An Approach to Temporal-Spatial Pattern Detection

Wei Wang, Jiong Yang, and Richard Muntz
Department of Computer Science
University of California, Los Angeles
{weiwang,jyang,muntz}@cs.ucla.edu

November 7, 1998

Abstract

The following approach is used to mine for temporal-spatial patterns in large data sets. We first present a parameterizable model for spatial-temporal sequences, then devise efficient ways to search for parameter values that will result in a good fit to (at least portion of) the data. Metrics for how well the model fits the data are defined which are similar to the support and confidence used in association rule mining and a new metric called *density* is proposed. A user specifies thresholds for these metrics and, based on properties of the class of models we are attempting to fit to the data, the search space can be drastically pruned. Experimental results on real and synthetic data sets demonstrate the efficiency of our algorithm.

1 Introduction

Data mining has became an active research area during recent years. The goal of data mining is to discover "interesting" information from a given large data set. Currently spatial data mining is a topic of increasing interest. As its name indicates, spatial data mining focuses on discovering interesting patterns in spatial data. We assume a spatial database consisting of objects, each with a time sequence of both spatial and non-spatial attributes. In general, spatial attributes are those which are functionally dependent on the location of the object, e.g., its longitude and latitude. All other attributes are non-spatial attributes. The value of both spatial and nonspatial attributes may evolve over time. A strong correlation of the evolution of some subset of these attributes may be very interesting in some applications. For example, if a hostile nation increases its border troops each month over the past six months, then this could be a very interesting evolution.

A model for a general class of temporal-spatial patterns will be defined as one way to present this kind of interesting evolution among spatial and non spatial attributes. The system will determine what instances from this class of patterns appear in the database. One naive approach to automated discovery of spatial patterns is to map the temporal-spatial pattern detection problem to the market basket problem and apply an existing association rule algorithm to find all relationships among attributes. However, this is not feasible due to the following differences between market basket data and temporal-spatial data.

- Number of attributes. In market basket data, each item can be viewed as an attribute and the relationships among attributes are of interest. The number of attributes can be between a few tens to a few millions. However, in a spatial data set, the number of attributes is usually much less than for a market basket application.

- Temporal Evolution. Most work done on mining association rules only considers a snapshot database or one in which time have already been deemed irrelevant. It does not take into account the temporal oriented relationships among attributes. The temporal-spatial pattern model must be general enough to capture relationships among evolutions of attributes.

- Attribute values. In market basket data, the attribute value is binary, 0 or 1. (One means that the item is present in the transaction while zero means that the item is not present in the transaction.) However, for spatial data, the attribute value domain can be a continuous interval over the real numbers. For example, the coordinates of the location of an object could be real numbers. Some attribute domain may be discrete integer. However, even in this case the cardinality of the domain of an attribute in spatial data is usually much larger than the binary domain of an attribute in the market basket data.

- Prior Knowledge about relationships among attributes. In market basket data, no prior knowledge of the data is assumed. Relationships among all attributes are mined. In spatial data, relationships between some attributes are known before the start of the mining process. For example, the distance between a person's house to a big city (e.g., Los Angeles or New York) is functionally dependent on the address of a person's residence. (We assume the location of the big city does not change.) It is useful to note that these existing apriori relationships and to prune the search so that these are not rediscovered.

In this paper, we introduce a new technique to discover temporal-spatial patterns of a certain type. First we separate the derived attributes (e.g., distance to the nearest hospital) from primary attributes (e.g., location of a house and location of the hospital). Only when the primary attributes change, can the derived attributes change. Therefore, we can first mine the relationships among primary attributes, then mine the relationships between primary attributes and derived attributes.

The remainder of the paper is organized as follows. Section 2 briefly discusses the related work. Temporal-Spatial patterns are defined in Section 3. General approach to discover such pattern is presented in Section 4. Section 5 and 6 investigate algorithms for dense region discovery and the valid pattern generation, respectively. Experimental results are shown in Section 7. Finally, conclusions are drawn in Section 8.

2 Related Work

2.1 Association Rules

Association rules were first introduced in a study of the *market basket* problem [Agr93]. The input data to an association rule discovery program consists a set of transactions, each of which consists of a set of items. The number of transactions in the input data file can range from a few thousand to tens of millions and the number of items in a transaction can be a few up to several thousands. The goal of the association rule discovery program is to generate a set of association rules in the form of $x_1 \cap x_2 \cap \ldots \cap x_m \rightarrow y$ where $x_1, x_2, \ldots, x_m$, and y are a set of items and the fraction of occurrences of y in a transaction, given that $x_1, x_2, \ldots, x_m$ appear in that transaction, is greater than a specified threshold. The threshold is called the *minimum confidence* and is usually specified by the user. Also, in order to present only the rules with certain significance, the user has to specify a *minimum support* so that only if the number of transactions containing all $x_1, x_2, \ldots x_m$, and y is greater than the minimum support, will the rule of $x_1 \cap x_2 \cap \ldots \cap x_m \rightarrow y$ appear in the final result.

The confidence measurement may be a little bit misleading because if two items both have very high frequency of occurrence, then the "confidence" that one item occurs given that the other item occurs could be high even if there is no relationship between these two items. For example, if both item x and y appear in 80% of transactions, respectively, then x and y will occur in at least 60% of all transactions even if they are not correlated. In order to solve this problem, researchers have proposed other metrics, such as correlation [Bri97b] and conviction [Bri97a]. Given an association rule $A \rightarrow B$, the correlation for this rule is $\frac{P(A \cap B)}{P(A) \times P(B)}$ while the conviction for this rule is $\frac{P(A) \times P(\neg B)}{P(A \cap \neg B)}$.

2.2 Spatial Trends

Much work has been done in this area in recent years. [Est97] [Est98b] develop a system to detect *spatial trends* and *spatial characterizations*. A spatial trend describes a regular change in one or more non-spatial attributes when moving away from a given starting point, whereas spatial characterization of a set of target objects is a description of the spatial and non-spatial properties which are typical for the target objects but not for the whole database. Not only the properties of the target objects but also the properties of their neighbors are considered during this process. A neighborhood graph is employed to facilitate the mining process. However, the work in [Est97] [Est98b] only considers one snapshot, and not the relationships among attributes and their evolutions.

2.3 Temporal Patterns

A new data mining problem, discovering sequential patterns, was introduced in [Agr95a] and [Agr96]. The input data is a set of sequences, called data-sequences. Each data-sequence is a list of transactions. Typically there is a transaction time associated with each transaction. A sequential pattern also consists of a list of sets of items. The problem is to find all sequential patterns with a user-specified minimum support, where the support of a sequential pattern is the percentage of data-sequences that contain the pattern. However, the attribute values are binary and the sequential patterns are limited in the sense that the class of patterns allow the representing the appearance of items, but not their absence. The inclusion of a user defined calendar is studied in [Ram98]. A user explicitly defines a calendar and interesting patterns are discovered based on the calendar. For example, if a user defines temporal subsequence to start on the days when the US government announces the unemployment rate as this calendar and the calendar is applied to the stock prices in New York Stock Exchange, then some interesting patterns can be discovered relating the reaction of stock prices to these announcements.

3 Temporal-Spatial Pattern

In this paper, we assume that the database consists of a set of objects, each of which has an unique ID and a location, at any instant of time represented, by a point in some space SP. Time varying attributes include

(a) spatial attributes, i.e., coordinates the object's position in SP, and (b) non-spatial attributes (with time stamps). Such attributes are referred to as *primitive object attributes*, and are associated with each object. We will view the database as a sequence of snapshots $S_1, S_2, \ldots$ of the objects and their attributes taken at some frequency. Data of this type can be found in many applications; processed satellite image data (after feature extraction) and census data are two examples. In addition to the primitive object attributes, each object can have a subset of *derived object attributes* which are functions of some subset of primitive object attributes. *Environment attributes* are object attributes derived from a set of primitive object attributes which explicitly includes the spatial attributes of the object. This set of primitive attributes are also referred to as the *base* attributes of the environment attributes. The number of derived object attributes is often much larger than the number of primitive object attributes. Note that only primitive object attributes are captured by the snapshots.

For example, assume the objects are people whose non-spatial attributes may include name, weight, height, salary, rent amount, etc. The spatial attributes store the home address of this person. All these attributes are primitive object attributes. The environment attributes associated with each object might include distance from a coast, distance to a big city, distance to a hospital, average rent and average income in the neighboring area, etc.

We consider a class of temporal-spatial patterns that can be viewed as a correlation among the evolution of object attributes.

Definition 3.1 *Given an attribute A_i, a* **qualification** *$Q(A_i)$ of order m describes the evolution of A_i over $m+1$ snapshots $S(m)$, which is represented as $Q_0(A_i) \rightarrow Q_1(A_i) \rightarrow \ldots \rightarrow Q_m(A_i)$ where $Q_0(A_i)$ is $(A_i \in R_{0_i}, 0)$ where $R_{0_i} = [l_{0_i}, u_{0_i}]$ is the allowed range in the value of A_i in the first snapshot in $S(m)$; whereas at the kth snapshot in $S(m)$ $(0 < k \leq m)$, $Q_k(A_i)$ can be one of the following two types.*

1. **absolute condition***: $(A_i \in R_{k_i}, k)$ where R_{k_i} is the range of allowed values of A_i at the kth snapshot in $S(m)$.*
2. **relative condition***: $(\Delta A_i \in R_{k_i}, k)$ where R_{k_i} is the allowed range of change of A_i and ΔA_i is the observed change from the $(k-1)$th snapshot to the kth snapshot in $S(m)$.*

Note that a *qualification* only describes possible evolutions of an attribute. It does not bind attribute evolution to a given set of snapshots. Neither does it imply that there exists some object whose attribute evolution satisfies the expression. For example, "salary starting at least at \$50,000 and increasing each year for 2 consecutive years" is a qualification of order 2 on *salary*, which can be represented as $(salary \in [\$50,000, \infty), 0) \rightarrow (\Delta salary \in (0, \infty), 1) \rightarrow (\Delta salary \in (0, \infty), 2)$. However, "salary starting at least at \$50,000 in 1990 and increasing each year for 2 consecutive years" is **not** a qualification by definition.

