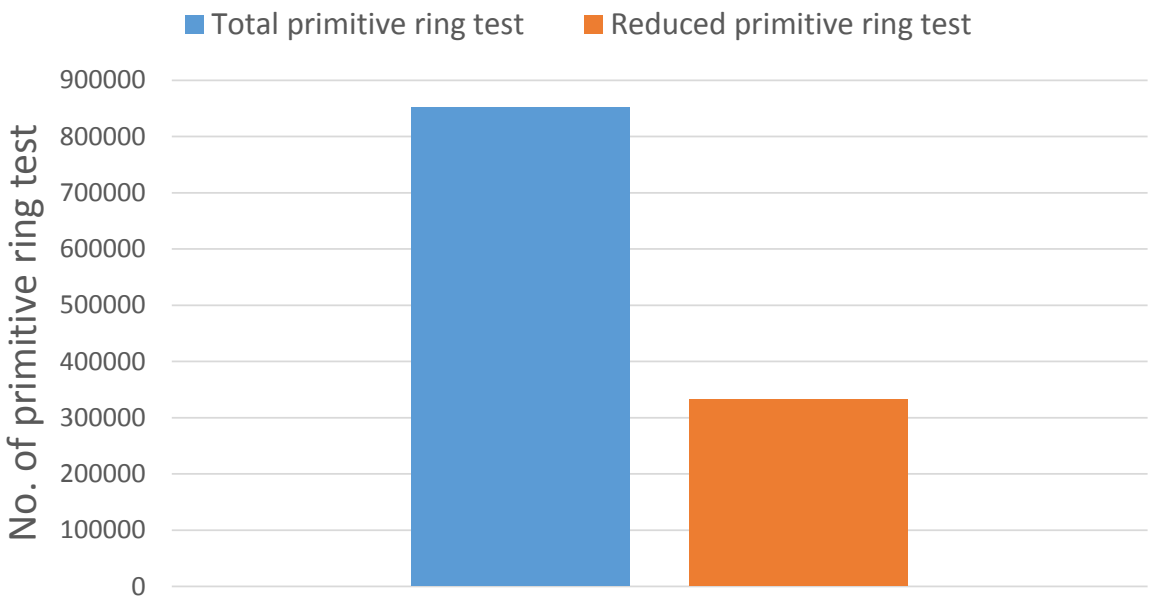

Figure 5.7: Comparison between existing ring counting algorithm and heuristic developed by us.

windows. It gives us the ability to obtain primitive ring statistics with virtually no upper limit on ring size and thus provides a powerful tool for the study of mid-range structure in materials with topological network structures, such as silicate liquid. All experiments of ToPoMine are done on an Intel(R) Core(TM) i7 @2.80 GHz PC with 4 GB of main memory running 64 bit Windows 7.

We developed a heuristic which made the primitive ring mining algorithm efficient in terms of comparing shortest distance path. The knowledge we gained while mining k-primitive ring is transferred to k+1 primitive ring mining process. This helped to gain the efficiency in running time. The algorithm has been tested on SiO_2 liquid.

Chapter 6
Conclusion and Summary

In this thesis, we studied about the dynamic behavior of Si and O atoms in SiO_2 liquid. By employing novel data mining techniques we studied on the movement pattern of Si and O atoms. In material science, micro-level (atomic-level) activities are the key to understand microscopic level properties of a type of material. We explored the ways to discover such knowledge by applying data mining particulary graph mining techniques.

In Chapter 1, we discussed about molecular dynamics and how data mining can help to gain insight on the dynamic behavior of materials. We also gave the motivation for the work. The motivation is driven by the interest in material science model and predict macroscopic properties e.g. strength, viscosity of a type of material. We indicated that the interesting activities, i.e. the activities that may form the atomic mechanism underlying a particular material property are often buried under a lot of other activities that are of no interest. Therefore we point out the fact that there is a great potential for data mining to play an important role in this type of discoveries.

In Chapter 2, we discussed about the preprocessing of raw data. We used the mechanism like Radial Distribution Frequency (RDF) to calculate coordination number and convolution theorem to smooth the data. We also discussed about the unfolding of atom coordinates to remove the periodicity in side the super cell.

