A Generalized Framework For Mining Spatio-temporal Patterns in Scientific Data

Hui Yang, Sameep Mehta, and Srinivasan Parthasarathy
Department of Computer Science and Engineering
The Ohio State University
{yanghu, mehtas, srini}@cse.ohio-state.edu

ABSTRACT

In this paper, we present a general framework to discover spatial associations and spatio-temporal episodes for scientific datasets. In contrast to previous work in this area, features are modeled as geometric objects rather than points. We define multiple distance metrics that take into account objects' extent and thus are more robust in capturing the influence of an object on other objects in spatial neighborhood. We have developed algorithms to discover four different types of spatial object interaction (association) patterns. We also extend our approach to accommodate temporal information and propose a simple algorithm to compute spatio-temporal episodes. We then develop algorithms to show that such episodes can be used to reason about critical events. We evaluate our framework on real datasets to demonstrate its efficacy. The datasets originate from three different areas, namely, Bioinformatics, Computational Molecular Dynamics and Computational Fluid Flow. We present results highlighting the importance of discovered patterns and episodes by using knowledge from the underlying domains. We also show that the proposed algorithms scale linearly as a function of dataset size.

1. INTRODUCTION

Analyzing spatial data is an important problem in many application domains, including geographical information systems, bioinformatics, scientific and engineering informatics, and computer aided design. The main difference between analyzing such data and data in a transactional form is that an object can influence the properties of objects located in the same spatial neighborhood [20] and therefore must be modeled in the analysis. Analyzing and reasoning about relationships among spatial objects is further complicated if the data is time varying in nature. Data produced from protein folding or fluid dynamics simulations, or geoinformatics datasets that track the behavior of intrusions are examples that have this additional constraint. Mining spatial relationships in these datasets is an important and interesting problem, since specific relationships may be indicators or predictors of upcoming events (e.g., vortex (hurricane) dissipation, crack propagation and amalgamation in materials).

Unfortunately, mining relationships among spatial objects is an extremely challenging task. First, almost all the related work done to date on this problem models each feature [1] as a point in a multi-dimensional space [13, 14, 27]. However, especially in physical sciences, the extent and shape of a feature can play an important role in determining its influence on neighboring objects. Second, there is a strong need to develop techniques to capture and reason about the interactions among the features. These interactions if captured properly can help domain experts to understand the underlying processes in very effective manner. Third, one needs to develop effective techniques to incorporate temporal information in the overall analysis and reason about them. Finally, recent technological advances in computational sciences have resulted in huge amounts of data. Traditional statistical approaches to model such interactions do not scale very well to large datasets. In this paper we propose a general-purpose framework to address these challenges.

We have noted that a feature's geometric properties such as shape and extent can play an important role in determining neighborhood relationships. For example as noted by Silver and Wang [19], two (or more) features can combine to form one new large feature. To capture and represent this event in a correct fashion, it is imperative that the change in shape and size is accounted for. Commonly used point based feature representation will not suffice. A straightforward solution to this problem would be to represent the feature by its Minimum Bounding Box (MBB). However MBBs are not ideal for all situations. For example vortices in fluid flows are very well represented by ellipsoids [17] but not by minimum bounding boxes. Alternatively, defect structures in materials may require irregular shape descriptors [12] such as landmarks[15]. Our framework thus supports multiple shape types. Figure 1 shows different representation schemes supported by out framework.

The interaction among spatially proximate features are important. An appropriate distance metric must be defined to detect when features are close enough to be considered interacting. Simply measuring the distance between centroids of two features is often inadequate for discovering such interactions. For example, in fluid flow, if two vortices are close to each other, they will possibly merge after a few time steps. However, the distance between their centroids can be very large. Therefore, using a centroid-based distance metric will miss this interaction and subsequently the merging event. The distance metric should thus be capable of taking into account the shape and extent of the features. We evaluate the use of four distance metrics and demonstrate that for different applications different metrics are appropriate.

In addition to distance metrics, we need a method to capture the nature of interactions. We define Spatial Object Association Patterns (SOAP) to characterize the interaction among different types of features (objects) at a given moment. We have identified four SOAP types, namely, *clique*, *star*, *sequence*, and *minLink* (Fig-

[1] We define a feature to be an object or entity of interest. A feature can have multiple attributes (including non-spatial ones) associated with it. For example, a vortex is a feature in a fluid flow simulation. Associated with this feature are spatial attributes such as location and non-spatial attributes such as velocity, swirl etc.

ure 2). Different SOAP types can characterize different types of interaction. For example, *clique* SOAPs in Molecular Dynamics data indicate the presence of compact defects. Compact defects are usually more stable and often govern the properties of materials. *Sequence* SOAPs on the other hand can indicate the formation of long extended defects from smaller point defects [16]. In this paper we propose fast and efficient algorithms for detecting frequently occurring SOAPs. Our algorithms are scalable and can handle large out-of-core datasets.

We have also developed a very simple approach to model evolving SOAPS or sets of features in data produced by scientific simulations. We define and identify three types of events to describe SOAPs' evolutionary behavior: *formation*, *dissipation*, and *continuation*. Formation and dissipation of a SOAP indicate the start and end of an interaction among involved features. Whereas continuation characterizes the stability of an interaction. The continuation of a SOAP from its formation to its next dissipation is abstracted as a *spatio-temporal episode*. Critical events such as merging can then be inferred by by analyzing these episodes. Furthermore, by combining episodes associated with multiple SOAP types, we can also model how the interactions among features change over time.

In summary we make the following contributions in this work:

1. We present robust techniques for modeling the shape and extent of features (objects).
2. We have developed fast algorithms for extracting frequent spatial object interactions through the design of appropriate distance functions and interaction types.
3. We have developed a simple yet effective approach for mining spatio-temporal episodes of SOAP patterns. We further demonstrate that an approach that combines information from multiple SOAP models is capable of reasoning about critical events.
4. We have empirically evaluated our approaches on real case study applications and show that the algorithms scale well and are capable of processing large datasets.

We validate our framework on three case study applications drawn from the scientific and engineering community. Two of the applications analyze scientific simulation data, namely, Computational Fluid Dynamics (CFD) and Computational Molecular Dynamics (CMD) and the other case study application is drawn from protein contact map analysis. The main challenges for these applications include feature detection, classification, and then extracting and modeling spatio-temporal or spatial interactions. Many techniques have been proposed to detect, extract and classify features from such data in the past [10, 11, 23, 24]. In this work we limit our discussion to the last aspect, namely, discovery of spatial or spatio-temporal patterns.

2. BASIC CONCEPTS

2.1 Spatial Feature Representation

We propose three different representation schemes: parallelepiped (or parallelogram in 2D), ellipsoid (or ellipse in 2D), and landmarks based representation, where landmarks are sampled boundary points [15]. These schemes can be used to model features from a variety of scientific domains. For instance, parallelograms are suitable to model non-local structures in protein contact maps. Whereas ellipsoids or ellipses are well-suited for vortices. Finally, landmarks are very effective to model highly irregular-shaped features such as defect structures in materials. The number of landmarks needed to represent a feature is domain dependent. The framework also supports elemental shapes such as lines and splines.

As shown in Fig. 1a-b, the shape descriptor of a parallelogram or an ellipse can be described as a vector $A_{basic} =< l_1, l_2, \theta >$. If landmarks are used (Fig. 1c), the shape descriptor is $A_{landmark}$=< (x_i, y_i):1≤ $i \leq \nu >$, where (x_i, y_i) is the position of the i^{th} landmark. These shape descriptors can be easily extended to 3-D cases.

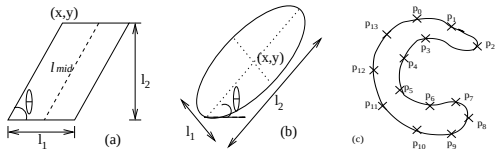

Figure 1: Shape Representations: (a)Parallelogram (b)Ellipse (c)Irregular

2.2 Dataset Representation

The dataset $\mathbb{D}$ consists of n features extracted from r>1 maps, denoted as M=$\{m_1, m_2, \ldots, m_r\}$. In time-varying data, the r maps correspond to r time frames or snapshots, which are taken at time $t_1, t_2, \ldots, t_r$ ($t_1 < t_2, \ldots, t_r$). For spatial data that involves multiple maps, one can arbitrarily assign a unique ID to each map. For example, the set of contact maps from different proteins can be treated in the same manner as that for time-varying data, where the arbitrarily assigned map ID is used in place of a snapshot's associated time. The n features in $\mathbb{D}$ are categorized into l types, which are governed by the underlying domain. The l feature types are identified by l unique labels, denoted as Σ=$\{c_1, c_2, \ldots, c_l\}$. A feature's geometric properties such as shape and size are captured based on one of the three representation schemes described in Section 2.1. Thus a feature f appearing at time t_i can be described as a vector f=< t_i, *location*, A_{geo}, *type* >, where *type*∈ Σ, *location* identifies f's position at t_i, and $A_{geo} \in \{A_{basic}, A_{landmark}\}$ models the geometric properties of f at t_i.

