**Categorization of visualization methods: Pixel-oriented** (m dimensions-m graphs or 1 graph-m fan-out dimensions)**, Geometric Projection** (Direct Visualization, Scatterplot and scatterplot matrices, Landscapes, Projection pursuit technique: Help users find meaningful projections of multidimensional data, Prosection views, Hyperslice, Parallel coordinates)**, Icon-based** (Chernoff Faces, Stick Figures)**, Hierarchical** (Dimensional Stacking, Worlds-within-Words, Tree-Map, Cone Trees, InfoCube)**, Visualizing complex data and relations** (Tag Cloud, social and information networks (non-numerical data) )

**Similarity, Dissimilarity (or distance), Proximity:**

**Minkowski distance:** d(i,j) = , where i = (xi1, xi2, …, xil) and j = (xj1, xj2, …, xjl)

d(i, j) > 0 if i ≠ j, and d(i, i) = 0 (Positivity), d(i, j) = d(j, i) (Symmetry), d(i, j) ≤ d(i, k) + d(k, j) (Triangle Inequality)

p = 1: (L1 norm) Manhattan (or city block) distance

p = 2: (L2 norm) Euclidean distance

p → ∞: (Lmax norm, L∞ norm) “supremum” distance

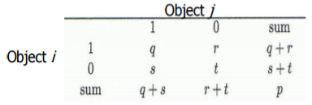
**Proximity Measure for Binary Attributes:**

**Distance** measure for **symmetric binary variables**: d(i,j) =

**Distance** measure for **asymmetric binary variables**: d(i,j) = (compare similar diseases with a set of medical tests)

**Jaccard coefficient**:

**Coherence(i,j)** =

**Proximity Measure for Categorical Attributes:**

Method1: **Simple matching** ( m: # of matches, p: total # of variables): d(I,j) =

Method2: **Use a large number of binary attributes**

**Ordinal Variable** (Can be discrete or continuous):

**Replace** an ordinal variable value by its rank: . **Map** the range of each variable onto [0, 1] by replacing i-th object in the f-th variable by ,

**Example:** freshman: 0; sophomore: 1/3; junior: 2/3; senior 1. Then distance: d(freshman, senior) = 1, d(junior, senior) = 1/3

**Attributes of Mixed Type:**

d(i,j) = , If f is **numeric**: Use the normalized distance. If f is **binary** or **nominal**: dij (f) = 0 if xif = xjf; or dij (f) = 1 otherwise. If f is **ordinal**, Compute ranks zif. Treat zif as interval-scaled.

**Cosine Similarity of Two Vectors:** (two text documents similarity)

Cos(

**Example:** d1 = (5, 0, 3, 0, 2, 0, 0, 2, 0, 0) d2 = (3, 0, 2, 0, 1, 1, 0, 1, 0, 1). d1 •d2 = 5 X 3 + 0 X 0 + 3 X 2 + 0 X 0 + 2 X 1 + 0 X 1 + 0 X 1 + 2 X 1 + 0 X 0 + 0 X 1 = 25

|6.481,

|4.12. Cos(

**KL Divergence**

or

Notice: if P : (a : 3/5, b : 1/5, c : 1/5). Q : (a : 5/9, b : 3/9, d : 1/9)

P′ : (a : 3/5 − ϵ/3, b : 1/5 − ϵ/3, c : 1/5 − ϵ/3, d : ϵ)

Q′ : (a : 5/9 − ϵ/3, b : 3/9 − ϵ/3, c : ϵ, d : 1/9 − ϵ/3)

**Data Quality Issues**

**Measures for data quality: A multidimensional view**

**Accuracy**: correct or wrong, accurate or not

**Completeness**: not recorded, unavailable,

**Consistency**: some modified but some not, dangling,

**Timeliness**: timely update?

**Believability**: how trustable the data are correct?

**Interpretability**: how easily the data can be understood?

**Data Cleaning**

**Incomplete**: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data (Occupation = “ ” (missing data))

**Noisy**: containing noise, errors, or outliers (Salary = “−10” (an error))

**Inconsistent**: containing discrepancies in codes or names(Age = “42”, Birthday = “03/07/2010”; Was rating “1, 2, 3”, now rating “A, B, C”; discrepancy between duplicate records)