In Chapter 3, we proposed an approach to analyze a given position-time series data produced by parallel molecular dynamics simulation for understanding the dynamical behavior of the material system. We developed a system to automate the detection of large atomic jumps in molecular dynamics simulations. We indicated that our automated analysis system can play an important role

in obtaining insight on atomic activities particulary for the liquid phase SiO_2 at relatively low temperature. Additionally, we showed that the detection of large movements allowed us to investigate the bond change that are more closely connected with atomic movements.

In Chapter 4, we presented the literature on Frequent Subgraph Mining (FGM). we mentioned that the research on FGM are mainly focussed at i) effective mechanism for generating candidate subgraphs (without generating duplicates) and ii) how best to process the generated candidate subgraphs so as to identify the desired frequent subgraphs in a computationally efficient way. We were mainly focussed in three most important part of FGM process: i) candidate generation strategy, ii) search strategy and iii) approach to frequency counting. We also, we addressed the problem of discovering pattern in atomic dynamics for material study. We considered the problem of pattern discovery in atom dynamics form material simulation. We proposed an event graph to model the atomic activities. We further proposed an algorithm to mine the frequent subgraphs in the event graph. We took advantage of the topological sort in generating candidate subgraphs during mining process. We showed the potential of our algorithm by experimenting on simulation data of SiO_2 system.

In Chapter 5, we studied, the topological structure of SiO_2. We proposed an efficient algorithm to mine the primitive ring. We showed the distribution of primitive rings for both big moved time windows and normal time windows, which we calculated using algorithm described in Chapter 3.

6.1 Future Directions

In Chapter 3, we have developed a novel and handy method to automatically detect atomistic events that accounts for material transport in molten silica system. Time series of atomic coordinates is usually very noisy, especially at high temperature and pressure, hindering detailed analysis. So developing a simple and robust analysis method is quite useful for computational materials science community. While our current approach deals with post data processing, it would be interesting to extend to it to perform in-situ analysis. Performing data analysis in real time will help us to perhaps save only those time windows in which potentially interesting atomic events

occur. Thus, reduced amounts of data (small subsets of data) can be saved for subsequent detailed analysis such as visualization of atomic movements and bond events. A real time data analysis is particularly important for very large atomic systems. However, this requires training our model with test data so that the appropriate threshold values could be estimated and implementation of our model for parallel processing. Moreover, our analysis consider a system with 72 atoms, so it will be interesting to investigate the larger system with millions of atoms with the efficient parallel implementation of our algorithm.

In Chapter 4, we proposed an event-graph model to model the atomic dynamics and proposed a graph mining algorithm to discover popular subgraphs in the event graph. We implemented and tested our algorithms on simulations of the material silica (SiO_2). The results show the potential of our mining technique for material-science discoveries. However, our event graph model is not only applicable for chemical compounds but also for other data models which can be represented in event graph model. So it would be interesting to investigate data from other fields which can be represented as event graph. Further more, comparing our ToPoMine algorithm with other pattern growth approach algorithm such as gSpan [73] in terms of candidate generation and canonical encoding technique would be a potential direction for research.

Primitive ring mining problem, which we discussed in Chapter 5 is a NP-complete problem. Therefore every existing algorithms for mining such rings are exponential. We can only come up with some heuristic which reduces the search space such that we can achieve some gain in execution time. Like other algorithms, we counted the primitive rings by enumerating them. However, if we could come up with an algorithm which could count the rings without enumerating them, then the complexity of algorithm can go down from exponential to polynomial time. This would be an another prospective domain to work on.

Bibliography

[1] Challenges and opportunities with big data. *A community white paper developed by leading researchers across the United States*.

[2] Interactive structure analysis of amorphous and crystalline systems manual. *http://isaacs.sourceforge.net/*.

[3] Scikit learn. *http://scikit-learn.org/stable/modules/mixture.html*.

[4] Theory of molecular dynamics simulation. *http://www.ch.embnet.org/MD_tutorial*.

[5] A survey on algorithms of mining frequent subgraphs. *International Journal of Engineering Inventions*, 2012.

[6] Rahaman A. *Phys. Rev.*, A136, 1974.

[7] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[8] B. J. Alder and T. E. Wainwright. *J. Chem. Phys.*, 27, 1957.