Note that in the rest of the paper, we refer to a feature corresponding to the above vector as a **spatial object**. We also assume the existence of an ordering among the l feature types: c_1<$c_2 \ldots < c_l$. Furthermore, we refer to a snapshot's associated time t_i as its ID.

2.3 Object-based Distance Metrics

The framework uses the following metrics to measure the distance between two objects o_i and o_j existing in the same snapshot.

- **Point-Point distance**: This is simply the Euclidian distance between object centroids.
- **Line-Line distance**: If o_i and o_j are parallelepipeds (or parallelograms), we first identify the line segment between the midpoints of the upper and lower surfaces (or sides) in each object, then compute the shortest distance between these two line segments as the line-line distance between o_i and o_j. If o_i and o_j are ellipsoids (or ellipses), the line-line distance is the shortest distance between the two major axes.

- **Boundary-Boundary distance**: This is the shortest pairwise distance between the sampled boundary points (landmarks) of o_i and o_j.

Notice that the last two metrics take objects' geometric properties into account. The framework also supports Hausdorff distance [2]. Since this distance is not applicable to the applications described in this article, we do not discuss it here.

Two objects o_i and o_j have a **closeTo** relationship if the distance between them is $\leq \varepsilon$, where ε is a user-specified parameter. Two objects are **neighbors** if they have a *closeTo* relationship. We also define the **isAbove** relationship between o_i and o_j. In a coordinate system, o_i is said to have a isAbove relationship with o_j, if the upper-left corner of o_i's Minimum Bounding Box (MBB)[2], denoted as (x_i, y_i, z_i), and the upper left corner of o_j's MBB, denoted as (x_j, y_j, z_j), meets the following condition: $(z_i > z_j) \vee [(z_i = z_j) \wedge [(y_i > y_j) \vee ((y_i = y_j) \wedge (x_i < x_j))]]$ in 3D, or $(y_i > y_j) \vee [(y_i = y_j) \wedge (x_i < x_j)]$ in 2D.

2.4 Spatial Object Association Pattern (SOAP)

A **Spatial Object Association Pattern (SOAP)** of size k, denoted as k-**SOAP**, characterizes the *closeTo* or *isAbove* relationships among k object types. The framework supports the discovery of four SOAP types: **star**, **clique**, **sequence**, and **minLink** (Figure 2). They can be abstracted as undirected graphs, where a node corresponds to an object-type $c_i \in \Sigma$, and an edge (c_i, c_j) indicates a *closeTo* or *isAbove* relationship between c_i and c_j. These SOAP types can also be represented as lists with different constraints for different SOAP types. We describe each SOAP type and its corresponding list representation as follows.

- **Star SOAPs** (Fig. 2a) have a *center* object-type, which is required to have a *closeTo* relationship with all the other object-types in the same SOAP. Let c_{cntr} be the center of a star k-SOAP p, and $\{c_{[i]}: i \in [1,k\text{-}1]\}$ be the other *k-1* object-types in p, where $c_{[1]} \leq \ldots \leq c_{[k-1]}$. The SOAP p can then be represented as the list p=(c_{cntr}, $c_{[1]}, \ldots, c_{[k-1]}$), where *closeTo*(c_{cntr}, $c_{[i]}$)=*true* ($i \in [1,k\text{-}1]$).
- **Clique SOAPs** (Fig. 2b) require a *closeTo* relationship hold between every pair of involved object-types in the same SOAP. Let $\{c_{[i]}: i \in [1,k]\}$ ($c_{[1]} \leq \ldots \leq c_{[k]}$) be the k object-types in a *clique* SOAP, it can then be described by the following list ($c_{[i]}: i \in [1,k]$): $\forall_{i,j \in [1,k]}$*closeTo*($c_{[i]}$, $c_{[j]}$).
- **Sequence SOAPs** (Fig. 2c) of size k, p=($c_{[i]}: i \in [1,k]$), satisfy two constraints: (1) *closeTo*($c_{[i]}$, $c_{[i+1]}$)=*true* and (2) *isAbove* ($c_{[i]}$, $c_{[i+1]}$)=*true*, where $1 \leq i \leq k\text{-}1$.
- **minLink SOAPs** (Fig. 2d) are a parameterized SOAP type, where the value of *minLink* is user-specified. Let *minLink=l*, *minLink* SOAPs include all the SOAPs that have *linkage*$\geq l$. The *linkage* of a k-SOAP p_k is defined as follows. Let n_i ($1 \leq i \leq k$) be the number of neighbors that the i^{th} object-type has in p_k, the *linkage* of p_k is then defined as $min\{n_i: (1 \leq i \leq k)\}$. Thus, we can represent a *(minLink=l)* k-SOAP by the following list: ($c_{[i]}: i \in [1,k]$):$\forall_{i \in [1,k]}$(#Neighbors Of $c_{[i]}$)$\geq l$, where ($c_{[1]} \leq \ldots \leq c_{[k]}$). Note that when *minLink=1*, all *star*, *clique*, and *sequence* SOAPs will be generated.

[2]For a 3D MBB, it is defined as the upper-left corner of its top surface.

A SOAP is said to be **autocorrelated** if an object-type occurs multiple times. For example, (c_1, c_1, c_2) is an autocorrelated 3-SOAP, where c_1 occurs twice. An **instance** of a SOAP p is the set of spatial objects that meet all the requirements specified by p. For instance, (o_i,o_j) is an instance of the *clique* 2-SOAP (c_1,c_2), where o_i*.type*=c_1, o_j*.type*=c_2, and *closeTo*(o_i,o_j)=*true*,

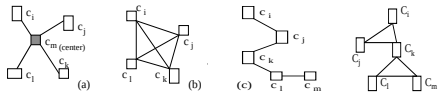

Figure 2: SOAP Types:(a)Star(b)Clique(c)Sequence(d)minLink=2

We define two measures **support** and **realization** to characterize the importance of a SOAP. The support of a SOAP p is the number of snapshots in the dataset where p occurs. Assume support*(p)*=*s*, let n_i be the number of p's instances in the i^{th} snapshot where p appears, realization(p)=$min\{n_i\}$. A pattern p is **frequent** if support (p) $\geq$ *minSupp*, and **prevalent** if realization(p) $\geq$ *minRealization*. Both *minSupp* and *minRealization* are user-specified parameters.

It is straightforward to show that frequent *star*, *clique*, *sequence*, and *(minLink=1)* SOAPs have the anti-monotone property [1]. This means that a k-SOAP cannot be frequent if one of its sub-SOAPs is not frequent. A **sub-SOAP** of a SOAP p corresponds to a sub-list of p's corresponding list. For a *star* SOAP, its sub-SOAPs also need to have the same center. However, when *minLink*>1, *minLink* SOAPs will not be anti-monotonic. We illustrate this by an example. Let *minLink=2*, the 2-SOAP (c_1,c_2) has linkage of 1, thus is not a valid minLink SOAP. However, the clique SOAP (c_1,c_2,c_3), which has linkage of 2, might be a valid *minLink=2* SOAP. Extra processing is needed to handle this case. For the first three SOAP types and *minLink=1* SOAPs, we leverage the anti-monotone property to efficiently prune the search space.

2.5 Spatio-temporal Episodes

Spatial relationships among objects (or features) evolve over time. As a result, SOAPs of different types also evolve over time. We identify the following three evolutionary events for a SOAP p:

- **Formation**: when the number of p's instances changes from zero to non-zero.
- **Dissipation**: when all p's instances become invalid. The dissipation of a SOAP can occur due to many reasons. For example feature(s) involved in a SOAP may cease to exist or merge into a new one. Another possible reason is end of interactions among the involved features.
- **Continuation** from time t_i to time t_{i+1}: if there exists at least one instance of p in each snapshot taken between t_i and t_{i+1}, including t_i and t_{i+1}.

Essentially, these events characterize the stability of interactions among different features. They are useful to model and subsequently predict features' future spatio-temporal behaviors.

Formation and dissipation events can occur to a SOAP many times. Thus a SOAP can exist in multiple disjoint temporal intervals, where each interval starts at a formation event and ends at a dissipation event. We refer to a SOAP's continuation in each of the above temporal intervals as a **spatio-temporal episode**. Let I_p^t be the

set of SOAP p's instances at time t, an episode of p in the *discrete* interval $[t_s, t_e]$ can then be described as: $E_p[t_s, t_e] = \{ I_p^t : t_s \leq t \leq t_e \}$. The framework identifies all the episodes associated with each frequent SOAP. It also extracts information to abstract how an interaction changes over time within an episode or across multiple episodes.

