Dynamic Classification of Defect Structures in Molecular Dynamics Simulation Data

S. Mehta* S. Barr† T. Choy† H. Yang* S. Parthasarathy* R. Machiraju*
J. Wilkins†
*Department of Computer Science and Engineering, Ohio State University
†Department of Physics, Ohio State University
Contact: {mehtas,srini,raghu}@cse.ohio-state.edu

October 5, 2004

Abstract

In this application paper we explore techniques to classify anomalous structures (defects) in data generated from ab-initio Molecular Dynamics (MD) simulations of Silicon (Si) atom systems. These systems are studied to understand the processes behind the formation of various defects as they have a profound impact on the electrical and mechanical properties of Silicon. In our prior work we presented techniques for defect detection [11, 12, 14]. Here, we present a two-step dynamic classifier to classify the defects. The first step uses upto third-order shape moments to provide a smaller set of candidate defect classes. The second step assigns the correct class to the defect structure by considering the actual spatial positions of the individual atoms. The dynamic classifier is robust and scalable in the size of the atom systems. Each phase is immune to noise, which is characterized after a study of the simulation data. We also validate the proposed solutions by using a physical model and properties of lattices. We demonstrate the efficacy and correctness of our approach on several large datasets. Our approach is able to recognize previously seen defects and also identify new defects in real time.

1 Introduction

Traditionally, the focus in the computational sciences has been on developing algorithms, implementations, and enabling tools to facilitate large scale realistic simulations of physical processes and phenomenon. However, as simulations become more detailed and realistic, and implementations more efficient, scientists are finding that analyzing the data produced by these simulations is a non-trivial task. Dataset size, providing reasonable response time, and modeling the underlying scientific phenomenon during the analysis are some of the critical challenges that need to be addressed.

In this paper we present a framework that addresses these challenges for mining datasets produced by Molecular Dynamics (MD) simulations to study the evolution of defect structures in materials. As component size decreases, a defect - any deviation from the perfectly ordered crystal lattice - in a semiconductor assumes ever greater significance. These defects are often created by introducing extra atom(s) in the Silicon lattice during ion implantation for device fabrication. Such defects can aggregate to form larger extended defects, which can significantly affect device performance in an undesirable fashion.

Simulating defect dynamics can potentially help scientists understand how defects evolve over time and how aggregated/extended defects are formed. Some of these defects are stable over a period of time while other are short-lived. Efficient, automated or semi-automated analysis techniques can help simplify the task of wading through a pool of data and help quickly identify important rules governing defect evolution, interactions and aggregation. The key challenges are: i) to detect defects; ii) to characterize and classify both new and previously seen defects accurately; iii) to capture the evolution and transitioning behavior of defects; and iv) to identify the rules that govern defect interactions and aggregation. Manual analysis of these simulations is a very cumbersome process. It takes a domain expert more than eight hours to manually analyze a very small simulation of 8000 time frames. Therefore, a systemic challenge is to develop an automated, scalable and incremental algorithmic framework so that the proposed techniques can support in-vivo analysis in real time.

In our previous work [11, 12, 14], we presented algorithms to address the first challenge. Here we address the second challenge coupled with the systemic challenge outlined above. The design tenets include not only accuracy and execution time but also both statistical and physical validation of the proposed models. We also present preliminary results

to show that our approach can aid in handling the third challenge.
The main contributions of our application case study paper are:

1. We develop a two-step incremental classifier that classifies both existing and new defects (generates a new class label).

2. We validate each step of our 2-step classifier theoretically, relying on both physical and statistical models.

3. We validate our approach on large (greater than 4GB) real MD simulation datasets and demonstrate both the exceptional accuracy and efficiency of the proposed framework.

4. We present initial results which show that our approach can be used to capture defect evolution and to generate labeled defect trajectories.

Our paper is structured in the following manner. Section 2 discusses the basic terminology of MD and related work. An overview of our framework is provided in Section 3. We present our algorithm in Section 4. Results on large simulation datasets are presented in Section 5. Finally, we conclude and discuss directions for future work in Section 6.

2 Background and Related Work

2.1 Background: In this section, we first define basic terms that are used throughout this article. Later, we describe pertinent related work. A **lattice** is an arrangement of points or particles or objects in a regular periodic pattern in three dimensions. The elemental structure that is replicated in a lattice is known as a **unit cell**. Now, consider adding a single atom to the lattice. This extra atom disturbs the geometric structure of lattice. This disturbance, comprised of atoms which deviate from the regular geometry of lattice is referred to as the **defect** structure. Defects created by adding an extra atom are known as single-interstitial defects. Similarly one can define **di-** and **tri-interstitial** defects by adding two or three single interstitial defects in a lattice respectively. Figure 1(a) shows a Si bulk lattice with a certain unit cell shaded differently (black). Figure 1(b) shows another lattice with a single interstitial defect. Figure 1(c) depicts two interstitials: in the lower left and upper right corners respectively of a 512-atom lattice. The different shades again represent separate and distinct defects.

We use the Object-Oriented High Performance Multiscale Materials Simulator (OHMMS) that some of us developed (primarily led by co-author Wilkins) [4] as our workhorse. The equation of motion as described by Newton's second law is used to determine atom locations. While the exact forces must be derived from quantum mechanical studies and computations, these classical equations serve as a suitable approximation.

2.2 Related Work: Traditionally, physicists have used ground energy and electrostatic potential to find defects in lattice. For example ab-initio methods are used to locate interstitial defects in a Si lattice [2, 19]. These methods exploit anomalies in the energy/potential fields available at all points in the lattice . However, the calculation and analysis of these energies/potential is very time consuming. The most pertinent work that is closely related to our own work employs the method of Common Neighbor Analysis (CNA) [3, 8]. CNA attempts to glean crystallization structure in lattices and uses the number of neighbors of each atom to determine the structure sought. However, it should be noted that the distribution of neighbors alone cannot characterize the defects, especially at high temperatures. Related to our work is the large body of work in biomolecular structure analysis [1, 10, 16, 20]. In these techniques, the data is often abstracted as graphs and transactions and subsequently mined. However, such an abstraction often does not exploit and explain many of the inherent spatial and dynamical properties of data that we are interested in. Moreover, while some of these techniques [17, 20], deal with noise in the data, the noise handling capabilities are limited to smoothing out uncorrelated and small changes in spatial location of atoms. Within the context of MD data, noise can also change the number of defect atoms detected, for essentially the same defect structure at different time frames. None of the methods in the biological data mining literature deal with this uncertainty. Matching of two structures has drawn a lot of attention in recent past. Zhang et al.[22], propose a protein matching algorithm that is rotation and translation invariant. This method relies on the shape of the point cloud and it works well for proteins given the relatively large number of atoms; the presence of a few extra atoms does not change the shape of point cloud. A potential match is not stymied by the presence of extra atoms. However, in MD simulation data, we are interested in anomalous structures which can be as small as just six atoms. Extra atoms that may be included in a defect given the thermal noise will skew a match significantly even if the two defects differ by one atom. Geometric hashing, an approach that was originally developed in the robotics and vision communities [21], has found favor in the biomolecular structure analysis community [15, 20]. Rotation and translations are well handled in this approach. The main drawback of geometric hashing is that it is very expensive because an exhaustive search for optimal basis vector is required. A more detailed discussion on various shape matching algorithms can be found in the survey paper by Veltamp and Hagedoorn [18]. We use statistical moments to represent the shape of a defect for initial pruning. The seminal work by Hu [13] described a set of seven moments which