[9] M. P. Allen and D. J. Tildesley. *Computer simulation of liquids*. MIT Press, 1989.

[10] Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroshi Sakamoto, and Setsuo Arikawa. Efficient substructure discovery from large semi-structured data. In *SIAM SDM*, pages 158–174, 2002.

[11] R. J. Bell and P. Dean. The structure of vitreous silica: Validity of the random network theory. *Philosophical Magazine*, 25(6):1381–1398, 1972.

[12] Noah C. Benson and Valerie Daggett. A comparison of multiscale methods for the analysis of molecular dynamics simulations. *The Journal of Physical Chemistry B*, 116(29):8722–8731, 2012.

[13] D. Bhattarai and B.B. Karki. Atomistic visualization: Space-time multiresolution integration of data analysis and rendering. *Journal of Molecular Graphics and Modelling*, 27:951:33–38, 2009.

[14] Dipesh Bhattarai. Space-time multiresolution approach to atomistic visualization. *PhD Dissertation in Computer Science, Louisiana State University, Baton Rouge, LA, USA*, 2008.

[15] Bidur Bohara and Bijaya B. Karki. Rendering particle trajectories with color-coded information for atomistic visualization. *The 12th IASTED International Conf. on Visualization, Imaging, and Image Processing (VIIP 12)*, 2012.

[16] Christian Borgelt and Michael R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 51–58, 2002.

[17] Soumen Chakrabarti, Byron E. Dom, David Gibson, Jon Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Mining the link structure of the world wide web. *IEEE Computer*, 32:60–67, 1999.

[18] Yun Chi, Yirong Yang, and Richard R. Muntz. Canonical forms for labelled trees and their applications in frequent subtree mining. *Knowl. Inf. Syst.*, 8(2):203–234, August 2005.

[19] Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, pages 231–255, 1994.

[20] Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. *J. Artif. Int. Res.*, 1(1):231–255, February 1994.

[21] Luc Dehaspe, Hannu Toivonen, and Ross Donald King. Finding frequent substructures in chemical compounds. In *4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 30–36, 1998.

[22] F. Diebold. On the origin(s) and development of the term big data. *Pier working paper archive, Penn Institute for Economic Research, Department of Economics, University of Pennsylvania.*, 2012.

[23] Wei Fan and Albert Bifet. Mining big data: Current status, and forecast to the future. *SIGKDD Explor. Newsl.*, 14(2):1–5, April 2013.

[24] Usama Fayyad, Gregory Piatetsky-shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[25] Gary William Flake, Robert E. Tarjan, and Kostas Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1:385–408, 2004.

[26] K. Goetzke and H.-J. Klein. Properties and efficient algorithmic determination of different classes of rings in finite and infinite polyhedral networks. *Journal of Non-Crystalline Solids*, 127(2):215 – 220, 1991.

[27] Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, Luigi Pontieri, and Domenico Sacca. Mining constrained graphs: The case of workflow systems.

[28] Thomas Grtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *IN: CONFERENCE ON LEARNING THEORY*, pages 129–143, 2003.

[29] Christoph Helma, Stefan Kramer, Luc, and De Raedt. The molecular feature miner molfea. In *In Proceedings of the Beilstein Workshop 2002: Molecular Informatics: Confronting Complexity; Beilstein Institut*, 2002.

[30] V.V. Hoang. Molecular dynamics simulation of amorphous sio2 nanoparticles. *J Phys Chem B*, 111(44):12649–56, 2007.

[31] Linn W. Hobbs, C. Esther Jesurum, Vinay Pulim, and Bonnie Berger. Local topology of silica networks. *Philosophical Magazine A*, 78(3):679–711, 1998.

[32] Jürgen Horbach and Walter Kob. Static and dynamic properties of a viscous silica melt. *Physical Review B*, 60:3169–3181, 1999.

[33] Hsun-Ping Hsieh and Cheng-Te Li. Mining temporal subgraph patterns in heterogeneous information networks. In *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, pages 282–287, 2010.

[34] Haiyan Hu, Xifeng Yan, Yu Huang, Jiawei Han, and Xianghong Jasmine Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21:213–221, 2005.

[35] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of the Third IEEE International Conference on Data Mining*, pages 549–552, 2003.

