PATTERN INFORMATION EXTRACTION FROM CRYSTAL STRUCTURES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Erhan Okuyan

December, 2005

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Uğur Güdükbay (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Oğuz Gülseren

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

PATTERN INFORMATION EXTRACTION FROM CRYSTAL STRUCTURES

Erhan Okuyan
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. Uğur Güdükbay
December, 2005

Determining crystal structure parameters of a material is a quite important issue in crystallography. Knowing the crystal structure parameters helps to understand physical behavior of material. For complex structures, particularly for materials which also contain local symmetry as well as global symmetry, obtaining crystal parameters can be quite hard. This work provides a tool that will extract crystal parameters such as primitive vectors, basis vectors and space group from atomic coordinates of crystal structures. A visualization tool for examining crystals is also provided. Accordingly, this work presents a useful tool that help crystallographers, chemists and material scientists to analyze crystal structures efficiently.

Keywords: crystal, crystallography, chemistry, material science, pattern recognition, primitive vectors, basis vectors, space group, symmetry.

ÖZET

KRİSTAL YAPILARDAN KALIP BİLGİSİ ÇIKARMA

Erhan Okuyan
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. Uğur Güdükbay
Aralık, 2005

Maddelerin kristal parametrelerinin belirlenmesi, kristallografi de önemli bir konudur. Kristal parametrelerinin bilinmesi, maddelerin fiziksel özelliklerinin anlaşılmasına yardımcı olur. Karmaşık yapılı maddelerde, özellikle eğer madde global simetri yanında lokal simetri de barındırıyorsa, kristal parametrelerinin belirlenmesi oldukça zor olabilir. Bu çalışmada, primitif vektörler, temel vektörler ve uzay grubu gibi kristal parametrelerini, kristal yapısını oluşturan atomların koordinat bilgilerini kullanarak belirleyecek bir araç ortaya çıkarılmıştır. Ayrıca, kristalleri incelemeye yarayan bir görüntüleme aracı da sunulmuştur. Dolayısıyla, bu çalışma, kristal bilimcilere, madde bilimcilere ve kimyacılara, kristal yapılarını daha verimli bir şekilde analiz etmeyi sağlayan faydalı bir araç sunmaktadır.

Anahtar sözcükler: kristal, kristallografi, kimya, madde bilimi, kalıp algılama, primitif vektörler, temel vektörler, uzay gurubu, simetri.

Acknowledgement

I would like to express my gratitude to my thesis supervisor Asst. Prof. Dr. Uğur Güdükbay for his encouragement, support and belief in my work. I would like to thank Asst. Prof. Dr. Oğuz Gülseren for giving his time to discuss various aspects of my thesis work.

I would like to thank Professors Şefik Süzer and Cemal Yalabık for valuable discussions.

Finally, I would like thank my family for their support and understanding throughout my thesis study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Obtaining parameters of crystal structures is a quite important issue in crystallography. Crystal structure of a material is closely related with its physical properties. Accordingly, obtaining crystal parameters of a material will help to understand its physical behavior. In material science, crystal parameters are used to classify materials. This classification helps to analyze materials effectively. Particularly, for complex cases such as alloys whose atomic ratios can change, such classification is quite useful to analyze physical properties.

In crystallography, mainly x-ray diffraction techniques are used to determine crystal structure of materials. This technique uses x-ray absorbance data to reveal crystal geometry [9, 11]. These techniques generally give satisfactory results. However, there are cases where obtained results are confusing or insufficient. In those cases, crystallographers have to test several crystal geometries manually. Scientists may also work on theoretical materials, where no sample is available. For those cases, x-ray diffraction techniques are not applicable. Accordingly, a tool that can find crystal parameters from atomic coordinates could be very useful.

The aim of this work is to extract pattern information in any crystal structure from raw atomic coordinates by calculating primitive vectors and basis vectors, and identifying the space group. This task is relatively easy for a human on

simple structures. However, it becomes quite hard for complex structures and usage of a computerized system becomes necessary. Another complication of this process is the existance of molecular structures in crystals. There are many molecular materials that form crystals. These molecular materials can be simple molecules such as H_2O, or they can be quite complex biological materials such as DNA or protein. Accordingly, a computerized approach is essential to handle such complex cases.

Since crystals are repeated patterns of atomic positioning in 3D space, it can be quite hard to understand crystal geometry. Accordingly, a 3D visualization tool is essential to understand crystal geometry. Secondary motivation of this work is to provide a good 3D visualization tool that allows users to explore crystal structure effectively. In this way, this work will be useful for people learning crystallography, as well as professionals. The visualization tool works on unit cell data that is either extracted from atomic coordinates or provided by the user directly. This tool allows observing unit cells in several angles, combining several unit cells to obtain larger crystal segments, showing or hiding several atom types, cutting crystal to obtain desired surfaces, dumping atomic coordinates that are shown, etc. Accordingly, the visualization tool helps scientists to understand crystal geometry effectively.

In this work, it is also assumed that sufficiently large volume of crystal structure is given as input data. It is assumed that atomic coordinates of atoms, which lie inside such volume, is generated within a small error margin and these atomic coordinates are used as input data. Each atoms type, which is given in input data, should also be identified. Since there can be more than one alternative combinations of primitive vector triplets and basis vector sets to define a crystal structure, the tool is designed to be semi-automatic so that it will ask the user the preferred primitive vector triplet alternative and preferred origin choice, throughout the analysis.

In this thesis, a framework that extracts parameters from crystal structure data is presented. Obtained crystal parameters are primitive vectors, basis vectors and space group number. Alternative unit cell parameters, such as the lengths of

primitive vectors and the angles between them are also calculated. The algorithms use atomic coordinates in crystal structure as input data.

The rest of the thesis is organized as follows. Chapter 2 presents terminology and related work. The details of the proposed algorithms are explained in Chapter 3, implementation details are explained in Chapter 4 and experimental results can be found in Chapter 5. Finally, Chapter 6 gives conclusions and possible future extensions.

Chapter 2

Background and Related Work

Crystal structure of a material is determined by relative positioning of atoms. The aim of this work will be to determine the relative positioning of atoms that is repeated throughout the crystal structure (pattern information). Since the subject is quite related to crystallography it is better to give some background information about it.

2.1 Background

In this section brief descriptions of some basic crystallographic terms, which are used frequently in this work, will be given. Descriptions are gathered from several sources, [2, 18, 9, 4, 11], and given in a summarized manner.

Crystal: Crystal is the term used for some solid material structure. It generally consists of single atoms or ions, but it may also contain molecules. In crystals atoms are placed at certain relative coordinates. They don't move under normal conditions. Every atom or molecule forming the crystal, interacts with all other atoms or molecules. Every unit structure attracts with each other with some physical forces. There are no bonds between any unit structures forming the crystal, hence there is no molecular formula either, as in H_2O formula. The

formula of a crystal simply represents the ratio of atoms or molecules forming the crystal. For example, $CaCl_2$ formula indicates that there are 2 Cl atoms for every Ca atom in $CaCl_2$ crystals. Crystals are formed by repetitions of unit cells, which can be defined as small identical construction blocks of crystals. Thus, crystal structures follow some pattern.

Unit Cell: Unit cells are small construction blocks of crystals. The shape of a unit cell can be rectangular box, hexagonal box or trapezoid. Basically, unit cells can be considered as a 3D geometric shape that can be placed side by side and fill some space completely. In general, the shape of a unit cell can be defined as a paralleloid. Accordingly six parameters are used to define a unit cell. In general representation, which consider unit cell shape as a paralleloid, there will be three different types of edges. The lengths of each edges and the angles between any two of these three edges will define the unit cell. In crystallographic terminology, lengths of these three edges are named as a,b and c. The angle between a and b is called γ, the angle between a and c is called β and the angle between b and c is called α. These six parameters define the unit cell. Another representation of unit cells is the vectoral representation. For a paralleloid, there will be three edges, intersecting at each corner of the paralleloid. Accordingly, if a corner of the paralleloid is considered as the origin, three vectors will be obtained from three edges which intersect at origin. These three vectors are called *Primitive Vectors*. The vectoral representation of unit cells is more popular and it is used in this work.

Basis: Basis defines the atomic placement within each unit cell. Crystal structure is formed by repetition of unit cells and the basis can be considered as the pattern that is repeated. Basis is represented as a list of vectors of atoms in unit cells. Basis includes a record for each atom which lies inside the unit cell. Each record contains information regarding this atoms type and its vectoral position. The vectoral position of an atom is calculated by using the same point as origin, which is used while calculating the primitive vectors.

Together with primitive vectors, basis defines the whole crystal structure. Basically, primitive vectors can be considered as the translation vectors. In a

crystal structure, any point which is obtained by translation of a basis atom by any integer combination of three primitive vectors will also contain an identical atom. For example, $CsCl$'s crystal structure is shown in Figure 2.1.

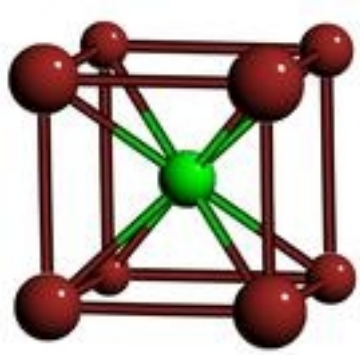

Figure 2.1: CsCl Unit Cell

Primitive Vectors in $\mathring{A}$ unit:

$$V_1 = [4.02, 0.0, 0.0]$$

$$V_2 = [0.0, 4.02, 0.0]$$

$$V_3 = [0.0, 0.0, 4.02]$$

Basis Vectors in $\mathring{A}$ unit:

$$B_1 = Cl, [0.0, 0.0, 0.0]$$

$$B_2 = Cs, [2.01, 2.01, 2.01]$$

The primitive vectors, V_1,V_2,V_3, and the basis vectors, B_1 and B_2, defines the crystal structure. Since Cl atom is at the origin, these vectors imply that there will be a Cl atom at every point which is an integer combination of V_1,V_2 and V_3, such as $(8.04, 4.02, 0.0)$,$(0.0, -4.02, 0.0)$ and $(4.02, 4.02, 4.02)$ points. Accordingly, Cs atoms will be placed at every point obtained by the translation of B_2 with any integer combination of V_1,V_2 and V_3, such as $(10.05, 6.03, 2.01)$,$(2.01, -2.01, 2.01)$ and $(6.03, 6.03, 6.03)$. All crystal structures can be defined in terms of 3 primitive vectors and a set of basis vectors.

Space Group: Unit cells show some symmetry properties. For example, $CsCl$ structure shown in Figure 2.1, has a rotational symmetry of 90 degrees. If you rotate the crystal structure, around one of the primitive vectors, by any integer multiple of 90 degrees, an identical placement would be obtained. There are several symmetry operations other than this rotational symmetry operation, such as mirror symmetries, translational symmetries, etc. Crystal structures are categorized into 230 groups, according to the symmetry operations they satisfy. These groups are called space groups. It is proven that any crystal structure must belong to one of these 230 space groups. Accordingly for any given crystal structure a corresponding space group can be found.

Crystallography: It is a branch of inorganic chemistry which studies crystals. It is particularly focused on the techniques that will reveal crystal structures of materials. Mainly, x-rays diffraction techniques are used for this purpose. Crystals diffracts x-rays with some particular angles, depending on the crystal geometry. Other than x-ray diffraction techniques, crystallography focuses on physical behaviors of crystals.

2.2 Previous Work

Extracting pattern information from atomic coordinates of a crystal structure is not a common problem. It can be used as an uncommon method in order to solve several problems one can met in crystallography. Accordingly, there is no significant amount of research directly related to this subject. However, since this subject is quite relevant to several other common subjects, there are several works that can partially help to this work. These works can be grouped into three main categories. The first category is crystallographic tools. For example, *Computational Crystallography Toolbox* [10], is one of the open source programs in this category. These tools allow users to define their own unit cell, by entering unit cell parameters. Users can examine atomic placements, perform several analyses, etc. Basically these tools help users to examine unit cell structure with every known detail and to understand the crystal structure more clearly. However,

since every essential unit cell parameter should be given to those tools as user input, their work is simply performing some parameter conversions, calculating some unit cell parameters which can be calculated using input parameters and providing a user interface.

The second category of previous works is crystallographic visualization tools. *RasMol* [17] program, is one of the well known example of this category. These tools aim to provide a good understanding on crystal geometry. They allow users to examine crystal structures from every aspect in 3D space. They provide several drawing models, such as ball model, ball-stick model, wire frame model, etc. They allow users to enable or disable showing some atom types, determining their colors and sizes, etc. They allow users to examine crystal structure other than unit cell perspective. In other words, by allowing to build multi-cells and allowing to cut the crystal structure according to user defined planes, these tools allow users to shape the crystal structure according to their desires. Several other properties can be added to this list. Basically it can be summarized that these visualization tools allow user to build his own crystal structure by giving unit cell parameters and shaping crystal according to his desires. They also provide a 3D visualization environment with numerous graphical alternatives. Generally crystallographic visualization tools are combined with crystallographic tools explained in the first category, in order to provide a more helpful utility. *Crystal Maker* [12] and *Crystal Builder* [13] programs are two important examples of such combinations.

Third type of works are related to pattern recognition, computer vision and 3d shape matching areas. Basically, since crystals follow some pattern, methods proposed in these areas can be used to find such patterns. Accordingly, several works done in these areas, such as several methods proposed in [15] and [14], can be used in this work.

In general there are several works partially related to this subject. However, since this work focusses on an uncommon problem, there are no directly related previous work, in our knowledge.

Chapter 3

Framework for Pattern Information Extraction

In any crystal structure, if a point is translated by any integer combinations of primitive vectors, an identical point is obtained. In order to two atoms being identical, these two atoms should belong to same atom type and their view of the crystal structure should be same. In other words, in order to two atoms A and B being identical, for every atom C in the crystal structure, there should be a corresponding atom C' with the same atom type with C, where C's relative distance to A is equal to C''s relative distance to B. Accordingly any atom A in a crystal structure, is identical to other atoms $A_{i,j,k}$, for all integer values of i,j and k, which the vector $A_{i,j,k} - A$ is equal to $V_1 \times i + V_2 \times j + V_3 \times k$ where V_1,V_2 and V_3 are primitive vectors. This observation leads to the fact that for two identical atoms A and B in any crystal structure, the vector obtained by coordinate differences of A and B will be an integer multiple of primitive vectors. Furthermore, by the definition of primitive vectors, for any atom A, there should be an identical atom $A_{i,j,k}$ for all integer values of i,j and k where $A_{i,j,k}$'s coordinate differs from A's coordinate by $V_1 \times i + V_2 \times j + V_3 \times k$. These observations are the heart of the proposed framework. Because these observations imply that if identical atoms can be grouped together, difference vectors between every pair of identical atoms in each group can be extracted. Set of these vectors

will include all integer combinations of primitive vectors. Accordingly, primitive vectors can be calculated by using these extracted vectors. Afterwards calculating the basis vectors and space group can be done. In this section, these procedures will be explained in detail.

3.1 Stages of Proposed Framework

In the first part of proposed framework, reading and indexing the atomic coordinates in input data are performed. Indexing should support retrieving points, which lie inside a given volume, efficiently. Second part of the framework is grouping the identical atoms together. Two atoms are considered identical if they belong to same atom type, and if they see the rest of the crystal structure same. With this definition, it is assumed that crystal structure is infinity big. This is not a realistic assumption. However since crystal structures are relatively quite big compared to atomic sizes, this assumption does not posses a practical problem. Grouping of identical atoms require detecting identical atoms and putting them in same group. After the grouping algorithm completed, vectors, which are the coordinate differences between every atom couple in every group, are extracted. Then, some of these vectors are eliminated in the filtering out redundant vectors phase, since they are not qualified to be a primitive vector. Afterwards, primitive vectors can be calculated. Then the user is asked to select a primitive vector triplet. Since there will be many vector triplets, which can be used as primitive vectors, asking user about his primitive vector preferences is a logical choice. In this way, the user is allowed select primitive vectors, which looks the best. Afterwards basis vectors can be calculated. However, this calculation requires the origin to be defined. In principle, any point can be used as origin and valid basis vectors can be calculated. However, users generally prefer to select some atoms position or some certain point which leads to a simple unit cell geometry, as origin. Accordingly, the user should be asked for origins position. For this purpose the clustering is done. The aim of the clustering process is grouping atoms, which can be used as a basis vector set. Afterwards, the coordinates of atoms of a cluster are shown to user so that he can select the origin. After the

origin is determined, the basis atoms can be found. At this point, the space group of the crystal structure can be identified. It is simply done by testing if every symmetry operations of each space group are supported by the crystal structure. Figure 3.1, summarizes the stages of the proposed framework.

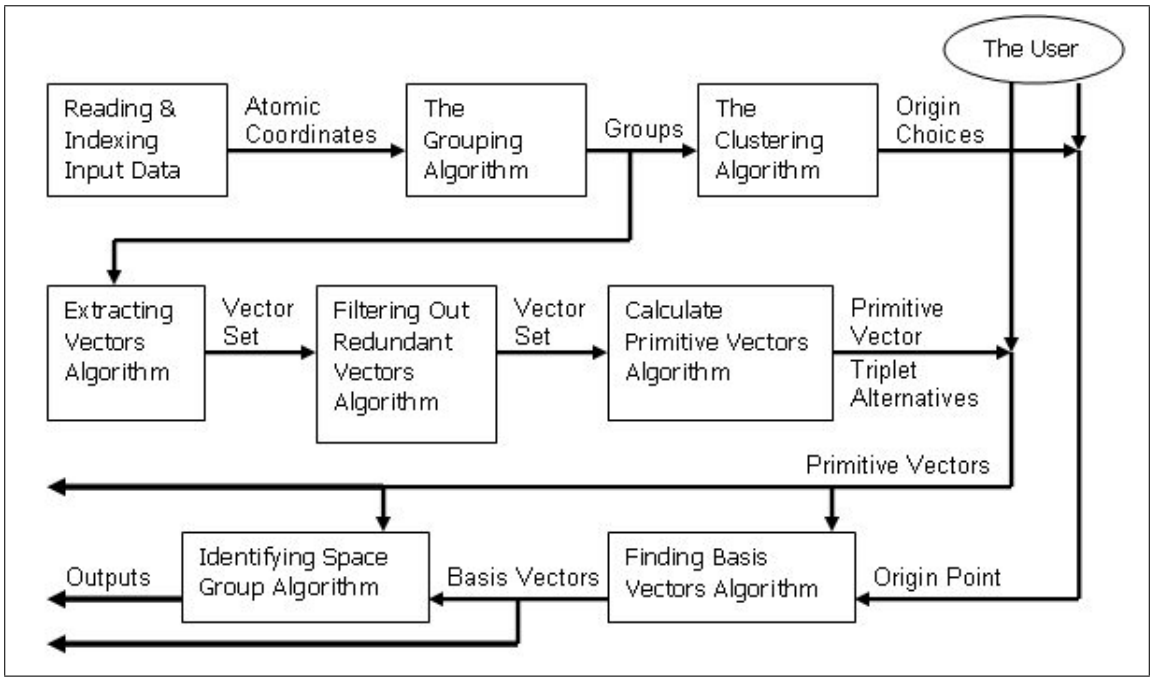

Figure 3.1: The flow diagram of the proposed framework

3.1.1 The Algorithm for Grouping Identical Atoms

Grouping identical atoms together is a quite crucial task for this analysis. For two atoms to be identical, they should belong to the same atom type and relative positioning of them to their neighbors should be the same. Theoretically A and B are identical atoms if for every atom C there exists another atom C' in the crystal structure with the same atom type of C, where C's vectoral distance to A is equal to C''s vectoral distance to B. In other words, for every atom around A, there should be a corresponding atom with same type around B with the exact relative positioning. Unless A and B are same atoms, this definition requires crystal structure to be infinitely big in order to A and B being identical.

However, practically it is sufficient to make sure there always are a corresponding atom around B for all atoms around A, which are closer to some relatively big distance.

If you translate any point in crystal structure by any integer multiples of primitive vectors, you will obtain an identical point. Let V_1,V_2 and V_3 be primitive vectors. These three vectors will define a paralleloid, which can be considered as the unit cell of crystal. If you take an atom, A, as the origin, then primitive vectors will define a volume V_A, which can be considered as the unit cell paralleloid starting from A's coordinates. Assume we are to check if A and B are identical. Then, it will be sufficient to check if the volumes V_A and V_B matches. Because, any point which does not lie inside V_A or V_B can be translated by integer combinations of primitive vectors to another point that lie inside these volumes. Accordingly, any point outside the volume has an identical corresponding point, which lie inside these volumes. Therefore, if any point outside these volumes will cause a mismatch, than it is guaranteed that some point inside the volume will also cause a mismatch. Accordingly, checking if the volumes of two atoms defined by primitive vectors matches, is sufficient to detect if these atoms are identical.

Since primitive vectors are unknown, it is not possible to determine the volumes of atoms that primitive vectors would define. However, trying to match some volumes, which include that volumes, will give correct results. In this work, cubic volumes around each atom were used. The reason for using a cubic volume is, searching all atoms in a rectangular boundary is much more efficient than searching all atoms in any random shaped volumes. In this work half of the edge length of this cube is called matching range. The boundary from minus matching range to matching range, at each axis around the atom is used as this atoms matching volume. The user is asked to determine a value for matching range parameter. User should select matching range parameter so that matching volume would be large enough to contain unit cell of crystal. Too low values may cause wrong results, while higher values increases the execution times. In general, users should make a safe guess about this parameter. For most cases, selecting a matching range value which make matching volume to cover about 10-20 atoms, will give correct results, since atoms which are not identical tend to

have significantly different positioning.

The algorithm for grouping identical atoms simply calculates matching volumes of each atom and it groups atoms with identical matching volumes. The algorithm tries to match each atom with previously found groups. If an atom matches with some group, it is included into this group. Otherwise, it forms another group. In order to calculate matching volume of an atom, it is necessary to make sure every part of the matching volume of this atom should be inside crystal segment given in the input data. Otherwise, incomplete matching volumes will be obtained and they would cause invalid mismatches. In this work, it is assumed that the input crystal structure is sufficiently big. In order to format the shape of the crystal structure and limiting the number of atoms that will be used during the analysis to a reasonable number, a parameter, cut out treshold, is introduced. While reading input coordinates, any atom, whose absolute x,y or z coordinate values exceeds cut out treshold value is ignored. Another parameter process range defines the volume whose boundaries are minus process range to process range at each axis. The atoms that lie inside this volume are actually analyzed. Process range parameter should be selected smaller than cut out treshold parameter by at least matching range. Accordingly, the volume defined by process range parameter lies inside the volume that is defined by cut out treshold parameter. Atoms which lie in the process volume, which is defined by process range parameter are guaranteed to have complete matching volumes. Both process range and cut out treshold parameters are asked to user. Too low values may not be sufficient to obtain the result, while too high values increase runtime.