Definition 3.2 *Given a set of attributes $A_1, A_2, \ldots, A_n$, a* **temporal-spatial pattern**[1] *of order m is an implication $Q(A_1) \cap Q(A_2) \cap \ldots \cap Q(A_{n-1}) \Longrightarrow Q(A_n)$ where $Q(A_i)$ is a qualification (of order m) of attribute A_i over the same set of $m+1$ consecutive snapshots.*

As with qualifications, a pattern describes a possible relationship among attributes. The left hand side of a pattern may contain zero or more attribute qualifications whereas the right hand side usually involves only

[1] We refer to "spatial-temporal patterns" as simply "patterns" in the rest of the paper.

one attribute. In this paper, we always assume that a pattern follows this format unless otherwise specified. A pattern is referred to as a *generalized* pattern if its right hand side involves multiple attributes.[2]

In general, for a given set of attributes, many patterns exist simultaneously. We need to identify which of these patterns are of interest. This is addressed in the following definitions which introduce metrics useful in quantifying how interesting a given pattern is.

For any pattern, its **support**, **density**, and **strength** are indications of frequency of occurrence, concentration of population and the degree of non-independence of this pattern with respect to some window size, respectively. The reason for specifying support and strength thresholds is similar to the reason for specifying support and confidence thresholds in association rule discovery. However, due to the nature of temporal-spatial data, users need to specify two more parameters. First, there should be enough samples everywhere within the range specified in the qualifications of a pattern to show that the pattern is valid over the entire range. Thus, the density parameter ϵ introduced for this purpose. Secondly, since snapshots are assumed in the input data. Some users may interested in the rules that may not have enough support if only one snapshot is considered at each evolution step, but will satisfy the support requirement if a larger number of consecutive snapshots are considered. To solve this problem, the user may choose to lower the support threshold, but this may cause another problems: some patterns without enough samples will also be reported. Therefore, the maximum window size is designed to let the user choose how large a set of consecutive snapshots should be considered together for the purpose of computing support, density, and strength. Window size is the number of snapshots that are taken into account to evaluate each evolution step in a pattern. For example, assume that, for a certain company, 10 female employees give birth each month on average. If the snapshot is taken once a month, then the pattern "a mother will come back to work after 6 months of giving birth" might be missed if the minimum support is 100 even though this pattern may be consistently true. However, if the user sets the window to be 12 months, it is likely that this pattern will be valid.

Definition 3.3 *Given a pattern $T : L_T \Longrightarrow R_T$ of order m, a window size $w \geq 1$, and a set of $m + w + 1$ consecutive snapshots $S_j, S_{j+1}, \ldots, S_{j+m+w}$, the* **support** *of T over $S_j, S_{j+1}, \ldots, S_{j+m+w}$ is defined to be*

$$\sum_{1 \leq k \leq w} N_k(T)$$

where $N_k(T)$ is the number of objects satisfying $L_T \cap R_T$ over snapshots $S_{j+k-1}, \ldots, S_{j+m+k-1}$.

For example, there are 10 snapshots total in Figure 1. The support of a pattern of order 3 with window size 3 over snapshot $S_2, S_3, \ldots, S_7$ is computed as follows. N_1, N_2, N_3 are calculated over snapshots $\{S_2, S_3, S_4, S_5\}$, $\{S_3, S_4, S_5, S_6\}$, and $\{S_4, S_5, S_6, S_7\}$, respectively. The support of the pattern is the summation of N_1, N_2, and N_3.

Note that in the case that the window size is greater than 1, an object may be counted more than once if the evolution of this object satisfies $L_T \cap R_T$ more than once at different time during $S_j, S_{j+1}, \ldots, S_{j+m+w}$.

Different methods can be used to capture the degree of non-independence. In this paper, we adopt correlation as the metric to measure the strength of the pattern [Bri97b].

[2] The approach presented in this paper can be easily extended to mine generalized patterns.

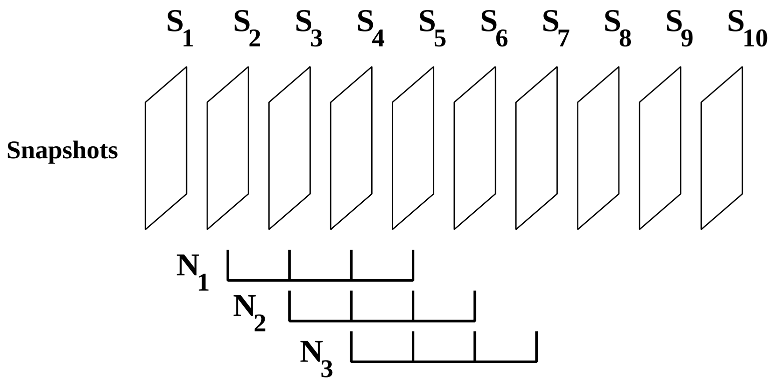


Figure 1: Support and Strength

Definition 3.4 *For a pattern $T : L_T \Longrightarrow R_T$, of order m, a window size $w \geq 1$, and a set of $m+w+1$ consecutive snapshots $S_j, S_{j+1}, \ldots, S_{j+m+w}$, the* **strength** *of T over $S_j, S_{j+1}, \ldots, S_{j+m+w}$ is defined as*

$$\frac{\sum_{1\leq k\leq w} N_k(L_T \cap R_T)}{\sum_{1\leq k\leq w} N_k(L_T) \sum_{1\leq k\leq w} N_k(R_T)}$$

where $N_k(L_T)$, $N_k(R_T)$, and $N_k(L_T \cap R_T)$ are, respectively, the number of occurrences of objects satisfying L_T, R_T, and $L_T \cap R_T$ over snapshots $S_{j+k-1}, \ldots, S_{j+m+k-1}$, respectively.

In the example shown in Figure 1, the snapshots used to calculate N_1, N_2, N_3 are again $\{S_2, S_3, S_4, S_5\}$, $\{S_3, S_4, S_5, S_6\}$, and $\{S_4, S_5, S_6, S_7\}$, respectively. Support and strength are not enough to describe meaningful patterns in the case where a qualification on an attribute can specify any subrange within its continuous domain. As shown in Figure 2(b), if we choose the range specified in the qualification of a pattern on attribute A_1 and A_2 to be their entire domains, then all objects satisfy the qualification, so that this pattern has the highest possible support and strength. However, it is clearly not of interest in any way. To eliminate this kind of pattern, we introduce the concept of *density*.

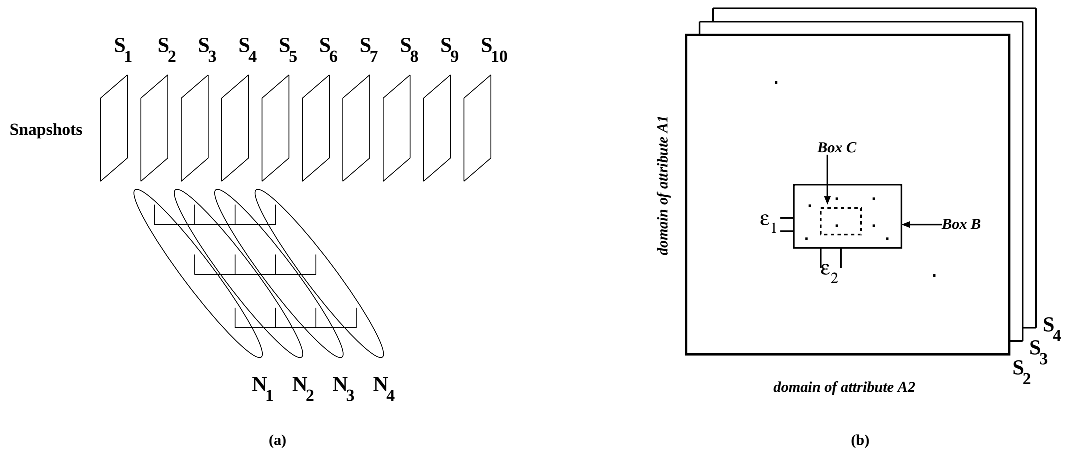


Figure 2: Density

Definition 3.5 *For a pattern $Q(A_1) \cap Q(A_2) \cap \ldots \cap Q(A_{n-1}) \Longrightarrow Q(A_n)$ of order m and window size $w \geq 1$*

over a set of $m+w+1$ *snapshots* $S_j, S_{j+1}, \ldots, S_{j+m+w}$*, we define the* **density** *to be*

$$\min_{0\leq k\leq m}\left(\min_{a_i\in R_{k_i}, 1\leq i\leq n} N_k(a_1, a_2, \ldots, a_n)\right)$$

where R_{k_i} *is the range defined in* $Q_k(A_i)$ *and* $N_k(a_1, a_2, \ldots, a_n)$ *is the sum of the number of objects at the* k*th,* $k+1$*st, ...,* $k+w-1$*th snapshots in* $S_j, S_{j+1}, \ldots, S_{j+m}$*, (i.e.,* $S_{j+k}, S_{j+k+1}, \ldots, S_{j+k+w-1}$*), satisfying, for each attribute* A_i*,*

1. $A_i \in [a_i - \epsilon_i, a_i + \epsilon_i]$ *if the absolute condition is used in* $Q_k(A_i)$ *and*
2. $\Delta A_i \in [a_i - \epsilon_i, a_i + \epsilon_i]$ *if the relative condition is used in* $Q_k(A_i)$

where ϵ_i *is a small constant.*

Intuitively, for each snapshot, the qualifications specified in a pattern can be viewed as a hypercube in a multi-dimensional space. Since the distribution of objects within such a hypercube is usually not uniform, the density here is used to capture the density of the most sparse region and is represented as the minimum number of objects falling into a small interval defined by the parameter ϵ_i. This is motivated by wanting to shrink the qualification condition and avoid having "empty regions" included in these patterns. For example, if the window size is 3, then at each evolution step, 3 snapshots are considered during evaluation of density as shown in Figure 2(a). Then, the density for Box B in Figure 2(b) is 1 because no matter where we move Box C, there exists at least one snapshot among $\{S_2, S_3, S_4\}$ such that at least one object on that snapshot is within Box C (i.e. $N_1 \geq 1$). Box B represents the qualification requirements. The dimensions of Box C are determined by the ϵ_i.