3. FRAMEWORK AND ALGORITHMS

An overview of the framework is given in Figure 3. In this article, we focus on four tasks enclosed by the dashed rectangle in the figure. We will also present solutions to facilitate spatio-temporal analysis and inferences.

We organize the dataset $\mathbb{D}$ in the following manner. The n objects are first grouped into l partitions, where each partition is composed of objects of the same type. Within each partition, objects are first ordered by the time they appeared, then by their locations at a certain time. This data organization is analogous to the vertical format used for association rule mining [26].

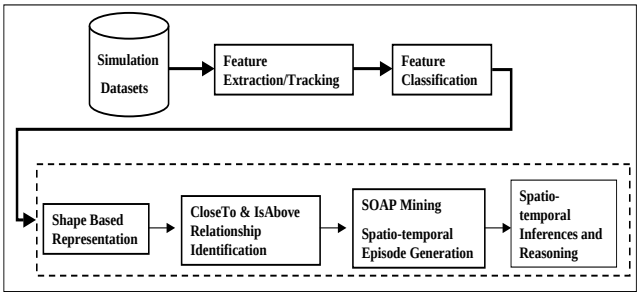

Figure 3: Overview of the framework

3.1 Equivalence Classes

Based on the list-based SOAP representation (Section 2.4), we organize SOAPs as equivalence classes. A **k-equivalence class**, denoted by k-**EquiClass**, is defined as the set of k-SOAPs that (1) are of the same SOAP type; and (2) have the same prefix, where the **prefix** of a k-SOAP consists of its first *k-1* elements. For instance, *star* SOAPs $< (c_1, c_1, c_1), < (c_1, c_1, c_2)$, and $< (c_1, c_2, c_4)$ will be organized into the same 3-EquiClass, with $< (c_1, c_1) >$ being the the prefix.

By using equivalence classes, our mining algorithms only need to compute the *closeTo* or *isAbove* relationships among objects once (,when generating 2-SOAPs). The equivalence classes also help to improve locality while generating SOAPS [26]. As a result, our algorithms can efficiently discover different types of SOAPs from large amounts of data.

3.2 Mining Star SOAPs

Figure 4 outlines the algorithm that discovers *star* SOAPs. The first step, *gen1SOAP*, generates frequent 1-SOAPs. For each object-type c_i, the procedure counts the number of snapshots that contain at least one object of type c_i. If the count $\geq$ *minSupp*, then p_1=(c_i) is a frequent 1-SOAP. The set of all frequent 1-SOAPs is denoted by P_1.

The next step, *gen2EquiClass*, discovers 2-SOAPs and organizes them into 2-EquiClasses (line 2). The pseudo-code of *gen2EquiClass* is described in Fig.5. A 2-EquiClass is generated for each frequent

Algorithm mine_starSOAP($\mathbb{D}$)
Parameters: *minSupp, minRealization, distType, ε*
1. $P_1 \leftarrow$ gen1SOAP(); //freq. and prev. 1-SOAPs
2. $EC_2 \leftarrow$ gen2EquiClass($\mathbb{D}$, P_1, *star, parList*); //parList: parameters
3. $k \leftarrow 2$;
4. while (1){
5. $EC_{k+1} \leftarrow \varnothing$; //the set of (k+1)-EquiClasses
6. foreach k-EquiClass $E_k \in EC_k$
7. foreach k-SOAP $p_{k,i} \in E_k$
8. $E_{k+1} \leftarrow \varnothing$ //the prefix of E_{k+1} is $p_{k,i}$
9. foreach k-SOAP ($p_{k,j} \in E_k$) $\wedge$ ($i \leq j$)
10. $p_{k+1} \leftarrow$ append($p_{k,i}$, lastElement($p_{k,j}$));
11. $S_{p_{k+1}} \leftarrow \{t_i : t_i$ contains $p_{k,i}$ and $p_{k,j}\}$;
12. foreach $t_i \in S_{p_{k+1}}$
13. if (countStarInstances(t_i, p_i, p_j)$\geq$ 1) occCnt++;
14. if (occCnt $\geq$ *minSupp*) $E_{k+1} \leftarrow E_{k+1} \cup \{p_{k+1}\}$;
15. if ($E_{k+1} \neq \varnothing$) $EC_{k+1} \leftarrow EC_{k+1} \cup E_{k+1}$;
16. if (*minRealization*>1) markPrevFreqSOAPs(E_{k+1});
17. if ($EC_{k+1} = \varnothing$) break; //terminate
18. k++; //increase SOAP size }//while(1)

Figure 4: Mining *Star* SOAPs

1-SOAP (Fig.5:line 2). The 2-EquiClass of 1-SOAP (c_i)$\in P_1$, denoted by E_{2,c_i}, contains frequent 2-SOAPs in the form of ($c_i, *$). To generate E_{2,c_i}, the procedure considers the following 2-SOAPs as candidates: (c_i, c_j), where (c_j)$\in P_1$ (Fig.5:line 3). For each candidate 2-SOAP p_2=(c_i, c_j), the procedure identifies all the snapshots where p_2 occurs (Fig.5:lines 7-12). An instance of p_2=(c_i, c_j) is an object pair (o_i, o_j), where (o_i.*type*=c_i) $\wedge$ (o_j.*type*=c_j) $\wedge$ (*distance*(o_i, o_j,*distType*) $\leq \varepsilon$). If p_2 occurs in $\geq$*minSupp* snapshots, then it is frequent and added to E_{2,c_i} (Fig.5:line 13).

Algorithm gen2EquiClass($\mathbb{D}$, P_1, $soapType$)
Parameters: *minSupp, minRealization, distType, ε*
1.$EC_2 \leftarrow \varnothing$; //the set of 2-EquiClasses
2.foreach (c_i)$\in P_1$ //P_1:ordered freq. 1-SOAPs
3. $E_{2,c_i} \leftarrow \varnothing$ //2-EquiClass with prefix of c_i;
4. foreach (c_j)$\in P_1$
5. if ((*soapType*$\notin$ {*star or sequence*}) $\wedge$ ($c_i > c_j$)) continue;
6. $p_2 \leftarrow (c_i, c_j)$; //a candidate 2-SOAP
7. $S_{p_2} \leftarrow \{t_i : (t_i$ contains c_i and $c_j)\}$;
7. foreach $t_i \in S_{p_2}$
9. foreach $(o_i, o_j) \in t_i : (o_i.label = c_i) \wedge (o_j.label = c_j)$
10. if ((*soapType=sequence*)$\wedge$($\neg$ isAbove(o_i, o_j))) continue;
11. if distance(o_i, o_j, *distType*)$\leq \varepsilon$) addInstance(p_2, (o_i, o_j));
12. if (cntInst(p_2, t_i) $\geq$ 1) occCnt++;
13. if (occCnt $\geq$ *minSupp*) $E_{2,c_i} \leftarrow E_{2,c_i} \cup \{p_2\}$;
14.if ($E_{2,c_i} \neq \varnothing$) $EC_2 \leftarrow EC_2 \cup E_{2,c_i}$;

Figure 5: 2-EquiClass Generation

The algorithm next discovers SOAPs of size>2 (Fig.4:lines 4-17). Two k-SOAPs in the same k-EquiClass are combined to construct a candidate (k+1)-SOAP. For each candidate (k+1)-SOAP p_{k+1} derived by appending the last element of $p_{k,j}$ to $p_{k,i}$ (Fig.4: line 10), the algorithm identifies all the snapshots where p_{k+1} occurs (Fig.4:lines 11-14). The i^{th} k-SOAP in a k-EquiClass is denoted as $p_{k,i}$. The procedure *countStarInstances*($t_i, p_{k,i}, p_{k,j}$) (Fig.4:line 13) computes the instances of p_{k+1} in snapshot t_i by combining instances of $p_{k,i}$ and $p_{k,j}$. Two instances from $p_{k,i}$ and $p_{k,j}$ are joined to produce a p_{k+1} instance if they have the same first k-1 objects and different last object. SOAPs with the same prefix are organized into one equivalence class as they are being generated (Fig4:lines 5,8,14,15). The mining process stops when all the frequent SOAPs have been discovered (Fig. 4:line 17).