Matching volume of an atom A is simply a list of all atoms, whose coordinate differences with A at each axis, are smaller than matching range value. This list contains these atoms relative coordinates to A and their atom types. Calculating the matching volume of an atom can be considered as a range search query with the corresponding boundary parameters. It is explained in data structures and indexing section in more detail. After matching volume of an atom is calculated, the list that define matching volume is sorted according to coordinate values of atoms in that list. Accordingly, while comparing two matching volume lists, linear scans of them would be sufficient.

```
GroupList=NULL;
foreach Atom A in ProcessVolume do
    MV=CalculateMatchingVolume(A);
    Sort(MV);
    foreach Group G in GroupList do
        if A.MatchingVolume matches G.MatchingVolume then
            G.Insert(A);
            break;
    if A is not matched to any group then
        G=new Group();
        G.MatchingVolume=MV;
        G.Insert(A);
```

Algorithm 1: The algorithm for grouping identical atoms

Ideally, the number of groups should be equal to the number of basis vectors. However, several reasons such as incomplete crystal segments, errors in crystal structures or errors in atomic coordinates may cause generation of more groups. Elimination of such groups is explained in error handling section. The algorithm for grouping identical atoms is given in Algorithm 1.

3.1.2 The Algorithm for Finding Primitive Vectors

If you translate any point in crystal structure by an integer combination of primitive vectors, you will obtain an identical point and vice versa. The grouping algorithm creates groups of identical atoms. Accordingly any atom inside a group can be translated to another atom in the same group, by some integer combination of primitive vectors. In other words, the vectoral distance between any two atom A and B in the same group is equal to some integer combination of primitive vectors. Accordingly, a list, which contains vectoral distances between all atom pairs in a group, can be created. This list contains all integer combinations of primitive vectors in some boundary. This list should also include all three primitive vectors, since a primitive vector itself is also an integer combination of primitive vectors. Therefore, it is possible to select any three vectors from created vector list and checking if they can produce all other vectors in the list

as their integer combinations. Accordingly, vector triplets, which can be used as primitive vectors, can be extracted from this list. Details of procedures are given in subsections.

3.1.2.1 The Algorithm for Extracting Vectors

Extracting vectors is a simple process. The algorithm simply takes each atom pair in a group, and adds the vectoral distance of these two atoms to the vectors list. If the algorithm for grouping identical atoms had been worked as expected, every group should produce non-conflicting vector lists. Since process volume cuts the crystal at some random place, it may cover some parts of some unit cells and it may leave other parts outside. Accordingly, number of atoms in each group will not be the same. Therefore, the lists produced from each group will not be identical. However, differences will be in terms of including or not including some vectors. Such vectors which appear at one groups list and which do not appear at other list will be big vectors since simpler vectors can be produced by closer pairs of identical atoms, which any selection of process volume can cover. These simpler vectors will be common for all lists. Accordingly, every list carries enough information to find the primitive vectors. So deriving the list for just one group is sufficient. However, for some cases such as presence of coordinate errors or structural errors of crystal, calculating a list for each group and merging these lists might be a better choice. Such cases will be explained in error handling section in detail.

Every atom pair in a group defines a vector. Considering there can be thousands of atoms in a group, the number of vectors that can be generated from a group can be quite large. Most of the genereted vectors would be identical to some other generated vector. Accordingly, number of distinct vectors would be much smaller. Nevertheless, the number of vectors can still be high. It is quite unlikely that a desired primitive vector set containing a long vector. Accordingly, eliminating long vectors would reduce the number of extracted vectors to a reasonable number. In this work, vectors whose length is larger than some predefined value are eliminated. The algorithm for extracting vectors is given in

Algorithm 2.

```
G=Most Crowded Group;
VLIST=NULL;
for i=0 to G.Count-2 do
    for j=i+1 to G.Count-1 do
        V=G.Atom[i] - G.Atom[j];
        if V.Length< C then VLIST+=V;
return VLIST;
```

Algorithm 2: The algorithm for extracting vectors

3.1.2.2 The Algorithm for Filtering Out Redundant Vectors

Three primitive vectors should be able to produce all other vectors as their integer combination. Consider two vectors V_1 and V_2. Moreover, let V_2 be $c \times V_1$, where c is an integer constant. Then V_2 cannot be a primitive vector unless c is equal to -1. This proposition can be proven by contradiction. Assume V_2 is a primitive vector together with P and Q vectors. If c equals to -1, $-1 \times V_2$ will produce V_1. Otherwise in order to produce V_1, P and Q should have an integer combination equal to $k \times V_1$, where k is equal to $i \times c - 1$ or $i \times c + 1$ for some integer value of i. This means that, P and Q have some linear combination that will produce V_2. Accordingly P,Q and V_2 are not orthogonal, thus cannot be a primitive vector triplet. So it is unnecessary to test a candidate primitive vector triplet which includes a vector, which is an integer multiple of another vector in the list. Accordingly removing such vectors from the vectors list will improve runtime performance. The first step of this procedure is sorting the vector list. Sorting is done according to the absolute x value first. The vectors with equal absolute x values are sorted according to their absolute y values and the vectors whose absolute x and absolute y values are equal are sorted according to their absolute z values. Such sorting is crucial for the algorithm. Because, in this way a vector V_2 which is an integer multiple of the vector V_1 is guaranteed to come later in the list. Accordingly, for any vector in the list only checking the rest of the list is sufficient. Even though complexity remains quadratic, runtime performance improves significantly. In this procedure, vectors that are -1 times of another

vector are also eliminated. The reason for this elimination is for any primitive vector triplets, you can multiply any of three primitive vectors by -1 and still obtain a valid primitive vector set. Accordingly, in this work instead leaving such vectors in the list and decreasing performance, eight different combinations of primitive vectors are calculated after primitive vectors are found. Since this approach helps to reduce the number of vectors in the list significantly, there is a significant performance improvement. The algorithm for filtering out redundant vectors is given in Algorithm 3.

```
Sort(VLIST);
V1=VLIST.FirstVector;
V2=NULL;
while V1 != NULL do
    V2=V1.NextVector;
    while V2 != NULL do
        tmp=V2.NextVector;
        if V2 is an integer multiple of V1 then
            Remove(V2);
        V2=tmp;
    V1=V1.NextVector;
return VLIST;
```

Algorithm 3: The algorithm for filtering out redundant vectors

3.1.2.3 The Algorithm for Calculating Primitive Vectors

After the redundant vectors are eliminated, a list of vectors is obtained which can form primitive vector triplet alternatives. Naive way to calculate the primitive vectors is taking every vector triples, which can be derived from the vector list and checking if every other vectors in the list can be produced in terms of integer combination of these vectors. However, this procedure has a $O(n^4)$ time complexity where n is the number of vectors in the list. Even though filtering redundant vectors reduces the list size significantly, still there will be many vectors in the list. Accordingly checking every vector triplet is not a desirable solution. Fortunately, a simplification is possible. Scientists generally prefer primitive vectors as small as possible. However, for some cases, since another three primitive vectors

present a more understandable geometric representation, users may prefer those vectors. Nevertheless, in any case desired primitive vector triplets will not contain too big vectors. Accordingly sorting the vector list according to the lengths of the vectors and limiting the set of vectors that can be in a primitive vector triplet can be an efficient solution. For this purpose, a parameter is asked to user. This parameter defines the set of vectors that will be used to derive candidate primitive vector sets. Three vectors that will form a candidate primitive vector triplet, will be selected from shortest vectors whose number is limited by the parameter taken from the user. Since setting this parameter to values around 100 is sufficient, this procedure becomes quite fast. The algorithm for calculating the primitive vector alternatives is given in Algorithm 4.

```
SortByLength(VLIST);
PVLIST=NULL;
VLEN=min(VLIST.Count,MaxNumOfPVCandidates);
for i=0 to VLEN-1 do
    for j=i+1 to VLEN-2 do
        for k=j+1 to VLEN-3 do
            isPV=true;
            for t=0 to VLIST.Count-1 do
                if VLIST[i],VLIST[j] and VLIST[k] cannot produce
                 VLIST[t] then
                    isPV=false;
                    break;
            if isPV then
                PVLIST+=new
                PrimitiveVector(VLIST[i],VLIST[j],VLIST[k]);
return PVLIST;
```

Algorithm 4: The algorithm for calculating primitive vector alternatives

In order to check if given three vectors V_1,V_2 and V_3 can produce the vector V, it is necessary to solve the following equation.

$$V = i \times V_1 + j \times V_2 + k \times V_3$$

Since all 4 vectors are 3 dimensional, given equation will result in a linear equation set of three equation with three unknowns; i,j and k. Solving such equation set is

a relatively easy operation and can be done quite fast. If integer solutions can be found for i,j and k, it is concluded that vectors V_1,V_2 and V_3 can produce vector V. If solutions are not all integers or no particular solution could be found, then it is understood that V cannot be produced by some integer combinations of given vectors, thus the vectors V_1,V_2 and V_3 can not be a primitive vector alternative.

After finding all vector triplets, which can be used as primitive vector sets, user is asked to select one or more primitive vector set alternatives. The algorithm for extracting basis vectors and identifying space group, continues according to the user's selections.

3.1.3 The Clustering Algorithm

Crystal structure is defined by primitive vectors and basis vectors. Basis vectors are atomic coordinate vectors of atoms, which lie inside the paralleloid defined by primitive vectors. Accordingly, in order to define basis vectors, determining the origin point is required. In principle, any point can be used as origin and the basis vectors, which perfectly define crystal structure together with the primitive vectors, can be calculated. However using a random point as origin is not a desired solution. Scientists usually prefer using a certain atoms coordinate as origin. For example, in Figure 2.1, one of the Cl atom's coordinate is used as origin. Accordingly Cl atoms are placed on corners of unit cell cube. If Cs atoms coordinates were used then figure would show Cs's on the corners and a Cl on the center of the cube. Another point could also be used as origin. Currently basis vectors of $CsCl$ structure are given in $\mathring{A}$ unit as;

$$B_1 = Cl, [0, 0, 0]$$

$$B_2 = Cs, [2.01, 2.01, 2.01]$$

Assume the middle point of Cs and Cl atoms were used as origin. Then basis vectors would be;

$$B_1 = Cl, [3.01, 3.01, 3.01]$$

$$B_2 = Cs, [1.00, 1.00, 1.00]$$

This definition will also be valid but unit cell structure will be harder to understand since this definition is not geometrically as powerful as first representation. Accordingly leaving origin selection to user is a better desicion.

Scientists generally select an atoms coordinates as origin. Since every atom in a group is identical, proposing one atom for each group is sufficient. However, proposing a random atom from each group is not a desired solution. Users should be able to observe relative positioning between proposed atoms. For example, in the $CsCl$ structure user should see coordinate differences between two neighbor Cl and Cs atoms are [2.01,2.01,2.01]. If the random atoms were proposed for each group, seeing these relations would not be possible. Accordingly, atoms should be clustered according to relative distances. Clustering should be done so that atoms in a cluster will be as close to each other as possible. Accordingly, the relative coordinates of atoms in a cluster can be observed easily.

The clustering is performed iteratively. First, each cluster has to have one atom from each group. Therefore, the initial step of the clustering process is, assigning each atom of the most crowded group to a different cluster. After that, remaining groups are iteratively processed. In order to process a group G, for all clusters and for all atoms in G, an atom-cluster pair is found whose atom to cluster center distance is minimum. Afterwards, a direction vector is defined by using this pair, as atoms relative coordinate according to the cluster center. After the direction vector is found, for all clusters C in the clusters list, if there is an atom A whose relative distance to C is equal to the direction vector, it is assigned to C. It is necessary to find a direction vector since the clusters should be identical. It is clear that cluster atoms should be as close as possible. However, this restriction is not sufficient. Consider the $CsCl$ structure. For each Cs atom there are 8 Cl atoms with the same minimum distance to the Cs atom. If instead of calculating the direction vector, the atom with the minimum distance were selected, any of these 8 Cl atoms could be used. Since for different clusters Cl atoms with different relative positioning can be selected, the clusters may not be identical. Accordingly, calculating direction vectors and performing assignments according to these vectors is necessary in order to obtain identical clusters. After assignments are completed, if there are clusters which no atoms

are assigned for the last group, these clusters are eliminated. The reason for this elimination is a cluster clearly has to contain one atom of each group. Otherwise, cluster will be incomplete. Around surfaces of crystals, such incomplete clusters can be seen. However, they are not suitable to be presented as origin alternatives to the user. Accordingly, these clusters are eliminated. After all clusters are obtained, they are sorted according to their center distances to origin of input data. The cluster with smallest center distance to origin is returned and shown to the user. Accordingly the simplest coordinates are shown to the user. The clustering algorithm is given in Algorithm 5.

```
ClusterList=NULL;
foreach Atom A in most crowded group do
    ClusterList+=new Cluster(A);
foreach Group G that is not processed do
    MinDist=∞ ;
    MinAtom=NULL ;
    MinCluster=NULL ;
    foreach Atom A in G do
        foreach Cluster C in ClusterList do
            D= distance between C and A ;
            if D < MinDist then
                MinDist=D;
                MinAtom=A;
                MinCluster=C;
    DV=MinAtom.Coordinates-MinCluster.Coordinates;
    foreach Cluster C in ClusterList do
        foreach Atom A in G do
            if A.Coordinates==C.Coordinates+DV then
                C.Assign(A);
                break;
    foreach Cluster C in ClusterList do
        if C.AtomCount < NumOfProcessedGroups then
            Remove(C);
Sort(ClusterList);
return First Cluster;
```

Algorithm 5: The clustering algorithm

The aim of this algorithm is simply providing user a candidate basis set to select the origin. Taking the atoms closest to the origin of input data from each group, could produce a good solution with much better runtime complexity. However, with the clustering algorithm used in this work, a geometrically more meaningful cluster will be obtained. Since this cluster will be closer to the desired basis set, seeing relations between atoms will be easier. After the coordinates of atoms in a cluster are shown to the users, they select the origin. Users can select one atom from the proposed list or they can enter coordinates of origin manually. After the origin is determined, the basis vectors can be found.

3.1.4 The Algorithm for Finding Basis Vectors

Basis vectors can be defined as the coordinates of atoms, which lie inside the unit cell paralleloid, defined by the primitive vectors and the origin. The data structure used in this work, can only answer rectangular boundary search queries. Accordingly, it is not possible to query only the atoms lying inside the unit cell paralleloid. However, the boundaries of the rectangular prism, which contains the unit cell paralleloid, can be calculated and these boundaries can be used to query the data structure. Assume x_i,y_i and z_i are x, y and z coordinates of ith primitive vector. Accordingly the minimum x value of the rectangular boundary will be the minimum of 0, x_1, x_2, x_3, $x_1 + x_2$, $x_1 + x_3$, $x_2 + x_3$, $x_1 + x_2 + x_3$ values while the maximum x value of the boundary is the maximum of given values. Minimum and maximum y and z values are found similarly. After finding the boundary values for the query, all atoms that lie inside this rectangular prism are obtained by querying the data structure. These atoms should be checked in order to see if they lie inside the unit cell paralleloid. Let V_1,V_2 and V_3 be primitive vectors. Any point P which lie inside the paralleloid defined by these primitive vectors can be expressed as $P = i \times V_1 + j \times V_2 + k \times V_3$ where i,j and k are numbers in the range of [0,1). Since P and the primitive vectors are all 3 dimensional vectors, given expression represents a 3 unknown 3 equation linear equation set. Solving that equation set for an atom and checking if i,j and k numbers are all in [0,1) range, will show if this atom lies inside the paralleloid thus if it is in the

basis set.

While performing such calculations, it is important to translate all coordinate values to the new coordinate system defined by the new origin value that user selected. An old coordinate value can be translated into new coordinate system by simply subtracting the origin's coordinate from the old coordinate values. Since it will be costly to translate every point to new coordinate system, it is better to do the translations whenever required during the procedure. The algorithm for finding basis vctors is given in Algorithm 6.

```
B=FindBoundaries(V1,V2,V3);
Translate(B,Origin);
AtomList=RangeSearch(B);
BasisList=NULL;
foreach Atom A in AtomList do
    if isInParalleloid(Translate(A,-Origin)) then
        BasisList+=A;
return BasisList;
```

Algorithm 6: The algorithm for calculating basis vectors

3.1.5 The Algorithm for Identifying Space Group

The space group of a crystal structure is determined by checking if it supports some symmetry operations. There are several symmetry operations, such as rotations, mirror operations, glide operations, etc. Crystal structure is tested to see which symmetry operations it supports. According to the set of symmetry operations it supports, it is classified into one of 230 predefined space groups. Any crystal structure should belong to one of these space groups [4]. The aim of this procedure is finding which space group that analyzed crystal structure belongs.

A symmetry operation can be considered as a 3D coordinate operation, which translates a point into an identical point. Consider a simple cubic lattice structure. In other words, consider a 3D mesh, which there is an identical atom at

every point with coordinates (i, j, k) where i,j and k are all integers. Take any point as the origin. Assume $(0, 0, 0)$ is selected as origin for simplicity. Afterwards rotate the crystal structure 90 degrees clockwise around z axis passing through the origin. This operation brings points with coordinates (a, b, c) into new coordinates $(b, -a, c)$. Since b,$-a$ and c are all integers and in the simple cubic lattice there are identical atoms at every point with integer coordinates, given symmetry operation is supported. Many symmetry operations are defined similar to the one given in the example. In general, any symmetry operation can be defined in terms of a rotation operation and a translation operation performed afterwards [11]. Accordingly, symmetry operations can be expressed by using a rotation matrix and a translation vector. Thus, applying a symmetry operation on a point can be expressed as a matrix vector multiplication and a vector addition.

For a crystal structure, in order to belonging to a space group, it should support a certain set of symmetry operations specific to this space group. A crystal structure may support all symmetry operations of more than one space groups. In those cases, the space group, which contain the highest symmetry operations is considered as the space group that crystal structure.There are 230 space groups. These space groups are ordered so that low symmetry groups have low group numbers and high symmetry groups have high group numbers. For example, cubic lattice class contains high symmetry groups. Space groups belonging to cubic lattice class supports more symmetry operations than any other space groups belonging to other classes. Accordingly, space groups 195-230 are used for cubic lattice class. On the other hand triclinic lattice class contains lowest symmetry space groups. The first and the second space groups are used for this class. Space group 1 contains only one symmetry operation, which contains an identity rotation matrix and a zero translation vector. Accordingly, every structure supports every symmetry operations of the first space group.

In order to identify the space group of a crystal structure, it should be tested to see if it supports every symmetry operation of each space group. If the crystal structure supports every symmetry operations of a space group, then it supports the space group. The space group with the highest group number, which crystal structure support, can be returned as the space group of the crystal structure.

Checking if a symmetry operation is supported by a crystal structure can be done by applying that operation on several crystal points, which covers the basis set. If the symmetry operation is supported, the translated point should be identical to the original point. There are two available coordinate systems for this procedure. The first one fractional coordinate system and the second one is Cartesian coordinate system. For each alternative, appropriate space group symmetry matrices and vectors should be used. If fractional coordinates are used, the Cartesian coordinates of each point that are to be tested, should be converted into the fractional coordinates. Afterwards symmetry operations can be applied, and the fractional coordinates of the translated point can be obtained. Then the coordinates of the translated point can be converted into Cartesian coordinates and checking if an identical atom is obtained can be done. However, if the Cartesian coordinates are to be used, then all symmetry matrices and vectors should be modified according to primitive vectors. In this work, the fractional coordinates are used. Converting a few test points coordinates into fractional coordinates is easier than modifying whole symmetry matrices and vectors. In addition, the symmetry matrices and vectors are generally given in fractional coordinates. Accordingly using fractional coordinates is easier and also more canonical way.

Testing if a crystal structure supports a symmetry operation might seem like a quite easy task. However, it has some complications. Firstly, using any primitive vectors will not work for each space group. For example, consider $NaCl$ structure. It's primitive vectors and basis vectors can be written as,

$$V_1 = [a, a, 0]$$

$$V_2 = [a, 0, a]$$

$$V_3 = [0, a, a]$$

$$B_1 = Na, [0, 0, 0]$$

$$B_2 = Cl, [0.5, 0.5, 0.5]$$

$NaCl$'s space group is given as 225 [7]. One of the symmetry operation of group 225, translates a point at *(x,y,z)* to the point *(x,y+0.5,z+0.5)*. Clearly this symmetry operation fails for Na atom at the origin, since there are no Na atoms

located at *(0,0.5,0.5)* point. Accordingly, this test show that $NaCl$ structure does not belong to 225^{th} space group. However, instead of using the primitive vectors given above, another vector set which define a cubic unit cell can be used. Then primitive vectors and basis vectors can be written as,

$$V_1 = [2a, 0, 0]$$

$$V_2 = [0, 2a, 0]$$

$$V_3 = [0, 0, 2a]$$

$$B_1 = Na, [0, 0, 0]$$

$$B_2 = Na, [0.5, 0.5, 0]$$

$$B_3 = Na, [0.5, 0, 0.5]$$

$$B_4 = Na, [0, 0.5, 0.5]$$

$$B_5 = Cl, [0.5, 0, 0]$$

$$B_6 = Cl, [0, 0.5, 0]$$

$$B_7 = Cl, [0, 0, 0.5]$$

$$B_8 = Cl, [0.5, 0.5, 0.5]$$

In this configuration, every point in the basis set support given symmetry operation. This example shows that primitive vector selection is important. In general, in order to test if the crystal structure belongs to a space group from the cubic lattice class, a vector set defining a cubic unit cell should be used. Besides, any vector set which result in some cubic unit cell cannot be used. Primitive vectors should define the minimal cubic unit cell. For example, consider the vectors

$$V_1 = [4a, 0, 0]$$

$$V_2 = [0, 4a, 0]$$

$$V_3 = [0, 0, 4a]$$

for $NaCl$ structure. This unit cell can be considered as a combination of eight previously defined unit cells putted together to form a bigger cube. The problem with this unit cell is that it supports every symmetry operation in 229^{th} space

group. 229^{th} space group contain 96 symmetry operations. 48 of them have *[0 0 0]* translation vectors. These 48 operations can also be found in 225^{th} group, so they are supported by $NaCl$ structure. Other 48 operations of 229^{th} group are, exactly same set of operations except they have *[0.5,0.5,0.5]* translation vectors, instead of *[0,0,0]* vectors. The problem is, *[0.5,0.5,0.5]* translations done with the vectors in multi unit cell is equivalent to the *[1,1,1]* translation done with the vectors which define minimal cubic unit cell. Accordingly, these 48 symmetry operations are trivially supported with the vectors defining the multi unit cell. Then, space group is found as 229 since it has higher group number. So, in order to test a cubic lattice class, a vector set that defines minimal cubic unit cell should be used. Similarly, in order to test other lattice classes, vector sets defining minimal unit cells of those classes should be used. There are seven lattice classes [2, 9, 18, 4]. Accordingly, seven different sets of vectors should be derived and used. In order to derive such vector sets, primitive vectors should be used. Each integer combination of primitive vectors defines a valid vector. Three orthogonal vectors define a valid unit cell. Accordingly, several choices of integer combinations of primitive vectors are used to define a set of valid vectors. Afterwards each combination of three vectors from the set of derived valid vectors is checked in order to see if these vectors define a unit cell belonging to one of these seven classes. If the answer is yes, this unit cell is recorded and these vectors are used in space groups tests belonging to this lattice class. In order to improve performance and guarantee to obtain the minimal unit cell, vector triplets that are to be checked are sorted before performing the checks. The algorithm is designed so that, the vector triplets with smaller vectors, are tested before the vector triplets with larger vectors. Accordingly, the minimal unit cells belonging to each lattice class are obtained before other unit cells belonging to same class. Therefore, once a unit cell is found for a class, other unit cells belonging to same class are discarded. The procedures used to identify the space group number are given in Algorithm 7 and 8.