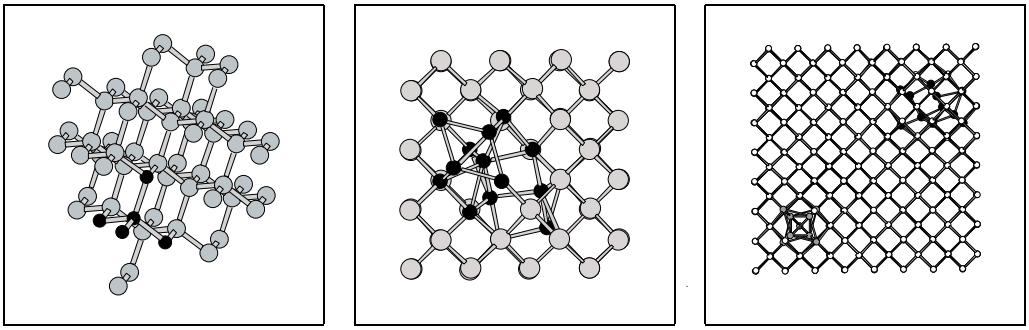


Figure 1: (a)Original lattice with unit cell marked (b)Lattice with one interstitial defect (c)Lattice with two interstitial defects

captures many of the features of a two-dimensional binary images. In recent work, Galvez [7] proposed an algorithm which uses shape moments to characterize 3D structures using moments of its two-dimensional contours. However, owing to relatively small number of atoms present in a typical defect, the contours or the corresponding implicit 3D surface are impossible to obtain with accuracy.

3 Dynamic Classification Framework

Figure 2 shows our framework for MD simulation data analysis. The framework is divided into 3 phases. *Phase 1 - Defection Detection*: this phase detects, spatially clusters (or segments) defects and handles periodic boundary conditions. Detailed explanation on this phase is given in our earlier work [11, 14, 12]. *Phase 2 - Dynamic Classification*: this phase classifies each defect found in *Phase 1*. This phase consists of three major components:

1. Generating a feature vector for each detected defect. This feature vector is composed of weighted statistical moments.
2. Pruning the defect classes based on the feature vector. This step provides a smaller subset of defect classes to which an unlabeled defect can potentially belong to.
3. An exact matching algorithm that assigns the correct class label to the defect. This steps takes into account the spatial position of the atoms. The defect is assigned a class label if it matches with any of the previously seen defects, otherwise the defect is considered new.

Phase 2 maintains 3 databases for all detected defects. Section 4 gives detailed information regarding these three databases and their update strategies.
The framework is made robust by modeling noise in both *Phase 1* and *Phase 2*. Our noise characterization models the aggregate movement of the defect structure and the arrangement of the atoms (in terms of neighboring bonds). Our framework can be deployed to operate in a streaming fashion. This is important since it enables us to naturally handle data in a continuous fashion while a simulation is in progress. *Phase 1* handles the entire frame and detects all the defects. Each defect is then pipelined into *Phase 2*. Thus, we are able to incrementally detect and classify detected defects while consistently updating the database in real time.
Phase 3 - Knowledge Mining: this phase uses the databases generated by *Phase 2*. These databases store the information about all the defects in a given simulation. These databases can be used to track and generate the trajectories of the defects, which can assist us to better understand the defect evolution process. Additionally, various data mining algorithms can be applied on the databases. Mining spatial patterns within one simulation can aid in understanding the interactions among defects. Finding frequent patterns across multiple simulations can help to predict defect evolution. In this paper, we describe *Phase 2* in detail. We also show some initial results for *Phase3*.

4 Algorithm

Our previous work [11, 12, 14] describes the defect detection phase in detail. Every atom in the lattice is labeled either as a bulk atom or as a defect atom. However, upon further evaluation we found that this binary labeling is not well suited for robust classification of defect structures. Therefore, we propose to divide the atoms into three classes based on their membership or proximity to a defect. We also validate the correctness of this taxonomy by using a physical model. Our taxonomy is:

- **Core-Bulk Atoms (CB):** The atoms which conform to the set of rules defined by the unit cell are *bulk* or non-defect atoms. Bulk atoms which are connected exclusively to other bulk atoms are labeled as core bulk atoms.

- **Core-Defect Atoms (CD):** The atoms which do not conform to the set of rules are *defect* atoms. All the defect atoms which are connected to more defect atoms than bulk atoms are labeled as core defect atoms. These

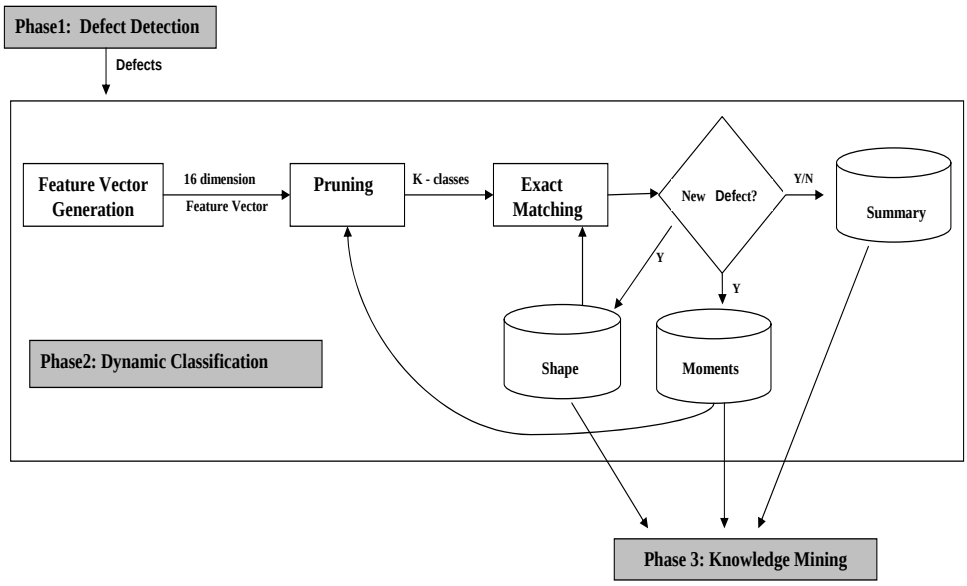


Figure 2: Defect Detection and Classification Framework

atoms dominate the shape and properties of a defect.