**Intentional** (e.g., disguised missing data)( Jan. 1 as everyone’s birthday?)

**Incomplete (Missing) Data**

**Data is not always available** (E.g., many tuples have no recorded value for several attributes, such as customer income in sales data)

**Missing data may be due to** (Equipment malfunction; Inconsistent with other recorded data and thus deleted; Data were not entered due to misunderstanding; Certain data may not be considered important at the time of entry; Did not register history or changes of the data)

**Missing data may need to be inferred**

**Handle Missing Data**

**Ignore the tuple**: usually done when class label is missing (when doing classification)—not effective when the % of missing values per attribute varies considerably

**Fill in the missing value manually**: tedious + infeasible?

**Fill in it automatically** with (a global constant : e.g., “unknown”, a new class?!; the attribute mean; the attribute mean for all samples belonging to the same class: smarter; the most probable value: inference-based such as Bayesian formula or decision tree)

**Noisy Data**

**Noise**: random error or variance in a measured variable

**Incorrect** **attribute** **values** **may be due to** (Faulty data collection instruments; Data entry problems; Data transmission problems; Technology limitation; Inconsistency in naming convention)

**Other data problems** (Duplicate records; Incomplete data; Inconsistent data)

**Handling Redundancy in Data Integration**

**Redundant data occur often when integration of multiple databases** (Object identification: The same attribute or object may have different names in different databases; Derivable data: One attribute may be a “derived” attribute in another table, e.g., annual revenue)

**Redundant attributes may be able to be detected by correlation analysis and covariance analysis**

**Careful integration of the data from multiple sources** may help reduce/avoid redundancies and inconsistencies and improve mining speed and quality

**Data Integration**

Correlation Analysis (for Categorical Data)

**Χ2 (chi-square) test:**

**Null hypothesis**: The two distributions are independent

The larger the value, the more likely the variables are related

Correlation does not imply causality

**Variance & Covariance**

**Correlation**

**Data Reduction**

**Obtain a reduced representation of the data set** (much smaller in volume but yet produces almost the same analytical results)

**Why data reduction?**—A database/data warehouse may store terabytes of data (Complex analysis may take a very long time to run on the complete data set)

**Methods for data reduction** (also data size reduction or **numerosity reduction**) (Regression and Log-Linear Models; Histograms, clustering, sampling; Data cube aggregation; Data compression)

**Parametric** (Assume the data fits some model, estimate model parameters, store only the parameters, and discard the data (except possible outliers), e.g., regression) vs. **Non-Parametric Methods** (Do not assume models; Major families: histograms, clustering, and sampling)

**Linear regression**: Y = w X + b; **Multiple regression**: Y = b0 + b1 X1 + b2 X2

Nonlinear regression; Log-linear model;

**Histogram Analysis** (Divide data into buckets and store average (sum) for each bucket; partitioning rules: Equal-width: equal bucket range or Equal-frequency (or equal-depth))

**Clustering** (Partition data set into clusters based on similarity, and store cluster representation (e.g., centroid and diameter) only)

**Sampling** (Key principle: Choose a **representative** subset of the data: Simple random sampling may have very poor performance in the presence of skew; Develop adaptive sampling methods, e.g., **stratified** sampling)

**Simple random sampling; Sampling without replacement; Sampling with replacement; Stratified sampling (**Partition (or cluster) the data set, and draw samples from each partition (proportionally, i.e., approximately the same percentage of the data))

**Data Cube Aggregation** (The lowest level of a data cube (base cuboid); Multiple levels of aggregation in data cubes; Reference appropriate levels; Queries regarding aggregated information should be answered using data cube, when possible)

**Data Compression** (String compression; Audio/video compression; Time sequence is not audio; Data reduction and dimensionality reduction may also be considered as forms of data compression)

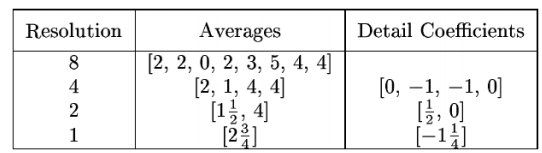
**Wavelet Transform**

**Discrete wavelet transform** (DWT) for linear signal processing, multi-resolution analysis

**Compressed approximation**: Store only a small fraction of the strongest of the wavelet coefficients

Similar to discrete Fourier transform (DFT), but better **lossy compression**, localized in space