Valid vectors are generated as integer combinations of primitive vectors. It is clear that infinitely many vectors can be generated by this approach. To limit the number of valid vectors to a reasonable number, only vectors generated by

```
ValidVects=NULL;
for i=-√K to √K do
    for j=i to √K do
        for k=j to √K do
            if i² + j² + k² < K then
                ValidVects+=i × V1 + j × V2 + k × V3;
Sort(ValidVects);
return ValidVects;
```

Algorithm 7: The algorithm for deriving valid vectors

using the integers i,j and k which satisfy inequality $i^2 + j^2 + k^2 < K$ where K is a predefined constant, as coefficients to primitive vectors are accepted as valid vectors. After valid vectors are defined, set of valid vectors are sorted according to their lengths. Accordingly, smaller vectors comes earlier.

The procedure starts with deriving the set of valid vectors. Then an integer array containing the index numbers of three vectors that will define a vector triplet is generated. Afterwards this array is sorted, so that vector triplets with small vector indices, thus small vector lengths, comes earlier in the list. Then according to this list, every vector triplet is checked to see if it defines a unit cell, belonging to one of seven lattice classes. If a vector triplet matches to a class, which no previous match has been found, then it is recorded.

In theory, for any lattice class, a unit cell can be generated from any primitive vector set. However, such unit cells are generally quite big unit cells. It is quite unlikely that crystal structure belongs to a space group of such classes with quite big unit cells. In this work, if a relatively small unit cell cannot be generated for a lattice class, then no space group belonging to this class are tested. Defining relatively small unit cell is done by limiting the integer multipliers of primitive vectors to derive the valid vectors. By this way, the number of valid vectors is also limited to a reasonable number, thus processing time is not effected badly.

After determining vector sets for each class, the test points should be gathered. In principle, set of test points should cover at least one identical point to each

```
ValidVects=DeriveValidVects();
PVCNT=0;
for i=0 to ValidVects.Count-1 do
    for j=i to ValidVects.Count-1 do
        for k=j to ValidVects.Count-1 do
            PVCoefficients[PVCNT][0]=i;
            PVCoefficients[PVCNT][1]=j;
            PVCoefficients[PVCNT][2]=k;
            PVCNT++;
Sort(PVCoefficients);
for i=1 to 7 do Classes[i]=NULL;
for i=0 to PVCNT do
    V1=ValidVects[PVCoefficients[PVCNT][0]];
    V2=ValidVects[PVCoefficients[PVCNT][1]];
    V3=ValidVects[PVCoefficients[PVCNT][2]];
    if V1,V2 and V3 are not orthogonal then continue;
    ClsId=ClassOf(V1,V2 and V3);
    if Classes[ClsId]!=NULL then
        continue;
    else
        Classes[ClsId]=SetOf(V1,V2,V3);
return Classes;
```

Algorithm 8: The algorithm for deriving unit cells of lattice classes

point in the basis set. However, using all atoms within some volume, which can contain a unit cell, will also work. In this work, a cubic volume around the origin, which is big enough to cover any possible unit cell, is determined and all atoms lying in this volume are used as test points. The rest of the procedure is simply testing every symmetry operation of space groups with every test points. Space groups are tested starting from the space group with highest group number. When crystal structure supports all symmetry operations of a space group, this space group is returned. The algorithm for identifying the space group is given in Algorithm 9.

The algorithm for identifying the space group starts with loading the space group data. Space group data consist of a small information and symmetry operations of each space group. Then primitive vectors defining unit cells of each

```
SpaceGroups=loadSpaceGroupData();
Classes=derivePVofClasses();
volume=determineBoundaries();
TestPoints=returnAllAtoms(volume);
for i=230 to 1 do
    S=SpaceGroups[i];
    UC=Classes[ClassOf(S)];
    if UC==NULL then continue;
    isSupported=1;
    foreach Symmetry operation M,V of S do
        foreach Point P in TestPoints do
            C=getFractionalCoordinatesOf(P,UC);
            Q=M×C+V;
            C=getCartesianCoordinatesOf(Q,UC);
            if There is no atom of type P at C then
                isSupported=0;
                break;
        if !isSupported then
            break;
    if IsSupported then
        return S;
```

Algorithm 9: The algorithm for identifying the space group

lattice class are generated as explained in the algorithm for deriving unit cells of lattice classes. After that, the volume containing the test points is defined. After the volume is determined, all atoms within the volume are used as test points.

Then for each space group starting from the one with the highest group number, test is performed. In order to perform the test, appropriate unit cell parameters for currently tested space group are determined. Afterwards for each symmetry operation, every test point is tested. Testing a point is simply done by applying the operation on the test point and checking if an atom with the same type exists in the coordinates of the translated point. Whenever all symmetry operations of a space group are supported by all test points, this space group is returned as the space group that crystal structure belongs.

3.2 Data Structures and Indexing

Data structures and indexing methods are quite important to solve this problem efficiently. Input data is queried several times throughout the analysis. Accordingly, efficiency of data structure effects both complexity and runtime performance significantly.

In the grouping algorithm, matching volumes of each atom is compared with matching volumes of previously found groups. Accordingly matching volumes of each atom should be found. Vector algorithms and the clustering algorithm do not use input data. The algorithm for finding basis vectors require finding all atoms lying inside the paralleloid defined by primitive vectors and the origin. The algorithm for identifying the space group performs several point search queries. Accordingly, data structure should answer queries that ask all atoms lying inside a sphere or a paralleloid and queries searching the atom at a given point. Basically, matching volume of an atom should contain all atoms that are closer than some certain distance to query atom. This definition defines a sphere whose origin is the center of the query atom. Fortunately, for the grouping algorithm, a volume, which contain defined spherical boundary, will also work. Using a bigger matching volume will reduce the performance, since there will be more data to compare. However, using cubic matching volumes instead of spherical ones become possible. There are many efficient data structures that can answer rectangular boundary search queries. However, the data structures that can answer spherical queries are not that efficient. Some methods that answer spherical queries use the data structures that answer rectangular range queries to index and query the data. While querying the data they query the bounding cube of the query volume and filters out undesired points afterwards. Some other methods use complex indexing techniques, which reduce the asymptotic complexity, but due to the complexity of the data structure, runtime performance will not be as improved. In addition, those methods will not be compatible with the queries required in the algorithm for finding basis vectors.

Querying random paralleloid volumes is not an easy task. However, for the analysis performed in this work, it has low importance. Since the basis vectors

are calculated once for each primitive vector set alternatives selected by the user, random paralleloid queries will be called a few times. Accordingly, using rectangular boundaries of bounding volume of this paralleloid will work sufficiently. Considering for most of the crystal structures angles between primitive vectors are in between 60 and 120 degrees, bounding volume will not be much bigger than the volume of the paralleloid. Accordingly using rectangular query volumes will be efficient enough.

To store and index input data there are several data structure alternatives. In this work, octree structure is used. In this work, the crystal structure given in the input data is assumed to be in cubic shape. Accordingly indexing the volume with the octree structure will be efficient. Since the crystal structures are homogeneous, the number of atoms per volume will not differ significantly at different parts of crystal. Therefore, the octree structure will be balanced by nature. Accordingly, the octree structure is quite suitable for indexing the input data.

Octree is a tree structure with eight children. Each node of the octree structure is associated with some cubic volume. Each child of an octree node is associated with one eight of its parents volume, formed by halving parents volume at each axis. Internal octree nodes contain child pointers. They do not contain actual records. Actual records are stored in leaf nodes. The number of records stored in a leaf node depends on the implementation. In this work, leaf nodes store only one record. The reason for that decision is, range queries that will be used in this work, will query relatively small volumes and several point searches will be done. If leaf nodes store more records, then since linear scan of each leaf node that intersect with the query volume would be required, lots of linear scans would be necessary compared to actual output size. Particularly, in the algorithm for identifying space group, lots of point searches will be required. Accordingly, one record per node approach will give better performance.

The octree structure without records is a simple root without any children. Accordingly, it is a leaf node. Records are iteratively inserted into the structure. If a record is inserted into an internal node, the corresponding child is found

and the insertion is recursively redirected to this child. If a record is inserted into an empty leaf node, it simply becomes that leaf node's data. If the leaf node is full, then it becomes an internal node and empty children leaf nodes are allocated. Then the data that this leaf node was carrying and the record that is to be inserted can be inserted on this internal node. Data insertion procedure for octree structures is given in Algorithm 10.

```
Insert(P,N)
if N is Leaf then
    if N is not empty then
        OldData=N.Data;
        ConvertIntoInternalNode(N);
        Insert(OldData,N);
        Insert(P,N);
    else
        N.Data=P;
else
    foreach Child C of N do
        if P lies in volume of C then
            Insert(P,C);
            break;
```

Algorithm 10: The algorithm for insertion into the octree structure

Converting a leaf node into an internal node can be done in several ways. The first way is reallocating the node so that it can be big enough to be an internal node. After that children leaf nodes are allocated. Another way is allocating a new internal node without allocating the first child. Then setting the leaf node to convert, as the first child and replacing the positions of newly allocated internal node and the old leaf node. This alternative will have the same effect with the first alternative with fewer allocations. The third way is defining node structure big enough to be an internal node or a leaf node. Thus, whenever a convertion is required, just allocating children nodes and setting required parameters would be sufficient. There can be several other alternatives but the differences will not be so major. In this work, the third approach is used because of its simplicity. The conversion procedure of a leaf node into an internal node is given in Algorithm

11.

```
ConvertIntoInternalNode(N)
N.Data=NULL;
for i=0 to i<8 do
    C=Allocate Child_i;
    C.Data=NULL;
    C.Children=NULL;
    if i%2==0 then
        C.xmin=N.xmin;
        C.xmax=(N.xmax+N.xmin)/2;
    else
        C.xmax=N.xmax;
        C.xmin=(N.xmax+N.xmin)/2;
    if i%4<2 then
        C.ymin=N.ymin;
        C.ymax=(N.ymax+N.ymin)/2;
    else
        C.ymax=N.ymax;
        C.ymin=(N.ymax+N.ymin)/2;
    if i%8<4 then
        C.zmin=N.zmin;
        C.zmax=(N.zmax+N.zmin)/2;
    else
        C.zmax=N.zmax;
        C.zmin=(N.zmax+N.zmin)/2;
```

Algorithm 11: The algorithm for converting a leaf node into an internal node in the octree structure

Allocating a node large enough to be a leaf node or an internal node, is not the best approach in terms of the runtime performance of octree creation procedure or the space requirement. However, better approaches will not improve the time and space requirements significantly and they introduce undesired code complexity. Mainly, the critical part of the data structures performance is the query times. Octree initialization and data insertion parts can be done quite fast and their performances are mostly limited by IO operations. In terms of the query times, all insertion methods are identical. Because, all operations used in the queries, are in-memory operations. Accordingly, the search technique just follows the

links. As long as the octree structure is preserved, query times will be identical. When a query is executed, the pointers in the records are set, so that the output of the query will form a linked list. Accordingly, while returning a query result, only a pointer to that linked list is returned. Since while answering a query no output is copied, the performance improves.

Range search queries can be efficiently answered by the octree structure. The search procedure finds all the points that lie inside the query boundary. Then it forms a linked list from these points and returns this linked list. The procedure that performs range search queries on the octree structure is given in Algorithm 12.

```
NodeSearch(Boundary,Node,LinkedListTail)
Result=NULL;
if Node is Leaf then
    if Node.Data==NULL then Return LinkedListTail;
    if Node.Data is in Boundary  then
        Node.Data.Next=NULL;
        LinkedListTail.Next=Node.Data;
        Return Node.Data;
    else
        Return LinkedListTail;
else
    Result=LinkedListTail;
    foreach Child C of Node do
        if Volume of C intersects with Boundary then
            Result=Search(Boundary,C,Result);
Return Result;


RangeSearch(Boundary)
Head=new Atom;
Head.Next=NULL;
Search(Boundary,Root,Head);
Return Head.Next;
```

Algorithm 12: The boundary search algorithm performed in the octree structure

The end of the linked list is given to each recursive function. If this function

finds some records that satisfies the query boundary, it appends these records to the end of the linked list obtained from the caller and returns the new end of the linked list. The range search algorithm performed on the octree structure runs the node search procedure, which recursively searches the structure.

There are other data structures used in this framework. In the grouping algorithm, created groups are stored in the arrays. Arrays are sufficient since they support efficient sorting algorithms such as quick sort and they allow linear scan of the records. Similarly, extracted vectors, found primitive vector sets, created clusters are all stored in arrays. Since only sorting and sequential scans are required on these data, using arrays is a quite efficient way. However, atom records should be stored differently. As explained earlier atom data's are stored in octree structure. However, it is also required to obtain all atoms belonging to some group, or some cluster. In order to be able to obtain such atoms efficiently, free next pointers are added to the atom record. One pointer is used to link atoms in the same group, one pointer is used for atoms in the same cluster and one pointer is used in linking octree query output. Accordingly, all atoms in a cluster or in a group connected with a linked list structure. A forth next pointer is also required to link all atoms which are in the process volume. Four pointers per atom record increases space requirement significantly, but it still is in the reasonable limits. However, an improvement is possible. At any time during the pattern extraction process, at most three pointers are actively used at the same time. Accordingly storing three pointers and sharing these three pointers, is possible. This approach also does not require any pointer swap operations. While the clustering algorithm is running no range search queries are required. After the clustering algorithm completed and the origin candidates are shown to user, there is no need to store cluster information. Accordingly the clustering algorithm can use the pointer that is used in the range search queries, without a problem.

Another data structure used in algorithms, is used to index matching volumes of groups during the grouping algorithm. For each matching volume, a distinct octree structure is created. In order to, not disturbing the main octree structure used in the range search queries, the records of such matching volumes

are duplicated. Accordingly, for each group there will be an independent octree structure of their matching volumes. The linked list structure returned from the range search is used as the atom's matching volume. It is not indexed since this operation requires duplication of records, which is quite costly. Since the number of groups is much smaller than the number of atoms, such approach minimizes the total complexity, while avoiding too much overhead.

3.3 Error Handling

Error handling is a quite important issue in this problem. Due to the sensitivities of the devices or the imperfections in crystal structures, input data can contain errors. Generally, scientists are able to generate ideal input data. However, due to several reasons they might not be able to generate ideal data. Also for some cases, scientists may prefer to introduce some errors in order to give some flexibility to the input. For example, the atomic radiuses cannot be known for certain [2]. An atoms radius changes small amounts in different materials. For example, Cl^- ion's radius is not the same in the $NaCl$ structure and in the KCl structure. Accordingly, several estimations are done in order to be able to give some approximate radius values of atoms. There are several measurement standards and thus several atom-radius tables, such as CPK's, ionic, covalent and Van-Der-Walls radiuses [5], S&P [16] and VFI radiuses [1], etc. Generally, scientists can obtain approximate radius values from the appropriate table. However, obtained values will have a small error margin. Accordingly, scientists may prefer to introduce some errors to input data, in order to cover atomic radius errors. These errors are mostly small position differences of atoms.

For some cases, another type of error, missing atoms, can be introduced. Missing atoms can be described as, the absence of an atom in a certain place in the crystal structure where it should exist in the perfect crystal [18]. These imperfections are seen quite frequently in the crystal structures, which are formed quite rapidly. Scientists might wonder if a crystal structure is in some certain form they expected and contain many imperfections, or it has another form.

For those cases, the analysis proposed in this work help scientists to reveal the actual pattern in the crystal data, thus reveal crystals form. Users can introduce this kind of errors to the input data for this kind of analysis. All these errors, causes the algorithms proposed in this framework to fail, if errors are not handled differently.

The most common errors are small atomic coordinate errors in the input data. Since the measurement devices have limited sensitivity and atomic radiuses cannot be known for certain, these types of errors can be seen quite often. It is logical to expect some small coordinate error in every coordinate values. The parameter, *EPS*, is used for this purpose. *EPS* represents the amount of maximum coordinate errors in each axis that can be seen in the input data. In other words, if an atoms coordinates are given as *[x,y,z]* in the input data, then actual coordinates can be; *[x± EPS,y± EPS,z± EPS]*. The value of *EPS* parameter depends on the introduced error range on atomic coordinates. Ideally, *EPS* should be 0. On the other hand, a value that is significantly smaller than the radius of the smallest atom in the input data, will also work accurately. Setting *EPS* parameter to high values may reduce the accuracy. It is recommended to set *EPS* to the maximum value of error margin in the atomic coordinates. It is also recommended to use better data if the coordinate errors are higher than %20 of the radius of the smallest atom.

Second type of error is the missing atoms in the crystal structure. Although this type of errors effect many algorithms discussed previously, they mostly affect the grouping algorithm. The following parts explain how the errors are handled for each algorithm.

1. **The Grouping Algorithm:** The grouping algorithm is the algorithm that is affected most from the errors. The grouping algorithm groups identical atoms together by comparing matching volumes of the atoms. Any error in the input data will cause two atoms that should belong to same group seeing crystal structure differently, thus not being identical. Accordingly the algorithm should be modified so that it will consider two matching volumes identical if the differences between these matching volumes can be caused

by errors. In the original algorithm, comparing two matching volumes is done by linear scans of two lists, which contain sorted relative coordinates of atoms lying in each matching volumes. Since the relative coordinates are assumed to be perfectly accurate in the original algorithm, two lists of matching volume's should be identical in terms of points in the lists and the order of these points, in order to being matched. However, since atom coordinates can contain errors, this method will not work. An atoms coordinates are given with an error margin $\pm EPS$ at each axis. Accordingly, relative coordinates of two atoms will be obtained with an error margin of $\pm 2EPS$. This error margin can easily change the order in the sorted matching volume list. So comparing two sorted lists with linear scans will not work. The naive way two compare two matching volume's would be, trying to find a corresponding point for every point in one of the matching volumes in the another matching volume. However, this quadratic time approach can be improved easily. For this purpose, indexes on the matching volumes are used.

The most important type of error for the grouping algorithm is missing atoms. Missing atoms in the crystal structures affects the grouping algorithm deeply. Since matching volume of an atom is a significantly big volume which contain several atoms, any atom is also placed in several other atoms matching volumes'. Accordingly, if any atom is missing, no atom which includes this missing atom in its matching volume will be able to match with its actual group. Accordingly, even a few number of missing atoms will cause the grouping algorithm to fail if no modifications are done. Another parameter is used to handle this type of errors. This parameter show how many atom mismatches are allowed while comparing two matching volumes. In other words, if an atom is missing, it will also be missing in some atoms matching volume. So while trying to find this atoms group, it will cause mismatches between this atoms matching volume and the groups matching volume, which this atom should belong to. So this parameter shows how many such mismatches are allowed to still consider compared matching volumes identical. In general, in the crystal structures, probability of an atom's being missing is very low. However, since there

are many atoms in a crystal, several missing atoms can be seen. Allowing a small number of mismatches, corrects most errors in the grouping algorithm. Probability of a mismatch occurring in a matching volume is a relatively small probability. However, occurring more than one mismatches in the same matching volume is a much smaller probability. So by setting this parameter to some small number, most of the error cases can be corrected without reducing the accuracy. With small modifications in the matching procedure and indexing method for the matching volume data, it is possible to handle most cases that cause errors in the grouping algorithm. Accordingly, obtaining sufficient information to continue pattern extraction process will be possible.

The number of groups will be much smaller than the number of atoms processed in the grouping algorithm. Accordingly, indexing matching volumes of the groups, and trying to find a corresponding point in the group's matching volume by using this index for every point in atom's matching volume, will be a good choice. Creating and maintaining the index on each group's matching volume will be easy and it will not bring too much overhead, since the number of groups would be low. Appropriate index structure would be the one that can answer range search queries with small ranges efficiently. Octree structure is quite suitable for this purpose.

The grouping algorithm, initially tries to find a group for the atom that it is processing. If it cannot find a corresponding group, this atom defines a new group. Accordingly in the original algorithm first atoms matching volume of each group is used as the matching volume of the group. If there were no errors, this approach works perfectly. However, first atoms matching volume can contain errors. Accordingly, this approach should be corrected. Atomic coordinate errors cause small errors in relative coordinates of the matching volumes. However, errors of this type are not quite important. In order to say two relative coordinates matches, the difference between their coordinate values should be smaller than $4EPS$ at each axis. Otherwise, it is certain that these relative coordinates do not match. As long as EPS value is small enough, coordinate errors will not cause any problem. Mainly, the reason to recommend using input data with error margin smaller than

%20 of the smallest atoms radius is, preventing such invalid matches. When an atoms group is found, it is possible to interpolate the groups matching volume and the atoms matching volume in order to obtain more accurate relative coordinates. However, it is not possible to reduce *4EPS* error margin even further. Accordingly, this interpolation will not have a significant effect. Instead, it brings lots of computational overhead. Thus, the interpolation of the matching volumes is not used in this work. Important problem comes from the missing atoms. If the matching volume of the first atom contains a missing atom, than this will cause a mismatch with any atom's matching volume. Allowing a small number of mismatches corrects most cases, but if the atom's matching volume also contains missing atoms, then this atom may not match to such group even though it should. A small modification corrects this problem. While testing if an atom matches with a group, if for a point P in the atom's matching volume, no corresponding point in the groups matching volume could be found, but atom matches to this group anyway, then P is inserted into the groups matching volume. Since atom's matching volume cannot contain extra atoms which should not exist, groups matching volume should have a missing atom corresponding to P. By inserting P, this missing can be fulfilled.