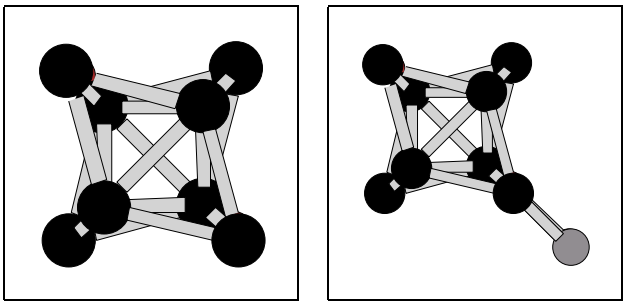
- **Interface Atoms (I):** These atoms form the boundary between a core bulk atom and a core defect atom. These atoms fail to conform the prescribed set of rules by a small marginally (i.e. the thresholds for bond lengths and angles, are violated in a marginal way) thus are marked as defect atoms; however, the majority of their nearest neighbors are core bulk. Thus, they form a ring between core bulk atoms and core defect atoms. Presence or absence of these atoms can considerably change the shape of a defect, which makes matching of defect structures very difficult.

Figure 4(a) illustrates all three types of atoms. The black atoms belong to core defect, white atoms form core bulk and the gray atoms are interface atoms. We next, justify this taxonomy.

Physical Validation: A lattice system can be represented by the Mechanical Molecular Model as follows: atoms are represented by spheres, and bonds by springs connecting these spheres. The energy of an atom in the lattice system is calculated by the following equation:

$$E_{total} = E_{length} + E_{angle} + E_{interactions}$$

where E_{length} and E_{angle} are the energies due to bond stretching and angle bending respectively. Also, each atom in the lattice interacts with every other atom. $E_{interactions}$ accounts for energy generated by these interactions. However, we can ignore $E_{interactions}$ for the MD datasets because OHHMS only considers the effect of first neighbors of



(a) (b))

Figure 3: (a) Original Defect (b) Detected Defect with extra atoms

an atom while solving MD equations. The physics behind springs can be used to calculate these energies. Hooke's law for springs states that the force exerted by a spring is proportional to the distance by which it is stretched. The energy of a spring can be derived by using relationship between energy and force. The energy is calculated by the following equation:

$$E = \frac{1}{2} K \delta^2 \tag{4.1}$$

where K is the spring constant and δ is the distance by which a spring is stretched from its uncompressed state.
E_{length} and E_{angle} for each atom can be computed by using appropriate spring constants, K_{length} and K_{angle} respectively. Information about the ideal bond length and bond angle present in the existing literature can be used

for uncompressed spring state. Drexer [5] lists the values for K_{length} and K_{angle} as 185 $Newton/meter$ and 0.35 $Newton/radian$ respectively for the Silicon lattice. For each atom we find the bond lengths and bond angles it forms with its first neighbors and then find the δs. Core bulk atoms deviate very little from the ideal bond angle and bond length, therefore their corresponding δs should be very low; whereas for core defect atoms the δs should be high. Since the energy is directly proportional to δ^2, core bulk atoms should have low energy whereas core defect atoms should have high energy.

To validate our taxonomy, we sampled 1400 frames from different simulations and calculated the energy for each atom in the lattice. Figure 4(b) shows the distribution of energy. It is clear from the distribution that the majority of the atoms have very low energy ($\in [0, 0.2]$). These atoms are core bulk atoms. The core defect atoms have very high energy (≥ 1.2). All the atoms which lie between low and high energy levels are interface atoms. Thus, this physical model clearly validates our taxonomy of atoms. Therefore we refine our original binary labeling [11] of individual atoms by further dividing defect atoms into core defect atoms and interface atoms.

Before describing the classification method, we discuss the challenges which need to be addressed to build a robust and efficient classifier for MD data. We list each of them and describe how they are addressed within the context of the proposed algorithm. For each proposed solution we also provide the physical validation using the Mechanical Molecular Model and properties of the lattice system.

4.1 Challenges and Proposed Solutions

4.1.1 Thermal Noise: Thermal agitations can cause atoms to change their spatial positions. Such changes can potentially have two kinds of effect on the defect structures:

1. The precise location of the atoms and their inter-pair distances will not be exactly the same from frame to frame. Thus the classification method should be tolerant to small deviations in the spatial positions.

2. The change in the spatial positions can also force a previously labeled bulk atom to violate the rules and be labeled as a defect atom (and vice versa) in the next time frame. Therefore the number of atoms in a defect can change, which in turn changes the overall shape of the defect structure which makes the classification task more difficult.

To address the first problem we consider a data driven approach to derive noise thresholds. From our study of physics, we know that the movement of each atom is influenced by the position and number of its neighbors. To model this behavior we define a random variable D_i:

$$D_i = \frac{1}{F+1}(M_i + \sum_{j=1}^{F} M_j)$$

where M_i is the movement of atom *i* between two consecutive time steps, having *F* first neighbors within a distance of $2.6\mathring{A}$ (bond length for *Si*), and $i \in [1, N]$, N being the total number of atoms in the lattice. We empirically observe that D_i can be effectively approximated by using a normal distribution with parameters, μ_{noise} and σ_{noise} (the average mean and standard deviation of all D_is). We found μ_{noise} to be very close to 0 (which is expected because a given atom cannot move very far from its original location between two consecutive time frames). The parameter σ_{noise} is used to model the effect of noise in the defect classification algorithm. From a set of randomly selected 4500 frames, we found the value of σ_{noise} to be $0.19\mathring{A}$.