To discover frequent SOAPs, the algorithm only needs to consider a SOAP's presence in a snapshot (Fig.4:line 13). Hence, some of the

Algorithm mine_sequenceSOAP($\mathbb{D}$)
Parameters: *minSupp, minRealization, distType,* ε
1. $P_1 \leftarrow$ genSOAP_1();//freq. and prev. 1-SOAPs
2. $EC_2\,(or\,P_2) \leftarrow$ gen2EquiClass($\mathbb{D}$, P_1, *sequence, parList*);//parList:parameters
3. $k \leftarrow 2$; //max. size of SOAPs discovered
4. while (1){
5. $P_{k+1} \leftarrow \varnothing$; //initialize the set of $(k+1)$-SOAPs;
6. foreach SOAP $p_{k,i} =< c_{[0]}, \ldots, c_{[k-1]} > \in P_k$
7. $E_{2,c_{[k]}} \leftarrow E_{2,c_{[k]}} \in EC_2) \wedge (E_{2,c_{[k]}}.prefix = c_{[k]})$;
8. foreach 2-SOAP $p_{2,j} \in E_{2,c_{[k]}}$
9. $p_{k+1} \leftarrow$ append($p_{k,i}$, lastElement($p_{2,j}$));
10 $S_{p_{k+1}} \leftarrow \{t_i : t_i$ contains both $p_{k,i}$ and $p_{2,j}\}$;
11. foreach $t_i \in S_{p_{k+1}}$
12. if (countSeqInstances($t_i, p_{k,i}, p_{2,j}$) $\geq$ 1) occCnt++;
13. if (occCnt $\geq$ *minSupp*) $P_{k+1} \leftarrow P_{k+1} \cup \{p_{k+1}\}$;
14. if ($P_{k+1} = \varnothing$) return; //terminate the process;
15. if (*minRealization*>1) markPrevFreqSOAPs(P_{k+1});
16. k++; //increase SOAP size}//while(1)

Figure 6: Mining *Sequence* SOAPs

discovered frequent SOAPs may not be prevalent if *minRealization*>1. In this case, the procedure *markPrevFreqSOAPs* is called to identify the SOAPs that are both prevalent and frequent (Fig4:line 16). It is necessary to keep SOAPs that are frequent but not prevalent. We explain this by one simple example. Let $\{a_1, b_1, b_2\}$ be the only objects in a snapshot and they are neighbors. Assume *minRealization=2*, in order to derive the two instances (a_1, b_1) and (a_1, b_2) of the 2-SOAP (a, b), the 1-SOAP (a) must be maintained even if its realization is 1 (<*minRealization*) in the snapshot.

Correctness: It is straightforward to show that the algorithm discovers all the frequent 1-SOAPs and 2-SOAPs. Thus to prove the algorithm is correct, we only need to show that every frequent k-SOAP (k>2) will be considered as a candidate SOAP. Assume p_k=($c_{[1]}, \ldots, c_{[k-2]}, c_{[k-1]}, c_{[k]}$) is frequent, then the two (k-1)-SOAPs $p_{k-1,i}$= ($c_{[1]}, \ldots, c_{[k-2]}, c_{[k-1]}$) and $p_{k-1,j}$= ($c_{[1]}, \ldots, c_{[k-2]}, c_{[k]}$) must also be frequent and in the same equivalence class. Thus the algorithm will consider p_k as a candidate. It is trivial to show that the procedure *countStarInstances* identifies all the instances of a candidate SOAP. □

3.3 Mining Clique SOAPs

Clique SOAPs can be discovered by applying two changes to the algorithm for mining *star* SOAPs (Fig.4). The first change is applied to *gen2EquiClass* (Fig.4:line 2) to eliminates 2-SOAPs in the form of (c_i, c_j):c_i>c_j, as they are not in lexicographic order (Fig. 5: line 5). The second change takes place at line 13 in Fig.4, which replaces the procedure *countStarInstance($t_i, p_{k,i}, p_{k,j}$)* with *countCliqueInstance($t_i, p_{k,i}, p_{k,j}$)*. The procedure identifies the instances of the candidate *(k+1)*-SOAP p_{k+1} in snapshot t_i, where p_{k+1} is obtained from combining two k-SOAPs $p_{k,i}$ and $p_{k,j}$ in the same equivalence class. Two instances from $p_{k,i}$ and $p_{k,j}$ are joined to produce an instance of p_{k+1} if they have the same first (k-1) objects and there is a closeTo relationship between the two instances' last object.

Correctness: The proof is similar to that for *star* SOAPs.

3.4 Mining Sequence SOAPs

The pseudo-code is described in Fig.6. Unlike mining *star* or *clique* SOAPs, which uses two k-SOAPs in the same equivalence class to generate a candidate (k+1)-SOAP (k≥2), the algorithm joins one k-SOAP and one 2-SOAP.

Algorithm mine_minLinkSOAP($\mathbb{D}$)
Parameters: *minLink, minSupp, minRealization, distType,*ϵ
1. $P_1 \leftarrow$ genSOAP_1();//freq. and prev. size 1 SOAPs
2. $EC_2\,(P_2) \leftarrow$gen2EquiClass($\mathbb{D}$, P_1, *minLink, parList*); //parList: parameters
3. $k \leftarrow 2$; //maximum SOAP size discovered
4. while (1){
5. $P_{k+1} \leftarrow \varnothing$; //initialize the set of $(k+1)$-SOAPs;
6. *blackList*$\leftarrow \varnothing$; //infreq. SOAPs;
7. foreach SOAP $p_k \in P_k$
8. $C_{cand} \leftarrow$ genCand(p_k); //class-labels that can potentially grow p_k
9. foreach $c \in C_{cand}$
10. $p_{k+1} \leftarrow$ append(p_k, c));
11. if (($p_{k+1} \in P_{k+1}$) $\vee$ $p_{k+1} \in$ *blackList*)) continue;
12. $Mp_{k+1} \leftarrow \{t_i : t_i$ contains p_k and $c\}$;
13. foreach $t_i \in M$
14. if (countInstances(t_i, p_k, c)$\geq$ *minRealization*) occCnt++;
16. if ($minLink > 1$){//check if p_{k+1} meets minLink
17. foreach instance e of p_{k+1}
18. if (getLinkage(e)$\geq$ $minLink$) cnt++;
19. if (cnt $\geq$ $minRealization$) occCntMinLink++;
20. }
21. }
22. if ((*minLink*>1)$\wedge$(occCntMinLink$\geq$*minSupp*)) p_{k+1}.*flag=true*;
23. if (occCnt $\geq$*minSupp*) $P_{k+1} \leftarrow P_{k+1} \cup \{p_{k+1}\}$;
24. else *blackList* $\leftarrow$*blackList*$\cup\{p_{k+1}\}$;
25. if ($P_{k+1} = \varnothing$) return; //terminate the process;
26. empty(*blackList*); //terminate the process;
27. if (*minRealization*>1) markPrevFreqSOAPs(P_{k+1});
28. k++; //increase SOAP size }

Figure 7: Algorithm-Mining *minLink* SOAPs

The first two steps (lines 1-2) discover all frequent 1-SOAPs and 2-SOAPs. The *isAbove* relationship is checked by the procedure *isAbove(o_i, o_j)* (Fig. 5:line 10). For each k-SOAP p_k=($c_{[1]}, \ldots, c_{[k]}$), the algorithm first locates the 2-EquiClass $E_{2,c_{[k]}}$ in which every 2-SOAP is in the form ($c_{[k]}, c_{[j]}$):closeTo($c_{[k]}, c_{[j]}$)$\wedge$isAbove($c_{[k]}, c_{[j]}$) (line 7). A set of candidate (k+1)-SOAPs are then generated by combining p_k with each 2-SOAP in $E_{2,c_{[k]}}$ (lines 8-9). Same as mining star or clique SOAPs, a candidate (k+1)-SOAP p_{k+1} is frequent if it appears in ≥*minSupp* snapshots (lines 10-13). The procedure *countSeqInstances($t_i, p_{k,i}, p_{2,j}$)* (line 12) identifies instances of p_{k+1} in snapshot t_i, where p_{k+1} is a candidate SOAP based on $p_{k,i}$ and $p_{2,j}$. Instances of p_{k+1} are computed by combining the instances of $p_{k,i}$ and $p_{2,j}$. The two instances $I_{k,i}$ and $I_{2,j}$, from $p_{k,i}$ and $p_{2,j}$ respectively, can be combined to produce a p_{k+1} instance if the last object in $I_{k,i}$ is the same as the first object in $I_{2,j}$. For the same reason explained before, the algorithm calls the procedure *markPrevFreqSOAPs* to label SOAPs being both prevalent and frequent if *minRealization*>1 (line 14). The algorithm stops when no more SOAPs can be discovered (line 15).

Correctness: For each frequent sequence k-SOAP p_k = ($c_{[1]}, \ldots, c_{[k-1]}, c_{[k]}$), the following two SOAPs must also be frequent: p_{k-1} = ($c_{[1]}, \ldots, c_{[k-1]}$) and p_2=($c_{[k-1]}, c_{[k]}$). Thus, p_k will be considered as a candidate and be discovered. It is trivial to show that the procedure *countSeqInstances* identifies all the instances of a candidate SOAP.□

3.5 Mining minLink SOAPs

The pseudo-code for mining *minLink* SOAPs is described in Figure 7. The algorithm starts with identifying all frequent and prevalent 1-SOAPs and 2-SOAPs (lines 1-2). All the features in a *minLink* SOAP are required to be in lexicographic order (see Section 3.1). Therefore, the procedure *gen2EquiClass()*, which generates 2-EquiClasses, prunes away 2-SOAPs that are not in lexicographic order (line 4 in Fig. 5).