Such modifications cover most of the cases that cause errors in the grouping algorithm. But still there can be errors. These errors cause some atoms not matching to the groups that they should match. Accordingly, these errors cause creation of unwanted groups. Since those error cases are rare, such unwanted groups contain a small number of atoms. Accordingly, by simply filtering out such groups, only desired ones can be obtained. For this purpose, a parameter, which defines the minimum number of atoms in the group, is used. If the number of atoms belonging to a group is smaller than this value, then this group is eliminated. In principle, the value of this parameter should be determined according to the input size and the expected number of groups to be created. For example, in $NaCl$ structure, there should be two groups. Actual groups contain about half of the atoms given in the input data. A reasonable input contains thousands of atoms. Unwanted groups mostly contain a few atoms. Therefore, using a value

about 20 should be ideal. However, if the input quality is poor, or the input crystal structure is not complete, there will be many groups with respectively high number of atoms. For those cases increasing the value of this parameter might work. The modified version of the grouping algorithm is given in Algorithm 13.

```
GroupList=NULL;
foreach Atom A in ProcessVolume do
    MV=CalculateMatchingVolume(A);
    foreach Group G in GroupList do
        MismatchCount=0;
        NoMatchedAtoms=NULL;
        foreach Atom B in MV do
            Boundary=B.Coordinates±4× EPS;
            R=RangeSearch(G.MatchingVolume,Boundary);
            if R==NULL then
                MismatchCount++;
                NoMatchedAtoms+=B;
            if MismatchCount > AllowedMatchCountDifference then
            break;
        if A.MatchingVolume matches G.MatchingVolume then
            G.Insert(A);
            foreach Atom B in NoMatchedAtoms do
                G.MatchingVolume.Insert(B);
            break;
    if A not matches to any group then
        G=new Group();
        GMV=Duplicate(MV);
        G.MatchingVolume=Index(GMV);
        G.Insert(A);
```

Algorithm 13: The algorithm for grouping identical atoms with the error handling mechanism

2. **Vector Operations:** Vector operations do not affected from errors as much as the grouping algorithm. No major modifications will be required to handle erroneous cases. Only changes should be done to handle possible differences that could be seen on vectors' values due to coordinate errors. Basically, a vector represents the relative coordinate differences between

two identical atoms. Accordingly a vectors coordinates will have an error margin of $\pm 2EPS$. This margin should be considered in the algorithm for filtering out redundant vectors and the algorithm for calculating primitive vectors. Just considering these error margins will be sufficient to handle error cases in vector operations. However, some improvements are possible.

In the vector extraction phase, several vectors are extracted. Most of these extracted vectors are duplicates. Consider $CsCl$ structure shown in Figure 2.1. In this structure, every Cl atoms are identical. Accordingly the relative coordinate differences between any two Cl atoms will be a vector. Consider we put crystal structure in our coordinate system, where Cs atoms are placed on the positions *(x,y,z)* where absolute values of x,y and z are all even, and Cl atoms are placed on the positions *(x',y',z')* where absolute values of x',y' and z' are all odd. In this coordinate system, any two atoms A and B with the coordinates *(x,y,z)* and *(x+i,y+j,z+k)* where i,j and k are all even, are identical and produces identical vectors. It is obvious that there will be lots of atom couples for any i,j and k values. If no errors existed, all these identical vectors would be equal. But since there is an error margin, two vectors that should be identical can differ as much as $4EPS$ at each axis. Taking the average values of the identical vectors will reduce the expected value of errors in extracted vectors. Obtained average vector values will still have an error margin of $\pm 2EPS$. It is not possible to reduce this value, since it is assumed that the error value is a random parameter. However, by taking averages, expected error value will be reduced significantly.

The missing atoms may cause some vectors, not being extracted. This is an unlikely case since each vector is duplicated many times. Only big vectors are not duplicated that frequently, but those vectors are not quite useful too. Accordingly, missing atoms will not cause a significant problem. On the other hand, an improvement is also possible. In the original algorithm, only most crowded group is used to extract vectors. Since there were no missing atoms or no coordinate errors, this approach works perfectly. However, extracting vectors from all groups will improve the accuracy for the cases where error is present. With this approach, most of the missing atom problems are corrected. Besides, more importantly, since more atom pairs

would produce the same vector, expected error range on each vector's coordinates will be reduced.

Small modifications are required to handle error cases in the algorithm for extracting vectors. For each extracted vector, a count parameter is also stored. This parameter show how many times this vector was extracted before. It is used to take averages. Another modification is using every group instead of only one as in the original algorithm. The algorithm for extracting vectors in the presence of errors is presented in Algorithm 14.

```
VLIST=NULL;
foreach Group G do
    for i=0 to G.Count-2 do
        for j=i+1 to G.Count-1 do
            V = G.Atom[i] - G.Atom[j];
            if V.Length ≥ C then continue;
            foreach Vector Y in VLIST do
                if V and Y are closer than 4EPS at each axis then
                    Y.Coordinates=(Y.Count×Y.Coordinates+V.Coordinates)/(Y.Count+1);
                    Y.Count++;
                    break;
            if V didn't matched to any previous Vector then
                V.Count=1;
                VLIST+=V;
return VLIST;
```

Algorithm 14: The algorithm for extracting vectors with error handling mechanism

Filtering out redundant vectors phase is quite similar to original one. Only difference is while deciding if a vector is integer multiple of other one, the differences that can be caused by errors are taken into consideration. Since the error ranges are known, testing if a vector is an integer multiple of another vector is a quite easy task. Accordingly, a small modification to handle this test is sufficient.

While calculating primitive vectors, procedure is a bit more complicated than filtering out redundant vectors phase. As explained earlier aim of this

procedure is testing primitive vector candidates consisting of three vectors, in order to see if they can produce all other vectors as their integer combinations. This operation is done by trying to find the integer values of i,j and k for the equation $V = i \times V_1 + j \times V_2 + k \times V_3$. Since each vector will have an error margin of $\pm 2EPS$, finding integer solutions might not be possible. However, since error margins will be much smaller than the sizes of the vectors, i,j and k values should be close to integer values if the primitive vector candidates can produce V. Accordingly, it is possible to solve this equation and eliminate primitive vector candidates which results i,j and k values which are not close enough to integer values. This elimination can eliminate most of the primitive vector set alternatives that should be eliminated. Afterwards another test is used to make the final decision. The equation given above can be rewritten as,

$$V \pm 2EPS = i \times (V_1 \pm 2EPS) + j \times (V_2 \pm 2EPS) + k \times (V_3 \pm 2EPS)$$

Accordingly, it can be written as

$$V \pm (2 \times EPS \times (i + j + k + 1)) = i \times V_1 + j \times V_2 + k \times V_3$$

Since the calculated i,j and k values cannot be too distant to their actual integer values, converting these values to the closest integer values is logical. Then these integer i,j and k values and the vector parameters can be used to test if the given equation holds. If the answer is yes, it is concluded that V_1,V_2 and V_3 can produce V as their integer combination. These modifications will be sufficient to handle errors in the calculating primitive vectors phase.

3. **The Clustering Algorithm:** The clustering algorithm does not affected from errors significantly. Only difference is, considering an atom could be found within some range rather than on a single point. This difference is handled similar to previous algorithms. Other than this difference, clustering works perfectly in erroneous cases.

4. **The Algorithm for Finding Basis Vectors:** Finding the basis vectors phase require major modification to handle erroneous cases. Given procedure for ideal input data cannot handle error cases. Basic assumption made

in the original algorithm was all basis vectors should be placed inside the unit cell paralleloid defined by the primitive vectors. However, since atomic coordinates can contain errors, a point that should be inside the paralleloid can be placed outside and a point that should be outside of the paralleloid can be placed inside. This is a quite common situation, since generally the origin and the primitive vectors are selected in a way to place the atoms at the corners, edges or faces of the unit cells. For example, for the $CsCl$ structure shown in Figure 2.1, Cl's coordinates are selected as origin. Accordingly a Cl atom is placed at each corner of the unit cell paralleloid. Only the one placed at the origin should be in basis list. With ideal data, simply returning every atom whose fractional coordinates are in $[0, 1)$ range at each axis, works perfectly. However, this distinction cannot work on the data containing coordinate errors.

Modifications should base on the fact that, equal number of atoms belonging to each group should be placed in every unit cell. For this purpose, the clustering algorithm can be used. Basically, the aim of clustering the data is providing user a candidate basis list. Accordingly, with some modifications a procedure that will return desired basis vectors can be obtained. Basic modification is done on the direction vector calculation part. In the original clustering algorithm, atom-cluster pairs with the smallest distances are found and used to calculate the direction vector. However, in finding basis vectors algorithm, direction parameters comes from the unit cell structure. In other words, direction vector should be selected so that every atom belonging to same cluster should belong to the same unit cell paralleloid. Another modification is using the origin of the unit cell as the cluster center, instead of using average coordinates of the atoms as in the original clustering algorithm. Rest of the procedure remains the same, except a few additions. In the original clustering algorithm, after the clusters are calculated, the atoms belonging to the cluster that is closest to origin, were returned. However, for finding basis vectors algorithm, this approach may not work. A cluster contains one atom per each group. On the other hand, user might use a vector set which is not primitive. For example, in face centered cubic crystal structure, primitive vectors are $V_1 = [0, a, a]$,$V_2 = [a, 0, a]$

and $V_3 = [a, a, 0]$. And this structure contains one basis atom placed at origin. But due to geometric simplicity users often prefer the vector set $V_1 = [2a, 0, 0]$,$V_2 = [0, 2a, 0]$ and $V_3 = [0, 0, 2a]$, with 4 basis atoms. Accordingly, user is allowed to enter such vector triplets manually in order to be used as primitive vector set. In order to handle such cases, some additions to original clustering algorithm are required. Firstly, user is expected to enter valid vectors. In other words, each vectors that the user enters should be some integer combination of the primitive vectors. Moreover, three vectors that the user enters should be orthogonal in order to define a valid volume. If the user enters valid vectors, then it is guaranteed that unit cell contains either the whole cluster or no part of that cluster. In other words, a unit cell will contain equal number of atoms belonging to each group. Since the clustering procedure used in the algorithm for finding basis vectors, determines the shape of each cluster according to the volume defined by given vectors, a unit cell can be filled completely with clusters and no part of those clusters left outside. After clusters are derived, for each cluster, average coordinates of atoms belonging to this cluster are assigned as center value. Then atoms of every cluster whose center lies inside the unit cell paralleloid defined by given vectors, are returned. Since if a cluster should be inside a unit cell, then the average coordinates of the atoms should also be inside the unit cell. Instead of using average coordinates, previously set center value or any atoms coordinates could also be used. However, using the average coordinates is safer in the presence of errors. Because, using an atoms coordinates can be risky since coordinate errors can put this atom outside of paralleloid incorrectly. The algorithm for finding basis vectors with error handling mechanism is given in Algorithm 15.

5. **The Algorithm for Identifying the Space Group:** Space groups are defined in order to present symmetry properties of crystal structures. A small difference in the primitive vectors or basis vectors can change the symmetry properties, thus space group significantly. For example, $NaCl$ structure and TlF structures are quite close structures. $NaCl$ has a cubic unit cell and its space group number is 225. TlF's structure can be considered as slightly distorted $NaCl$ structure. It's unit cell is not cubic. Axis

lengths differs a small amount, making unit cell orthorhombic. Accordingly, it cannot support symmetry operations that cubic unit cells can support, thus its space group number is 69. Distortions in TlF are as small as a certain level of noise can cause. Accordingly in the presence of errors, it is quite easy to confuse $NaCl$ and TlF structures. In general, it can be said that identifying space group in the presence of error is quite difficult. Giving wrong results is quite possible. If the error tolerance were set to a high level, then distorted materials would be treated as higher symmetry materials. Otherwise, high symmetry materials would be treated as low symmetry materials. Accordingly it is strongly recommended to use ideal data, and set the EPS parameter to a quite low value, in order to identify the space group of a material correctly.

There are a few modifications possible for identifying the space group in the presence of errors. In the original algorithm, some point searches are performed, in order to find some certain atoms. However, presence of errors can change the atoms coordinates. Therefore, instead of performing point searches, range searches with small boundaries determined according to error margins should be performed. This modification is simply a trivial correction to handle coordinate errors. Another modification that could help is, instead of finding the highest symmetry space group and finishing the procedure; every space group could be tested. Afterwards all space groups, which are supported by the crystal structure can be listed as a possible space group. Then, the user could manually analyze those alternatives. These two modifications help to identify space group number in the presence of errors. However, unfortunately, there are no possible solutions, that can help to distinguish structural distortions and distortions caused by errors. Accordingly identifying the space group procedure cannot be considered reliable in the presence of errors.

```
FindBasisVectors(Vects,Origin)
ClusterList=NULL;
foreach Atom A in most crowded group do
    C=new Cluster(A);
    C.Center=A.Coordinates;
    ClusterList+=C;
P=The paralleloid defined by Vects and starting from Origin;
C=Find the closest cluster to Origin whose center lies inside P;
DV=C.Coordinates-Origin.Coordinates;
foreach Cluster C in ClusterList do
    C.Center-=DV;
foreach Unprocessed Group G do
    MinDist=∞ ;
    MinAtom=NULL ;
    MinCluster=NULL ;
    foreach Atom A in G do
        foreach Cluster C in ClusterList do
            if A lies inside paralleloid defined by Vects starting from
            C.Center then
                D= distance between C and A ;
                if D < MinDist then
                    MinDist=D;
                    MinAtom=A;
                    MinCluster=C;
    DV=MinAtom.Coordinates-MinCluster.Coordinates;
    foreach Cluster C in ClusterList do
        if There is an atom A in G at C.Coordinates+DV then
            C.Assign(A);
    foreach Cluster C in ClusterList do
        if C.AtomCount < NumOfProcessedGroups then
            Remove(C);
ReturnList=NULL;
foreach Cluster C in ClusterList do
    C.Center=Average atom coordinates of C;
    if C.Center lies inside P then
        ReturnList+=C.AtomList;
return ReturnList;
```

Algorithm 15: The algorithm for finding basis vectors with error handling mechanism

Chapter 4

Implementation

In this chapter, implementation details will be described. In the first section, programming environment will be introduced. In the second section, data structures used will be explained. Finally, in the third section, the algorithms will be discussed. The complexity analysis of the algorithms will also be done in this section.

4.1 Programming Environment

The name of the tool we implemented is *BilKristal*. The implementation consist of three sub-programs, *Analyzer*, *VisualizationTool* and *UserInterface*. *Analyzer* program performs the pattern extraction analysis and calculates the unit cell parameters. *VisualizationTool* uses unit cell parameters and provides a good visualization tool to observe the crystal structure. *UserInterface* program provides a good user interface to the user. It also handles interactions with the user and other sub-programs. Main reason to have three different sub-programs instead of one, is the desire to obtain high performance program with good user interface. As known well, *C* language is one of the best languages in terms of performance. Programs that require performance, are generally written by using *C* language. Since the pattern information extraction from the crystal structures is a hard

job, which require lots of computation power, using a high performance language is essential. For the visualization part, it is not quite essential to use a high performance language since; usually graphic cards determine the performance of the visualization phase. Accordingly, the effect of the programming language on the performance would not be much. Nevertheless, using a high performance language is preferable. So, *Analyzer* and *VisualizationTool* programs are written in *C* language. Since these two programs are not logically bounded, they are written as two separate programs instead of one program. With this approach, the implementation phase becomes simpler and a little performance improvement is obtained.

UserInterface program provides interactions with the user. Accordingly, it requires a good user interface. *C++ with .NET platform* provides the necessary environment for developing programs with good user interfaces. *C++ .NET* language is a relatively fast language too. Accordingly, *UserInterface* program is implemented in *C++ .NET* language. By separating these programs, instead of writing one program, the implementation phase become easier and simpler.

In the following sections, these three sub-programs will be explained in more detail.

4.1.1 Analyzer

Analyzer is the program that executes the pattern extraction algorithms. The unit cell parameters are found by the *Analyzer* program. It obtains initial parameters, such as the input data path and the analysis constants such as *EPS*, as argument list while it is started. It receives other parameters, which are determined throughout the analysis, such as the origin choice or the primitive vector selections of the user, via the input. It gives its results by writing on temporary files. It also writes several commands indicating the results are written or the program is waiting for the input, the percentage values showing the progress of the analysis and several other comments, to output. Simply, the *Analyzer* reads

the input data, performs the analysis according to the given parameters and outputs the results. *Analyzer* is written in *C* language by using *Microsoft Visual C++ 6.0* development tool. It is a console application and it is not designed to be a stand-alone program.

4.1.2 VisualizationTool

VisualizationTool is responsible for visual representation of the crystal structure. It uses the unit cell parameters as input. It provides several features to visualize the crystal structure more effectively. It is also written in *C* language by using *Microsoft Visual C++ 6.0* development tool. It also uses *OpenGl* graphic libraries to handle the graphics. *Glut* library is also used as well as the standard OpenGl libraries in the *VisualizationTool* program. Similar to *Analyzer* program, *VisualizationTool* is also a console application and does not have a user interface other than created display window. And, it is not designed to be a stand-alone program too.

The *VisualizationTool* program obtains required parameters from a file, which is created by the *UserInterface* program. This file contain the unit cell parameters such as the primitive vectors and the basis vectors as well as atom parameters such as the color and the radius values of each atom type in the basis set. Initially a single unit cell is shown. Afterwards according to the users choices, graphics is altered. The user informs the *VisualizationTool* program about his choices by using two different ways. The first one is using the interactions with display window. This method is implemented by using *Glut*s event handling mechanisms. Display window is capable to understand the mouse and the keyboard inputs. The user can use the mouse to rotate the 3D environment. In order to perform the rotation, simply dragging the screen is sufficient. The user can also use the arrow keys, and PageUp, PageDown keys to rotate the screen around three different axes. The user also can change the camera's distance to the center point of the 3D environment by using Home and End keys. Accordingly, camera can get closer or distant to the crystal structure. There are also other keys designed as shortcut keys, which are programmed to perform several actions.

VisualizationTool can also interact with user via the *UserInterface* program. Users can interact with the user interface that *UserInterface* program provided. The *UserInterface* redirects the users commands to the *VisualizationTool* program via writing into *VisualizationTool*'s input. The *VisualizationTool* program contains a thread, which checks it's input continuously for incoming commands. Accordingly, users interaction with the user interface provided by the *UserInterface* program can be understood.

There are several actions that the user can perform with the *VisualizationTool*. Most important ones can be explained as follows;

1. **Creating Multi-Cells:** The first type of action is the creating multi-cells. The *VisualizationTool* basically draws a single unit cell by translating the coordinate system by some integer combination of the primitive vectors and drawing each basis atom afterwards. Defining multi-cells is done by determining the repetition numbers for each primitive vector. These repetition numbers represent the integer coefficients that are used to determine the translation vector of the coordinate system. There are two repetition numbers for each primitive vector. First number is the minimum value and the second one is the maximum value of those integer coefficients. In general, a unit cell is drawn for each integer coefficient combinations, whose integer coefficients for each primitive vector lies in between those minimum and maximum values. For example, consider 0 is used as the minimum repetition numbers and 1 is used as the maximum repetition number for all primitive vectors. Then 8 unit cells are drawn which lies in the unit cell that primitive vectors, $2V_1$, $2V_2$ and $2V_3$ define. Similarly consider the case that minimum and maximum repetition values for each vector are V_1:0,0 , V_2:1,3, V_3:-1,1. Then 9 unit cells are drawn, which lies in the unit cell that primitive vectors, V_1, $3V_2$ and $3V_3$ define and using $V_2 - V_3$ point as the origin.

2. **Changing Drawing Options:** The user is allowed to change some drawing options. The first option is the draw size. By default, each atom is drawn with it's original size. However, generally crystal structures are well

packed and understanding the crystal structure as it is, can be hard. Scaling each atoms radius to a smaller value can help the user to understand the crystal structure. Accordingly, the user is allowed to select a scaling ratio. The second option is enabling or disabling to show sticks. Sticks are used to connect touching atoms, when they are scaled to a smaller size and not touching anymore. Generally, they are quite helpful and desired. However since the users are not forced to give radius values of atoms in our work, the stick model might not be accurate. Accordingly showing the sticks might not be desired. User can enable or disable to show sticks. The third draw option is enabling or disabling to draw unit cells, complete. By default, for a single unit cell, only the basis atoms are drawn. However, generally, these basis atoms do not give information about the unit cell geometry. Consider the $CsCl$ structure shown in Figure 2.1. It only contain 2 basis atoms, one Cs and one Cl. Accordingly, these two atoms do not provide a cubic unit cell geometry. In general, for single unit cells, the desired drawing is the one given in the Figure 2.1, which is a cube where there are Cl atoms at each corner and a Cs atom at the center. In this view, some atoms belonging to the neighbor unit cells are also included into the drawing. This view can be considered as the set of all atoms, which are placed inside or on any faces of the unit cell paralleloid that the primitive vectors define. By enabling or disabling the draw unit cells complete option, user can obtain this view or only basis atoms. Both views can be desirable for several situations. Another two drawing options are enabling or disabling to show controls and primitive vectors on the display window. If the drawing controls is enabled, list of keyboard shortcuts that can be used on the display window are shown. Similarly, if drawing the primitive vectors are enabled, they are drawn on the display window.

3. **Enabling or Disabling to Draw Certain Atom Types:** There is a part in the user interface which can be considered as the legend. At this part, every atom types in the crystal are listed. Colors of those atom types are also indicated, so that user can identify them. In addition, a checkbox for each atom type that the user can enable or disable drawing of that atom type, are also included. Accordingly, user can show or not show some

certain atom types, thus examine the crystal structures with different views.