Physical Validation: The noise threshold can be validated by using the Mechanical Molecular Model. The bond energy B of Si-Si bond is 52 $Kcal/mol$, which is the amount of energy needed to break a Si-Si bond. Using Equation 4.1 we can compute the maximum distance a Si atom can be displaced before the bond is broken. Essentially we solve the following equation for the value of δ :

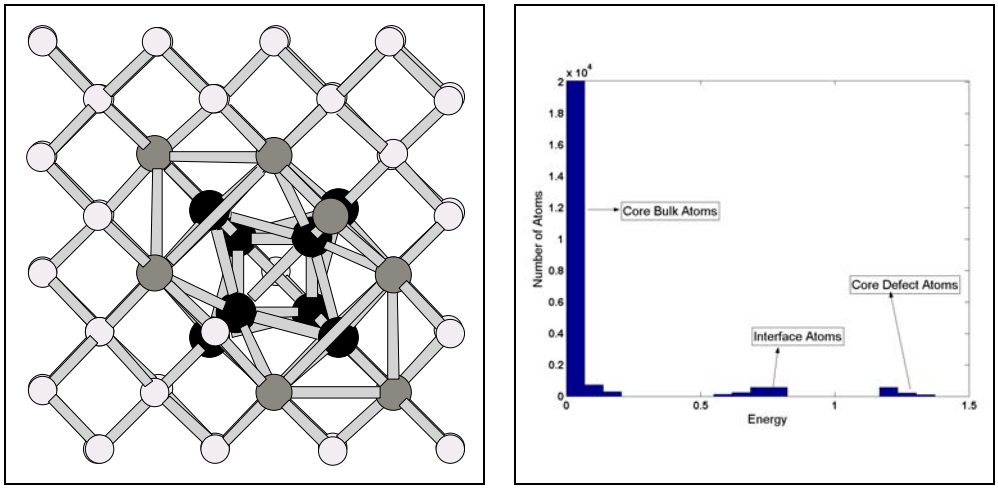
$$B \leq \frac{1}{2} K \delta^2$$

By substituting the values of K and B we found the value of δ to be $0.2\mathring{A}$ which implies that two bonded atoms cannot be moved more than $0.2\mathring{A}$ apart without breaking the bond between them. Thus, the empirically observed value is very close to the theoretical value given by the physical model.

To solve the second problem posed by thermal noise, we propose a weighting mechanism. The weighting mechanism is based on the following two observations:

- *Observation 1:* In two consecutive time frames the core defect atoms cannot change considerably.
- *Observation 2:* Interface atoms can make a transition from bulk to defect (and vice-versa) very quickly.

Figure 3(a) and Figure 3(b) show the defect detected from two different frames after applying local operators. The defect in Figure 3(b) has extra atoms (interface) but the core defect (black atoms) remains unchanged. Therefore, a *weighting mechanism* is proposed to reduce the influence of interface atoms relative to that of core atoms within a defect structure. Essentially, the weight assigned to each atom in a given defect is proportional to the number of its first neighbors in the defect structure. Thus core defect atoms contribute more to defect classification than interface atoms. These weights are also used for handling translations (described below) and for computing the feature vector (weighted moments).



(a) (b)

Figure 4: (a) Taxonomy of atoms (b) Energy Plot

Physical Validation: *Observation 1* can be explained as follows. Each atom in the lattice interacts most with its first neighbors. The more the number of first neighbors of an atom, the more connected it is and hence the more restricted its movement. The core defect atoms have high connectivity with other defect atoms, which makes it more difficult for them to move very far in a short period of time.
Observation 2 can be explained as follows. Interface atoms labeled as defect (or bulk), usually fail (or conform to) the set of rules by a small margin. A very small variation in their spatial locations can change their labels. These interface atoms, however, are very loosely connected to the core defect atoms. Most of their first neighbors are core bulk atoms.
To summarize, over a period of time core defect atoms will change considerably less than interface atoms. Therefore more emphasis (weight) should be given to core defect atoms while matching two defect structures. This is precisely what our weighting mechanism does.

4.1.2 Translational and Rotational Invariance:

Translations and rotations pose another problem in defect classification. The same defect can occur in different positions and orientations in the lattice. To classify a defect correctly, translations and rotations should be resolved before assigning the class. We next present our approach to attain translational and rotational invariance.

Feature Vector Generation: We describe the shape of a defect by using statistical moments. We chose to use all first, second and third order moments. Third order moments capture skewness in defects. To account for the interface atoms we calculate weighted moments instead of simple moments. *(Recall that the weighting mechanism assigns high weights to core defect atoms and low weights to interface atoms)*. The feature vector comprising of weighted moments of a defect is calculated as :

$$W_{mnp} = \frac{1}{\sum_{j=1}^{N} w_j} \sum_{i=1}^{N} w_i * D_{ix}^m * D_{iy}^n * D_{iz}^p$$

$$where\ m+n+p \leq 3$$

where D_{ix} is the *x*-ordinate of i^{th} atom of defect *D*. An important property of this feature vector is that it is translation invariant if the weighted center of mass (given by the first three weighted moments) is translated to zero.
Since all the rotations in the lattice are symmetry operations (see below for an explanation of symmetry operation), rotational ambiguity can be resolved easily by applying the appropriate permutation on the feature vector. For example, if the defect is rotated by 180 degrees across the X-plane (a mirror transform), all the moments involving an odd power of the X-component will change sign. In a similar fashion, all the rotations can be resolved by checking the pre-defined permutations of original moments. There are a total of 3 first-order moments, 6 second-order moments and 10 third-order moments. Of these, since the center of mass is translated to the origin (to deal with translations), W_{100}, W_{010} and W_{001} are all zero. Therefore we have a 16-dimensional feature vector represented by D_w.

Physical Validation: Interface atoms change the shape of the defect, which can change the center of mass considerably. Therefore using center of mass without the weights can assign a new class label to a defect even if the core defect is not new. Thus core defect atoms should contribute more

towards the calculation of the center of mass. A lattice cannot be rotated in arbitrary directions. The only rotations possible in the lattice are those which *carry the lattice onto itself*. This means that after rotation each atom of the lattice is exactly at a position occupied by an atom prior to the rotation. These rotations are known as *symmetry operations* [9]. Under this constraint, only a finite number of rotations are possible in lattices. For example, in the Si lattice system only 24 types of rotations are possible.