The threshold for N_k and ϵ_i are parameters specified by the user. For example, ϵ_i could be chosen such that the user is indifferent to exact values of attribute A_i in this range. Then the threshold for N_k can be the number of objects that are expected to fall into the indifferent interval for any "interesting" pattern. In general, this threshold should be larger than the expected number of objects which could fall into the interval if the values of attributes were independently and uniformly distributed. The threshold for N_k can be specified as an integer (e.g., 10) or a function of the average density (e.g., $5 \times$ *avg.density*) For example, a user may want to set ϵ_{salary} to be \$1000 on the attribute *salary*. Then the threshold of N_k can be chosen to be, say 12, (i.e. $2 \times$ *avg.density*) or 30 (i.e., $5 \times$ *avg.density*) if the domain of salary is (0, \$500,000) and there are 3000 employees.

In general, a pattern is of interest only if it occurs for a significant number of objects, usually referred as large *support*, and has a certain degree of density and strength[3]. Users are required to specify the desirable thresholds for support, density, strength, and window size.

Definition 3.6 *A* **valid pattern** *is a pattern satisfying the specified requirements of support, density, and strength (within the maximum window size).*

A pattern T may be valid for a period of time, referred to as the *valid period*, which is a maximum set of consecutive snapshots $VP = \{S_v, \ldots, S_w\}$ ($v < w$) such that T is valid for any $m+1$ consecutive snapshots $S_k, \ldots, S_{k+m}$ but is invalid for snapshots $S_{k-1}, \ldots, S_{k+m-1}$ and $S_{w-m+1}, \ldots, S_{w+1}$, where m is the order of

[3] Outlier is another type of interesting pattern which only involves very few number of objects. However, we are only concerned with patterns with relatively high support in this paper due to space limitations

this pattern and $v \le k \le w - m$. The **length** of the valid period VP, denoted by $length(VP)$, is defined as $w - v$.

For example, if snapshots are taken once a year on general information of employees, then $((salary \in [\$50,000, \$70,000], k) \rightarrow (\Delta salary \in [\$10,000, \$20,000], k+1)) \Longrightarrow (\Delta rent\_per\_month \in [0, \$200], k+1)$ may be a pattern of order 2 whose valid period is {1990, 1991, 1992, 1993} with support 12400, density equal to twice of the average density, and strength 1.5. The interpretation of this pattern is that from 1990 to 1993 those employees whose salary were between \$50000 and \$70000 and received a raise between \$10000 and \$20000 moved to a place where the rent is higher by at most \$200 per month within the same year.

Due to space limitations, we only consider the case when the maximum window size is 1 in the rest of this paper. However, all results can be extended to the case where the maximum window size larger than 1. Before introducing some important properties held by patterns, we first define some useful concepts on qualifications and patterns.

Two qualifications of order m on attribute A_i, $Q(A_i) = Q_0(A_i) \rightarrow Q_1(A_i) \rightarrow \ldots \rightarrow Q_m(A_i)$ and $Q'(A_i) = Q'_0(A_i) \rightarrow Q'_1(A_i) \rightarrow \ldots \rightarrow Q'_m(A_i)$, are *compatible* iff $Q_k(A_i)$ and $Q'_k(A_i)$ use the same type of condition for all k where $0 \le k \le m$. $Q(A_i)$ is a *sub-qualification* of $Q'(A_i)$ (and $Q'(A_i)$ is a *super-qualification* of $Q(A_i)$) iff $Q_k(A_i)$ and $Q'_k(A_i)$ are compatible and the range specified in $Q_k(A_i)$ is either the same as or a subrange of that in $Q'_k(A_i)$ for all k where $0 \le k \le m$. $Q(A_i)$ is a *proper* sub-qualification of $Q'(A_i)$ ($Q'(A_i)$ is a *proper* super-qualification of $Q(A_i)$) iff $Q(A_i)$ is a sub-qualification of $Q(A_i)$ and $Q(A_i) \neq Q'(A_i)$. $Q(A_i)$ and $Q'(A_i)$ are *disjoint* iff $Q(A_i)$ and $Q'(A_i)$ are compatible and there does not exist a qualification $Q''(A_i)$ such that $Q''(A_i)$ is a sub-qualification of both $Q(A_i)$ and $Q'(A_i)$.

Given a set of qualifications QS, all of order m, a qualification $q_{min} \in QS$ is a *minimum qualification* of QS iff there does not exist another qualification $q' \in QS, q' \neq q_{min}$ such that q' is a proper sub-qualification of q_{min}. A qualification $q_{max} \in QS$ is a *maximum qualification* of QS iff there does not exist another qualification $q' \in QS, q' \neq q_{max}$ such that q' is a proper super-qualification of q_{max}. The default set of qualifications of order m is the set of all possible qualifications in this paper unless otherwise specified. In this case, any qualification whose specified ranges are points (i.e., only a single value for each attribute is allowed at each step) is a minimum qualification. The qualification which allows the entire attribute domains as the ranges (i.e., attributes can take any value in their domains) is the maximum qualification.

Given a non-minimum qualification q, a set of sub-qualifications of q, is called a *partition of* q iff one of the following conditions is satisfied.

1. $partition(q)$ consists of only the qualification q itself.
2. $\exists\, q_1, q_2$ such that
   (a) the minimum super-qualification of $\{q_1, q_2\}$ is q;
   (b) q_1 and q_2 are disjoint; and
   (c) for any sub-qualification q' of q, either q_1, q' are not disjoint or q_2, q' are not disjoint.

   and $partition(q) = partition(q_1) \cup partition(q_2)$.

A partition of q is illustrated in Figure 3. Assume q is of order 1, which represents the entire box. Q_0 and Q_1 are two evolution steps specified in the qualification of an attribute. q_1 and q_2 are disjoint, any

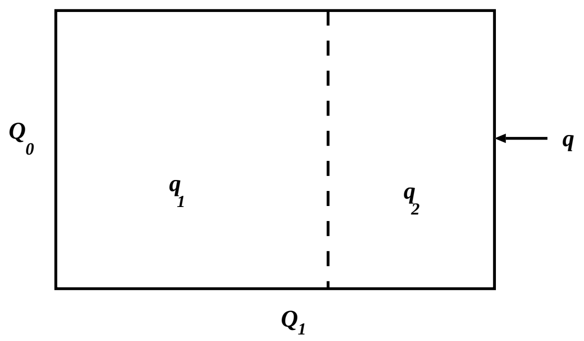


Figure 3: Partition of Qualification q

subqualification of q will intersect with at least one of q_1 and q_2, and q is the minimum superqualification of both q_1 and q_2.

For any two patterns of order m, $T : Q(A_1) \cap Q(A_2) \cap \ldots \cap Q(A_{n-1}) \Longrightarrow Q(A_n)$ and $T' : Q'(A_1) \cap Q'(A_2) \cap \ldots \cap Q'(A_{n-1}) \Longrightarrow Q'(A_n)$, T and T' are *compatible* iff $Q(A_i)$ and $Q'(A_i)$ are compatible for all $1 \leq i \leq n$. T is called a *subpattern* of T' iff for each attribute A_i, $Q(A_i)$ is either the same as $Q'(A_i)$ or a sub-qualification of $Q'(A_i)$. T' is called a *superpattern* of T in such a case. It is obvious that any pattern is a subpattern and a superpattern of itself. T is called a proper subpattern of T' (T' is called a proper superpattern of T) iff T is a subpattern of T' and $T \neq T'$. T and T' are *disjoint* iff T and T' are compatible and there does not exist a pattern T'' such that T'' is a subpattern of both T and T'.

Given a set of patterns TS, a pattern $T \in TS$ is called a *minimum* pattern of TS iff it is not a proper superpattern of any other pattern in TS. Similarly, a pattern $T \in TS$ is called a *maximum* pattern of TS iff T is not a proper subpattern of any other pattern in TS. Again, the default set is assumed to be the set of all possible patterns unless otherwise specified.

Definition 3.7 *Given two patterns T_1 and T_2 where T_1 is a subpattern of T_2 a* **pattern interval** (T_1, T_2) *is the set of all patterns T such that T is a superpattern of T_1 and is a subpattern of T_2. T_1 and T_2 are called the* **subbound** *and* **superbound** *of this pattern interval, respectively.*

Definition 3.8 *Given two pattern intervals $TI_1 = (T_1, T_2)$ and $TI_2 = (T_3, T_4)$, if either T_3 belongs to TI_1 or T_1 belongs to TI_2, then the* **intersection** *of TI_1 and TI_2 is (T_5, T_6) where T_5 and T_6 are minimum common superpattern of T_1, T_3 and maximum common subpattern of T_2, T_4, respectively. Otherwise, their intersection is $\emptyset$.*

Given a non-minimum pattern T, a set of subpatterns of T is called a *partition of* T, and denoted *partition*(T), iff one of the following two conditions is satisfied.

1. *partition*(T) consists of only one pattern, i.e., T.
2. $\exists$ T_1 and T_2 such that
   (a) the minimum super-pattern of $\{T_1, T_2\}$ is T;
   (b) T_1 and T_2 are disjoint; and
   (c) for any subpattern T' of T, either T_1, T' are not disjoint or T_2, T' are not disjoint.

and $partition(T) = partition(T_1) \cup partition(T_2)$.