To generate a *(k+1)*-SOAP , the algorithm uses one k-SOAP and one 1-SOAP. For each k-SOAP p_k, the procedure *genCand(p_k)* (line 8) is called to collect the set of features (1-SOAPs) that have a *closeTo* relationship with at least one feature in p_k. These features can be easily identified from 2-EquiClasses(EC_2). Let C be the set of all such 1-SOAPs. A candidate *(k+1)*-SOAP p_{k+1} is then generated by appending a feature $c_i \in C$ to p_k (line 10). However, the same p_{k+1} can be generated multiple times. For example, consider the following 2-SOAPs $p_1 :< c_1, c_2 >$ and $p_2 :< c_1, c_3 >$, p_1 can be joined with c_3 to form a 3-SOAP $< c_1, c_2, c_3 >$. Similarly c_2 can be joined with p_2 to obtain the same 3-SOAP. Therefore, the SOAP will be considered twice. To avoid redundant computation, the algorithm checks if a candidate SOAP has been discovered before processing it. The candidate $(k+1)$-SOAP can be either frequent or infrequent. If it is frequent, it can be found in P_{k+1}, the set of *(k+1)*-SOAPs discovered so far. The algorithm also maintains a *blackList* to keep all the infrequent candidate *(k+1)*-SOAPs that have been processed so far (lines 6, 24, and 26). In Line 11, the algorithm searches both P_{k+1} and the *blackList* to make sure that a candidate SOAP is not re-processed. It proceeds to process p_{k+1} if it is not in P_{k+1} or the *blacklist* (lines 12-16). The procedure *countInstances(t_i, p_k, c)* (line 14) identifies p_{k+1}'s valid instance in snapshot t_i by combining instances from p_k and c_i. An instance I_k of p_k can be combined with an object of type c_i to produce an instance of p_{k+1} if they meet the following conditions: (1) o_i is not in I_k; (2) o_i has a *closeTo* relationship with at least one object in I_k; and (3) the *linkage* of I_{k+1} is $\geq$*minLink*.

Note that when *minLink*> 1, in order to discover all frequent *(k+1)*-SOAPs, the algorithm needs to identify all the frequent k-SOAPs with *linkage* ≥ 1. We explain this by an example. Let *minLink=2*, to find the clique 3-SOAPs (c_1,c_2,c_3) (linkage=2), the algorithm needs to join the 2-SOAP (c_1,c_2) and c_3. Thus, (c_1,c_2) cannot be pruned away even though its linkage is 1 (<*minLink*). As a result, some of the discovered SOAPs can fail to meet the *minLink* criteria. To address this issue, the algorithm computes a pattern's *linkage* value (line 18) and flags those patterns that have *linkage*$\geq$*minLink* (line 22). For the same reason stated before, when *minRealization*>1, the algorithm calls the procedure *markPrevFreqSOAPs* to label SOAPs being both prevalent and frequent (line 14). The algorithm stops when no more SOAPs can be discovered.

Correctness: It is straightforward to show that the algorithm discovers all frequent 1-SOAPs and 2-SOAPs. Let p_{k+1} = ($c_{[1]}$, ..., $c_{[k]}$, $c_{[k+1]}$) be a frequent $(k+1)$-SOAP that has linkage$\geq$ *minLink*. There must exist a k-SOAP p_k, of which all elements belong to p_{k+1}. Furthermore, it is frequent and has linkage≥ 1, i.e., p_k must have been discovered. Assume p_k=($c_{[1]}$, ..., $c_{[k]}$), since $c_{[k+1]}$ must be a neighbor of at least one element in p_k, the SOAP p_{k+1} thus will be considered as a candidate and subsequently discovered. It is trivial to show that countInstances() identifies all the instance of p_{k+1}.□

3.6 Analyzing Spatio-temporal Episodes

For each discovered SOAP, its spatio-temporal episodes can be constructed by identifying its associated formation and dissipation events. Such events can be easily derived by checking the the SOAP's presence at a given time. Let λ be the number of episodes associated with SOAP p, then its episodes correspond to λ disjoint time intervals $\{[t_s^i, t_e^i]: 1 \leq i \leq \lambda\}$, where t_s^i and t_e^i mark p's i^{th} formation and dissipation.

We use episodes to address two important issues. First, to make inferences on critical events such as the merging of multiple features. Second, to model how the interactions among a certain set of features evolve over time.

Algorithm: Spatio-temporal Episode Analysis
Input: $\mathbb{E}$: spatio-temporal episodes of different SOAP types

Analysis 1: Infer critical events
1.1 foreach episode $E_p[t_s^i, t_c^i] \in \mathbb{E}$
1.2 Foreach instance I_p^t: $t \in [t_s^i, t_e^i]$
1.3 $A_p^t \leftarrow$ size of the MBB of I_p^t
1.4 Fit a simple linear regression model over (A_p^t, t):$t \in [t_s^i, t_e^i]$

Analysis 2: Model interacting history of F=$\{c_1, \ldots, c_i\}$
2.1 $E_F \leftarrow$ all episodes in $\mathbb{E}$ that are associated with F
2.2 Sort E_F in increasing order of $e.t_s$:$e \in E_F$

Figure 8: Spatio-temporal episode analysis

To address the first issue, we analyze episodes individually. The pseudo-code is described in Figure 8. Let $E_p[t_s^i, t_e^i]$be an episode associated with SOAP p. For each of p's instance appearing during $[t_s^i, t_e^i]$, we compute the size of the instance's minimum bounding box (MBB) (line 1.3), where the MBB of a SOAP instance encompasses every object in the instance. We then apply a simple linear regression model to model the trend that an instance's MBB varies over time (line 1.4). In the model, time is considered to be an explanatory variable, and the size of an instance's MBB as a dependent variable. Inferences can then be drawn based on this trend. For instance, if the MBB of an instance decreases from t_s^i to t_e^i, it is very possible that the involved objects will merge at its corresponding SOAP's dissipation.

To model the interacting behavior among a set of features of interest, denoted as F=$\{c_1, \ldots, c_i\}$, we take the following two step (See Figure 8). We first find all the episodes that are associated with F (line 2.1). These episodes can be associated with different SOAP types. We then order all these episodes by their formation time (line 2.2). The resulting ordered episode sequence is then used to model the interactions among features in F.

3.7 Optimizations

1. Neighborhood-based pruning: This strategy facilitates fast identification of an object's neighbors. To find all the neighbors of an object o in snapshot t_i, the algorithm first identifies o's *neighborhood* in t_i, which is a spherical (or circular) area around o with radius=ε (the distance threshold). The algorithm then only considers the objects in o's neighborhood to locate its neighbors, as all objects outside the neighborhood cannot be o's neighbors. Objects in o's neighborhood can be efficiently located due to the data organization scheme described earlier.

2. Short-circuiting: This strategy targets at eliminating infrequent candidate SOAPs at an early stage. For each candidate SOAP p_{cand} derived from combining two SOAPs of smaller size, p_i and p_j, the strategy works as follows. It first intersects the two intervals $[p_i.m_{min}, p_i.m_{max}]$ and $[p_j.m_{min}, p_j.m_{max}]$, which correspond to the range of snapshot IDs that p_i and p_j are associated with. If the intersected interval has less than *minSupp* snapshots, p_{cand} is infrequent. Otherwise, it proceeds to collect the set of snapshots, denoted as M, that contain both p_i and p_j. If $\|M\| <$*minSupp*, then p_{cand} is infrequent. If p_{cand} is still a candidate at this point, the algorithm proceeds to compute its instances in each snapshot

$t_i \in M$. However, snapshots in M are processed in increasing order of (#instances of p_i)+(#instances of p_j). (The more instances a snapshot has, the more time will be required to generate the instances of p_{cand}.) A counter *occCnt* is maintained to indicate the number of snapshots which contain p_{cand} so far. After some point (e.g., when 25% of the snapshots have been processed), the algorithm starts to regularly compare *minSupp* with the sum of *occCnt* and the number of snapshots in M to be processed. If the sum is less than *minSupp*, p_{cand} is infrequent and can be discarded.

3. Efficient Data Structures: Efficient data structures are also used to make the algorithms more efficient, both in performance and storage requirement. Specifically, we use bit-sets and fast bit-set operations [5] to realize fast spatial-join, which is implemented by the *countInstances* procedure in each algorithm.