4.1.3 Shape Based Classification: While matching two defect structures, the classifier should take into account the positions of all individual atoms in the defects. This atom-to-atom matching is relatively expensive. Furthermore because of large number of defect classes present in simulation datasets, it would be unrealistic to carry out such an atom-to-atom matching for all classes at each and every time step. Therefore a scheme is needed to effectively reduce the number of candidate classes on which an exhaustive atom-to-atom matching is performed.
We address this challenge by adopting a two step classification process. The first step uses weighted moments to find a smaller subset of defect classes to which the unlabeled defect can potentially belong and passes it to the next step. Weighted moments (feature vector) are used because moments are known to capture the overall shape of an object very well [13]. The second step then finds the closest class by taking into account the positions of the atoms and their arrangement in three dimensional space. In essence, both steps use the information about the shape of a defect. The first step uses the high level information about defect structure whereas the next step refines it by matching individual atoms. We achieve the desired efficiency because the first step is computationally very cheap and reduces the search space considerably for the next step. Experimental results to corroborate this are shown in Section 5.

Physical Validation: The majority of the physical properties of a defects are governed by its shape. Most of the stable defects seen so far have a very compact shape. Unstable structures tend to re-organize the atoms to form such a compact structure. The movement of the defect in the lattice is also governed by the shape of the defect.

4.1.4 Emergence of new defect classes: The underlying motivation of our effort is to discover information which can assist scientists to better understand the physics behind defect evolution, ideally in real time. This defect evolution can result in new defect classes which are not in existing literature. This requires the classification process to be dynamic [6]. The classifier should be dynamic in the classical sense, as in new streaming data elements can be classified, but should also be dynamic in the sense that new classes(defects) if discovered can be added to the classifier model in real time. The new defect should be available when the next frame is processed.
We next present our two-step classification process which integrates all the proposed solutions to the above-mentioned challenges.

4.2 Two Step Classification Algorithm: This phase classifies the defect(s) detected in $Phase1$. Given a defect **D**, the goal is to find the type **T** of this defect. If **D** does not match any of the previously seen defects in the simulation, it is labeled as a new defect and stored in the databases $\mathbf{ID}_{shape}$ and $\mathbf{ID}_{moment}$, where **ID** is a unique simulation identification number, $\mathbf{ID}_{shape}$stores the actual three dimensional co-ordinates and $\mathbf{ID}_{moment}$ stores the weighted central moments (feature vector) of the defect structure. These databases store all the unique defects detected in the current simulation. The label of a new defect is of the form *d*efect_i_j, indicating that the new defect is the j^{th} defect in the i^{th} frame of the simulation. If **D** is not new then a pointer to the defect class which closely matches **D** is stored. Besides these two databases a summary file is generated which stores names of all detected defects in the simulation along with corresponding frame numbers. We now proceed to describe the two steps of our classifier in detail.

4.2.1 Step 1 - Feature Vector based Pruning: We use a variant of the KNN classifier for this task. The value of *K* is not fixed: instead, it is determined dynamically for each defect. Given the feature vector (D_W) of a defect *D*, we compute and sort the distances between D_W and ID_{M_i}, where ID_{M_i} is the moment vector of the i^{th} defect in ID_M. All classes having distances less than an empirically-derived threshold are chosen as candidate classes. **Step 2**, then, works on these K classes only. If no class can be selected, D is considered as a new defect. Databases ID_{shape} and ID_{moment} are updated immediately, so that D is available when the next frame is processed.
In a similar fashion, one can use Naive Bayes and Voting based classifiers. Like the *KNN* classifier, these classifiers also provide metrics which can be used to select the top K candidate classes. More specifically, a Naive Bayes classifier provides the probabilities of a feature vector belonging to each class, and a voting based classifier gives the number of votes for each class. The top K classes can then be chosen based on probabilities and votes. We chose VFI as our voting based classifier. As for other types of classifiers, such as the decision tree-based ones, it is not trivial to pick K candidate classes, therefore they are not considered in this work.
From the three applicable classifiers, the *KNN* classifier is chosen because it gives the highest classification accuracy, as described in Section 5. Besides its high accuracy, the *KNN* classifier is incremental in nature. In other words, there is

no need to re-build the classification model from scratch if a new class is discovered. In contrast, Naive Bayes and VFI will require the classification model to be re-built every time a new class is discovered.

The *K* candidate classes are passed to **Step 2**. The representative shapes of these *K* classes are matched using an exact shape matching algorithm based on the Largest Common Substructure (LCS). Next, we explain the main steps of our exact matching approach.

4.2.2 Step 2 - Largest Common Substructure based algorithm: Assume, A is a defect of unknown type and B is the defect representing one of the candidate classes from **Step 1**. The defects are mean centered and the rotation is resolved. We next describe all the steps of the **LCS** algorithm in detail.

- **Atom Pairs Formation:** The defects are sorted w.r.t. their x-ordinate. Two atoms *i* and *j* in defect A form an atom pair A_{ij} if distance(A_i,A_j) $\leq$ *bond length*. This step uses the information about neighbors and connectivity calculated in *Phase 1*. These atom pairs are calculated for both defects. For each atom pair A_{ij}, we store the projection onto X,Y and the Z-axes represented by A_{ijx}, A_{ijy} and A_{ijz} respectively.

- **Find matching Pairs:** For each pair A_{ij} we find all pairs B_{kl} such that

$$|A_{ijx} - B_{klx}| \leq \sigma_{noise}$$
$$|A_{ijy} - B_{kly}| \leq \sigma_{noise}$$
$$|A_{ijz} - B_{klz}| \leq \sigma_{noise}$$

  where threshold σ_{noise} is obtained as explained in Section 4.1.1.

  We represent this equality of atom pairs as $A_{ij} \leftrightarrow B_{kl}$, which implies that the length and orientation of the bond formed by atoms i and j of defect A is similar to the bond formed by atoms k and l of defect B.

  By comparing each projection separately, we intrinsically take care of both: bond length and orientation.

- **Find Largest Common Substructure (LCS):** The rules generated in the previous step are used to find the largest common substructure between two defects. We use a region growing based approach to find LCS.

  The pseudo code for finding LCS is shown in Figure 5.

  Before explaining each step in detail, we define the notion of **compatible substructures:**

  Two substructures U and V are considered compatible w.r.t. the rule $A_{ij} \leftrightarrow B_{kl}$, if the last atom added to U is the i^{th} atom of defect A and the last atom added to V is the k^{th} atom of defect B.

  Being compatible implies that the two substructures have the same number of atoms and the orientation of atoms (which defines shape) is approximately same (within noise thresholds).