Figure 4 illustrates the concept of partitioning patterns. In this example, there are three patterns A, B, and C. Two attributes are involved and the regions satisfying the qualifications form rectangles at each snapshot. (If more attributes were involved, they would form a hypercube.) The partition is based on a division of the allowed range on attribute A_2 snapshot 2 so that all minimum patterns flow though one of the partitioned regions. Looking ahead, our search algorithm follows the reverse path of some partitions. This enables us to search all possible candidates exactly once.

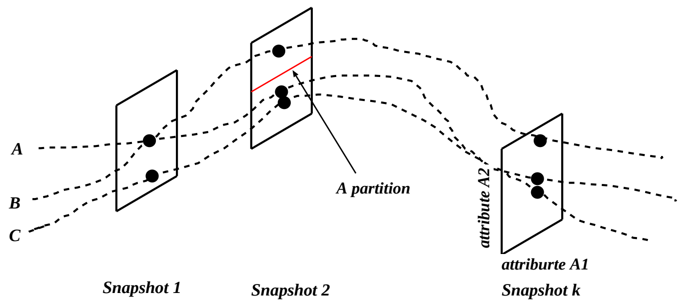


Figure 4: Pattern Partition

There are some important properties held by the class of patterns we have defined that will be useful in pruning the search for valid patterns. In the following discussion assume that all patterns are of order m and over snapshots $S_1, S_2, \ldots, S_s$ where $s \geq m$ unless otherwise specified.

Property 3.1 *Let T be a non-minimum pattern and $partition(T)$ be a partition of T. Then there must exist a pattern $T' \in partition(T)$ such that $strength(T') \geq strength(T)$.*

Proof. The proof is by induction. If $partition(T)$ only consists of one pattern T, then it is trivial that $strength(T) \geq strength(T)$. If $partition(T) = \{T_1, T_2\}$, then only the qualification of one attribute, say A_i, has been subdivided. Since the strength of $T : L \Longrightarrow R$ is defined as

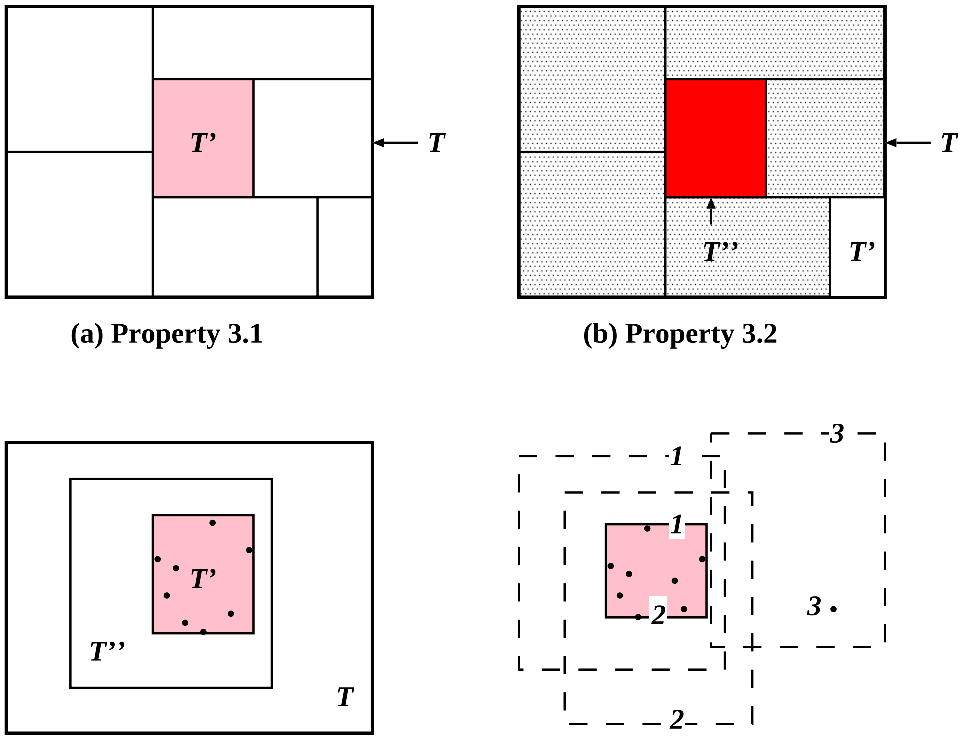
$$strength(T) = \frac{N(L \cap R)}{N(L)N(R)}$$

A_i can be either in the left hand side L or in the right hand side R. Without loss of generality, we assume that A_i is in the left hand side L. Then we have

$$\begin{aligned} strength(T_1) &= \frac{N_1(L \cap R)}{N_1(L)N(R)} \\ strength(T_2) &= \frac{N_2(L \cap R)}{N_2(L)N(R)} \end{aligned}$$

where $N(L \cap R) = N_1(L \cap R) + N_2(L \cap R)$ and $N(L) = N_1(L) + N_2(L)$ by the definition of partition. Without loss of generality, we can assume that $strength(T_1) < strength(T)$. Then

$$\begin{aligned} \frac{N_1(L \cap R)}{N_1(L)N(R)} &< \frac{N(L \cap R)}{N(L)N(R)} \\ N_1(L \cap R)N(L) &< N_1(L)N(L \cap R) \\ N(L \cap R)N(L) - N_2(L \cap R)N(L) &< N(L)N(L \cap R) - N_2(L)N(L \cap R) \\ N_2(L)N(L \cap R) &< N_2(L \cap R)N(L) \\ \frac{N(L \cap R)}{N(L)N(R)} &< \frac{N_2(L \cap R)}{N_2(L)N(R)} \end{aligned}$$



(a) Property 3.1

(b) Property 3.2

(c) Property 3.3 and 3.4

(d) Property 3.5

Figure 5: Illustration of Properties

and hence $strength(T) < strength(T_2)$. Therefore, this property holds for $partition(T) = \{T_1, T_2\}$. Assume that this property holds for $partition(T)$ which consists of $k > 0$ patterns. Then for a partition which consists of $k+1$ patterns, there exist two disjoint sets of subpatterns T_1 and T_2 of T such that $\{T_1, T_2\}$ is a partition of T. Then, either $strength(T_1) \geq strength(T)$ or $strength(T_2) \geq strength(T)$. Without loss of generality, we assume that $strength(T_1) \geq strength(T)$. Since $partition(T_1)$ consists of at most k patterns, there must exist a pattern $T' \in partition(T_1)$ such that $strength(T') \geq strength(T_1)$. Then we have $strength(T') \geq strength(T)$. Thus, this property holds. □

This property is illustrated in Figure 5(a). Informally, this property shows that for any non-minimum pattern, there exists a subpattern which has at least the same strength. Looking ahead, this implies that, in the search algorithm, we can start from a minimum pattern whose strength satisfies the required threshold.

Property 3.2 *Let T be a non-minimum pattern and $partition(T)$ be a partition of T. If there exists a pattern $T' \in partition(T)$ such that $strength(T') < strength(T)$, then there exists another pattern $T'' \in partition(T)$ such that $strength(T'') > strength(T)$.*

Proof. Let T_2 be the subpattern of T and superpattern of T' such that the strength of T_2 is still less than the strength of T but its immediate parent T_1 in the partition of T (i.e. $\{T_2, T_3\}$ is a partition of T_1) has strength greater than or equal to $strength(T)$. (Note that $T_2 = T'$ is possible.) Therefore, $strength(T_2) < strength(T_1)$. By definition of strength and the same algebraic manipulation used in the proof of Property 3.1, $strength(T_3) > strength(T_1) \geq strength(T)$. By Property 3.1, there exists a pattern $T_4 \in partition(T_3)$ such that $strength(T_4) \geq strength(T_3) > strength(T)$. Therefore, this property holds.

□

Figure 5(b) illustrates this property. This property is almost as same as Property 3.1, but with one small difference. If there exists one subpattern whose strength is less than the superpattern, then there exists another subpattern whose strength is strictly greater than the superpattern. This property is not used directly in the search algorithms, but is used to prove the following property.

Property 3.3 *Let T be a non-minimum pattern and Ψ be the set of all minimum proper subpatterns each with strength greater than or equal to $strength(T)$. Let T' be the minimum superpattern of patterns in Ψ. If $T' \neq T$, then $strength(T') > strength(T)$.*

Proof. By Property 3.1, $\Psi \neq \emptyset$. We can always construct a partition of T such that $T' \in partition(T)$. Since all minimum proper subpatterns whose strength is at least that of T are subpatterns of T', then for all minimum subpatterns T_1 of T which are not subpatterns of T', $strength(T_1) < strength(T)$. Therefore, by Property 3.1, for all patterns $T_2 \in partition(T) - \{T'\}$, $strength(T_2) < strength(T)$. By Property 3.2, $strength(T') > strength(T)$. □

In other words, for any pattern T, we find all subpatterns in T whose strength is at least T. Let T' be the minimum superpattern of that set. Then $strength(T') > strength(T)$. This property is illustrated in Figure 5(c) where the minimum trends with strength greater than or equal to $strength(T)$ are represented by dots.

Property 3.4 *Let T be a non-minimum pattern and Ψ be the set of all minimum proper subpatterns where each has strength greater than or equal to $strength(T)$. Let T' be the minimum superpattern of patterns in Ψ. For any pattern T'' such that T' is a subpattern of T'' which is a subpattern of T, $strength(T') \geq strength(T'') \geq strength(T)$.*

Proof. If $T'' = T$, it is trivial that $strength(T'') \geq strength(T)$. Otherwise, we can always construct a partition such that $T'' \in partition(T)$. Since all minimum proper subpatterns with strength at least that of T are subpatterns of T'', then for all minimum subpatterns T_1 of T which are not subpatterns of T'', $strength(T_1) < strength(T)$. Therefore, by Property 3.1, for all patterns $T_2 \in partition(T) - \{T''\}$, $strength(T_2) < strength(T)$. By Property 3.2, $strength(T'') > strength(T)$. Similarly, $strength(T') \geq strength(T'')$. Thus, this property holds. □

This property is very important. It says that the change of strength from a subpattern to its superpattern is monotonically decreasing unless there is another strong pattern present. For example, in Figure 5(c), $strength(T') \geq strength(T'') \geq strength(T)$. In other words, if there exists a supperpattern of T'', denoted by T''', such that $strength(T''') > strength(T'')$, then there has to exist a pattern outside T'', but inside T''' such that its strength is larger than T''. Looking ahead, this property enables us to devise our search algorithm in a way such that once it detects a pattern that does not satisfy the strength requirement, then it skips all its superpatterns until a superpattern includes a new subpattern whose strength is larger than the threshold.