4. **Defining Cut Planes:** Another important feature that the *Visualization-Tool* provides is defining cut planes. A cut plane can be described as a plane where crystal structure is divided into two parts. It is defined by three numeric values and a cut operation. These three numeric values are given in the crystal indexing system [9] and they define the plane. The crystal indexing system considers primitive vectors as its three main axes. In order to define a plane in the crystal indexing system, values where this plane intersects with each of these main axes should be calculated. Inverses of those values according to multiplication, defines this plane. For example, consider primitive vectors, V_1,V_2 and V_3 are assumed to be main axes. And let plane P intersects each main axes at $(iV_1, 0, 0)$,$(0, jV_2, 0)$ and $(0, 0, kV_3)$ points respectively. Then $(\frac{1}{i}, \frac{1}{j}, \frac{1}{k})$ are the values that define the plane P in the crystal indexing system. Cut operations are simple comparison operators such as $>$,$<$,$\geq$,$\leq$,$=$. Any atom, which does not satisfy a cut operation, will not be drawn. In order to define cut planes, the crystal indexing system is used because it is the common indexing system for planes in crystallography. The user is allowed to define as many cut planes as he desires. Accordingly, he can shape the crystal structure into any convex polyhedral shape he desires.

5. **Dumping Atomic Coordinates:** Dumping the atomic coordinates into a file is another important action that the user can perform. After performing several operations, users might want to obtain the list of atomic coordinates. User might want to use those coordinate values as input to other utilities. Accordingly, user is allowed to dump atomic coordinates of all atoms that are currently shown on the screen, into a file. User can select to dump fractional coordinates or Cartesian coordinates.

6. **Animations:** User can use the animation options to see the crystal structure in three dimensional way. In order to have the sense of depth from a scene on the computer screen, several methods are available. Some of these methods require special hardware, such as 3D glasses. However, with

motion, 3D view can be obtained much easier. In this work, small animations are used to provide the motion that is required to obtain the 3D view. Animations used in *VisualizationTool* are simple combinations of rotations around three main axis. Since the aim of the animation is helping the user to observe the crystal structure in a 3D environment, different types of animations might be distracting, thus they weren't used in this work. The user can select animation style and speed. 5 different animation styles are defined. These styles are the rotations around three axes, rotations around a user defined vectos and a random combination of rotations around main axis. In general, even though the animation property is quite simple, it works quite efficiently. The user can observe the crystal structure quite efficiently with the help of the animation property.

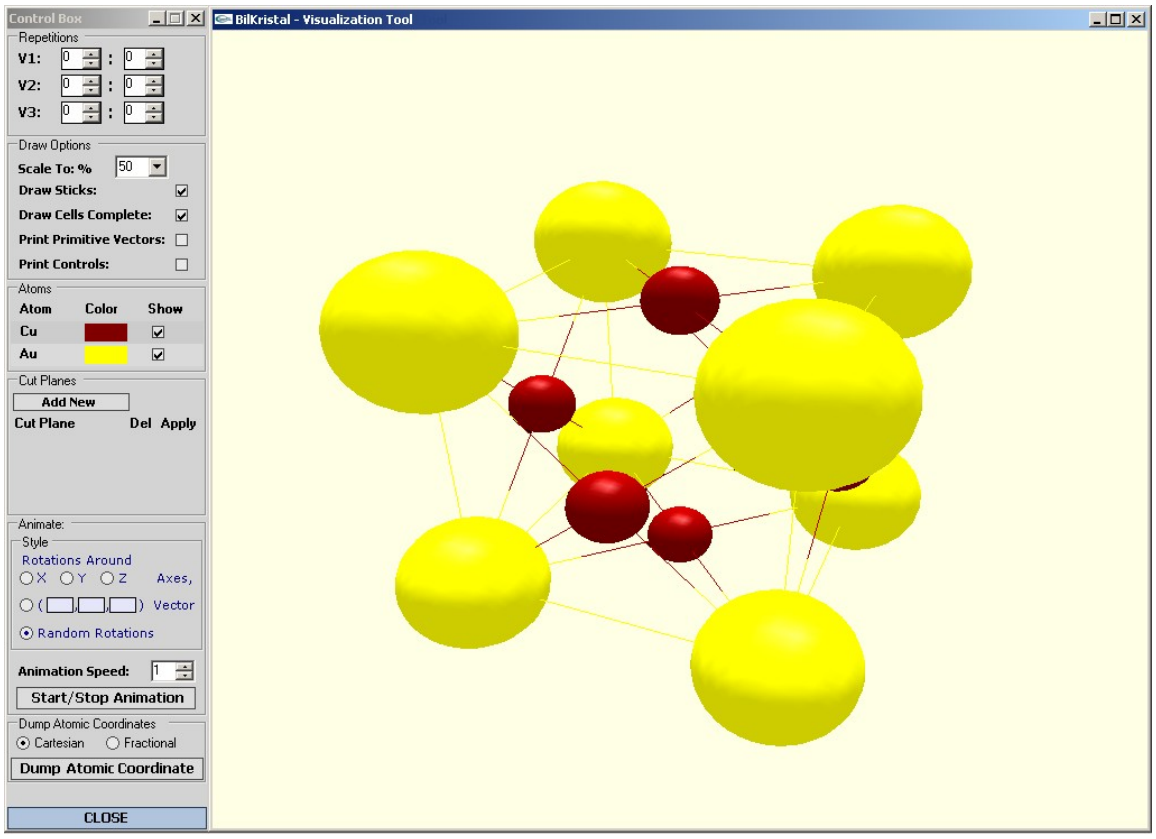

Figure 4.1: The crystal visualization tool screenshot

In Figure 4.1, a screenshot of the *VisualizationTool* program can be seen. The toolbox at the left belong to the *UserInterface* program and display window at the right belongs to the *VisualizationTool* program.

4.1.3 UserInterface

UserInterface is the main program that the user interacts with. It provides a user interface for the *Analyzer* and the *VisualizationTool* programs. It is written in *C++ .NET* language by using *Microsoft Visual Studio .NET 2003* development tool. It handles the interactions with the user. It receives input data and analysis parameters from uthe ser and it runs the *Analyzer* program accordingly. In order to execute the *Analyzer* program it uses the *Processing* libraries of .NET environment. The *Analyzer* program's console window is not shown to user. The input and the output of the *Analyzer* program are redirected to the *UserInterface* program, thus interactions between those two programs are done via these IO streams. The *Analyzer* program uses it's output to inform *UserInterface* about it's current state of the analysis. For example, the *Analyzer* program informs the *UserInterface* about which algorithm is finished, the completed percentage of the current algorithm, etc. Temporary files are also used to exchange mass amount of data, such as the list of possible primitive vectors. The *UserInterface* shows the primitive vector alternatives, the origin alternatives and the results to the user according to the data that the *Analyzer* provided. Similarly it transmits the user's choice of primitive vector alternatives, origin, etc. to the *Analyzer* program. The *UserInterface* program also informs the user about the progress of algorithms.

After the *Analyzer* program finishes, results are shown to user. The user can see the unit cell structure extracted from the input data by selecting the visualize option. Then the *UserInterface* program runs the *VisualizationTool* program with the corresponding parameters. The *UserInterface* program also provides a user interface that allow the user to change some visual properties of the unit cell structure, to combine several unit cells, etc. Interactions between the *UserInterface* and the *VisualizationTool* programs are done in a similar way

to the *UserInterfaces* interaction with the *Analyzer*. However, in this case users can also interact with the *VisualizationTool* program directly.

The *UserInterface* program also provides an interface to run the *VisualizationTool* program according to the unit cell parameters that user provided directly. Accordingly, the *BilKristal* tool can also be used as a crystallographic visualization tool.

4.2 Data Structures

Several data structures are used in this implementation. Most of these data structures, are used in the *Analyzer* program. In the *UserInterface* program, some of the data structures that are used in the *Analyzer* program are also used. In the *VisualizationTool* program, mainly three different data structures are used. In Appendix A, these data structures can be seen.

Most important data structures that are used in the *Analyzer* program can be listed as follows.

- **Point structure:** The basic data structure used in this framework is the *point* structure. It represents an atoms coordinates given in the input data. It also contains information representing the atom type.
- **Octree node structure:** The second data structure used in *Analyzer* program is the *octreeNode* structure. This structure is used to index atomic coordinates in octree structure. It contain fields showing the volume assigned to this node, pointers to children nodes, pointer to data record, etc.
- **Group structure:**Another data structure used in the algorithms is the *group* structure. It is used by the algorithm for grouping identical atoms. It contains fields representing the atom type of this group and matching volume of this group. It also stores linked list pointers connecting every atom belonging to this group.

- **Vector structure:** The *vector* structure represents the vectors. These structures are created in the algorithm for extracting the vectors and they are used throughout the vector operations. This structure contain three fields that represent the vectors lengths at each axis.

- **Primitive vector structure:** The *primitiveVector* represents a primitive vector set. It is mainly used in the vector operations. It contain three *vector* structures representing three vectors of the primitive vector set. It also contain several other fields representing some unit cell parameters that this *primitiveVector* structure define.

  Since three vector structures stored in the *primitiveVector* are sufficient to define the unit cell perfectly, other fields of the *primitiveVector* structure are redundant. However, since sorting primitive vector structures is required, those redundant data are included to the data structure. Many valid primitive vector alternatives will be calculated during the analysis. Sorting these alternatives and finding the most user friendly one would be an important task. Sorting the primitive vectors require using those redundant values. Accordingly, instead of calculating those values several times during the sorting procedure, calculating them once and spending some extra memory space is preferred in this work.

- **Cluster structure:** The *cluster* structure is used to represent clusters. This structure contains fields representing the center of the cluster. The structure also stores a linked list connecting every point belonging to this cluster.

- **Basis point structure:** The *basisPoint* structure represents a basis vector. This structure contains a *point* structure that stores the coordinates of the basis vector. This structure also store fractional coordinates of the basis vector.

- **Space group structure:** The *spaceGroup* structure contains information about a space group. The structure contains some inforation about the space group, such as the group number an a description. The structure also stores data representing symmetry operations of the space group.

The *UserInterface* program also uses several data structures. However, most of these data structures are identical with the ones used in the *Analyzer* program. Since these two programs interact with each other, they should agree on the data structures that are to be exchanged. Other than those duplicate data structures, the *UserInterface* program uses simple data structures such as, linked lists and arrays.

The *VisualizationTool* uses three important data structures. These data structures can be listed as follows.

- **Point structure:** The *point* structure is quite similar to the *point* structure used in the *Analyzer* program. It represents an atom that is to be drawn. Accordingly, this structure contains coordinate values of the atom. This structure also stores other parameters representing the atom type such as the color or the radius of the atom.
- **Stick structure:** The *stick* structure represents the sticks that are drawn between two touching atoms. It is a quite simple structure. This structure simply contains two pointers to two atoms which the stick should be drawn in between.
- **Cut Plane Structure:** The *cutPlane* structure represents the cut planes that user defines. This structure simply stores plane parameters in the crystal indexing system and the cut operation.

4.3 Algorithms

The algorithms are mostly used in the *Analyzer* program. In the *UserInterface* program mostly user interface routines and IO procedures are used. In the *VisualizationTool* program, simple procedures that are used to perform event handling and drawing are written. These procedures are implemented using general programming techniques. Accordingly in this section, algorithms that are used in the *Analyzer* program are explained.

4.3.1 Reading Input Data

In this work, it is assumed that the input data contain a record for each atom in the crystal structure. A record contains an atom type descriptor and atomic coordinates of the atom. A record should be written in one line in the data file. Two sample lines are given as follows.

```
Na -5.66 2.83 -14.15
Cl -2.83 5.66 -11.32
```

The *Analyzer* simply reads the input data line by line. It checks if the atomic coordinates lie inside some cubic volume defined according to the starting parameters. If atomic coordinates are outside that volume, it skips that line and continues. Otherwise, a point data is created with the information given in this line. A unique id number is assigned to each distinct atom type descriptor. These unique id numbers are used as the atom type id number throughout the algorithms. After the id number is assigned to the point data, program checks if the atomic coordinates lie inside the processing volume. If the coordinates lie inside, created point structure is inserted into the link list that stores points that are to be analyzed. Otherwise, point is inserted into another linked list. While reading the data, the maximum and the minimum values of x,y and z coordinates of the atoms, which are not skipped, are also noted. These values will be used while initializing the octree structure.

Algorithmic complexity of this procedure is linear time. More precisely it can be written as $O(N + S)$ where N is the number of lines which are not skipped and S is the number of lines that are skipped.

4.3.2 Indexing Input Data

Indexing the input data is done by using the octree structure. A specific implementation of the octree structure is used in this work. Firstly, volumes that are assigned to each node are constant and should be specified while creating the

node. Some implementation of the octree structure, use dynamic volumes, but since the crystal structures can be considered homogeneous, using static volumes will not bring an overhead. Instead, with achieved simplicity the runtime performance improves. Another property of the octree structure that is used in this work is it contains only one data record per node and only leaf nodes store data. In this way, swap operations are not needed. For a random data, one data per node rule can be considered as a quite bad choice since the high number of points can be placed in a small volume. For those cases, the access times increases significantly. However for homogeneously distributed data, such as the crystal data, this approximation allows to access any point in logarithmic time. There are some search techniques, such as storing multiple data records per node while keeping them sorted, which allow logarithmic access times with modified search routines. However, one record per leaf node approach is preferred in this work because of its simplicity. Some extra space is required in this implementation. However, such space requirement is not quite high, thus acceptable.

The first step of indexing input data is initializing the octree structure. The initialization can be described by simply creating the root node and assigning some maximum volume that root node. The maximum and the minimum values of x, y and z coordinates, which are calculated while reading the input data, are used to define the volume used in initialization. After the initialization completed, created point structures are inserted into the octree structure iteratively. An insertion has logarithmic time complexity. Accordingly inserting all points will have $O(Nlog(N))$ complexity. The space requirement will be linear.

4.3.3 The Algorithm for Grouping Identical Atoms

In the algorithm, matching volumes of each atom that are to be analyzed are calculated. Since the crystal data is homogeneous, the number of atoms in each matching volume can be considered constant. If we call this constant M, than obtaining the matching volume of an atom has a time complexity of $O(log(N) + M)$, since accesses to the boundaries of the matching volume can be done in logarithmic time and maintaining the linked list of the output can be done in

linear time.

After the matching volume of an atom is obtained, for each previously found group atom's matching volume is compared with group's matching volume, in order to see if they match. Matching volume's of the groups are indexed with the octree structure. So checking if two matching volumes matches can be done in $O(Mlog(M))$ time. If the matching volumes matches, then the atom's group is determined. Atoms can be inserted into the atom list of the corresponding group in constant time and the procedure continues with the next atom. However, if the atom does not match with any group, a new group should be created. Creation of a new group requires copying of atom's matching volume into group's matching volume, and indexing it. This procedure can be executed in $O(M + Mlog(M))$ time.

If we call G to the number of groups that will be generated throughout the grouping algorithm and A to the number of atoms that are to be analyzed, then the time complexity will be $O(Alog(N)+AM+GAMlog(M)+GM+GMlog(M))$. The first part, $Alog(N) + AM$, represents the complexity of obtaining matching volumes of every atom that are to be analyzed. The second part, $GAMlog(M)$, shows the matching volume comparisons of every atom with every group. Finally, the last part, $GM + GMlog(M)$, represents the creation times of each group. In the worst case, number of groups can be equal to N, which is the case that every atom defining a group. In that case, the complexity will be $O(NAMlog(M))$. However, this is a very unlikely case. Most of the time, the number of groups that will be generated is a small number, if a valid crystal data is used. Thus, for the average case, G can be considered as a small constant value. Accordingly, the complexity reduces to $O(Alog(N) + AMlog(M))$.

4.3.4 Vector Operations

The vector operations contain four algorithms. The first algorithm is the algorithm for extracting vectors. In this procedure, for every group G, for each atom belonging to this group, a vector which represents the relative distance between

this atom and the reference point of G, is created. Totally $A - G$ vectors are created and accordingly this procedure has $O(A)$ time complexity where A is the number of atoms to analyze. After the vectors are generated, they are sorted. By applying a few linear scans on the sorted list, vectors, which are identical, are combined. Since the sorting has a time complexity $O(Alog(A))$, this algorithm has an overall time complexity of $O(Alog(A))$.

The second algorithm is the algorithm for filtering out redundant vectors. This operation is done by comparing every pair of vectors produced by the extracting vectors algorithm, in order to see if one of the vectors is integer multiple of the other one. Such vectors which are integer multiples of other vectors are eliminated. Another sorting is performed after eliminating those vectors. The time complexity of this algorithm will be, $O(V^2)$ where V is the number of vectors produced in the extracting vectors algorithm. This leads to a worst case complexity of $O(A^2)$, since V can be as much as $A - G$ and G can be 1. Even though the quadratic time complexity seems high, the runtime performance of this algorithm is much better. The extracting vectors algorithm often produces much less number of vectors than A. Accordingly, runtime performance can be expected to be quite reasonable, even though the quadratic worst-case complexity.

The third algorithm is the algorithm for calculating primitive vector candidates. For this purpose, every triplets of vectors is tested in order to see if they can produce every vector in the vector list as their integer combinations. This algorithm has a time complexity of $O(A^4)$. However, the number of vectors that can be in a vector triplet, is limited to a relatively small constant. Accordingly worst case complexity reduces to $O(P^3A)$, where P is equal to this constant.

The fourth algorithm is the algorithm for purifying primitive vector candidates. This procedure is simply a sorting, which brings the primitive vector candidates, which look nicer to the user to front. Accordingly, it has a time complexity of $O(PVClog(PVC))$, where PVC is the number of primitive vector candidates calculated by the previous algorithm. Since the upper bound of PVC is $P \times (P - 1) \times (P - 2))$, the worst case complexity of this algorithm can be written as $O(P^3log(P))$.

Overall worst case time complexity of the vector operations is,

$$O(Alog(A)) + O(A^2) + O(P^3A) + O(P^3log(P))$$

where P is equal to the parameter representing maximum number of vectors which can form a primitive vector candidate. Since $log(P)$ is much smaller than A, the given worst case complexity reduces to $O(A^2 + P^3A)$.

4.3.5 The Clustering Algorithm

In the clustering algorithm, firstly the groups are sorted according to the number of atoms in their lists. Then a cluster is created for every atom in the most crowded group. This operation takes $O(G_0)$ time, where G_i represents number of atoms belonging to i^{th} group. After that, every group is processed concurrently. To process a group a direction vector is calculated. Direction vector can be described as the minimum distance between a cluster center and an atom belonging to the group that is currently processed. Finding the direction vector require to check every cluster-atom pairs and it can be done in $O(G_0G_i)$ time, since the number of clusters can be as much as G_0. After the direction vector is calculated, for each cluster an atom is found whose atom to cluster center distance is equal to the direction vector. This operation also has $O(G_0G_i)$ time complexity. After the assignments are completed, the procedure continues with the next group. Accordingly, overall time complexity for the clustering algorithm is $O(G_i)+O(\sum_{i=1}^{G-1} G_0G(i))$. This expression reduces to $O(G_0 * (A+1-G_0)$ when the summation is evaluated. This expression is maximal when $G_0 = (A+1)/2$. Accordingly the worst case complexity is $O(A^2)$.

4.3.6 The Algorithm for Finding Basis Vectors

This algorithm is quite similar to the clustering algorithm. The first part of the algorithm is creating initial clusters for each atom of the most crowded group. Since the size of the most crowded group can be at most A, this part has $O(A)$ complexity. Then the origins are assigned to each cluster as cluster centers. This

procedure requires scanning of all clusters in order to find the closest suitable cluster to calculate the direction vector. Afterwards another scan is done for assigning cluster centers. Accordingly this part also has a $O(A)$ time complexity. After that, for each group, atoms of the processing group are assigned to a cluster as in the clustering algorithm. Only difference is, while finding the direction vector, the atom should lie inside the unit cell paralleloid defined by given vectors starting from the cluster center. This constraint requires a constant time check, thus the complexity of this part of the algorithm is identical to the same part given in the clustering algorithm. After calculating clusters, each cluster is scanned to assign the average coordinates of atoms as cluster centers. While doing this scan, clusters are also checked in order to see if they should be in the output. This procedure has $O(A)$ time complexity. Since this algorithm brings additional $O(A)$ time complexity to the clustering algorithm whose complexity is $O(A^2)$, this procedures worst case complexity is also $O(A^2)$.

4.3.7 Identifying Space Group

In order to say a crystal structure matches to a space group, every point in the set of test points should support every symmetry operation of that space group. While applying each symmetry operation, appropriate vector set should be used. In other words, in order to test a space group belonging to some lattice class, the vector set, which define the minimal unit cell belonging to same lattice class, should be used. Since every space groups are tested, vector sets that define unit cells belonging to each lattice class should be generated. This operation is done by creating a set of vectors, which are derived from the primitive vectors as their integer combinations and testing several vector triplets from this set. In this implementation, total number of such vector triplets that are tested is limited to a reasonable number, 39,711. The combinations of the derived vectors, which are to be used, are pre-computed. This limitation may cause not being able to find vectors that define the unit cells belonging to some lattice classes. However, it is essential to limit this number in order to be able to finish this procedure in reasonable time. Since these 39,711 vector sets are selected so that they cover

all small unit cells, it is quite unlikely that this approach causes invalid results, since it is quite unlikely for a big unit cell defining the crystal characteristics.

Selecting the set of test points is done according to the primitive vectors. A volume that is guaranteed to cover at least one atom identical to each atom in the basis set, should be used. This restriction can be guaranteed by covering all atoms, which lie inside the unit cell paralleloid defined by the primitive vectors. Accordingly, the boundary of the test volume is defined according to the primitive vectors. The maximum and the minimum values in the set of 0, $V_1.x$, $V_2.x$, $V_3.x$, $V_1.x + V_2.x$, $V_1.x + V_3.x$, $V_2.x + V_3.x$, $V_1.x + V_2.x + V_3.x$ values are used as the boundary values of x axis. Boundaries of y and z axes can be found similarly. Then the atoms that lie inside this boundary, are used as test points. However, the number of total test points is also limited to a relatively high number, 1000. The reason for this restriction is to avoid unnecessarily high number of test points that can be obtained due to irregular unit cells. This approach might theoretically cause errors, but it is a quite unlikely case.