  The algorithm starts by finding all **compatible substructures** U and V w.r.t to the rule $A_{ij} \leftrightarrow B_{kl}$ (Line 4). The length of U (and V) is increased by 1 and atom j (and l) is added. Lines 5-10 of Figure 5 show this process. However, if no **compatible substructures** are found then a new substructure U (and V) is initialized with atoms i and j (k and l). Lines 11-16 in Figure 5 refer to this case. The same process is then repeated for all the rules.

```
1  Input : All rules
2  For each rule : A_ij ↔ B_kl
3  {
4   Find Compatible substructures U and V
5    If U and V found
6     {
7       Length = Length+1;
8       U[Length] = j;
9       V[Length] = l;
10    }
11   else
12    {
13      Create new U and V;
14      Store i and j in U;
15      Store k and l in V;
16    }
17 }
```

Figure 5: Pseudo code for finding Largest Common Substructure

  This method also provides the correspondence between atoms. Atoms in U and V have a one-to-one relationship between them.

- **Similarity Metric Computation:** The **L**argest **S**tructure **(LS)** is then chosen from the common substructures. We use the following metric to determine the similarity between A and B:

$$Sim(A, B) = \frac{2 * \|LS\|}{\|A\| + \|B\|}$$

  This similarity is calculated between *A* and all the K candidate defect classes. The class which gives the maximum similarity greater than a user defined threshold is chosen as the target class. If the maximum similarity is less than the user defined threshold the defect

is considered new and both the databases, ID_{shape} and ID_{moment} are updated. The summary database is updated for each defect (previously seen or new).

5 Experiments and Results

In this section we present the results of our framework. As noted earlier we use OHMMS (see Section2) to generate the datasets. We first, show the advantage of weighted moments over unweighted moments by comparing the accuracies of various classifiers. Next, we demonstrate the accuracy of the LCS algorithm bootstrapped with different classifiers: *KNN, Naive Bayes and VFI*. Later, we show the scalable aspects of our framework by deploying it on very large datasets (in the giga-byte range). Finally, we present preliminary results demonstrating how our two-step classifier can help us gain a better understanding of defect evolution.

5.1 Robust Classification: To illustrate the importance of using weighted moments as opposed to unweighted moments, we performed the following experiment: a total of $1,400$ defects were randomly sampled across multiple simulations conducted at different temperatures. The noise in the simulation depends on the temperature at which the lattice is simulated. Therefore two defects belonging to the same class can have different number of atoms and/or different positions of atoms depending on the temperature, even though their core defect shape remains approximately the same. This sampling strategy ensures that no two defects of the same class are exactly the same. Each defect, in this experiment, belongs to one of the fourteen classes of single interstitial defects that are known to arise in Si.

For comparison purposes, we tried nine different classifiers. Figure 6 clearly demonstrates that all classifiers perform better when weighted moments were used. Classification accuracies of *VF1*, *KNN (K=1)* and *Decision tree* based classifiers are comparable (close to 90%). *SMO* (SVM based classifier), also provided good accuracy (85%) but it was quite slow; classifying 1,400 defects took over 25 minutes. On average the classification accuracy increased by 8% when weighted moments were used.

Next, we present the classification accuracies of *Naive Bayes, KNN and VFI*. These classifiers are modified to pick the K most important classes dynamically (as explained in Section 4). Figure 7 shows the results for this experiment. *KNN* with weighted moments outperforms all other classifiers by achieving an accuracy of 99% whereas *Naive Bayes* is the least accurate with an accuracy of 86%. Again, weighted moments outperform unweighted moments.

An important point to note is that all the $1,400$ defects used for this experiment were labeled manually by a domain expert. However, in actual simulation data there are no predetermined labels since new classes can be created as the simulation progresses. Also there is no training data to build the initial model for *Decision tree* and *Naive Bayes* classifiers. For the purpose of this experiment, we artificially divided the dataset into training (90%) and testing data (10%) for all the classifiers that require training data to build model. Classifier accuracies are averaged over 10 runs of the classifiers.

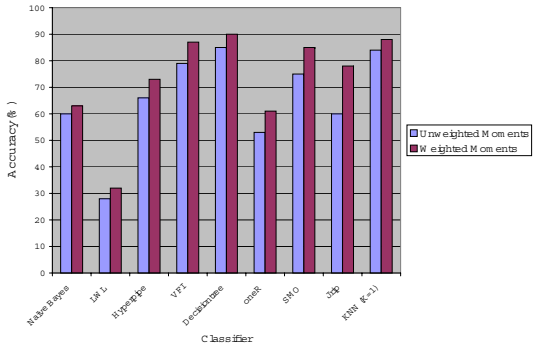


Figure 6: Accuracies of various classifiers

Only *KNN* and *VFI* can discover new classes in real time. Both classifiers calculate a similarity metric for classification: *distance in the case of KNN and votes for VFI*. If this similarity metric is less than a user defined threshold, a new class label can be assigned to the defect. However, *VFI* will have to build the whole classification model from start whenever a new defect class is discovered. Since large number of defect classes can be created in a simulation, rebuilding the classification model repeatedly will degrade the performance considerably.

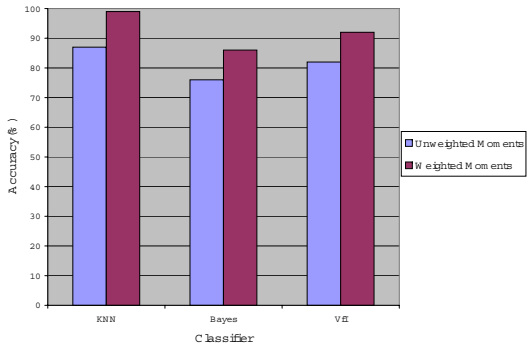
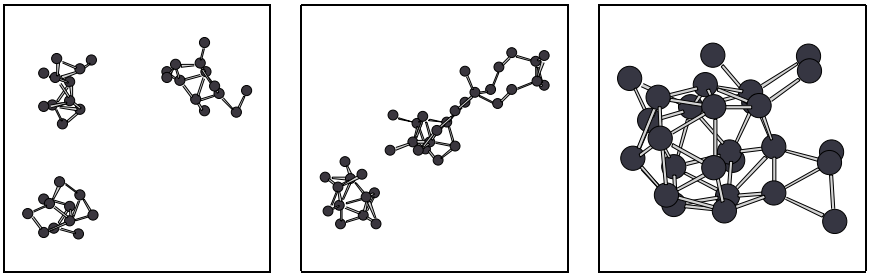


Figure 7: Classification accuracy with Dynamic K

Thus the *LCS* algorithm bootstrapped with the *KNN* classifier using weighted central moments is the best choice in terms of accuracy and efficiency.