Property 3.5 *Given a threshold of strength α, let Ψ be the set of all minimum patterns with strength at least α. Then the set Θ of all patterns whose strength is at least α consists of a set of pattern intervals, where for each pattern interval (T_1, T_2), there exists a subset of patterns in Ψ, referred to as $\Psi(T_1)$, such that*

1. T_1 is a minimum superpattern of patterns in $\Psi(T_1)$;

2. $strength(T_2) \geq \alpha$;

3. for any minimum pattern $T_3 \in \Psi$, if T_3 is a subpattern of T_2, then $T_3 \in \Psi(T_1)$;

4. for any pattern T_4 such that T_4 is a proper superpattern of T_2 and $strength(T_4) \geq \alpha$, there exists a minimum pattern $T_5 \in \Psi$ such that T_5 and T_2 are disjoint.

Proof. First, we prove that every pattern $T \in \Theta$ belongs to a pattern interval satisfying the above conditions. Let $\Psi(T)$ be the set of minimum patterns whose strength is at least α and T_1 be the minimum superpattern of patterns in $\Psi(T)$. Since $strength(T) \geq \alpha$, there exists a pattern T_2 such that

1. T_2 is a superpattern of T;

2. $strength(T_2) \geq \alpha$;

3. for any minimum pattern $T_3 \in \Psi$, if T_3 is a subpattern of T_2, then $T_3 \in \Psi(T_1)$;

4. for any pattern T_4 such that T_4 is a proper superpattern of T_2 and $strength(T_4) \geq \alpha$, there exists a minimum pattern $T_5 \in \Psi$ such that T_5 and T_2.

It is obvious that T belongs to pattern interval (T_1, T_2).

Secondly, we show that if a pattern T belongs to a pattern interval (T_1, T_2) satisfying the above conditions, then $T \in \Theta$. By Property 3.4, $strength(T) \geq strength(T_2) \geq \alpha$. Therefore, this property holds. □

Intuitively, there are many patterns satisfying the strength requirement. However, these patterns can be represented more compactly as intervals. In turn, each interval is represented by two patterns, a subbound pattern and a superbound pattern where the subbound pattern is a subpattern of the superbound pattern. The subbound pattern for each intervals has to be minimum superpattern of some minimum patterns whose strength satisfies the requirement. Moreover, if any superbound pattern is expanded, then (1) another minimum pattern which meets the strength requirement will become a subpattern of it, or (2) the strength of the superbound pattern falls below the requirement. This ensures that if we start from the subbound patterns and expand until the previous two conditions are encountered, we can find all patterns which satisfy the strength requirement. Figure 5(d) shows 3 pattern intervals satisfying some strength requirement. The solid boxes denote the subbounds while the dash box contour the superbounds. The dots represent those minimum trends with sufficient strength. Our search algorithm will be presented in later sections.

The previous properties concern strength only. However, if the density and support are considered also, then the search process can be improved significantly. For example, if there is some region in the attribute space whose density is below the requirement, then none of its superpatterns will be a valid pattern. This property is illustrated in Figure 5(d).

The next property concerns the valid period. If we find two patterns with the same order which are valid in two different, but overlapped sets of snapshots, then we should combine them together in some way.

Property 3.6 *Let TIS_1 and TIS_2 be the sets of valid pattern intervals of order m over $(S_j, S_{j+1}, \ldots, S_{j+k-1})$ and $(S_{j+1}, S_{j+2}, \ldots, S_{j+k})$ respectively, where $k > m$. Then, the set of valid pattern intervals of order m over $(S_j, S_{j+1}, \ldots, S_{j+k})$ consists of all pattern intervals which are in the non-empty intersection of one pattern interval in TIS_1 and another pattern interval in TIS_2.*

Proof. By definition, any pattern T of order m which is valid over $(S_j, S_{j+1}, \ldots, S_{j+k})$, T must belong to a valid pattern interval TI_1 over $(S_j, S_{j+1}, \ldots, S_{j+k-1})$ and a valid pattern interval TI_2 over $(S_{j+1}, \ldots, S_{j+k})$, respectively. Then, the intersection of TI_1 and TI_2 can not be empty since it must contain T. On the other hand, For any valid pattern interval TI_1 over $(S_j, S_{j+1}, \ldots, S_{j+k-1})$ and TI_2 over $(S_{j+1}, \ldots, S_{j+k})$, if the intersection of TI_1 and TI_2 is not empty, then those patterns which belong to the intersection have to be valid over $(S_j, S_{j+1}, \ldots, S_{j+k})$. Therefore, the property holds. □

This property suggests that we can start from the pattern intervals which are valid on $m + 1$ snapshots (m is the order of the pattern), then we try to find pattern intervals on $m + 2$ snapshots by combining two valid pattern intervals of order m whose valid periods are overlapped.

4 General Approach

Since there are usually many attributes in an application, users are allowed to specify the set of attributes and the maximum order for patterns in order to limit the space of potential patterns and control the complexity of the mining process. The default setting of the interesting attributes is the set of all attributes.

Given a sequence of snapshots, our objective is to mine out a set of patterns that meet specified thresholds for support, density, and strength. Since there are infinitely many subranges for a given continuous range (i.e. attribute domain), there exist infinitely many possible patterns for a given set of attributes and usually only a small subset of them are valid. Searching and evaluating all these candidates is inefficient and not practical. Moreover, the number of valid patterns may be infinite as well. Therefore, instead of finding all valid patterns, our goal is to find subbounds and superbounds of valid pattern intervals since they are more interesting to the users. Any pattern T is a valid pattern iff there exists a valid pattern interval (T_1, T_2) such that T is a subpattern of T_2 and a superpattern of T_1. However, there still could be infinitely many valid pattern intervals. To address this problem, we quantize each numerical attribute domain. Each continuous domain is divided into small intervals of the same length, which is used as the granularity we consider during the mining process. These small intervals are referred to as *indifferent intervals* in the remaining of this paper. Users are allowed to specify the length of such indifferent interval. Different values within the indifferent interval are not distinguished. The results we obtain will be an approximation to the solution with continuous domains.

Mining valid pattern intervals involves two steps.

1. Find region(s) whose density exceeds the threshold specified by the user.

2. Locate all valid pattern interval(s) within such region(s).

Assume that the set of primitive object attributes we consider are $\{A_1, A_2, \ldots, A_p\}$ among which A_1 and A_2 are spatial attributes, e.g., longitude and latitude respectively. In addition, we have a set of e environment attributes $EA_1, \ldots, EA_e$ which are derived from the spatial attributes. Usually, for an object, the value of these attributes will not change unless the corresponding base attributes (i.e., spatial attributes in this case) evolve[4]. A binary indicator is set to indicate whether the spatial attributes change. Let A denote the set of all primitive attributes and indicators.

[4] In this paper, we do not consider those environment attributes that might evolve independently due to space limitations.

For each attribute A_i whose domain is $D(A_i)$, we consider both the *absolute* and *relative* quantification on attributes. We divide both $D(A_i) = [D_{i_l}, D_{i_u}]$ and $D(\Delta A_i) = [D_{i_l} - D_{i_u}, D_{i_u} - D_{i_l}]$ into small intervals $[D_{i_l}, D_{i_l} + d_i), [D_{i_l} + d_i, D_{i_l} + 2d_i), \ldots, [D_{i_l} + \lfloor \frac{D_{i_u} - D_{i_l}}{d_i} \rfloor \times d_i, D_{i_u}]$ and $[D_{i_l} - D_{i_u}, \lfloor \frac{D_{i_l} - D_{i_u}}{d_i} \rfloor \times d_i), \ldots, [-2d_i, -d_i), [-d_i, 0), [0, 0], (0, d_i], (d_i, 2d_i], \ldots, (\lfloor \frac{D_{i_u} - D_{i_l}}{d_i} \rfloor \times d_i, D_{i_u} - D_{i_l}]$ of length d_i, respectively. For example, if the domain of employee's salary is $[0, \$500000]$, and $D(\Delta salary)$ is therefore $[-\$500000, \$500000]$, then the granularity considered could be intervals of length \$1000, which means that $D(salary)$ and $D(\Delta salary)$ are divided into intervals $[0, \$1000), [\$1000, \$2000), \ldots, [\$499000, \$500000]$ and $[-\$500000, -\$499000), \ldots, [\$1000, 0), [0, 0], (0, \$1000], \ldots, [\$499000, \$500000]$, respectively.

Since we do not distinguish values within the same interval, we use $N(x)/d_i$ to approximate the density of such an interval x where $N(x)$ is the number of objects whose value of attribute A_i is in the interval x.

We first mine the correlations among the set of primitive attributes together with the indicator (for spatial attributes) except for the correlations between the indicator and the base attributes it monitors, i.e., the spatial attributes. If a pattern exists between an environment attribute and other primitive attributes, then we expect a pattern to be detected involving the indicator and those primitive attributes. Therefore, we need a way to tell whether there could be a pattern involving environment attributes. For any attribute $A_i \in A$, the evolution from snapshot S_k to S_{k+1} can be represented in two ways.