Also, the SOAP mining algorithms identify frequent k-SOAPs ($k > 2$) by processing one snapshot (or map) a time. Thus the algorithms can efficiently handle large out-of-core datasets.

4. EXPERIMENTAL EVALUATION

In this section we evaluate our framework on datasets from three different scientific domains, namely, Bioinformatics, Computational Molecular Dynamics and Computational Fluid Flow. We also evaluate the performance and scalability of the framework on large datasets.

4.1 Case Study 1: Protein Datasets

This dataset consists of 8,732 contact maps, corresponding to 8,732 different proteins taken from Protein Data Bank (PDB) [4]. This dataset contains no temporal information. For a protein with N amino acids, its contact map C is a $N \times N$ binary matrix. The position $C(i, j)$ is set to 1 if the distance between the i^{th} and j^{th} residues is less than a threshold and 0 otherwise [21]. We use 6Å as the threshold as suggested in the literature [25]. A *contact map feature* is composed of a set of matrix positions, where each position and at least one of its eight neighbors contain a 1 [23] (see Figure 9 for simple example). A simple region growing approach is used to find the features in contact maps. We then use an entropy-based clustering algorithm to cluster features into l groups (or classes) [6]. Please refer to Yang et al. [23] for more details about contact maps and feature generation.

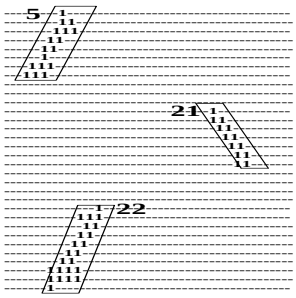

Figure 9: Features extracted from proteins 1a0i (ID from PDB)

A total of 1,009,755 features are extracted from the 8,732 contact maps. These features are clustered into 28 classes. Many of these features correspond to well-known protein secondary structures and have been validated by domain experts. We represent each feature by its minimum bounding parallelogram and label the parallelogram by the feature's class ID. For instance, the three features extracted from protein 1a0i (ID from PDB) are classified into three classes, referred to by their identifiers 5, 21, and 22. Figure 9 shows three features (from one part of a large contact map) represented as parallelograms. Table 1 describes the characteristics of the input dataset.

#maps	#objType	#objects	Avg. #obj/map
8732	28	1,009,755	115

Table 1: Description of Protein Contact Map Dataset

We used the protein structural hierarchy[3] described in the database of Structural Classification of Proteins (SCOP)[4] to evaluate the quality of our results. We find that different types of SOAPs can actually distinguish different protein folding structures. Table 2 lists the β-protein folds that are distinguished by each SOAP type, where SOAPs are generated based on the L-L distance. The folds in bold are those that are associated with only one SOAP type. Whereas other folds in the table are distinguished by two or more SOAP types. Folds in other protein classes such as α-protein show a similar trend. For example, the α-protein fold "Cyclin-like" is primarily associated with sequence SOAPs.

Star	Immunoglobulin-like beta-sandwich Concanavalin A-like lectins/glucanases Trypsin-like serine proteases **Cupredoxin-like** **Acid proteases** **Cysteine proteinases**
clique	Immunoglobulin-like beta-sandwich Concanavalin A-like lectins/glucanases
Sequence	Immunoglobulin-like beta-sandwich Concanavalin A-like lectins/glucanases Trypsin-like serine proteases **Lipocalins** **Nucleoplasmin-like/VP**

Table 2: List of β-protein folds associated with each SOAP type, distType=L-L

4.2 Case Study 2: Molecular Dynamics Simulation Datasets

The Molecular Dynamics (MD) dataset is generated using the Object-oriented High-performance Multi-scale Multi-resolution Simulator for material science (OHMMS) [16]. A slice of Silicon (Si) lattice is simulated to understand the creation of stable structures and dimer rows. The dataset consists of 4000 Silicon slices, corresponding to 4000 snapshots. There are 94 atoms in each slice.

4.2.1 Creation of stable structures

Figures 10(a) and (b) show the slice of Si lattice at *t=0* and at *t=3999* respectively. At *t=0*, there are no stable structures in the lattice. However, stable structures are formed at *t=3999* and can be easily spotted from Figure 10(b). These structures corresponds to the pentagon-like rings formed by five Si atoms. Our framework was able to discover these structures automatically by mining *minLink=1* SOAPs from the dataset. Two atoms are considered as neighbors if the distance between them $\leq$ 2.8Å, which is a value suggested by domain experts. Our framework was able of detect the formation to these structures. These structures were discovered by using minLink SOAP type with *minlink =1*.

[3]We use the first two levels: $l_1 : class$ and $l_2 : fold$ in our experiments

[4]http://scop.mrc-lmb.cam.ac.uk/scop/

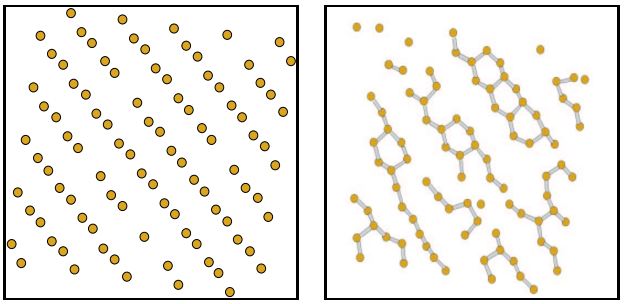

Figure 10: (a)Si surface at t_0 (b)Si surface at t_{3999}

4.2.2 Creation of dimer rows

A dimer is defined as a pair of connected atoms. Please note that not all pairs of bonded atoms qualify as a dimer. *The atoms should be oriented at an angle (around 45 degrees) to be considered a dimer.* Orientation of features is thus very important here and should be considered for mining spatial patterns. A dimer row is created when individual dimers align themselves in a particular fashion. For example, Figure 11(c) shows a detected dimer row (enclosed in the rectangle). The figure also shows other dimer rows, which are at different stages of evolution. Dimer rows were discovered by mining sequence SOAPs. As suggested by domain experts, 2 dimers are considered as neighbors if their distance is $\leq$ 5.0Å. This is our distance threshold.

By discovering different types of spatial association patterns, *minLink* and *sequence* in this case, we were able to find two very different types of meaningful structures in the same dataset. This validates our belief in characterizing different types of interactions by different types of association patterns.

SOAP evolution in this dataset is illustrated in Figure 11(b) and Figure 11(c). Figure 11(b) shows two 2-SOAPs. These SOAPs have no interaction between them, however after few time steps a new SOAP is created between them. This new SOAP acts as a link between the other SOAPs and the whole structure - consisting of 5 dimers- is discovered as a sequence SOAP. This episode points to amalgamation of two features and creation of a new larger one. Such events are automatically discovered by our toolkit.

4.3 Case Study 3: Vortex Simulation Datasets

The vortex dataset is generated by implementing a simplistic version of the algorithm proposed by Christian [7]. The dataset consists of 1970 snapshots, with around 200 vortices in each snapshot. The number of vortices in each frame is changing over time, as an existing vortex can dissipate, new vortex can be created, or two vortices can merge to form a new vortex. Figure 12 shows three simulation snapshots at different times. Figure 12(a) shows the initial configuration of the vortices with no SOAPs. Figure 12(b) shows the creation of a SOAP. This SOAP is formed because the two vortices are moving towards each other and eventually their distance falls in the the distance threshold.

Figure 12(c) shows a very interesting result. The two vortices involved in the SOAP came closer and eventually merged into a new vortex. This event is captured by the SOAP's dissipation (resulting from an amalgamation of two features within the SOAP). Note that SOAPs' dissipation does not necessarily imply dissipation of features. We were also able to make inference on such events by identifying the changing trend of the corresponding instance's MBB.

Figure 13 shows two different types of SOAPs identified in the snapshot taken at t_{991}. One can observe that they capture two very different interactions that can occur to multiple vortices.

4.3.1 Spatio-temporal Episode Evaluation

As mentioned earlier, combining episodes associated with different SOAP types can be used to model the evolving nature of interactions among different features. For the same set of features, the formation of different SOAP types at different time indicates a potential change to their interacting behavior, since different types of SOAPs characterize different types of interactions.

Figure 14 demonstrates how the interactions among four vortices evolve over time and are captured by combing multiple episodes formed at different time. The four vortices form a *clique* SOAP at t_0. This *clique* SOAP continues for 11 time steps and then evolves into a *star* SOAP at time t_{12}, which continues for another 15 time steps (Figure 14(b)). A clique SOAP points to the presence of compact spatial patterns, where each feature is interacting with every other feature. The transition of a clique SOAP to Star implies that features are moving way from each other and may cease to interact after some time. Figure 14(c) demonstrates exactly this behavior.

Other transitions such as from *star* to *clique* are also identified for different sets of features. In the case that a SOAP evolves from star to clique, we observe that features are moving towards each other and thereby increasing the level of interactions. This type of transition can also point to a critical event, namely the merging of two individual features.