The algorithm for identifying space group requires knowing symmetry operations of each space group. In this implementation, we obtained the space group data of 273 space groups (some space groups have more than one form depending on the origin or the axis selections), from Bilbao Crystallography Server [6]. The source of symmetry operations was stated as International Tables for Crystallography, 1982 [8], which can be considered as the most common reference tables used in crystallography. Downloaded data are converted into a more suitable form, a data file. Every time the *Analyzer* program is started, this data file is read and the space group data are loaded into the data structure explained in data structures section.

Checking if a symmetry operation is supported on a point, consists of two parts. The first part is applying the symmetry operation and obtaining the translated point. The second part is checking if an identical point exists at the coordinates of the translated point. The first part has constant time complexity while the second part has the logarithmic time complexity since it require a point search. There are 273 space groups to test and the number of symmetry

operations of each space group is constant. Since the number of test points are also bounded by a constant value, the complexity of the algorithm for identifying space group is $O(log(N))$. The number of vector sets that are tested in order to find a vector set for each lattice class is also constant. Accordingly, the part that finds those vector sets has a constant time complexity. Accordingly overall complexity of this algorithm can be written as $O(log(N))$.

4.3.8 Overall Complexity

Overall complexity can be found by adding individual complexities. Accordingly, overall complexity is,

$$O(N+E)+O(Nlog(N))+O(Alog(N)+AMlog(M))+O(A^2+P^3A)+2\times O(A^2)+O(log(N))$$

where N represents the number of points in the octree structure, A represents the number of points to analyze, E represents the number of lines in the input data that are skipped, M represents the average number of points in the matching volume and P represents the maximum number of vectors which can form a primitive vector to test. Obtained expression can be collapsed into

$$O(E+Nlog(N)+AMlog(M)+A^2+P^3A)$$

as a more compact form of the average case complexity.

Chapter 5

Results and Performance

In this section, we tested our framework with several test data. We tried to cover materials on every crystallographic class. In test environment section, test data will be explained in detail. In experimental results section, outputs of the tool are compared with the actual crystal parameters. Partial outputs of the algorithms are also discussed. In the performance evaluation section, execution times of each stage of the framework for each input data will be given and these times are compared with the average case complexities of the algorithms. In the error handling section, a set of test data with different error margins are used to examine the error tolerance of the framework.

5.1 Test Environment

In order to test the framework, several test data are generated. At least one material belonging to each crystallographic class is tested. For low symmetry classes, such as triclinic and monoclinic lattices, randomly chosen primitive vector sets and basis vector sets are used. Accordingly, a more complex input data could be obtained and the algorithms could be tested more effectively. For other crystallographic classes, the real crystal parameters of actual materials are used. Crystal parameters are obtained from [7]. Data generation is done automatically

by using a program. To generate the data, each basis vector is translated by some integer combinations of the primitive vectors. If the translated point lies inside a predefined cubic volume centered at the origin, the point is written to the file. This operation is repeated with different integer combinations of the primitive vectors, until the predefined volume fills completely. Accordingly, cubic crystal segments are obtained. Boundaries of this predefined volume are selected as -20.0 to 20.0 at each axis. In this way, it is guaranteed to obtain input data which can be used by the default threshold parameter, 20.0, used in the *Analyzer* program. Since any point, which does not lie inside this predefined volume, is not included into the output, there will be incomplete unit cells. In addition, the atomic ratios in the input data will not be equal to the atomic ratios in the unit cell. Accordingly, generated data do not leak any extra information about the material. For each material, two sets of data are generated. The first set is the ideal input data and generated as explained. The second data contains coordinate errors and missing atoms. Generation of the second set of data is done similar to the first set. Basic difference is after calculating the actual position of a point, a small amount of noise is added to the coordinate values. Value of this noise is set to ± 0.03. This value is sufficiently large for a generated data. Accordingly testing with this level of noise should be sufficient to evaluate the framework. Other difference is, even though a point is qualified to be written into the output file, with 0.0001 probabilities it does not written. With the given probability a few missing atoms are expected in the input data.

Nine real and two randomly generated materials are used in the testing phase. Real materials are, $NaCl$, La_2O_3, Cu_3Au, PtS, Al_3Ti, Mg, $CoSn$, αHg and TlF. For triclinic and monoclinic lattice classes randomly generated unit cells are used. For triclinic class, a big unit cell is generated. 14 basis vectors with five different atom types are randomly distributed inside the unit cell. For monoclinic class a smaller unit cell is derived. Six basis vectors with three different atom types are used. Analysis are applied on these material's ideal and noisy data. During the analysis, default values of the analysis parameters are used. Detailed information about materials and the test results can be found in the experimental results section.

5.2 Experimental Results

In this section test results of eleven different materials are given. The tests are performed on both ideal and noisy data. As mentioned earlier, the tool proposes several choices to the user throughout the analysis, such as primitive vector set alternatives or origin choices. Throughout the tests, the most user-friendly primitive vector set and the origin are selected. The results are rounded to two decimal digits. In the first part, calculated primitive vectors and basis vectors are given and compared with the actual values. Besides, small descriptions of each structure are given in this part. In the second part, the outputs of the intermediate stages are given and discussed. Finally, space group results are discussed in the third part.

5.2.1 Primitive Vectors and Basis Vectors

Extracted primitive vectors and basis vectors information for each structure are given as follows:

1. $NaCl$:

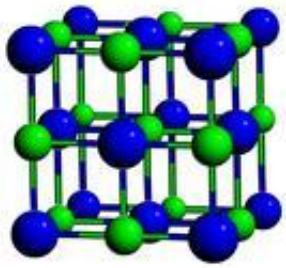

Figure 5.1: $NaCl$ Unit Cell

$NaCl$ has a cubic unit cell as shown in Figure 5.1. However, primitive unit cell is smaller. As given in the Table 5.1, three equal length vectors with 60 degrees of α,β and γ angles, present a more compact unit cell.

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[0 2.83 2.83]	[0 2.83 2.83]	[0 2.81 2.82]
V_2	[2.83 0 2.83]	[2.83 0 2.83]	[2.82 0 2.80]
V_3	[2.83 2.83 0]	[2.83 2.83 0]	[2.82 2.82 0]

Table 5.1: Primitive vectors of $NaCl$ structure

As shown in Table 5.1, with the ideal data, primitive vectors are calculated without errors and with the noisy data small distortions occur. However, considering the error margin in atomic coordinates is ± 0.03, which results in an error margin of ± 0.06 on vectors, given distortions are quite acceptable. Accordingly primitive vectors are calculated successfully for both data.

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	Na,[0 0 0]	Na,[0 0 0]	Na,[0 0 0]
B_2	Cl,[2.83 2.83 2.83]	Cl,[2.83 2.83 2.83]	Cl,[2.82 2.82 2.86]

Table 5.2: Basis vectors of $NaCl$ structure

Similar to the primitive vector calculation, basis vectors were also accurately found with the ideal data while small but acceptable distortions are observed with the noisy data. So for this material the tool worked successfully with either data.

2. Cu_3Au:

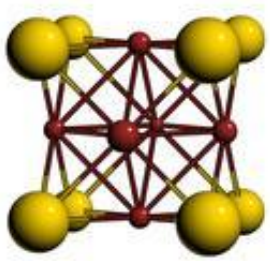

Figure 5.2: Cu_3Au Unit Cell

Cu_3Au has a cubic unit cell shown in Figure 5.2. Primitive vectors also represent a cubic unit cell. A unit cell contains one Au and three Cu atoms as the basis vectors. For Cu_3Au, primitive vectors are calculated accurately

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[3.14 0 0]	[3.14 0 0]	[3.13 0 0]
V_2	[0 3.14 0]	[0 3.14 0]	[0 3.13 0]
V_3	[0 0 3.14]	[0 0 3.14]	[0 0 3.13]

Table 5.3: Primitive vectors of Cu_3Au structure

with the ideal data and a small distortion which is about 0.01, was observed with the noisy data.

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	Au,[0 0 0]	Au,[0 0 0]	Au,[0 0 0]
B_2	Cu,[0 1.57 1.57]	Cu,[0 1.57 1.57]	Cu,[0.03 1.55 1.55]
B_3	Cu,[1.57 0 1.57]	Cu,[1.57 0 1.57]	Cu,[1.55 -0.03 1.58]
B_4	Cu,[1.57 1.57 0]	Cu,[1.57 1.57 0]	Cu,[1.57 1.54 -0.01]

Table 5.4: Basis vectors of Cu_3Au structure

Basis vectors are calculated correctly with ideal data. With the noisy data at most 0.03 differences are seen at the coordinate values. This value is not higher than the noise level in the input data, thus acceptable.

3. La_2O_3:

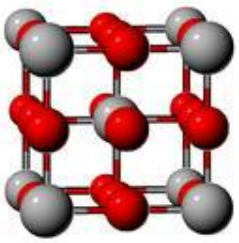

Figure 5.3: La_2O_3 Unit Cell

La_2O_3 has a cubic unit cell shown in Figure 5.3. Similar to the $NaCl$ structure, primitive vectors of this structure, define a more compact unit cell.

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[-2.57 2.57 2.57]	[-2.57 2.57 2.57]	[-2.57 2.57 2.55]
V_2	[2.57 -2.57 2.57]	[2.57 -2.57 2.57]	[2.55 -2.56 2.57]
V_3	[2.57 2.57 -2.57]	[2.57 2.57 -2.57]	[2.60 2.55 -2.58]

Table 5.5: Primitive vectors of La_2O_3 structure

Similar to previous structures, primitive vectors are found accurately with the ideal data and with the noisy data, acceptable results are obtained.

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	La,[0 0 0]	La,[0 0 0]	La,[0 0 0]
B_2	O,[2.57 0 0]	O,[2.57 0 0]	O,[2.56 0.01 -0.01]
B_3	O,[0 2.57 0]	O,[0 2.57 0]	O,[0.02 2.54 0.01]
B_4	O,[0 0 2.57]	O,[0 0 2.57]	O,[0.01 0.03 2.53]

Table 5.6: Basis vectors of La_2O_3 structure

Basis vectors are found accurately with ideal data. With noisy data, some distortions observed, but results are still acceptable.

4. PtS:

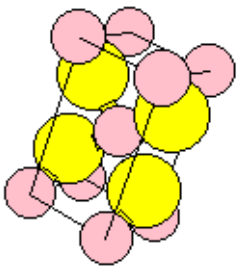

Figure 5.4: PtS Unit Cell

PtS has a tetragonal unit cell shown in Figure 5.4. The primitive vectors also define a tetragonal unit cell.

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[1.48 0 0]	[1.48 0 0]	[1.46 0 0]
V_2	[0 1.48 0]	[0 1.48 0]	[0 1.47 0]
V_3	[0 0 3.29]	[0 0 3.29]	[0 0 3.28]

Table 5.7: Primitive vectors of PtS structure

While finding the primitive vectors program gave accurate results with the ideal data and the results with 0.02 error margin are obtained with the noisy data. Accordingly, both results are acceptable.

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	Pt,[0 0.74 0]	Pt,[0.74 0 1.65]	Pt,[0 0 0]
B_2	Pt,[0.74 0 1.65]	Pt,[0 0.74 0]	Pt,[0.75 0.76 1.65]
B_3	S,[0 0 0.82]	S,[0 0 2.47]	S,[-0.02 0.72 0.84]
B_4	S,[0 0 2.47]	S,[0 0 0.82]	S,[0.01 0.71 2.46]

Table 5.8: Basis vectors of PtS structure

As shown in Table 5.8, correct results are obtained with ideal data. However, order of basis vectors changed. Since basis vectors define a set, order is not important. Accordingly, results obtained with ideal data are accurate. To obtain basis vectors with the noisy data, a different origin has been used. Accordingly, all basis points are translated to a new point. Selected origin point was, B_1 of the actual basis vectors. If every basis point of the actual basis vector set is translated by -B_1, the following basis set is obtained.

$$B_1 = Pt, [0, 0, 0]$$

$$B_2 = Pt, [0.74, -0.74, 1.65]$$

$$B_3 = S, [0, -0.74, 0.82]$$

$$B_4 = S, [0, -0.74, 2.47]$$

Since the y coordinates of some basis vectors are negative, those vectors should be translated by V_2. Those translations produce the new set:

$$B_1 = Pt, [0, 0, 0]$$

$$B_2 = Pt, [0.74, 0.74, 1.65]$$

$$B_3 = S, [0, 0.74, 0.82]$$

$$B_4 = S, [0, 0.74, 2.47]$$

Results obtained by noisy data are equivalent to this set with acceptable error margins. Accordingly, acceptable basis vectors are found with both data.

5. Al_3Ti:

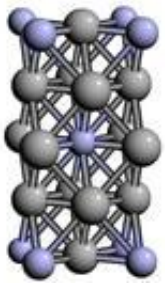

Figure 5.5: Al_3Ti Unit Cell

Al_3Ti also has a tetragonal unit cell shown in Figure 5.5. However, it's primitive vectors represent a more compact unit cell.

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[1.81 0 0]	[1.81 0 0]	[1.80 0 0]
V_2	[0 1.81 0]	[0 1.81 0]	[0 1.80 0]
V_3	[0.91 0.91 1.91]	[0.91 0.91 1.91]	[0.89 0.90 1.90]

Table 5.9: Primitive vectors of Al_3Ti structure

As shown in Table 5.9 acceptable results are obtained with both data. With ideal data, results are accurate and with noisy data, small distortions observed.

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	Ti,[0 0 0]	Ti,[0 0 0]	Ti,[0 0 0]
B_2	Al,[0.91 0.91 0]	Al,[0.91 0.91 0]	Al,[0.92 0.93 -0.01]
B_3	Al,[0.91 0 0.95]	Al,[0.91 0 0.95]	Al,[0.90 0.02 0.96]
B_4	Al,[0 0.91 0.95]	Al,[0 0.91 0.95]	Al,[0 0.89 0.92]

Table 5.10: Basis vectors of Al_3Ti structure

Similar to the previous structures, with the ideal data accurate results and with the noisy data acceptable results are obtained.

6. Mg:

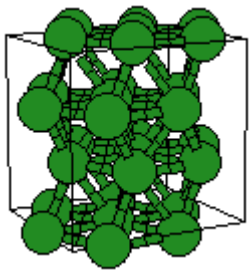

Figure 5.6: Mg Unit Cell

Mg has a hexagonal unit cell shown in Figure 5.6. Its crystal structure is called hexagonal closed pack (hcp), which is proven to be one of the densest packing for identical atoms [3].

As shown in Table 5.11, with ideal data, exact values are obtained. However, with noisy data situation is different. If $-V_1$ was used instead of V_1 in actual primitive vectors, an equivalent primitive vector set to the one obtained with noisy data would be obtained. Accordingly, the tool has found another valid primitive vector set with noisy data. Actually, such output is intentionally

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[0.86 -1.49 0]	[0.86 -1.49 0]	[0.83 1.48 0]
V_2	[0.86 1.49 0]	[0.86 1.49 0]	[-0.85 1.50 0]
V_3	[0 0 2.81]	[0 0 2.81]	[0 0 2.81]

Table 5.11: Primitive vectors of Mg structure

selected with noisy data, in order to demonstrate the tool's ability to find alternative primitive vector sets.

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	Mg,[0.86 0.50 0.7]	Mg,[0 0 0]	Mg,[0 0 0]
B_2	Mg,[0.86 -0.50 2.1]	Mg,[0.86 0.50 1.40]	Mg,[-0.03 1.01 1.41]

Table 5.12: Basis vectors of Mg structure

In actual basis vectors no atoms coordinate was chosen as the origin. However, one of the atoms coordinate is used as basis with the the. Accordingly, the calculated basis vectors and the actual basis vectors have different values. Let define new basis vector set based on actual vector set by using B_1's coordinates as origin. Then the following set would be obtained;

$$B_1 = Mg, [0, 0, 0]$$

$$B_2 = Mg, [0, -1.00, 1.40]$$

Afterwards if B_2 is translated with V_2 then the following set is obtained.

$$B_1 = Mg, [0, 0, 0]$$

$$B_2 = Mg, [0.86, 0.50, 1.40]$$

This basis vector set is identical to the set that is obtained with the ideal data. Accordingly, obtained results are accurate with ideal data.

The basis vector set obtained with the noisy data is accurate, if actual basis vector set id convertible to the obtained basis vector set with a set of

translations. If B_2's coordinates are considered as origin, then the obtained set would be,

$$B_1 = Mg, [0, 1.00, -1.40]$$

$$B_2 = Mg, [0, 0, 0]$$

If B_1 is translated by V_3, following set is obtained

$$B_1 = Mg, [0, 1.00, 1.40]$$

$$B_2 = Mg, [0, 0, 0]$$

which is identical to the basis vector set obtained with noisy data aside from small noise. Accordingly, the results can be considered accurate.

7. $CoSn$: $CoSn$ has a hexagonal unit cell shown in Figure 5.7. As in Mg structures two different primitive vector alternative is used to obtain the results.

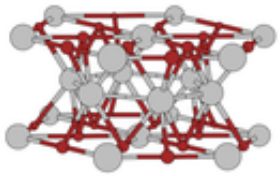

Figure 5.7: $CoSn$ Unit Cell

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[1.23 -2.13 0]	[-1.23 2.13 0]	[1.23 -2.12 0]
V_2	[1.23 2.13 0]	[1.23 2.13 0]	[1.23 2.12 0]
V_3	[0 0 4.02]	[0 0 4.02]	[0 0 4.02]

Table 5.13: Primitive vectors of $CoSn$ structure

As shown in Table 5.13, primitive vectors obtained with the noisy data are equivalent to actual primitive vectors aside from small distortions. However,

for primitive vectors obtained with ideal data, $-V_1$ of the actual primitive vectors are used as V_1. Accordingly, both alternatives are valid and acceptable.

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	Sn,[0 0 0]	Sn,[0 0 0]	Sn,[0 0 0]
B_2	Sn,[1.23 0.71 2.01]	Sn,[0 1.42 2.01]	Sn,[1.27 0.71 2.01]
B_3	Sn,[1.23 -0.71 2.01]	Sn,[0 2.84 2.01]	Sn,[1.24 -0.69 2.03]
B_4	Co,[0.62 -0.36 0]	Co,[-0.62 1.07 0]	Co,[0.63 -1.07 -0.02]
B_5	Co,[0.62 0.36 0]	Co,[0.62 1.07 0]	Co,[0.62 1.05 -0.01]
B_6	Co,[1.23 0 0]	Co,[0 2.13 0]	Co,[1.23 0.03 -0.03]

Table 5.14: Basis vectors of $CoSn$ structure

$CoSn$ structure has six basis vectors. Luckily, one of them was used as origin. While getting the results, the same atom is selected as origin, in order to make the validation of the results easier. Since the primitive vectors obtained with the noisy data are equivalent to actual primitive vectors, obtained basis vector set is also quite similar. Some distortions are observed but it is easy to see these two sets are identical. However, for the basis set obtained with the ideal data, further checks are needed. Using fractional coordinates is a more convenient way to see such changes. Basis set of actual basis vectors with the fractional coordinates are,

$$B_1 = Sn, [0, 0, 0]$$
$$B_2 = Sn, [\frac{1}{3}, \frac{2}{3}, \frac{1}{2}]$$
$$B_3 = Sn, [\frac{2}{3}, \frac{1}{3}, \frac{1}{2}]$$
$$B_4 = Co, [\frac{1}{2}, 0, 0]$$
$$B_5 = Co, [0, \frac{1}{2}, 0]$$
$$B_6 = Co, [\frac{1}{2}, \frac{1}{2}, 0]$$

If $-V_1$ were used to calculate fractional coordinates instead of V_1, fractional coordinates of the actual basis set would be,

$$B_1 = Sn, [0, 0, 0]$$

$$B_2 = Sn, [\frac{-1}{3}, \frac{2}{3}, \frac{1}{2}]$$
$$B_3 = Sn, [\frac{-2}{3}, \frac{1}{3}, \frac{1}{2}]$$
$$B_4 = Co, [\frac{-1}{2}, 0, 0]$$
$$B_5 = Co, [0, \frac{1}{2}, 0]$$
$$B_6 = Co, [\frac{-1}{2}, \frac{1}{2}, 0]$$

Translating these vectors with V_1 results in the following set;

$$B_1 = Sn, [0, 0, 0]$$
$$B_2 = Sn, [\frac{2}{3}, \frac{2}{3}, \frac{1}{2}]$$
$$B_3 = Sn, [\frac{1}{3}, \frac{1}{3}, \frac{1}{2}]$$
$$B_4 = Co, [\frac{1}{2}, 0, 0]$$
$$B_5 = Co, [0, \frac{1}{2}, 0]$$
$$B_6 = Co, [\frac{1}{2}, \frac{1}{2}, 0]$$

This vector set is identical to the vector set obtained with ideal data. Accordingly, basis vector results are acceptable for both data.

8. αHg:

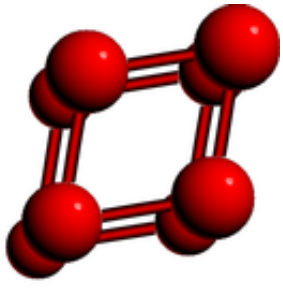

Figure 5.8: αHg Unit Cell

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[2.0 0.33 0.33]	[1.99 0.33 0.33]	[0.34 1.95 0.29]
V_2	[0.33 2.0 0.33]	[0.33 1.99 0.33]	[1.96 0.34 0.33]
V_3	[0.33 0.33 2.0]	[0.33 0.33 1.99]	[0.37 0.27 1.98]

Table 5.15: Primitive vectors of αHg structure

αHg is one form of Hg crystals. It can be considered as cubic structure, which is distorted by increasing the diagonal length. The unit cell of αHg can be seen in Figure 5.8.

Primitive vectors that are obtained with ideal data have small errors in lengths, probably caused by double's precision and the rounding errors. Noisy data result contain more distortions. However, error margins are acceptable. Accordingly, primitive vectors obtained by either data are valid and acceptable. αHg has one basis vector, which is placed at the origin.

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	Hg,[0 0 0]	Hg,[0 0 0]	Hg,[0 0 0]

Table 5.16: Basis vectors of αHg structure

Accordingly, both results are trivially accurate.

9. TlF:

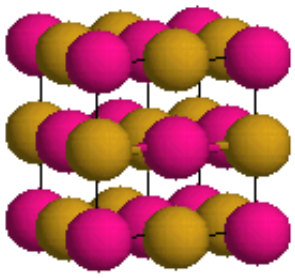

Figure 5.9: TlF Unit Cell

TlF has a similar structure with $NaCl$ structure as can be seen from Figure

5.1 and Figure 5.9. However, its unit cell is not cubic like the $NaCl$ unit cell. TlF has an orthorhombic unit cell. It can be considered as the distorted form of the cubic unit cell of $NaCl$.