- absolute condition: $(A_i \in R_{i_k}, k) \rightarrow (A_i \in R_{i_{k+1}}, k+1)$
- relative condition: $(A_i \in R_{i_k}, k) \rightarrow (\Delta A_i \in R_{i_{k+1}}, k+1)$

In both cases, the evolution of an attribute can be mapped to a point in a two dimensional space. After applying the quantization, there are either $\frac{|D(A_i)|^2}{d_i^2}$ or $\frac{2 \times |D(A_i)|^2}{d_i^2}$ distinct evolutions respectively, where $\mid D(A_i) \mid$ is the span of the domain of A_i. For a given $A' \subseteq A$, the domain space is $\Pi_{A_i \in A'} D(A_i) \times (D(A_i) + D(\Delta A_i))$ for a pair of snapshots S_k and S_{k+1}, which is a $2\times \mid A' \mid$ dimensional space, where $\mid A' \mid$ is the cardinality of A'. For example, quantification on salary could be either $[\$100000, \$110000]$ at 1990 to $[\$130000, \$150000]$ at 1991 or $[\$100000, \$110000]$ at 1990 with a $[\$5000, \$10000]$ rise at 1991.

Due to the computational complexity of the problem, we need an approach to mine patterns efficiently without losing much precision. The general scheme for mining patterns of order m from s given snapshots $S_1, \ldots, S_s$ consists of the following three steps.

1. For each $m + 1 (m \geq 1)$ consecutive snapshots $S_k, S_{k+1}, \ldots, S_{k+m}$ where $1 \leq k \leq s - m$, find all valid pattern intervals (of order m).
2. Find all candidate pattern intervals of order m with longer valid periods, then verify them.
3. If any pattern involves the indicator for spatial attributes, all environment attributes of the objects involved are calculated and the pattern will be refined.

We will elaborate each step of this general outline in the immediately following sections.

5 Dense Region Discovery

In this section, we focus on mining patterns of order m with valid period length at least m from a given sequence of s snapshots $S_1, S_2, \ldots, S_s$.

Since the two types of attribute evolution condition (i.e., relative condition and absolute condition) are not totally independent from each other, it is more efficient to organize a coordinated search for regions with object density greater than or equal to the threshold. We can begin by searching dense regions with respect to the relative qualification and use the result to constrain the search space for dense regions with absolute qualification. For example, if $A_{i_k} \in (l_{i_k}, u_{i_k})$ and $\Delta A_{i_{k+1}} \in (l_{i_{k+1}}, u_{i_{k+1}})$, then $A_{i_{k+1}} \in (l_{i_k}+l_{i_{k+1}}, u_{i_k}+u_{i_{k+1}})$. The basic unit for the dense region is a hypercube of size $\Pi_{A_i \in A'} d_i^2$. Counters are used in this process. Similar to large item set counting in the association rule problem, the projection of any dense region in the domain space of A' to the domain space of any subset of A', say B, has to be a dense region. Therefore, instead of searching the whole space, we can facilitate the search by constructing a candidate set of dense regions from the dense regions in the domain space of a smaller subset of attributes.

Any proposed approach to counting large itemsets can be applied to discovering dense regions [Agr93] [Bri97a]. Due to space limitations, we do not elaborate on them in this paper. The basic procedure we use is presented in Algorithm 5.1. If j attributes are considered over $m+1$ snapshots $S_k, \ldots, S_{k+m}$, then the quantized space is of $j \times (m+1)$ dimensions. The basic cell is a hypercube of size $\Pi_{A_i \in A} d_i^{m+1}$ and describes a possible object evolution with respect to attributes in A. For an object o, $cell(o, A)$ denotes the cell (in such a quantized space) which describes the evolution of o (with respect to attributes in A) over snapshots $S_k, \ldots, S_{k+m}$. D_j and C_j represent the sets of dense cells and candidate dense cells involving j attributes, respectively.

Algorithm 5.1 *Find Dense Regions*

DENSE_REGION($S_k, S_{k+1}, \ldots, S_{k+m}, DENSITY$)
/* *find the dense region(s) for snapshots* $S_k, S_{k+1}, \ldots, S_{k+m}$ */
{
  $D_1 \leftarrow \emptyset$
  $C_1 \leftarrow$ *the set of all cells involving only one attribute*
  Initialize the counter for each cell in C_1 *to 0*
  for $j \leftarrow 1$ **to** p **do** /* p *is the total number of attributes* */
    for *each object* o
      for *each set of* j *attributes* A
        if $cell(o, A) \in C_j$
        then $cell(o, A).counter \leftarrow cell(o, A).counter + 1$
    for *each cell* $c \in C_j$
      if $c.counter \geq$ **Density**
      then $D_j \leftarrow D_j \cup c$
    if $D_j = \emptyset$
    then return
    $C_{j+1} \leftarrow$ *Construct* $(j+1,\ D_j)$
}

Construct$(j+1, D_j)$ builds C_{j+1} from D_j. This procedure is similar to that used in counting large item sets.

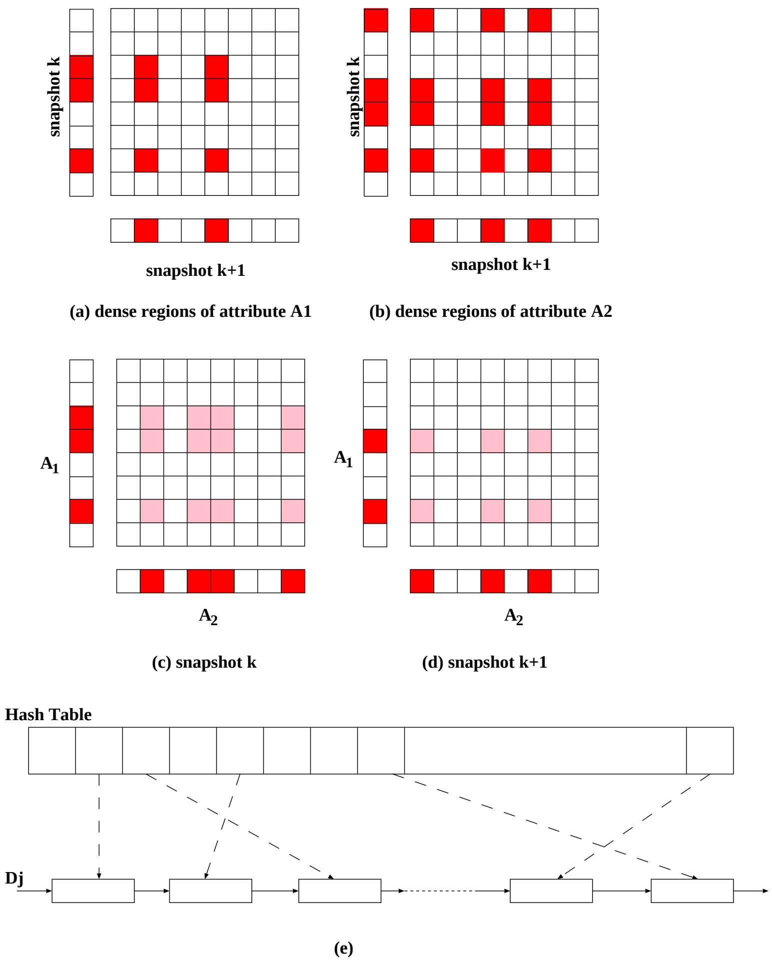


Figure 6: Constructing C_{j+1} from D_j

As an example of Algorithm 5.1, consider 2 snapshots S_k and S_{k+1} and two attributes A_1 and A_2. Figure 6(a) and (b) illustrate the dense regions (indicated by dark shaded cells) in D_1 involving A_1 and A_2, respectively. The candidate set C_2 for the dense region involving both A_1 and A_2 are the "cross product" of the dense regions shown in Figure 6(a) and (b). The projections of C_2 on snapshot S_k and S_{k+1} are shown (indicated by light shaded cells) in Figure 6(c) and (d) respectively. Again, many strategies have been proposed for this task. Here we just explain a simple way to build C_{j+1}. In this approach, D_j is always maintained in a sorted order. We go though the list D_j, and for each cell c in D_j, if for each cell c' in D_j following c is sorted order and c' differs from c by one attribute, we create a new cell c'' which consists of all attributes involved in both c and c'. Then we find all cells in D_j that are different from c'' by one attribute. A hash table is employed to index all cells in D_j as shown in Figure 6(e). If there are $j-1$ such cells (excluding c and c'), then we make c'' a candidate cell in C_{j+1}.

6 Valid Pattern Generation

For each dense region involving j attributes $A_1, A_2, \ldots, A_j$, for each possible format $Q(A_1) \cap Q(A_2) \cap \ldots \cap Q(A_{k-1}) \cap Q(A_{k+1}) \cap \ldots \cap Q(A_j) \Longrightarrow Q(A_k)$, we search the set of possible valid pattern intervals. Given such an attribute A_k serving as the right hand side, only one pattern corresponds to each cell, whose strength can be calculated uniquely. In the following discussion, *pattern*(c) and *cell*(T) are used to denote the corresponding pattern of cell c and the corresponding cell (or supercell) of pattern T, respectively. The finest pattern we consider is the pattern generated from only one dense minimum cell. We regard these patterns as minimum patterns. By the properties of pattern, any valid pattern is a superpattern of at least one such finest pattern whose strength is also at least the required strength. Based on this fact, we begin from the set of dense cells whose strength is at least the required strength, referred to as DC. Let DS and DT be the set of minimum supercells of a subset of cells in DC (i.e., an element of the powerset of all cells in DC) and the set of patterns whose corresponding cell is in DS and whose strength is at least the required strength, respectively. For each pattern T in DT, we search for its minimum valid superpatterns which can serve as the subbounds of some valid pattern intervals. Breadth first search is employed. Beginning from *cell*(T), the range is expanded in one direction for one dimension by one unit at each step, within this dense region, until the number of objects is greater then the support while the strength is still greater than the requirement. Then this pattern, referred as to T_{sub}, is taken as a subbound for valid pattern interval(s). In order to find the superbound(s), searching continues in the same manner until a pattern T_{super} is found such that all of its superpatterns either violate the strength requirement or another dense cell in DC becomes included. Then the pattern T_{super} is taken as a superbound of the valid pattern interval. Note that multiple superbounds might exist for the same subbound. For each such superbound T_{super}, a valid pattern interval is constructed as (T_{sub}, T_{super}). Any pattern belonging to this interval is guaranteed to be valid. However, during the expansion process, if the strength of the candidate pattern falls below the requirement before the support is met or the number of objects is still below the support when another dense cell in DC is met, then there is no valid pattern containing this range. This process repeats on every pattern in DT. It follows from the properties of patterns that this procedure can find all valid patterns. Algorithms to search valid pattern intervals are presented in Algorithm 6.1.