(a) (b) (c)

Figure 8: Transitions in Three Interstitials(a) 1^{st} Frame (b) 20000^{th} frame (c) $130{,}000^{th}$ frame

5.2 Scalable Classification - Large Simulations: We use three large datasets, namely **Two Interstitials**, **Three Interstitials**, and **Four Interstitials** for these experiments. Table 1 summarizes the number of frames, size of the dataset, total number of defects present in the simulation and number of unique defect classes identified by our framework. For all three datasets, our framework was able to correctly identify all the defect structures. However, given the paucity of space we only present an in-depth analysis of the **Three Interstitials** dataset. Similar results were also obtained for other datasets.

Dataset	Number of Frames	Size (in GB)	Total Defects	Unique Defects Detected
Two Interstitials	512,000	4	350,000	2,841
Three Interstitials	200,200	6	320,000	1,543
Four Interstitials	297,000	10	410,000	3,261

Table 1: Datasets Used in Evaluation

This simulation starts with three disconnected interstitial defects. The defects move around in the lattice during the first $19,000$ time frames. However, at the $20,000^{th}$ time frame two of the defects join and form a 'new' larger defect. This larger defect does not change for a long period of time. However, at the $130{,}000^{th}$ time frame the third defect joins the 'new' defect and forms a single large defect which remains unchanged until the end of the simulation. Figure 8 shows the evolution of the defects in the simulation. For the rest of this paper we refer to changes in defect shape or type as "transitions".

Though these transitions occur over a large period (thousands of time steps), atoms do not stay at the exact same position in two consecutive frames due to thermal noise. Such thermal agitations can also cause bulk atoms to be labeled as defect atoms (and vice versa). As a result, there exist marginal fluctuations in the shape of a defect from frame to frame. However, the effect of these changes on weighted central moments is relatively small. For example, in the **Three Interstitials** dataset, the total number of defect instantiations in the simulation was around $320,000$. However, our classifier detected only $1,543$ unique defect classes. These $1,543$ defects capture the actual transitions as verified by a domain expert. To reiterate, the use of weighted moments minimizes false positives and ensures robust classification.

The use of weighted moments and pruning in **Step 1** also allows our approach to achieve good scalability. Finding the *LCS* is a relatively expensive algorithm, therefore we want to use it as infrequently as possible. In most cases the number of candidate classes K from **Step 1** (*KNN* classifier) of our dynamic classifier is less than 3. For example, in the **Two interstitials** dataset $2,841$ unique defects were found however, the LCS algorithm only evaluates less than 3 closest matches. This underlines the usefulness of the pruning step of our classifier. The discovery of all the unique defect classes demonstrates that the correct defect class is not pruned away. To summarize, pruning based on weighted moments provides scalability to the framework without affecting the accuracy.

Many of these defects are not stable, i.e, they may exist for as few as 100 time frames; however these unstable defect structures are extremely important since they allow one to understand the physics behind the creation of, and transitioning to, stable structures. We can easily eliminate these unstable structures from our repositories by either maintaining simple counts or by time averaging the frames. However, using both these techniques will result in loss of transition information. To illustrate this point we took the same **Three Interstitials** dataset and averaged it over every 128 frames. In this averaged data, we found only 18 unique defects. It turns out that we found all the possible stable structures, but the actual transitioning behavior was lost.

5.2.1 Timing Results: Figure 9 shows the time taken by OHHMS to complete the simulation and time taken by our framework to analyze the data. The figure also shows the individual time taken by *Phase1* (defect detection) and *Phase2* (classification). *Phase1* takes around 45% of the time and

Phase2 requires the rest of the time. All the experiments are carried out on Pentium 4 2.8GHz dual processor machine with 1 GB of main memory. Our classifier can analyze the data almost 1.5 times faster than the data generation rate. This allows us to analyze the data and build the defect databases in real time without dropping/losing any frames. Another advantage is that we are not required to store the large simulation file (of the order of 15GB) on disk. All the needed information about defect type(s), number(s) and transitions, can be obtained from the simulation databases and the summary file.

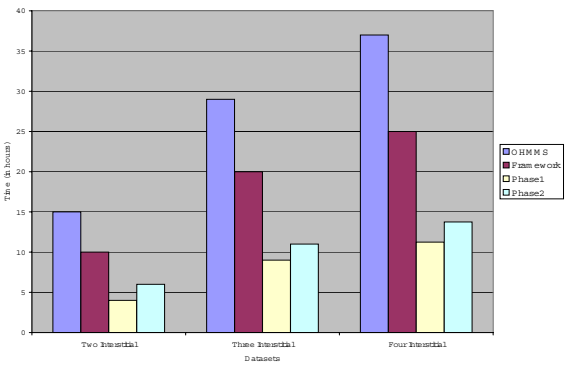


Figure 9: Timing Results

Next we show how the results produced by the framework can be used for tracking and understanding the movement of defects in the simulation.

5.3 Meta-stable Transitions: The transitions between two meta-stable defect structures are even more important than the creation of transient structures. We, next present experimental results on a simulation that depicts the transition of a single defect to another defect.

This is a fairly small but extremely useful simulation. The simulation has $1,300$ frames with 67 atoms in each frame and one interstitial defect. We were able to detect the 50 unique defects which actually capture the transitions from defect of type *I*3us-01 to *I*3us-03. (These labels are provided by domain experts). The defect does not break into multiple parts; therefore, we do not have to deal with the correspondence problem in this case[1]. Again, all these results have been verified by our domain expert by manually checking every frame of the simulation.

5.4 Generating defect trajectories: From the summary database produced at the end of simulation analysis, we can glean important information about the movement of a defect in lattices. The summary database provides information to construct a defect's motion trajectory over a period of time. We use a $10,000$ frame simulation to show this. In this simulation the defect moves in the $-z$ direction through the lattice, reaches the end of the lattice and then stays in the xy plane. We found 70 unique defects in this simulation. All the detected defects are labeled as one of these 70 classes. Most of these defects were highly unstable. We plot the (x,y,z) coordinates of the all the detected defect's weighted centroid at each time stamp. Figure 10 clearly shows the movement of the defect in the $-z$ direction. This idea can be extended to a multiple defect simulation. Since the defects in the summary database are labeled therefore, it should be fairly easy to construct multiple trajectories for multiple defects. By studying these labeled trajectories, one can gain more insight on how a defect evolves and interacts with other defects over time.

[1]Correspondence allows the labeling of two defects with the same class label at two different time epochs.

6 Conclusions

In this application case study, we propose a two-step classifier to classify the defects in large scale MD simulation datasets. The classifier is scalable and incremental in nature. New classes of defects can be discovered and added to classifier model in real time. The approach is also robust to noise (inherent to MD simulations). We present various noise handling schemes and validate these schemes using a physical model and properties of the lattice systems. We demonstrate the capabilities of our approach by deploying it on very large datasets ($\geq$ 4GB). We were able to find a very small number of unique defect classes from these large datasets. These unique classes capture the defect transitions very well.