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[0 1.17 1.08]	[0 1.17 1.08]	[0 1.17 1.06]
V_2	[1.11 0 1.08]	[1.11 0 1.08]	[1.11 0 1.07]
V_3	[1.11 1.17 0]	[1.11 1.17 0]	[1.11 1.14 0]

Table 5.17: Primitive vectors of TlF structure

TlF has primitive vectors similar to the $NaCl$ structure. The vector set obtained with the ideal data are accurate and vector set obtained with the noisy data has acceptable errors.

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	Tl,[0 0 0]	Tl,[0 0 0]	Tl,[0 0 0]
B_2	F,[0 0 1.08]	F,[0 0 1.08]	F,[0.03 0.06 1.06]

Table 5.18: Basis vectors of TlF structure

Similar to $NaCl$, TlF also has two basis vectors. With ideal data, accurate results are obtained. With noisy data obtained error margin is acceptable.

10. Randomly Generated Monoclinic Material: This material is randomly generated. It has a monoclinic unit cell. Unit cell parameters are predetermined and basis vectors are randomly placed inside the unit cell. a, b and c parameters are set to 4.0,5.0 and 6.0 respectively while β parameter is set to 72 degrees.

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[4.00 0.00 0.00]	[4.00 0.00 0.00]	[3.99 0.00 0.00]
V_2	[0.00 5.00 0.00]	[0.00 5.00 0.00]	[0.00 4.99 0.00]
V_3	[1.85 0.00 5.71]	[1.85 0.00 5.71]	[1.84 0.00 5.71]

Table 5.19: Primitive vectors of random monoclinic data

As Table 5.19 imply, primitive vectors are calculated accurately with ideal data and almost accurately with noisy data.

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	0,[2.27 3.35 1.94]	0,[2.27 3.35 1.94]	0,[2.27 3.34 1.95]
B_2	0,[0.44 3.45 1.37]	0,[0.44 3.45 1.37]	0,[0.42 3.45 1.35]
B_3	1,[4.27 2.90 3.54]	1,[4.27 2.90 3.54]	1,[4.26 2.87 3.55]
B_4	1,[3.39 0.25 2.57]	1,[3.39 0.25 2.58]	1,[3.39 0.25 2.58]
B_5	2,[4.37 1.35 3.48]	2,[4.37 1.35 3.48]	2,[4.40 1.37 3.46]
B_6	2,[4.42 4.75 2.40]	2,[4.42 -0.25 2.40]	2,[4.40 -0.22 2.42]

Table 5.20: Basis vectors of random monoclinic data

In order to generate this unit cell, three atom types are used and two basis vectors of each type are generated. Since the data generation is random, none of the points is considered as the origin in the generated basis set. In order to make comparisons easier, *(0 0 0)* point were also chosen as origin in the program. Accordingly, basis vectors given in actual data and basis vectors obtained from the outputs should match without requiring any translations. Basis vectors obtained with ideal data matches to the actual basis vectors except one place, B_6. Instead obtaining [4.42 4.75 2.40] value as B_6, vector set obtained with the ideal data contains [4.42 -0.25 2.40] value. These two coordinates are identical coordinates since they differ by V_2. Accordingly, given output is scientifically correct and acceptable. However, this point does not lie inside the unit cell paralleloid defined by the primitive vectors. Accordingly preferred point would be [4.42 4.75 2.40]. This problem is caused by error handling mechanisms used in the algorithm for finding basis vectors. Small coordinate errors can cause a basis vector, which should be on a corner, on an edge or on a face of the unit cell to be left outside. Accordingly, in order to handle such cases, basis vector coordinates are allowed to have small negative values. Since the ideal data is also analyzed by assuming it can contain errors, [4.42 -0.25 2.40] value is allowed and preferred over [4.42 4.75 2.40] since it is closer to the origin of the unit cell. Since this situation is scientifically correct and the desired vector can be easily generated manually, this problem is not

quite important, thus results are perfectly acceptable.

Basis vectors obtained with noisy data are quite similar to the ones obtained with the ideal data. Basis vectors contain acceptable errors. Accordingly, for random monoclinic data acceptable results are obtained with both ideal data and noisy data.

11. Randomly Generated Triclinic Material: This data is randomly generated with predefined unit cell parameters. In order to generate the unit cell, a,b and c are set to 7.0,6.0 and 4.0 respectively and α,β and γ are set to 118, 81 and 75 degrees respectively. As the parameters imply, a relatively big triclinic unit cell is obtained. Volume of this unit cell is significantly bigger than any other unit cells previously analyzed. Accordingly, this generated data contain less information about the pattern of the crystal. Therefore, it can be considered as a good test data in order to observe the performance of framework with relatively poor input data.

	Actual Primitive Vectors	Ideal Data Output	Noisy Data Output
V_1	[7.00 0.00 0.00]	[7.00 0.00 0.00]	[6.99 0.00 0.00]
V_2	[1.55 5.80 0.00]	[1.55 5.80 0.00]	[1.55 5.79 0.00]
V_3	[0.63 -2.11 3.34]	[0.63 -2.11 3.34]	[0.63 -2.11 3.33]

Table 5.21: Primitive vectors of random triclinic data

As Table 5.21 imply primitive vectors are calculated accurately with ideal data and almost accurately with noisy data.

While generating this unit cell, 14 basis vectors are generated belonging to four different atom types. Accordingly, a big, more complex unit cell is obtained. During the analysis, similar to the test done with random monoclinic data, *(0 0 0)* point is selected as the origin in order to make comparisons easier. Obtained basis vectors matched to actual data except three vectors. Found B_3, B_8 and B_{12} vectors differ from the ones in the actual data. These differences are identical to the ones discussed in the random monoclinic data results part. As in the monoclinic data, these differences come from the error handling mechanisms of the finding primitive

	Actual Basis Vectors	Ideal Data Output	Noisy Data Output
B_1	0,[4.12 3.17 1.14]	0,[4.12 3.17 1.14]	0,[4.15 3.16 1.11]
B_2	0,[1.22 3.49 0.80]	0,[1.22 3.49 0.80]	0,[1.20 3.47 0.77]
B_3	0,[6.45 5.00 1.17]	0,[4.89 -0.80 1.17]	0,[4.87 -0.79 1.18]
B_4	1,[6.70 -0.06 0.37]	1,[6.70 -0.06 0.37]	1,[6.72 -0.04 0.39]
B_5	1,[6.75 2.05 2.07]	1,[6.75 2.05 2.07]	1,[6.75 2.04 2.06]
B_6	1,[4.84 -0.66 1.50]	1,[4.84 -0.66 1.50]	1,[4.82 -0.67 1.50]
B_7	2,[6.47 0.27 2.04]	2,[6.47 0.28 2.04]	2,[6.45 0.30 2.06]
B_8	2,[8.11 4.62 1.40]	2,[6.56 -1.18 1.40]	2,[6.55 -1.15 1.41]
B_9	2,[3.02 0.16 3.04]	2,[3.02 0.16 3.04]	2,[3.01 0.19 3.01]
B_{10}	2,[0.64 -1.00 1.77]	2,[0.64 -1.00 1.77]	2,[0.63 -1.01 1.74]
B_{11}	2,[7.84 4.31 0.70]	2,[7.84 4.31 0.70]	2,[7.82 4.29 0.69]
B_{12}	2,[1.99 -0.96 3.17]	2,[1.37 1.15 -0.17]	2,[1.34 1.13 -0.14]
B_{13}	3,[4.14 0.01 2.37]	3,[4.14 0.01 2.37]	3,[4.17 0.01 2.36]
B_{14}	4,[3.81 3.75 0.40]	4,[3.81 3.75 0.40]	4,[3.79 3.77 0.40]

Table 5.22: Basis vectors of random triclinic data

vectors algorithm. Accordingly, results are scientifically correct and can be considered accurate. Results obtained with the noisy data are quite close to the ones obtained with the ideal data. Accordingly, both results can be considered accurate.

As shown in the experimental results, framework is quite successful with ideal data. Almost all cases exact primitive vectors and basis vectors are found. For some cases, coordinate errors with error margin ± 0.01 are obtained. However those cases are quite rare and they possibly caused by the double precision sensitivity and the rounding. For some test data, some basis vectors were replaced with another identical basis vector, which lies outside the unit cell. This situation occurred due to the error handling mechanisms in the finding basis vectors algorithm. Considering the given output is still scientifically correct and the desired vectors can be obtained either manually or repeating the analysis with lower EPS values, such cases can be considered unimportant. Accordingly, with the ideal data obtained primitive vector and basis vector results can be considered accurate.

With noisy data, obtained results were quite similar to the ones that are obtained with the ideal data. Some distortions with small error margins are observed in the results. The level of distortions was acceptable considering the noise levels in the input data. Accordingly the framework can be considered succesfull with the noisy data.

5.2.2 Results of Intermediate Stages

As explained previously, there are several algorithms used during the analysis that give intermediate results. In this section, the results of these intermediate stages are analyzed, in order to examine each algorithms efficiency individually. The first algorithm to examine is the algorithm for grouping identical atoms. In this algorithm, initially each group is derived. Afterwards groups with small number of atoms are eliminated. The first column in Table 5.23, represents the total number of generated groups and the second column represents the number of groups left after the elimination of weak groups. The second and the third algorithms are, algorithm for extracting vectors algorithm and al algorithm for filtering out redundant vectors. The third column represents the number of vectors generated after the execution of the extracting vectors algorithm and the fourth column represents the number of vectors left after the redundant vectors are filtered out. Other algorithms do not give quantifiable intermediate results. However, commenting on the results of the calculating primitive vector alternatives algorithm, purify primitive vectors algorithm and the clustering algorithm is possible.

As shown in Table 5.23, first two columns contain the same values for all input types. This observation states that no groups were found whose atom count is smaller than some certain number, thus no elimination of groups took place. Those values are also equivalent to the number of atoms in the primitive unit cells for each input type. Accordingly, the grouping algorithm worked perfectly and produced desired results for all input data. The reason for not obtaining any weak groups is the quality of input data. The input data used in the testing phase are generated so that the crystal structure is complete in the volumes that

	Number of Generated Groups	Number of Reduced Groups	Number of Extracted Vectors	Number of Reduced Vectors
$NaCl$	2	2	42	21
$NaCl$ (noisy)	2	2	50	22
La_2O_3	4	4	26	13
La_2O_3 (noisy)	4	4	26	13
Cu_3Au	4	4	80	37
Cu_3Au (noisy)	4	4	80	37
PtS	4	4	292	121
PtS (noisy)	4	4	292	124
Al_3Ti	4	4	328	137
Al_3Ti (noisy)	4	4	343	156
Mg	2	2	280	121
Mg (noisy)	2	2	296	158
$CoSn$	6	6	98	43
$CoSn$ (noisy)	6	6	98	43
αHg	1	1	296	124
αHg (noisy)	1	1	296	219
TlF	2	2	774	331
TlF (noisy)	2	2	774	384
Monoclinic	6	6	21	10
Monoclinic (noisy)	6	6	22	10
Triclinic	14	14	18	8
Triclinic (noisy)	14	14	19	9

Table 5.23: The results of the intermediate stages for different materials

are used in the analysis. However, if this were not the case, then the first column of the table would contain higher values, and the second column were storing the same values. Accordingly, some weak groups would have been eliminated.

The third column contain values representing the number of vectors after extraction of vectors completed, and the fourth column contain values representing the number of vectors left after redundant vectors has been filtered out. These two columns were given in order to see the efficiency of the filtering out redundant vectors algorithm. The algorithm for filtering out redundant vectors eliminates the vectors that are an integer multiple of another extracted vector. Accordingly,

the aim of this algorithms is increasing the performance by reducing the number of vectors in the vector set that the analysis will continue. As the table implies, the numbers varies significantly for different input types. Accordingly, no general comment can be made. However, for most cases, the algorithm eliminates half of the extracted vectors. Another observation that can be made is, for all input types sufficient number of vectors are extracted. This observation is important due to an optimization, in the extracting vectors algorithm. While extracting vectors, if an extracted vectors length is higher than some respectively high predefined value, it is discarded. The reason for such elimination is, the long vectors are highly unlikely to be a primitive vector. Accordingly, eliminating quite long vectors will improve the performance without causing the data loss. The values in the third column show that the predefined value used to eliminate long vectors is selected appropriately.

After the redundant vectors are filtered out, every primitive vector candidate, which consist of vector triplets, are tested in order to see if they can be used as a primitive vector set. The maximum number of vectors, that can be used to form a vector triplet that is to be tested is limited to some number whose default value is 150. If the number of vectors left after the redundant vectors are filtered out, is higher than this value, then the shortest vectors are used. As shown in the table, some values in the fourth column is higher than the dafault value of this parameter, 150, and some of them are smaller. For the ones that are smaller, the algorithm for filtering out redundant vectors, clearly increases the performance significantly. However, for the other ones no performance improvement is achieved. Even though no performance increase is obtained for such cases, still filtering out redundant vectors is useful, since it eliminates the vectors, which cannot be in any primitive vector set. Executing this algorithm helps the smallest vectors that will be used to form primitive vector alternatives to test, being more meaningful. Accordingly, it helps the quality.

In general, the algorithm for extracting vectors and the algorithm for filtering out redundant vectors can be considered quite efficient. Since, the framework was able to calculate the primitive vectors accurately, for all input data, results of these two algorithms can be considered accurate too.

The algorithm for purifying primitive vectors works on the output of the calculating primitive vectors algorithm. The aim of this algorithm is sorting the primitive vector alternatives, so that the most desired forms of the primitive vectors comes to the top places. While running the analyzer program, after this algorithm completed, the program lists the primitive vector alternatives to the user. The user selects at least one alternative in order to continue to the analysis. Performance of the algorithm for purifying primitive vectors determines the order of the list. In general, with the ideal input data, the most desired primitive vector alternatives could be found at the top places. For some cases, small modifications are needed in order to obtain a better-looking structure, such as swapping two vectors or multiplying a vector by -1. However, desired primitive vector set could be obtained quite easily. Considering this analysis is designed to be a semi-automatic analysis, the algorithm for purifying primitive vectors could be considered quite successful with the ideal data. However, with noisy data, obtained results were not as satisfactory. As given in previous sections, in the presence of error, primitive vectors contain small distortions. Accordingly, a,b,c,α,β and γ parameters of the unit cells also contain some distortions. These parameters are used to determine which primitive vectors are more preferable by the user. For example, instead of random angle values, some certain angle values are preferred, such as 90 degrees or 120 degrees. In the presence of errors, since those parameters will contain distortions, determining the order of primitive vector alternatives also becomes harder. With the noisy data, the desired primitive vector alternative could not be found at the top places for all cases. For some cases, generally with high error levels, most desired form could not be found in the list at all. Accordingly, with noisy data, user occasionally needs to examine several alternatives to pick the best and he rarely needs to select a less desired alternative. However, for most cases , user was able to select a desired primitive vector alternative with a small effort, with the reasonable level of error. Accordingly the algorithm for purifying primitive vectors can be considered successful.

In general, the algorithm for calculating primitive vector alternatives is not an algorithm to judge. It simply applies a certain procedure on a given set and

returns the output. For almost all cases, it's output is certain and correct. However, for some cases, which the input data with high error margins were used, this algorithm may present some primitive vector alternatives, which actually should not be validated. For example, consider a simple cubic structure, where there is an atom placed at the coordinates (i, j, k) where i,j and k are all integers. Then one primitive vector alternative would be $V_1 = [1.0, 0.0, 0.0]$,$V_2 = [0.0, 1.0, 0.0]$ and $V_3 = [0.0, 0.0, 1.0]$. Consider a vector $Y = [1.0, 0.1, 0.0]$, exists in the set of vectors along with V_1. Ideally, this case should never happen, but if the error margins are high, it is possible to see such cases. Then the vector set, V_1,Y and V_3 would be tested. Since $V_2 = 10 \times (Y - V_1)$, this vector set can generate any vector that the vector set V_1,V_2 and V_3 can generate. Accordingly, such vector sets are considered as primitive vector alternatives. These cases are observed with the input data with high error margins and it is not a common case. Such kind of primitive vector alternatives could easily be distinguished and eliminated by the user since they define unit cells with quite low volumes. This type of errors is one of the reasons that the tool is designed to be semi-automated. Otherwise, this type of primitive vector sets might cause to obtain wrong results.

5.2.3 Results of the Space Group Identification Stage

The algorithm for indentifying the space group is affected from the errors significantly. In the presence of errors, accuracy of the space group information may drop significantly. Since the presence of errors is too crucial for this algorithm, three different tests are performed for each test material. In the first test, ideal data are used and EPS is set to a low value, 0.001. In the second test the ideal data and the default EPS values are used. Finally, in the third test the noisy data and default EPS values are used. Test results are given in Table 5.24.

For cubic structures; $NaCl$,La_2O_3,Cu_3Au, tetragonal structures; PtS,Al_3Ti and hexagonal structures; Mg,$CoSn$, the space groups are found correctly in all three tests. None of these structures are some distorted form of a higher symmetry structure. Accordingly, these materials are in the highest symmetry form that their structures allow. There is no way that these materials can be confused with

	Known Space Group Number	Ideal Data EPS=0.001	Ideal Data Default EPS	Noisy Data Default EPS
$NaCl$	225	225	225	225
La_2O_3	229	229	229	229
Cu_3Au	221	221	221	221
PtS	131	131	131	131
Al_3Ti	139	139	139	139
Mg	194	194	194	194
$CoSn$	191	191	191	191
αHg	166	166	166	166
TlF	69	69	138	138
Monoclinic	1	1	1	1
Triclinic	1	1	1	1

Table 5.24: The results of the space group identification stage

another higher symmetry structure even in the presence of reasonable level on noise. Thus, their space groups were identified accurately. The space group of the trigonal structure, αHg, has also been identified accurately in all three tests. However, it is necessary to stress out possible errors that can be observed with structures similar to αHg. The αHg structure can be considered as a distorted simple cubic structure. It's unit cell can be considered as a simple cube, whose diagonal length increased. Accordingly α, β and γ angles, which are the angles between each primitive vector pair, are all equal and smaller than 90 degrees. αHg structure has different forms, depending on such angles. According to the environmental properties such as the temperature and the pressure, those angles change their values within some range. Accordingly, αHg's unit cell might become quite close to simple cube unit cell. In the test data, those angles were set to 70 degrees. Accordingly, the unit cell parameters differ from simple cubes unit cell parameters more than the errors can cause. Therefore, the space group could be identified accurately in all three tests. However, if those angles were set to a value close to 90 degrees, such as 88 degrees, it is quite likely to obtain 221^{st} space group, which represents the simple cube structure in the second and in the third tests.

For monoclinic data, space group is found as 1, which is a triclinic space group. This might seem erroneous, but since basis vectors are randomly assigned inside the monoclinic unit cell, generated crystal structure show no symmetry what so ever. Accordingly, this structure belongs to first space group. While generating this data, the main concern was testing other algorithms of the framework, such as the algorithms for calculating finding primitive vectors and basis vectors. Accordingly, instead of seeking monoclinic symmetry, data is designed to present more complex crystal structure, which will test those algorithms of the framework more efficiently. The algorithm for identifying space group is mainly tested by using close structures such as Mg and $CoSn$ and all three structures in the cubic lattice class, etc. These structures symmetrically differs a small amount, as the close space group numbers indicate. Identifying those small differences constitutes the main challenge for the algorithm for identifying the space group. Accordingly not using a test data showing monoclinic symmetry, does not posses a problem. In general, the algorithm identified the space group of randomly generated test data, accurately in all three tests.

Some problems are observed with TlF structure. TlF is a distorted version of $NaCl$ structure. In other words, a high symmetry structure is converted into a low symmetry structure due to small distortions. In the second and in the third tests, default EPS values are used, which informs the *Analyzer* program about the possibility of errors in the input data. Accordingly, algorithm performs its analysis in the flexibility of given error margin. This flexibility leads TlF structure to match a higher symmetry group. Cases like TlF, are the main reason to strongly recommend using ideal data and setting the EPS parameter to a low value, in order to get reliable space group result.

In general, the space group results can be considered successful. For all test materials except TlF, correct results are obtained for all three tests. It would be safe to say, for materials, which are not some distorted form of a higher symmetry structure; results can be considered accurate even in the presence of reasonable level of noise. However, for materials which are some distorted forms of a higher symmetry structure, identified space group may not be reliable unless ideal data is used and the EPS parameter is set to a low value.

Results of this algorithm are characteristically different from the other algorithms used in analysis. For example, if the input were containing errors, then these errors affects the values of primitive vectors. The effect will be proportional to the error margin in the input data, thus with reasonable error margins, calculated primitive vectors can be sensitive enough. However, errors might cause a structure to match a completely irrelevant space group. Accordingly, result obtained with the presence of errors, might not give any meaningful information about the space group of the material at all. Accordingly, using ideal data and quite low EPS values are strongly recommended. However, if it is not possible, user should manually check other lower symmetry space groups that the tool also returns.

5.3 Performance Evaluation

In this section, the runtime performance of *Analyzer* program is discussed. In Table 5.25, execution times of each procedure, obtained with each test data are given. Runtime values are given in terms of milliseconds. These values were logged by the *Analyzer* program and they only represent the time passed during the execution of the specified algorithm. While testing runtime performances, *UserInterface* program was also running, in order to provide user interactions. Accordingly, this program might cause some fluctuations in the runtime performances, due to its CPU requirements. However, since this program was not performing heavy calculations, this effect should be small.

First two columns represent the number of atom coordinates in the input data. First row represents the number of all atoms in the input data while second row represents number of all atoms that are to be analyzed. The tool discards any atom, which does not lie inside some predefined volume. While generating the data, no such atoms were generated, in order to keep the performance evaluation phase simpler. Accordingly, no atoms in the input data were discarded.

The third column represents the time required to read the input data and

the fourth column represent the time spent while indexing the data by using the octree structure. Both values increases while the number of atoms in the input data increases. In general, reading the data requires much more time than indexing the data. Since IO operations are much slower than in-memory operations, this situation is normal.

The fifth column represents the time spent while grouping identical atoms. The grouping algorithm is based on comparing matching volumes, in order to determine each atoms group. Since during the tests, the default matching range parameter is used, the matching volumes of the materials that are denser in terms of the number of atoms per volume, contains more atoms. Accordingly, those materials would require more time trying to match atom's and group's matching volumes.