Algorithm 6.1 *Find Valid Pattern Interval*

Valid_Pattern_Interval (D_j, $SUPPORT$, $STRENGTH$)
{
  $result \leftarrow \emptyset$
  for *each dense region dr in* D_j
    for $k \leftarrow 1$ **to** j **do**
    /**mining pattern of* $Q(A_1) \cap Q(A_2) \cap \ldots \cap Q(A_{k-1}) \cap Q(A_{k+1}) \cap \ldots \cap Q(A_j) \Longrightarrow Q(A_k)$ */
      $DC \leftarrow$ *set of cells whose strength are at least* $STRENGTH$
      $DS \leftarrow$ *set of minimum supercells of any subset of cells in* DC
      for *each* $c \in DS$ **do**
        if $pattern(c).strength \geq STRENGTH$
        then $result \leftarrow result \cup Exam(c, dr, SUPPORT, STRENGTH)$
  return *result*
}
Exam (c, dr, $SUPPORT$, $STRENGTH$)

```
{
    result ← ∅
    subbound ← Mine_subbound((c, dr, SUPPORT, STRENGTH)
    for each sb ∈ subbound do
        superbound ← Mine_Superbound(c, dr, DC, SUPPORT, STRENGTH)
        for each spb ∈ superbound do
            result ← (sb, spb)
    return result
}
Mine_Subbound (c, dr, SUPPORT, STRENGTH)
{
    subbound ← ∅
    if c.support < SUPPORT
        Candidate ← Expand(c)
        for each c' ∈ Candidate do
            if within(c', dr) and pattern(c').strength > STRENGTH and pattern(c').support > SUPPORT
            then subbound ← subbound ∪ Mine_Subbound(c', dr, SUPPORT, STRENGTH)
            else if pattern(c').strength > STRENGTH and c'.support > SUPPORT
            then subbound ← subbound ∪ pattern(c')
    return subbound
}
Mine_Superbound (c, dr, DC, SUPPORT, STRENGTH)
{
    superbound ← ∅
    if pattern(c).strength > STRENGTH
        Candidate ← Expand(c)
        for each c' ∈ Candidate do
            if within(c', dr) and c'.strength > STRENGTH and exclude(c, c', DS)
            then superbound ← superbound ∪ Mine_Superbound(c', dr, SUPPORT, STRENGTH)
        if superbound = ∅
        then superbound = pattern(c)
    return superbound
}
```

After obtaining pattern intervals whose valid period is of length m where m is the order of the pattern, we can find all patterns with longer valid period. Candidates for pattern interval of order m whose length of valid period is k ($k > m$) can be constructed from pattern intervals of order m whose length of valid period is $k-1$. It is obvious that a valid pattern interval TI over snapshots $S_j, S_{j+1}, \ldots, S_{j+k+1}$, has to be an intersection of two existing valid pattern intervals over $S_j, S_{j+1}, \ldots, S_{j+k}$, and S_{j+1}, S_{j+k+1} respectively because patterns belonging to TI have to be valid over both $S_j, S_{j+1}, \ldots, S_{j+k}$ and S_{j+1}, S_{j+k+1}. Therefore, if there are n_j valid pattern intervals over $S_j, S_{j+1}, \ldots, S_{j+k}$ and n_{j+1} valid pattern intervals over $S_{j+1}, \ldots, S_{j+k+1}$, then the number of candidate pattern intervals over $S_j, S_{j+1}, \ldots, S_{j+k+1}$ is at most $n_j \times n_{j+1}$. We only need to verify whether these candidate pattern intervals are valid during $S_j, S_{j+1}, \ldots, S_{j+k+1}$.

The formal presentation of the algorithm is in Algorithm 6.2. $Valid\_Period(TS, m, s)$ returns the set of valid pattern intervals of order m with longer valid period over s snapshots. $Valid\_Intersection$ returns the set of valid pattern intervals of patterns in TS_1 and TS_2 over snapshots $S_{j_1}, \ldots, S_{j_2}$. $min\_Superpattern(t_1, t_2)$ and $max\_Subpattern(t_1, t_2)$ return the minimum common superpattern and maximum common subpattern of t_1 and t_2, respectively.

Algorithm 6.2 *Valid Period*

```
Valid_Period (TS, m, s)
{
   currentTS ← TS /* TS is an array of sets of valid patterns where each TS[k]
                     is a set of patterns valid from snapshot S_k */
   Result ← TS
   for j ← m to s do
      for k ← 1 to s − j do
         Result_j[k] ← Valid_Intersection(currentTS[k], currentTS[k + 1], m, k, k + j)
         Result ← Result_j[k]
         currentTS ← Result_j
   return Result
}
Valid_Intersection (TS_1, TS_2, m, j_1, j_2)
{
   Result ← ∅
   for every two pattern intervals (t_1, t_2) ∈ TS_1 and (t_3, t_4) ∈ TS_2 do
      if t_1 is a subpattern of t_4 or t_3 is a subpattern of t_2
      then
         t_5 ← min_Superpattern(t_1, t_3)
         t_6 ← max_Subpattern(t_2, t_4)
         if t_5 is a subpattern of t_6
         then result ← result ∪ (t_5, t_6)
   return Result
}
```

By Property 3.6, this procedure guarantees to explore all valid patterns.

Claim 6.1 *Given a set of valid patterns of order m whose valid period is of length m based on a set of snapshots $S_1, S_2, \ldots, S_s$, Algorithm 6.2 returns all valid patterns of order m with longer valid period.*

If any pattern involves the indicator for environment attributes, then the value of the environment attributes for those involved objects are calculated. Patterns can be refined by a similar algorithm as presented above. However, the search space is much smaller in this case. As we will see in experimental results that this is much more efficient than treating environment attributes the same as primitive attributes.

7 Experimental Results

We have implemented a prototype of our pattern discovery algorithm in C. The experiments were run on an Ultra Sparc10 workstation with 128 MB main memory, one 300MHZ CPU, and running Solaris 2.5. Our pattern detection algorithm consists of two phases, (1) dense region discovery, and (2) valid pattern discovery. We apply our search algorithm on different data sets, synthetic and real. We discovered that on average, 40% of the all execution time is spent on the first step while 60% time is spent on the second step. A significant amount of time has been spent in the first step. However, the first step is similar to the large item set counting and much research has been done on this area. We therefore omit the analysis of the former phase, and rather focus on the latter one. In this section, we chose correlation as the metric of the strength.

7.1 Efficiency Analysis

We first compare our algorithm with an algorithm that exhaustively searches the entire space. For the purpose of testing the performance of our algorithm with respect to the specified minimum support, we synthetically generated one data set. The synthetic data set contains 50,000 objects and there are 7 primary attributes associated with each object and 15 snapshots overall. 20% of all objects are uniformly distributed on all attributes while the remaining 80% are used to embed some specific patterns. The length of an indifferent interval was set to 1% of the domain length and the density threshold was set to $2 \times avg.density$. Support varies from 2% to 10%. Figure 7(a) shows the ratio of the number of reported valid patterns vs. the number of patterns that are examined and the ratio of the number examined patterns vs. the number of total possible patterns (that could be present in the dense region discovered in phase 1) with respect to the support threshold. It is evident that all patterns searched by our search algorithm are in the final result, which means that our search algorithm did not waste too much time. On the other hand, our search algorithm only searches about 1% of all patterns in the dense region.

Figure 7(b) shows how the above two ratios vary with respect to the strength threshold. The support is fixed at 4% and the strength varies between 1.2 to 1.6. Again almost all patterns searched by our algorithm are valid. In the next section, we will show our algorithm achieves a high recall ratio.

7.2 Recall versus Performance

The indifferent interval can be adjusted to speed up the process. However, this can degrade the results. In this section, we analyze the degradation as we vary the sizes of the indifferent intervals. Three synthetic data sets were generated, each of which consists of 100,000 objects and 10 snapshots. Each object has 5 attributes. We embedded 500 patterns up to order 5 in each data set. Figure 8 describes the variation of the average overall response time versus the recall of the results. The recall is represented by the percentage of embedded patterns that are reported by our search algorithm. (The precision of our algorithm is 100%, i.e., all reported trends are valid.) The indifferent intervals are chosen as a fraction of the attribute domain. The

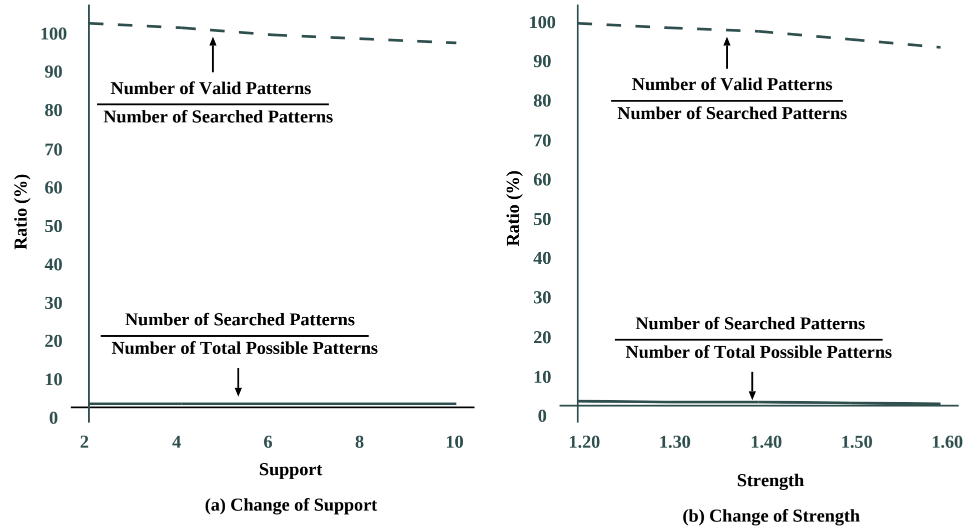


Figure 7: Efficiency of the Search Algorithm

number indicated on the curve represents the length of the indifferent interval as a fraction of the total span of the attribute domain. In this experiment, the density, support, and strength are chosen as $3 \times avg.density$, 10%, and 1.3, respectively.