[36] Jun Huan, Wei Wang, and Jan Prins. Spin: Mining maximal frequent subgraphs from graph databases. In *In KDD*, pages 581–586, 2004.

[37] W. Humphrey, A. Dalke, and K. Schulten. VMD: visual molecular dynamics. *Journal of molecular graphics*, pages 33–38, 1996.

[38] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, pages 13–23, 2000.

[39] Nikita Jain and Vishal Srivastava. Data mining techniques: A survey paper. *International Journal of Research in Engineering and Technology*, 2013.

[40] Benneyan JC. An introduction to using computer simulation in healtcare. *J Soc Health Syst.*

[41] Chuntao Jiang, Frans Coenen, and Michele Zito. A survey of frequent subgraph mining algorithms. 2004.

[42] Ya juan Jin, Bao hua Yang, and Yuan he Huang. Molecular orbital studies on the rings composed of {D2d} {C36} cages. *Computational Materials Science*, 36(4):474 – 479, 2006.

[43] Yiping Ke, James Cheng, and Wilfred Ng. Correlation search in graph databases. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 390–399, New York, NY, USA, 2007. ACM Press.

[44] Brian P. Kelley, Roded Sharan, Richard M. Karp, Taylor Sittler, David E. Root, Brent R. Stockwell, and Trey Ideker. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proceedings of the National Academy of Sciences*, 100(20):11394–11399, 2003.

[45] Nikhil S. Ketkar. Subdue: compression-based frequent pattern discovery in graph data. In *in OSDM 05: Proceedings of the 1st international workshop on open source data mining*, pages 71–76. ACM Press, 2005.

[46] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46:604–632, 1999.

[47] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 313–320, 2001.

[48] Michihiro Kuramochi and George Karypis. Grew-a scalable frequent subgraph discovery algorithm. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, pages 439–442, 2004.

[49] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.*, pages 243–271, 2005.

[50] V. Lakshmanan and J. Kain. A Gaussian mixture model approach to forecast verification. *Weather and Forecasting*, 25(3):908–920, 2010.

[51] K. Lakshmi. Frequent subgraph mining algorithms- a survey and framework for classification.

[52] Ju Li. Atomistic visualization. In *Handbook of Materials Modeling*, pages 1051–1068. Springer Netherlands, 2005.

[53] Yuhua Li, Quan Lin, Gang Zhong, Dongsheng Duan, Yanan Jin, and Wei Bi. A directed labeled graph frequent pattern mining algorithm based on minimum code. In *Proceedings of the 2009 Third International Conference on Multimedia and Ubiquitous Engineering*, pages 353–359, 2009.

[54] Gelin B. R. McCammon, J. A. and M. Karplus. *Nature (Lond.)*, 267, 1977.

[55] Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 647–652, 2004.

[56] Tom O'Haver. Smoothing. *http://terpconnect.umd.edu/ toh/spectrum/Smoothing.html*.

[57] Carl Edward Rasmussen. The infinite gaussian mixture model. In *In Advances in Neural Information Processing Systems 12*, pages 554–560. MIT Press, 2000.

[58] Douglas Reynolds. Gaussian mixture models. *MIT Lincoln Laboratory*.

[59] J. Rino, I. Ebbsj I, R. Kalia, A. Nakano, and P. Vashishta. Structure of rings in vitreous sio2. *Phys Rev B Condens Matter*, pages 3053–3062, 1993.

[60] N. Vandana Sawant, Ketan Shah, and Vinayak Ashok Bharadi. Survey on data mining classification techniques. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, pages 1380–1380. ACM, 2011.

[61] Shobhit S. Shakya, Bijaya B. Karki, and Jian Zhang. Analyzing molecular dynamics scattered data for large atomic movements. *Computational Materials Science*, 95(0):198 – 206, 2014.

[62] A. Sharma, Xinlian Liu, P. Miller, Aiichiro Nakano, Rajiv K. Kalia, Priya Vashishta, Wei Zhao, T.J. Campbell, and A. Haas. Immersive and interactive exploration of billion atom systems. In *Virtual Reality, 2002. Proceedings. IEEE*, pages 217–223, 2002.

[63] Dennis Shasha, Jason T. L. Wang, and Rosalba Giugno. Algorithmics and applications of tree and graph searching. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 39–52. ACM, 2002.