It is stated that one of the most dominant part of the grouping algorithm's runtime complexity is $O(GAMlog(M))$, where G represents the number of groups, A represents the number of atoms to process and M represents the number of atoms in each atoms matching volumes. Since the volumes of the crystal segments used in the analysis are equal for every test material, values in the first column are roughly proportional to the number of atoms per volume values of each material. Thus, they are proportional to the M value. As shown in Table 5.25, materials with high atom count values also have much higher execution times of the grouping algorithm. Since there are other parameters acting on the algorithm's runtime complexity, atom count and the execution time relation is not quite clear. In general, the grouping algorithm can be considered sufficiently fast, since the runtime performances are in reasonable limits. However, for some cases this algorithm can be one of the algorithms that have a major contribution to the overall execution time.

Sixth seventh and eight columns represent vector operations, the algorithm for extracting vectors, the algorithm for filtering out redundant vectors and the algorithm for calculating primitive vector alternatives respectively. The algorithm for extracting vectors turns out to be quite fast. It took less than 50 milliseconds for any material. The algorithm for filtering out redundant vectors was

even faster. For most cases, it's runtime was even immeasurable. It is mentioned that even though the worst-case complexity of the algorithm for filtering out redundant vectors is higher than the algorithm for extracting vectors, the runtime performance would be better. These runtime results support that prediction. The algorithm for calculating primitive vector alternatives is the dominant time consuming vector operation. Basically, it checks every vector triplets obtained from the vector set, in order to see if they could produce all other vectors in that vector set by their integer combinations. In order to reduce the tuntime complexity, an optimization is done by limiting the number of vectors that can be used to form a primitive vector alternative, to a relatively small number. Accordingly, this algorithm's runtime complexity is reduced to reasonable levels. In the table, eighth column show irregular values for different input data. Cross checking the runtime performance of this algorithm with the number of vectors remained after the redundant vectors are filtered out, which is given in Table 5.23, show that the input materials which require higher processing times for this algorithm also has higher number of vectors left after the algorithm for filtering out redundant vectors completed. In general, the algorithm for calculating primitive vector alternatives can be considered as the most time consuming operation after the grouping algorithm.

The ninth column shows the clustering times. For some input materials such as αHg, the clustering is quite fast. During the clustering algorithm, each atom of the first group defines a cluster. Since the αHg structure has only one group, it's clustering is trivial, thus quite fast. The clustering algorithm performs cross checks with all atoms belonging the group that is processing and each cluster, for every group other than the first one. Therefore, since having too many groups will reduce the number of clusters, it will decrease the processing time. For random triclinic data, which has 14 groups and $CoSn$ which has 6 groups, the clustering times are much smaller than other structures. In general, if the number of atoms to process is high, then the clustering time also tend to be high. However if the number of groups is high or equal to one, the clustering time drops significantly.

The tenth column represents the processing times of the identifying the space

group algorithm. This algorithm does not depend on the input material significantly. The runtime complexity of this algorithm is logarithmic time. Therefore, in theory, all inputs materials should have similar processing times. However, some optimizations change this situation. Finding one symmetry operation which is not supported by the crystal structure is sufficient to conclude that the crystal structure does not support the space group. Accordingly, checking the rest of the symmetry operations can be avoided. As the table implies, the inputs whose processing times are higher are generally higher symmetry structures. Since a higher symmetry structure tends to support more symmetry operation, it benefits from the short circuit property described above much less. Accordingly, it has higher processing times.

The eleventh column represents the runtime of the the finding basis vectors algorithm. Since this algorithm is quite similar to the clustering algorithm, the runtime results are quite similar to the clustering times.

In general, the dominant time-consuming algorithm was the grouping algorithm for most cases. The algorithm for calculating primitive vector alternatives and the clustering algorithm also demanded significant processing times for some input materials. However, it is not possible to generalize time requirements of the algorithms since they depend on inputs characteristics significantly. In general, the runtime performance of the whole analysis can be considered quite satisfactory. Whole analysis completed under 30 seconds for any input data. This value can be considered quite successful considering the algorithms were focused on the accuracy rather than the runtime performance.

Memory requirement of algorithms was quite reasonable throughout the tests. The *Analyzer* program used under 40 Megabytes of memory during the tests with any input data. The *UserInterface* program also required about 20 Megabytes of memory. The *VisualizationTool* program required about 10 Megabytes of memory. Accordingly, the overall system require at most 60 Megabytes of memory at anytime throughout it's usage. This amount is quite reasonable considering the memory capacities of recent computers. Accordingly, memory requirements can also be considered quite satisfactory.

5.4 Error Handling

The algorithms can perform quite well in the presence of the reasonable level of coordinate errors. The noisy data used in the tests were containing $\pm 0.03\mathring{A}$ coordinate errors at each axis. This level of noise is sufficient to cover almost all of the cases that this framework should handle. On the other hand, in this section, the proposed framework is tested with further levels of coordinate errors in order to see the algorithm's error tolerances. During the following tests, the calculated primitive vectors and the basis vectors are examined. Since it is stated that the space group information might be unreliable under the presence of errors, the space group results obtained from the following tests were not included into discussion. The $NaCl$ structure is used as the test structure. The error margins, $\pm 0.2\mathring{A}$, $\pm 0.4\mathring{A}$, $\pm 0.6\mathring{A}$, $\pm 0.8\mathring{A}$ and $\pm 1.0\mathring{A}$ are used to generate the test data. While performing the analysis, the error margin values used during the data generation phase are also given to the tool as the EPS value. If the default EPS value, which is equal to $0.05\mathring{A}$, were used, then no calculations could be done. Since the $NaCl$ structure is used, the target primitive vectors will be $V_1 = [2.830, 2.830, 0.000]$, $V_2 = [2.830, 0.000, 2.830]$ and $V_3 = [0.000, 2.830, 2.830]$ in $\mathring{A}$ unit. This form of the primitive vectors is the most desired one but there are also other valid primitive vector alternatives that the *Analyzer* program can find. The target basis vectors in the fractional coordinates should be $B_1 = Na, [0.0, 0.0, 0.0]$ and $B_2 = Cl, [0.5, 0.5, 0.5]$, considering the coordinates of a Na atom is selected as the origin. Similarly, there are also other valid basis vector sets. Each test performed with these error margins will be described individually.

1. **$\pm 0.2\mathring{A}$ Error Level:** With this error margin, the grouping algorithm produced 2 groups and eliminated none. The primitive vectors are calculated as $V_1 = [2.630, -2.862, 0.000]$, $V_2 = [2.758, 0.000, 2.773]$ and $V_3 = [0.000, -2.780, 2.803]$. These primitive vectors are in the same form with the target primitive vectors, but this form is also valid. The values can be considered sufficiently close to the target primitive vectors. A vector can contain $\pm 2EPS$ coordinate errors, which in this case is equal to ± 0.4. Since the highest coordinate error is found as 0.2, the primitive vector

results can be considered satisfactory. The basis vectors were found as, $B_1 = Na, [0.0, 0.0, 0.0]$ and $B_2 = Cl, [0.55, 0.52, 0.48]$. These values can also be considered correct within this error margin. Accordingly, the framework could be considered quite successful to handle this level of error.

2. $\pm$**0.4**$\mathring{A}$ **Error Level:** With this error margin, the grouping algorithm produced 2 groups and eliminated none. The primitive vectors are calculated as $V_1 = [2.555, 2.793, 0.000]$, $V_2 = [-0.170, 2.790, 2.655]$ and $V_3 = [2.690, 0.124, 2.778]$. These primitive vectors are in the form of target primitive vectors. The values can be considered sufficiently close to the target primitive vectors. The highest difference of the coordinate values is calculated as 0.275, which is quite small considering the possible difference of $\pm 2EPS$, which is 0.8. The basis vectors were found as $B_1 = Na, [0.0, 0.0, 0.0]$ and $B_2 = Cl, [0.55, 0.51, 0.55]$. These values can also be considered correct within this error margin. Accordingly, the framework handles this error level successfully.

3. $\pm$**0.6**$\mathring{A}$ **Error Level:** With this error margin, the grouping algorithm produced 2 groups and eliminated none. The primitive vectors are calculated as $V_1 = [2.778, 2.773, -0.102]$, $V_2 = [0.000, 2.845, 2.765]$ and $V_3 = [2.837, -0.056, 2.798]$. These primitive vectors are also in the form of the target primitive vectors. The values can be considered sufficiently close to the target primitive vectors. The highest difference of the coordinate values is calculated as 0.102, where it can be as much as $\pm 2EPS$, which is 1.2. The basis vectors were found as $B_1 = Na, [0.0, 0.0, 0.0]$ and $B_2 = Cl, [0.50, 0.42, 0.55]$. These values can also be considered correct within this error margin. Increased precision in the calculating primitive vectors part while increasing the error margin can be considered as an interesting coincidence. In the framework, some techniques are used to improve the quality of the vectors, such as averaging close vectors in order to reduce the effects of high errors. Accordingly, it is quite unlikely to calculate primitive vectors, which contain errors close to $2EPS$. However, no techniques that will work more precisely with the data containing higher error margin were used. Thus, this situation can be considered as just a coincidence.

4. $\pm$**0.8**$\mathring{A}$ **Error Level:** With this error margin, the grouping algorithm still produced 2 groups and eliminated none. However, in this test no remotely acceptable primitive vectors were calculated. The tool proposed several primitive vector alternatives but none of them were close to any valid form of the actual primitive vectors. Accordingly, calculating primitive vectors part failed with this error margin. After selecting some proposed primitive vector alternative, the tool proposed a list of atoms to choose as the origin. This means, the clustering could be done. Since the list were containing one Na and one Cl atoms whose relative distance show that these atoms were neighbors, the clustering can also be considered correct. However, the tool was unable to calculate any basis vectors. Accordingly, the framework failed within this error margin.

5. $\pm$**1.0**$\mathring{A}$ **Error Level:** The results obtained with this error margin is quite similar to the ones obtained with $\pm$0.8 error margin. Only the clustering part worked correctly. Calculated primitive vectors were unacceptable and no basis vectors could be found. Accordingly, the framework failed for this error margin too.

In general, the error tolerance of the framework can be considered quite well. The tests show that, even with the data containing $\pm$0.6 error margin, accurate results are obtained for $NaCl$ structure. Considering Na's radius is 1.16 and Cl's radius is 1.67 in $NaCl$ structure, $\pm$0.6 error level can be considered as a quite large value. It was recommended that the error margin should be smaller than 25% of the radius of the smallest atom in the input data. For $NaCl$ structure, analysis were successful with the error margins even more than 50% of the radius of the smallest atom in the input data. However, for more sensitive materials, such as crystals in hexagonal lattice class, 25% can be considered a safer upper limit. The primitive vectors of these crystals are closer to each other and the matching volumes of atoms in different groups are closer. Accordingly, the coordinate errors can cause problems more easily for those structures. However, with the error margin less than 25% of the radius of the smallest atom in the input data, the analysis would be successful.

Even though the framework has high error tolerance, it is recommended to use ideal data if possible. The main reason for that recommendation is, despite analysis can succeed with erroneous data, the primitive vectors and the basis vectors will contain small errors. These errors would in acceptable ranges, but they will not completely accurate. With ideal data, the tool can calculate exact values. Another reason to use the ideal data is the space group operations. The space group identification stage is effected from the errors significantly. Therefore, with erroneous data relying on obtained space group results might be deceptive. Accordingly, whenever possible, ideal data should be used.

5.5 Discussion

The results, the performance evaluation and the error analysis show that the proposed framework and the algorithms that are used in different stages are quite successful. Always correct results are obtained with ideal data. The runtime performances were also quite good. However, with noisy data some problems are observed. The basic problem was small deviations in the results. The primitive vectors and the basis vectors were containing small errors. These errors were in acceptable ranges considering the error margin of the input data. However, there were other inconveniences. As known, this work is designed to be a semi-automatic system. The user is expected to give some inputs throughout the analysis. After the analysis is started, the user is asked about two things, the primitive vector sets to continue the analysis with and the origin choice. The tool is designed so that it will present better alternatives of primitive vector sets in higher places. Actually, sole functionality of the algorithm for purifying primitive vector alternatives is ordering valid primitive vector alternatives according to users preference. With the ideal data, this algorithm was quite successful. Desired primitive vector alternatives could be found on one of the top places. However, with the noisy data, finding the desired vectors could be harder. Since during the tests, the desired vectors were manually selected, this difficulty was not reflected in the experimental results section. Another problem with the noisy data is the possibility of obtaining invalid primitive vector sets. As explained previously,

these invalid primitive vectors do not cause any significant problem since the user can identify them quite easily. However, they cause inconvenience to user. Finally, the last but may be the most important problem with the noisy data was the space group results. During the tests, the presence of noise, or informing the tool about the possibility of noise by using default EPS value, caused incorrect results with TlF structure. Even though all other structure's space groups were identified accurately with the noisy data, TlF example shows that the space group results might contain errors if ideal data and low EPS values were not used. Since TlF structure is a slightly distorted form of a higher symmetry structure, the algorithm for identifying space group's sensitivity to the errors is quite natural. Accordingly, the algorithm can be considered accurate. However, user should be aware of this sensitivity of this algorithm, while using the tool.

	total # of points	# of points to process	reading input	indexing input	grouping algorithm	extracting vectors	filtering out vectors	finding primitive vectors	clustering	finding space group	finding basis vectors
$NaCl$	3371	1331	130	0	40	10	0	20	10	141	0
n$NaCl$	3375	1331	151	10	40	10	0	20	20	30	0
La_2O_3	3375	2197	140	0	81	10	0	10	30	180	20
nLa_2O_3	3375	2197	140	10	80	10	0	10	30	191	20
Cu_3Au	7813	4631	181	20	300	10	10	80	120	71	70
nCu_3Au	7807	4626	200	20	371	10	0	90	130	30	70
PtS	36396	17598	500	121	12958	20	10	2484	1612	30	741
nPtS	36169	17594	541	120	12738	20	0	3616	2243	21	742
Al_3Ti	41513	20213	571	110	16494	30	0	3204	2083	20	792
nAl_3Ti	41509	20209	601	130	17195	20	0	3685	2674	90	821
Mg	17752	8954	311	30	2333	20	0	2273	641	20	361
nMg	17750	8953	331	40	2473	20	0	5008	541	10	351
$CoSn$	17107	8979	300	50	2043	10	0	181	340	50	160
n$CoSn$	17105	8784	301	50	2113	10	0	170	321	30	158
αHg	8865	4573	201	20	330	10	0	2053	0	20	0
nαHg	8863	4573	221	20	360	10	10	4817	0	10	0
TlF	46397	22707	621	140	14020	40	20	5729	5147	161	2033
nTlF	45995	22703	650	131	15001	40	10	3858	5168	20	2041
Monoclinic	3360	1824	160	10	50	10	0	0	10	20	0
nMonoclinic	3360	1825	161	10	50	10	0	10	10	20	0
Triclinic	6631	3460	221	30	280	10	0	40	30	30	10
nTriclinic	6620	3445	230	20	280	0	0	0	20	10	10

Table 5.25: The execution times of the stages of the framework for different materials

Chapter 6

Conclusion

The aim of this work is to extract pattern information from crystal structures by using atomic coordinates. Determining pattern information of crystal structures, such as primitive vectors, basis vectors and space group, has great importance in crystallography, chemistry and material sciences, since physical behavior of materials are directly related to those crystal parameters. This work provides a tool that can help scientists to identify and classify crystal structures. This work also provides a good crystal visualization tool that allows scientists to observe crystal structures in 3D environment.

Since this subject is not quite common, there were no directly related previous works. Accordingly, a new framework is proposed. Some approaches that are used in other areas such as 3D shape matching and pattern recognition are adapted to this problem and some new approaches are devised. There are two main challenges for proposing the framework. The first challange is obtaining accurate results while achieving reasonable runtime performance. Several critical decisions, such as limitations on inputs of intermediate stages, are taken for this purpose. Several computational optimizations are also proposed. The second challenge is handling erroneous input data. In order to overcome this problem the algorithm for finding basis vectors is rewritten and other algorithms are modified.

This work is tested with several data showing various characteristics. Test

data was generated by using real crystal parameters, with different error levels. Some randomly generated data are also included into tests. Experimental results and error analysis show that framework is capable to give accurate results even in the presence of reasonable levels of errors. Runtime performances were also quite satisfactory. Accordingly, proposed framework can be considered efficient and accurate.

6.1 Future Work

In this work, a tool, which extracts pattern information from crystal coordinates, is presented. The results were quite satisfactory with ideal data. On the other hand, small distortions were observed with noisy data. Since the levels of distortions were smaller than the error margin that errors in the input data could cause, the tool was considered successful with noisy data. Even though the levels of distortions were acceptable, the precision of the results can be improved. Several techniques that can improve the precision of output, were also proposed. Some of those techniques were not used in the implementation in order to achieve good runtime performance. On the other hand, since runtime performance turned out to be quite satisfactory, those techniques can be used to improve the precision of results with erroneous data.

This work can also be considered as a crystallographic visualization tool. The visualization tool provided in *BilKristal*, presents several features, such as defining multi cells, defining cut planes, using animations etc. Even though most of the required features are included into this tool, it still can be improved. Currently the visualization part of the *BilKristal* tool, can be considered as a practical tool that performs required operations in a simple way. However, various options, different graphical model choices etc. could improve the visualization tool.

The main technique used in crystallography is x-ray diffraction analysis. Accordingly, integration of this work with x-ray diffraction techniques can be quite useful for crystallographers. Since most of the data that crystallographers use are

based on x-ray diffraction analysis, such integration could be quite helpful. Together with the improvements described before, a quite powerful crystallography tool can be obtained.

Bibliography

[1] VL. Indenbom, BK. Vainshtein, VM. Fridkin. *Structure of Crystals, 3rd Edition*. Springer Verlag, Berlin, 1995.

[2] James R. Bowser. *Inorganic Chemistry*. Brooks/Cole, Pacific Grove, California, 1993.

[3] J. Conway, N. Sloane. *Sphere Packings, Lattices, and Groups*. Springer-Verlag, New York, 1993.

[4] Duncan McKie, Cristine McKie. *Essentials of Crystallography*. Blackwell Scientific Publications, 1986.

[5] WP. Reinhardt, E. Clementi, DL. Raimondi. Cpk Atomic Radii. *Phase Transitions: A Multinational Journal*, 38:2686–, 1963.

[6] J. M. Perez Mato, A. Kirov C. Capillas, S. Ivantchev, E. Kroumova, M. I. Aroyo and H. Wondratschek. Bilbao Crystallographic Server: Useful Databases and Tools for Phase-Transition Studies. *Phase Transitions: A Multinational Journal*, 76:155–70, 2003.

[7] Naval Research Laboratory Center for Computational Materials Science. Crystal lattice structures, 1995.

[8] Theo Hahn. *International Tables for Crystallography*. D. Reidel Pub. Co, Norwell, MA, U.S.A, 1987.

[9] Christopher Hammond. *Basics of Crystallography and Diffraction*. Oxford University Press, New York, 1997.

[10] Computational Crystallography Initiative. Computational Crystallography Toolbox, 2005.

[11] M.F.C. Ladd. *Symmetry in Molecules and Crystals*. Ellis Horword Ltd., West Sussex, 1989.

[12] CrystalMaker Software Limited. Crystalmaker software: Crystal and Molecular Structures Visualization and Diffraction, 2005.

[13] A. Lin. The MOE Crystal Builder, 2005.

[14] Szymon Rusinkiewicz, Michael Kazhdan, Thomas Funkhouser. Symmetry Descriptors and 3D Shape Matching. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 115–123, New York, NY, USA, 2004. ACM Press.

[15] Monique Pavel. *Fundamentals of Pattern Recognition*. Marcel Dekker Inc., New York, 1989.

[16] CT. Prewitt, RD. Shannon. *Acta Crystallographica*, 25:925–946, 1969.

[17] Roger Sayle. Rasmol.

[18] Dieter Schwarzenbach. *Crystallography*. John Wiley and Sons Ltd., West Sussex, England, 1996.

Appendix A

Data Structures

Data structures used in *Analyzer* program are listed as follows.

1. **Point structure:**

```
typedef struct point{
    int type;
    double x;
    double y;
    double z;
    struct point *next;
    struct point *next2;
    struct point *gnext;
}point;
```

2. **Octree node structure:**

```
typedef struct octreeNode{
    struct octreeNode *children[8];
    point *data;
    double xmax;
    double xmin;
```

```
    double ymax;
    double ymin;
    double zmax;
    double zmin;
    struct octreeNode *parent;
} octTreeNode;
```

3. **Group structure:**

```
typedef struct group{
    int atomtype;
    point *ref;
    point *idlist;
    point *list;
    int idatcnt;
    int atomcnt;
    struct group *next;
}group;
```

4. **Vector structure:**

```
typedef struct VECTOR{
    double x;
    double y;
    double z;
}VECTOR;
```

5. **Primitive vector set structure:**

```
typedef struct PRI_VECTS{
    VECTOR v1;
    VECTOR v2;
    VECTOR v3;
    double V1L;
    double V2L;
```

```
    double V3L;
    double VLAVG;
    double volume;
    double alpha;
    double beta;
    double gamma;
    double AngleValue;
    int VectorValue;
    struct PRI_VECTS *next;
}PRI_VECTS;
```

6. **Cluster structure:**

```
typedef struct cluster{
    double cx;
    double cy;
    double cz;
    point *newpnt;
    int curnumofpnts;
    point *list;
    struct cluster *prev;
    struct cluster *next;
}cluster;
```

7. **Basis point structure:**

```
typedef struct BasisPoint{
    point bp;
    double fx;
    double fy;
    double fz;
} BasisPoint;
```

8. **Space group structure:**

```
typedef struct SpaceGroup{
    int groupNumber;
    char name[256];
    int numOfOperations;
    int numOfTotalOperations;
    double *RMatrix;
    double *TVector;
}SpaceGroup;
```

Data structures used in *VisualizationTool* program are listed as follows.

1. **Point structure:**

```
typedef struct point{
    double x;
    double y;
    double z;
    double radius;
    int R;
    int G;
    int B;
}point;
```

2. **Stick structure:**

```
typedef struct stick{
    point *P1;
    point *P2;
    struct stick *next;
}stick;
```

3. **Cut Plane structure:**

```
typedef struct cutPlane{
```

```
        double i;
        double j;
        double k;
        int operation;
        struct cutPlane *next;
    }point;
```