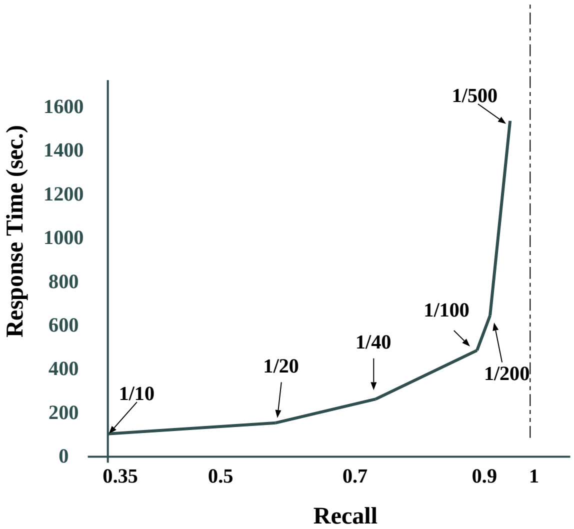


Figure 8: Response Time vs. Recall

It is clear that when indifferent intervals are large, a significant number of patterns are missing. With smaller intervals, the recall improves, the response time degrades. However, when we choose the length of an indifferent interval to be 1% of that of attribute domain, the response time is still within an acceptable length, but 90% recall can be achieved.

7.3 Environment Attributes

The previous experiments dealt with primary attributes. In this section, we report on experiments on with environment attributes. As mentioned before, environment attributes are treated differently from the primary attributes. The effect is illustrated in Figure 9. A synthetic data set is generated for this experiment. The data set consists of 15 snapshots, 100,000 objects. Each object has five primary attributes. The number of environment attributes varies from 1 to 4. Let Res_e and Res_p be the response time when treating environment attributes in the same way as normal primary attributes, and the response time when treating environment attributes via the indicator from primary ones, respectively. Then the speed up factor is defined as $\frac{Res_e - Res_p}{Res_p}$. The result in Figure 9 shows over 10 times speed up is achieved by treating these two kinds of attributes differently as was described in Section 6.

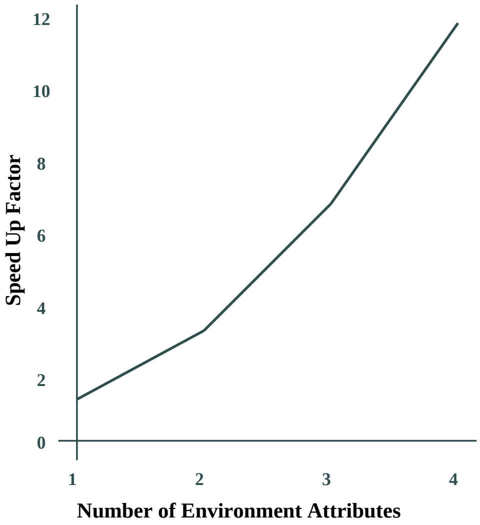


Figure 9: Environment Attributes

7.4 Real Data Set

We also ran our mining algorithm on a real data set. Each object represents a person. The primary attributes are the age, the title of that person, the salary of that person, family status (single, married, head of household), and his house location (zip code). There are two environment attributes, the distance between the person's house and his/her working address, and the distance between the person's house and a major city. All the distances are computed based on the zip codes. We select all attributes as interesting. There are 20,000 objects and 10 snapshots. The snapshot is taken once a year, from 1986 to 1995. We set the indifferent interval to be 1% of the domain range and the thresholds for support, density, and strength to be 3% (i.e., 600 objects), $2 \times avg.Density$, and 1.3, respectively.

It takes approximately 260 seconds to mine the patterns. 347 pattern intervals are discovered total. Due to space limitation, we only can present a few interesting patterns. One pattern discovered is that "People receiving a raise tends to move further away from the big city from 1987 to 1993." This pattern may be explained as the price of a house in the suburb area is more expensive and the area is more safe during 1987 to 1993. The other is that "If people with salary range between \$70,000 to \$100,00 and they get a raise, the range of the raise will likely be from \$7,000 to \$15,000 from 1986 to 1995."

8 Conclusion

We introduce an approach to detect tempo-spatial pattern given a sequence of snapshots of the spatial database. We take into account not only the support and strength, but also the density as well. Our pattern discovery algorithm does not require the user to provide any specific information on what attributes are related and how they are related. Also we separate the derived attributes from the primary ones so that an additional large speed up is achieved. Experimental results show that our discovery algorithm can efficiently discover temporal-spatial patterns embedded in the data set.

References

[Agr93] R. Agrawal, T. Imilienski, and A. Swami. Database mining: a performance perspective. *IEEE Trans. on Knowledge and Data Engineering*, 5(6):914-925, 1993.

[Agr95a] R. Agrawal, R. Srikant. Mining Sequential Patterns. *Proc. of the Int'l Conference on Data Engineering (ICDE)*, Taipei, Taiwan, March 1995.

[Agr95b] Rakesh Agrawal and Giuseppe Psaila. Active data mining. *Proc. 1st Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, 3-8, 1995.

[Agr96] R. Srikant and R. Agrawal. Mining sequential patterns: generalizations and performance improvements. *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT)*, Avignon, France, March 1996.

[Bri97a] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic Itemset counting and implication rules for market basket data. *Proc. ACM SIGMOD Conf. on Management of Data*, 1997.

[Bri97b] S. Brin, R. Motwani, C. Silverstein. Beyond market baskets: generalizing association rules to correlations. *Proc. ACM SIGMOD Conf. on Management of Data*, 1997.

[Est96] Martin Ester, Hans-Peter Kriegel, Jrg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 226-231, 1996.

[Est97] Martin Ester, Hans-Peter Kriegel, and Jrg Sander. Spatial data mining: a database approach. *Proc. 5th Intl. Symp. on Spatial Databases (SSD)*, 47-66, 1997.

[Est98a] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. *Proc. 24th Intl. Conf. on Very Large Data Bases (VLDB)*, 1998.

[Est98b] M. Ester, A. Frommelt, H.-P. Kriegel, and J. Sander. Algorithms for characterization and trend detection in spatial databases. *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 1998.

[Fel97] Ronen Feldman, Yonatan Aumann, Amihood Amir, and Heikki Mannila. Efficient algorithms for discovering frequent sets in incremental databases. *Proc. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, 59-66, 1997.

[Gal97] Antony Galton. Continuous change in spatial regions. *Proc. 3rd Intl. Conf. on Spatial Information Theory (COSIT)*, 1-13, 1997.

[Gru98] Stephane Grumbach, Philippe Rigaux, and Luc Segoufin. The DEDALE system for complex spatial queries. *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 213-224, 1998.

[Han97] Jiawei Han, Krzysztof Koperski, and Nebojsa Stefanovic. GeoMiner: a system prototype for spatial data mining. *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 553-556, 1997.

[Hor97] K. Hornsby and M. J. Egenhofer. Qualitative representation of change. *Proc. 3rd Intl. Conf. on Spatial Information Theory (COSIT)*, 15-33, 1997.

[Kno96a] Edwin M. Knorr and Raymond T. Ng. Finding aggregate proximity relationships and commonalities in spatial data mining. *IEEE Trans. on Knowldge and Data Engineering*, 8(6): 884-897, 1996.

[Kno96b] Edwin M. Knorr and Raymond T. Ng. Extraction of spatial proximity patterns by concept generalization. *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 347-350, 1996.

[Kno97] Edwin M. Knorr, Raymond T. Ng, and David L. Shilvock. Finding boundary shape matching relationships in spatial data. *Proc. 5th Intl. Symp. on Spatial Databases (SSD)*, 29-46, 1997.

[Kop96] Krzysztof Koperski, Junas Adhikary, and Jiawei Han. Spatial data mining: progress and challenges. *SIGMOD Workshop on Research Issues on data Mining and Knowledge Discovery (DMKD)*, 1996.

[Ng94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. *Proc. 20th Intl. Conf. on Very Large Data Bases (VLDB)*, 144-155,1994.

[Ram98] Sridhar Ramaswamy, Sameer Mahajan, and Abraham Silberschatz. On the discovery of interesting patterns in association rules. *Proc. 24th Intl. Conf. on Very Large Data Base (VLDB)* 1998.

[Wan97] Wei Wang, Jiong Yang, and Richard Muntz. STING: a statistical information grid approach to spatial data mining. *Proc. 23rd Intl. Conf. on Very Large Data Bases (VLDB)*, 186-195, 1997. http://dml.cs.ucla.edu/~weiwang/paper/VLDB97.ps

[Wan99] Wei Wang, Jiong Yang, and Richard Muntz. STING+: an approach to active spatial data mining. to appear *Proc. 15th Intl. Conf. on Data Engneering (ICDE)*, 1999.

[Wid96] Jennifer Widom and Stefano Ceri. *Active Database Systems*, Morgan Kaufmann, 1996.

[Zai98] Osmar R. Zaiane, Jiawei Han, Ze-Nian Li, Sonny H. Chee, and Jenny Chiang. MultiMediaMiner: a system prototype for multimedia data mining. *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 581-583, 1998.

[Zan97] Carlo Zaniolo et al. *Advanced Database Systems*, Morgan Kaufmann, 1997.

[Zha96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 103-114, 1996.