We are currently working on solving the correspondence problem in the context of multiple defects. This will enable us to build an automated system to capture important events such as *defect disintegration* and *defect amalgamation*. Another future goal is to understand the interactions among defects in a simulation. Towards this goal, we plan to model the movement of defect as trajectories, tagged by defect class labels, and analyze these trajectories. We also plan to apply other data mining techniques including frequent itemset mining and spatial patterns mining to gain more insight in the actual defect evolution process.

References

[1] C. Borgelt and M. Berthold. Mining Molecular fragments: Finding relevant substructures of molecules. In *ICDM*, 2002.

[2] S.J. Clark and G.J. Ackland. Ab initio calculation of the self interstitial in silicon. *Physical Review Letters vol. 56*, 1997.

[3] A. S. Clarke and H. Jnsson. Structural changes accompanying densification of random hard-sphere packings. *Physical Review E vol.47*, pages 3975–3984, 1993.

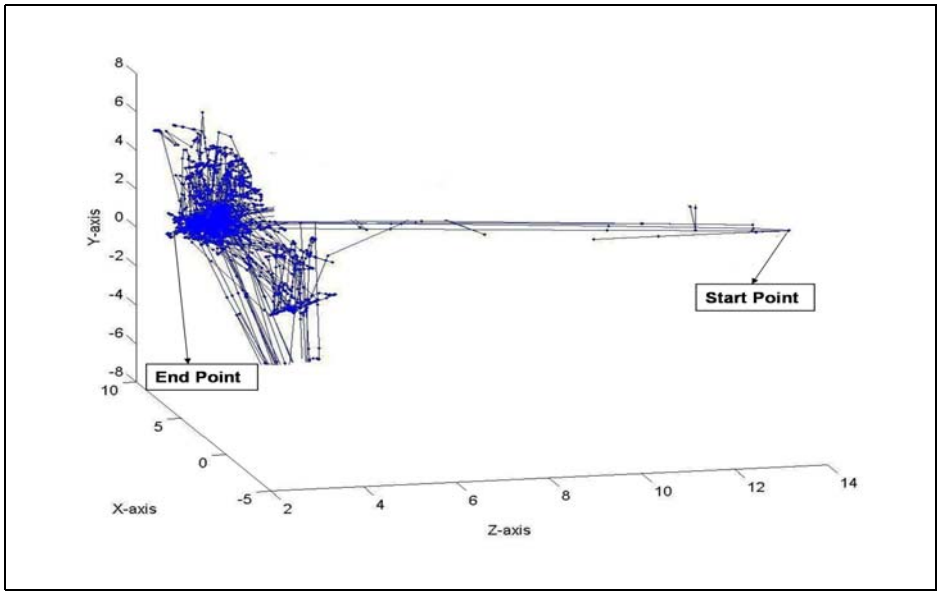


Figure 10: Capturing the movement of defect

[4] D. A. Richie, Jeongnim Kim, and John W. Wilkins. Real-time multiresolution analysis for accelerated molecular dynamics simulations. In *American Physical Society March Meeting*, 2001.

[5] K. Eric Drexler. *Nanosystems: molecular machinery, manufacturing, and computation*. Wiley Publishers, 1992.

[6] L. Spencer G. Hulten and P. Domingos. Mining time-changing data streams. In *Knowledge and Data Discovery (SIGKDD)*, 2001.

[7] J.M. Galvez and M. Canton. Normalization and shape recognition of three-dimensional objects by 3d moments. *PR*, 26:667–681, 1993.

[8] H. Jnsson and H. C. Andersen. Icosahedral ordering in the lennard-jones liquid and glass. *Physical Review Letters vol. 60*, pages 2295–2298, 1988.

[9] Charles Kittel. *Introduction to Solid State Physics*. John Wiley and Sons, 1971.

[10] L. Dehapse, H. Toivonen and R. King. Finding Frequent substructures in chemical compounds. In *Knowledge Discovery and Data Mining*, 1998.

[11] M. Jiang, T.-S. Choy, S. Mehta, M. Coatney, S. Barr, K. Hazzard, D. Richie, S. Parthasarathy, R. Machiraju, D. Thompson, J. Wilkins, and B. Gatlin. Feature Mining Algorithms for Scientific Data . In *SIAM*, 2003.

[12] Sameep Mehta, Kaden Hazzard, Raghu Machiraju, Srinivasan Parthasarathy, and John Wilkins. Detection and visualization of anamolous strcutures in molecular dynamics simulation data. In *IEEE Conference on Visualization*, 2004.

[13] M.hu. Visual Pattern Recognition by Moment Invariants. In *IRE Trans Information Theory*, pages 179–187.

[14] R. Machiraju, S. Parthasarathy, J. Wilkins, D. Thompson, B. Gatlin, D. Richie, T. Choy, M. Jiang, S. Mehta, M. Coatney, and S. Barr. Mining of Complex Evolutionary Phenomena, Next Generation Data Mining. In *NGDM*, 2003.

[15] R. Nussinov and H. Wolfson. Efficient Detection of three dimensional Structural Motifs in Biological Macromolecules by Computer Vision Techniques. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 88, Dec 1, 1991.

[16] S. Djoko, D. Cook and L. Holder. Analyzing the benefits of domain knowledge in substructure discovery. In *Knowledge Discovery and Data Mining*, 1995.

[17] S. Parthasarathy and M. Coatney. Efficient Discovery of Common Substructures in Macromolecules . In *ICDM*, 2002.

[18] R. Veltkamp and M. Hagedoorn. State-of-the-art in shape matching. Technical Report UU-CS-1999-27, Utrecht University, the Netherlands, 1999.

[19] R.J. Needs W.K. Leung and G. Rajagopal. Calculation of silicon self interstitial defects. *Physical Review Letters vol. 83*, 1999.

[20] X. Wang, J. Wang, D. Shasha, B. Shapiro, S. Dikshitulu, I. Rigoutsos and K. Zhang. Automated discovery of active motifs in three dimensional molecules. In *Knowledge Discovery and Data Mining*, 1997.

[21] Y. Lamdan and H. Wolfson. Geometric Hashing : a general and efficient model-based recognition scheme. In *Proceedings of the second ICCV*, pages 238–289, 1988.

[22] C. Zhang and T. Chen. Efficient Feature Extraction for 2D/3D Objects in mesh representation. In *ICIP*, 2001.