4.4 Performance Evaluation

In this section we present the results on scalability of our framework. We also present result highlighting the importance of our optimization schemes. All the experiments were carried out on a Pentium 1.7GHz machine of 1GB main memory. We applied the SOAP mining algorithms to a large Molecular Dynamics dataset produced by OHMMS. The dataset consists of $387,999$ slices of Silicon lattice and is of size 1.5GB. We first identify all the dimers in each slice and then apply the framework to find different types of interactions among dimers.

4.4.1 Scalability Results

The fact that we can process the data incrementally allows the approach to scale well and handle out of core datasets. Figure 15 shows the time taken to discover all types of SOAPs at different support thresholds for the 1.5GB dataset. First of all, even for a large dataset we are able to mine all SOAPs in reasonable amount of time. Time increases as support threshold decreases, because more SOAPs will be generated at a lower support value. However, this increase in time is linear. For example, at 20% support we take 180 seconds to discover all the *star* SOAPs, the time just increases by 50 seconds when it is at a very low support threshold 0.5%. Figure 15 shows the performance results on the 1.5GB dataset. Please note that minLink SOAP subsumes all other SOAP types. As a result the algorithm can compute all the other SOAP types within roughly the same time (negligible overhears) it takes to compute min-link patterns.

4.4.2 Optimization Evaluation

We next present experimental results to show the significance of using different optimizations. The most time-demanding compo-

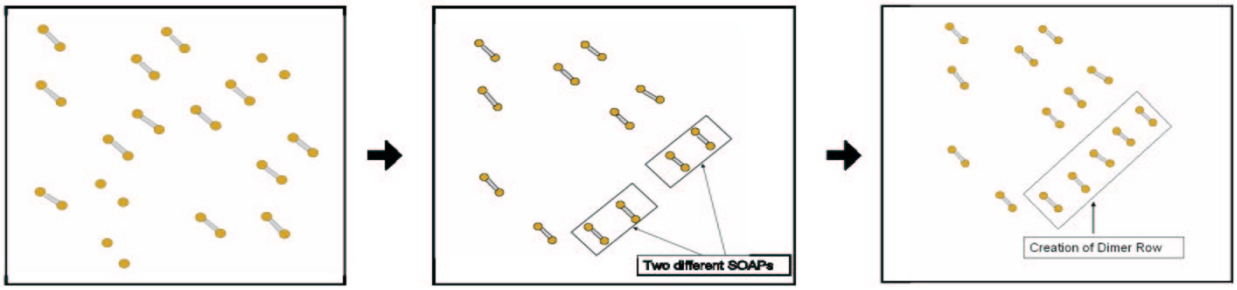

Figure 11: (a)Dimers on Si surface at t_{2000} (b)Dimers on Si surface at t_{2500} (c) Dimer on Si surface at t_{3999}

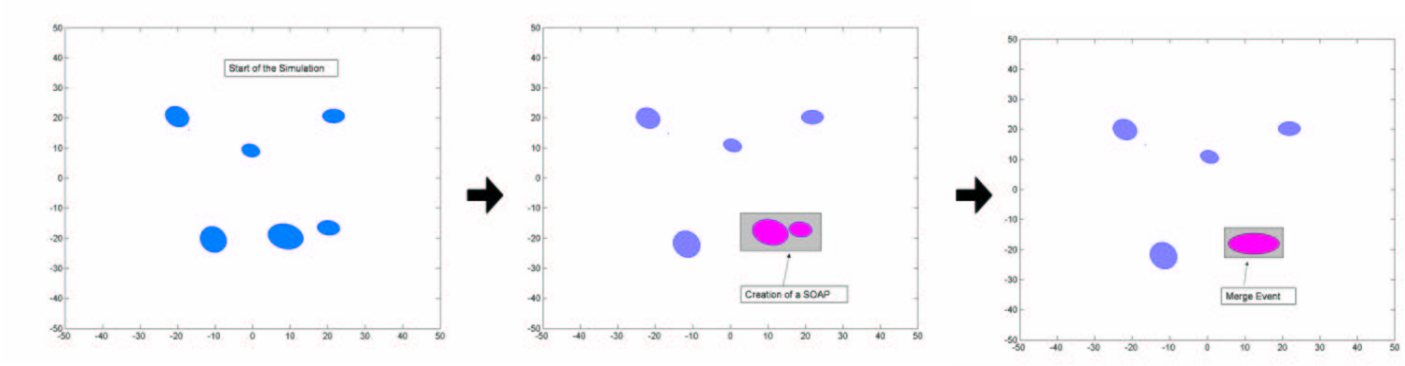

Figure 12: (a)Vortices at t_0 (b)SOAP Formation at t_{90} (c) Vortex Merging at t_{104}

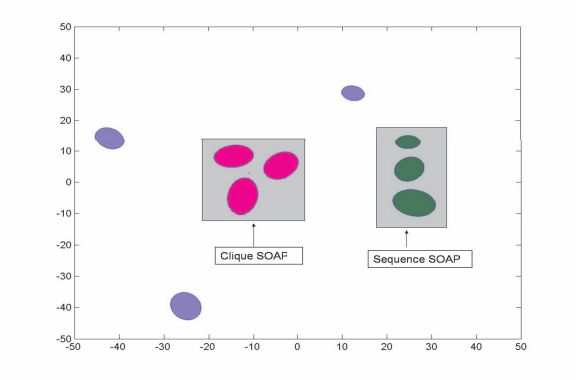

Figure 13: Different Vortex SOAPs are discovered at (t_{991}).

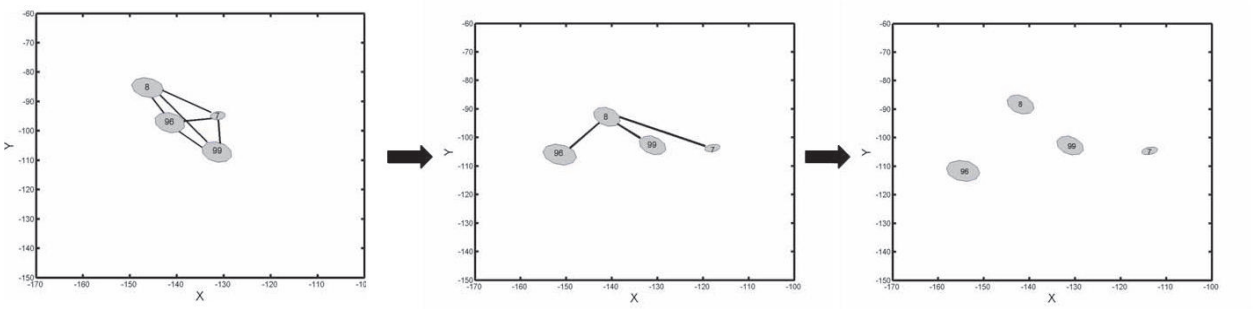

Figure 14: Evolving SOAP: (7 8 96 99) starts as a Clique (t_0), evolves into a Star (t_{20}), eventually out of interaction range (SOAP dissipation)(t_{28})

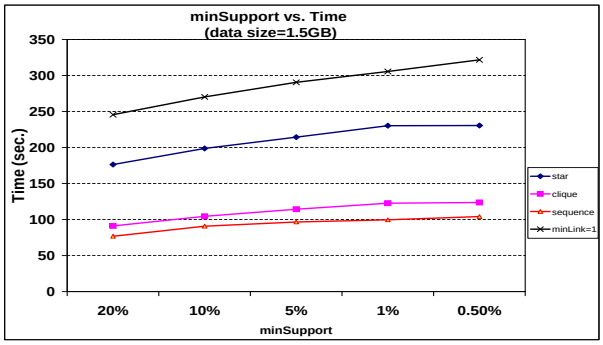

Figure 15: Running Time on 1.5GB Dataset (minDist=5Å)

nent in mining frequent SOAPs is to identify valid SOAP instances for a SOAP candidate in each map (or snapshot). This is especially true when the dataset involves large number of maps. Thus, the main target of our optimization strategies is to prune away infrequent SOAPs as early as possible. Table 3 summarizes numbers of SOAP candidates that are pruned away at an early stage, referred to as *Early-Prunes* in the table. All results are based on the 1.5GB MD dataset with *minSupp= 5%*. The last column in Table 3 lists the ratio between the number of early-pruned candidates and total number of candidates. The average ratio is around 40% which is roughly in line with the savings in computational time.