[64] Lander Simulation and Training Solutions. History of computer simulation.

[65] Steven W. Smith. The scientist and engineer's guide to digital signal processing. *http://www.dspguide.com/ch6/2.htm*.

[66] F. H. Stillinger and A. Rahman. *J. Chem. Phys.*, 60, 1974.

[67] John E. Stone, Kirby L. Vandivort, and Klaus Schulten. GPU-accelerated molecular visualization on petascale supercomputing platforms. In *Proceedings of the 8th International Workshop on Ultrascale Visualization*, UltraVis '13, pages 6:1–6:8. ACM, 2013.

[68] Lini T. Thomas, Satyanarayana R. Valluri, and Kamalakar Karlapalem. Margin: Maximal frequent subgraph mining. *ACM Trans. Knowl. Discov. Data*, 10:1–42, 2010.

[69] Jianyong Wang, Zhiping Zeng, and Lizhu Zhou. Clan: An algorithm for mining closed cliques from large dense graph databases. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *ICDE*, page 73. IEEE Computer Society, 2006.

[70] Takashi Washio and Hiroshi Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, July 2003.

[71] Dong Xin, Hong Cheng, Xifeng Yan, and Jiawei Han. Extracting redundancy-aware top-k patterns. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 444–453. ACM, 2006.

[72] Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S. Yu. Mining significant graph patterns by leap search. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 433–444, 2008.

[73] Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 721–724, 2002.

[74] Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 286–295. ACM, 2003.

[75] Xifeng Yan, Philip S. Yu, and Jiawei Han. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 766–777. ACM, 2005.

[76] Xifeng Yan, X. Jasmine Zhou, and Jiawei Han. Mining closed relational graphs with connectivity constraints. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 324–333, New York, NY, USA, 2005. ACM.

[77] Chang Hun You, Lawrence B. Holder, and Diane J. Cook. Graph-based data mining in dynamic networks: Empirical comparison of compression-based and frequency-based subgraph mining. In *Proceedings of the 2008 IEEE International Conference on Data Mining Workshops*, pages 929–938, 2008.

[78] Xianglong Yuan and A.N. Cormack. Efficient algorithm for primitive ring statistics in topological networks. *Computational Materials Science*, 24(3):343 – 360, 2002.

[79] Xianglong Yuan, Vinay Pulim, and Linn W Hobbs. Molecular dynamics refinement of topologically generated reconstructions of simulated irradiation cascades in silica networks. *Journal of nuclear materials*, 289(1):71–79, 2001.

[80] Mohammed J. Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2002.

[81] Mohammed J. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. *in IEEE Transaction on Knowledge and Data Engineering*, 17:1021–1035, 2005.

[82] Mohammed J. Zaki and Ching-Jui Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Trans. on Knowl. and Data Eng.*, 17(4):462–478, April 2005.

[83] Cheng Zhang, Bhupesh Bansal, Paulo S. Branicio, Rajiv K. Kalia, Aiichiro Nakano, Ashish Sharma, and Priya Vashishta. Collision-free spatial hash functions for structural analysis of billion-vertex chemical bond networks. *Computer Physics Communications*, 175(5):347, 2006.

[84] Cheng Zhang, Bhupesh Bansal, Paulo S. Branicio, Rajiv K. Kalia, Aiichiro Nakano, Ashish Sharma, and Priya Vashishta. Collision-free spatial hash functions for structural analysis of billion-vertex chemical bond networks. *Computer Physics Communications*, 175(5):339–347, 2006.

[85] Dongju Zhang, Mingwen Zhao, and R. Q. Zhang. Two- and three-membered-ring hybrid structures of silica nanoclusters. *The Journal of Physical Chemistry B*, 108(48):18451–18454, 2004.

Vita

Shobhit S. Shakya was born in Bheri zone, Nepal, in February, 1984, to Bharat Kumar Shakya and Girija Shakya. He received his Bachelors degree in Computer Engineering from Tribhuvan University, Nepal 2006. He worked as a software engineer for some time before joining Louisiana State University, Baton Rouge, USA in January 2009 for doctorate degree in computer science. He worked as a research assistant to Dr. Jian Zhang while working toward the doctoral degree. His doctoral work involved machine learning and graph mining.