SOAP Type	# Total	# Early-Prunes	Gain%
STAR	646	221	34.2
CLIQUE	367	165	44.9
SEQUENCE	327	132	40.3

Table 3: Optimization Results

4.5 Impact of Distance Metrics

In order to capture different interactions among evolving objects (or spatial relationships in spatial data), it is critical to use an appropriate distance metric. In this section we present results demonstrating the importance of using distance metric that is capable of taking objects' shape and extent into account. We compare the number of SOAPs generated from using an object-based distance metric and that from the commonly used point-based metric. Due to space constraint, we only present results generated from the most informative shape representation scheme, which is parallelogram in the protein case, and ellipse in the vortex case.

Tables 4-5 show the number of SOAPs discovered based on two different distance metrics on protein datasets, centroid-centroid (C-C) distance in Table 4 and boundary-boundary (B-B) distance in Table 5. SOAPs are further grouped according to their associated protein classes. Using B-B distance, we are able to find more SOAPs, 406 in total as compared to only 235 SOAPs when C-C distance is used. The difference is even more radical in small proteins. With point-point distance only 42 SOAPs were discovered in small proteins. However, it increases to 142 if boundary-boundary distance is used. We would like to point out that most of these identified SOAPs are also verified by domain experts

Similar observations can also be made on vortex datasets. As mentioned earlier, the ellipse based shape representation and boundary-boundary distance make the most sense in this case. To justify this, we compare the numbers of SOAPs discovered by using B-B distance and C-C distance. As shown in Table 6, we are able to

Type	#(α)	#(β)	#(small)	#(peptide)
Star	2	103	19	7
Clique	-	37	11	6
Sequence	4	30	12	4
Total	6	170	42	17

Table 4: SOAPs in major protein groups, dist=C-C

Type	#(α)	#(β)	#(small)	#(peptide)
Star	8	145	91	13
Clique	2	39	26	5
Seq.	6	40	25	6
Total	16	224	142	24

Table 5: SOAPs in major protein groups, dist=B-B

get twice as many SOAPs based on Boundary-Boundary distance compared to Centroid-Centroid distance. Results on MD datasets were very similar in spirit, however due to lack of space we are not presenting the results.

Type	(C-C)	(B-B)
Star	586	1315
Clique	169	329
Seq.	2166	6240
Total	2921	7885

Table 6: #SOAPs Discovered in Vortex Data

5. RELATED WORK

Spatial object association patterns proposed in this work share some commonalities with collocation patterns, which identify features that are frequently located in the same neighborhood. However, most work on collocation pattern mining is limited to applications where subjects of interest can be represented as points [18, 13, 14, 27]. Furthermore, all these points are located in a single map, which correspond to a snapshot in our work. Recently, Xiong *et. al.* proposed an approach to discover collocation patterns for extended spatial objects including line-strings and polygons [22]. For each object, they identify its neighborhood and then consider objects that have overlapped neighborhood as candidate collocation patterns. This is similar to the *Clique* SOAPs defined in this article. However, in their approach they only consider objects located in the same map.

Our research on spatio-temporal association patterns shares some of the objectives with approaches for spatial reasoning. Bailey-kellogg and Zhao [3]. propose a methodology for reasoning about such problems called qualitative spatial reasoning (QSR). Their work is methodology driven and mainly focuses conceptual topics such as data representations and manipulations. They also discuss the use of different spatial primitives to model objects of different shapes and spatial relationships among objects [3]. Our framework is an efficient realization of their conceptual methodology for scientific data. In addition we also support the discovery of spatio-temporal episode patterns that is not explicitly considered in their work. Fernyhough et al. implemented techniques to detect events by identifying frequently occurring spatial relationships [9, 8]. However, their proposed technique only considers pair-wise relationships. Thus interactions involving more than two features will be missed by only considering pair-wise relationships.

6. CONCLUSION

In this paper, we present a general framework for mining spatial associations and spatio-temporal episodes for scientific datasets. Features are modeled as geometric objects rather than points. We de-

fine multiple distance metrics that take into account objects' shape and extent and thus are more robust in capturing the influence of an object on other objects in its spatial neighborhood. We have developed algorithms to discover four different types of spatial object association patterns across multiple maps. We also extend our approach to mine for spatial temporal episodes and thereby present a methodology for reasoning about critical events.

Empirical results on three real case study applications, drawn from the scientific and engineering disciplines serve to validate the framework. We show that the discovered interactions are meaningful and can be used to uncover important spatial and spatio-temporal patterns in the underlying scientific domain. We further demonstrate that the different association pattern types our framework when used in conjunction can provide an effective mechanism to support qualitative spatio-temporal reasoning. Performance studies carried out on large out-of-core datasets indicate that the framework is both efficient and scalable.

We are currently extending the framework to include other types of association patterns, for example, patterns concerning both topological and neighborhood relationships. We are also implementing new approaches towards more robust spatio-temporal inferences and reasoning. Our framework also currently targets the discovery of important frequent spatial interactions, but in many cases rare interactions can also be important. We plan to extend our work to address this limitation.

Acknowledgements: We thank John Wilkins for providing and helping validate the MD results, and thank R. Machiraju, D. Thompson, K. Marsolo and D. Polshakov for valuable comments, discussion and validation of results pertaining to CFD and Protein data.

7. REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[2] M. J. Atallah. A linear time algorithm for the hausdorff distance between convex polygons. *Information Processing Letters*, 17:207–209, 1983.

[3] Chris Bailey-Kellogg and Feng Zhao. Qualitative spatial reasoning: extracting and reasoning with spatial aggregates. *AI Mag.*, 24(4):47–60, 2004.

[4] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank, 2000.

[5] C. Burdick, M. Calimlim, and J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *ICDM 01*, 2001.

[6] C. Cheng, A.W. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *KDD'99*, pages 84–93. ACM Press, 1999.

[7] J.P. Christian. Numerical simulation of hydrodynamics by the method of point vortices. pages 189–197, 1971.

[8] A. G. Cohn and S. M. Hazarika. Qualitative spatial representation and reasoning: an overview. *Fundam. Inf.*, 46(1-2):1–29, 2001.

[9] J H Fernyhough, A G Cohn, and D C. Hogg. Event recognition using qualitative reasoning on automatically generated spatio-temporal models from visual input. In *IJCAI97 Workshop on Spatial and Temporal Reasoning*, 1997.

[10] C. Henze. Feature detection in linked derived spaces. In *In IEEE Conf. on Visualization*, 1998.

[11] M. Jiang, T. Choy, S. Mehta, S. Parthasarathy, R. Machiraju, D. Thompson, J. Wilkins, and B. Gatlin. Feature mining paradigms for scientific data. In *SIAM*, 2003.

[12] S. Mehta, K. Hazzard, R. Machiraju, S. Parthasarathy, and J. Wilkins. Detection and visualization of anamolous structurs in molecular dynamics simulation data. In *In IEEE Conf. on Visualization*, 2004.

[13] Y. Morimoto. Mining frequent neighboring class sets in spatial databases. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 353–358. ACM Press, 2001.

[14] R. Munro, S. Chawla, and P. Sun. Complex spatial relationships. In *The Third IEEE International Conference on Data Mining (ICDM2003)*, 2003.

[15] C. R. Rao and S. Suryawanshi. Statistical analysis of shape of objects based on landmark data. 93(22):12132–12136, Oct. 1996.

[16] D.A. Richie, J. Kim, and J.W. Wilkins. Real-time multiresolution analysis for accelerated molecular dynamics simulations. 2001.

[17] A. Sadarjoen, F.H. Post, and D.C. Banks et al. Selective visualization of vortices in hydrodynamic flows. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 419–422. IEEE Computer Society Press, 1998.

[18] S. Shekhar and Y. Huang. Discovering spatial co-location patterns: A summary of results. *Lecture Notes in Computer Science*, 2121:236+, 2001.

[19] D. Silver and X. Wang. Volume tracking. In Roni Yagel and Gregory M. Nielson, editors, *IEEE Visualization '96*, pages 157–164, 1996.

[20] Waldo R. Tobler. A computer movie simulating urban growth in the detroit region. 1970.

[21] M. Vendruscolo and E. Domany. Recovery of protein folding from contact maps. 1997.

[22] Hui Xiong, Shashi Shekhar, Yan Huang, Vipin Kumar, Xiaobin Ma, and Jin Soung Yoo. A framework for discovering co-location patterns in data sets with extended spatial objects. Apr. 2004.

[23] H. Yang, K. Marsolo, S. Parthasarathy, and S. Mehta. Discovering spatial relationships between approximately equivalent patterns. August 2004.

[24] Kenneth Yip. Structural inferences from massive datasets. In *IJCAI (1)*, pages 534–541, 1997.

[25] M.J. Zaki. Mining protein contact maps. In *BIOKDD 2002*, 2003.

[26] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. Technical Report TR651, 1997.

[27] X. Zhang, N. Mamoulis, D. W. Cheung, and Y. Shou. Fast mining of spatial collocations. In *KDD '04: Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 384–393. ACM Press, 2004.