**Method**: (Length, L, must be an integer power of 2 (padding with 0’s, when necessary); Each transform has 2 functions: smoothing, difference; Applies to pairs of data, resulting in two set of data of length L/2; Applies two functions recursively, until reaches the desired length)



**Why Wavelet Transform**

**Use hat-shape filters** (Emphasize region where points cluster; Suppress weaker information in their boundaries)

**Effective removal of outliers** (Insensitive to noise, insensitive to input order)

**Multi-resolution** (Detect arbitrary shaped clusters at different scales)

**Efficient** (Complexity O(N))

**Only applicable to low dimensional data**

**Data Transformation**

A function that maps the entire set of values of a given attribute to a new set of replacement values s.t. each old value can be identified with one of the new values

**Methods** (**Smoothing**: Remove noise from data; **Attribute/feature construction** : New attributes constructed from the given ones; **Aggregation**: Summarization, data cube construction; **Normalization** :( Scaled to fall within a smaller, specified range: min-max normalization; z-score normalization; normalization by decimal scaling); **Discretization**: Concept hierarchy climbing

**Normalization**

**Min-max normalization**: to [new\_minA, new\_maxA]:

**Z-score normalization** (μ: mean, σ: standard deviation):

**Normalization by decimal scaling**: , Where j is the smallest integer such that Max(|ν’|) < 1

**Discretization**

**Three types of attributes**: (**Nominal**—values from an unordered set, e.g., color, profession; **Ordinal**—values from an ordered set, e.g., military or academic rank; **Numeric**—real numbers, e.g., integer or real numbers)

**Discretization: Divide the range of a continuous attribute** into intervals (Interval labels can then be used to replace actual data values; Reduce data size by discretization; Supervised vs. unsupervised; Split (top-down) vs. merge (bottom-up); Discretization can be performed recursively on an attribute; Prepare for further analysis, e.g., classification)

**Data Discretization Methods**

**Binning** (Top-down split, unsupervised)

**Histogram analysis** (Top-down split, unsupervised)

**Clustering analysis** (Unsupervised, top-down split or bottom-up merge)

**Decision-tree analysis** (Supervised, top-down split)

**Correlation** (e.g., χ2) analysis (Unsupervised, bottom-up merge)

Note: All the methods can be applied recursively

**Dimensionality Reduction**

**Dimensionality reduction methodologies** (**Feature selection**: Find a subset of the original variables (or features, attributes); **Feature extraction**: Transform the data in the high-dimensional space to a space of fewer dimensions)

**Some typical dimensionality methods** (Principal Component Analysis; Supervised and nonlinear techniques (Feature subset selection; Feature creation))

**Principal Component Analysis (PCA)**

**Attribute Subset Selection** (Redundant attributes; Irrelevant attributes)

**Heuristic Search in Attribute Selection**

**Attribute Creation (Feature Generation)**

**Data Warehousing and On-line Analytical Processing**

**Data Warehouse—Subject-Oriented**

Organized around major subjects, such as customer, product, sales

Focusing on the modeling and analysis of data for decision makers, not on daily operations or transaction processing

Provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process

**Data Warehouse—Integrated**

**Constructed by integrating multiple, heterogeneous data sources** (relational databases, flat files, on-line transaction records)

**Data cleaning and data integration techniques are applied.** (Ensure consistency in naming conventions, encoding structures, attribute measures, etc. among different data sources (Ex. Hotel price: differences on currency, tax, breakfast covered, and parking); When data is moved to the warehouse, it is converted)

**Data Warehouse—Time Variant**

**The time horizon for the data warehouse is significantly longer than that of operational systems** (Operational database: current value data; Data warehouse data: provide information from a historical perspective (e.g., past 5-10 years))

**Every key structure in the data warehouse** (Contains an element of time, explicitly or implicitly; But the key of operational data may or may not contain “time element”)

**Data Warehouse—Nonvolatile**

**Independence** (A physically separate store of data transformed from the operational environment)

**Static: Operational update of data does not occur in the data warehouse environment** (Does not require transaction processing, recovery, and concurrency control mechanisms; Requires only two operations in data accessing: (initial loading of data and access of data))

**OLTP vs. OLAP**

**OLTP: Online transactional processing** (DBMS operations; Query and transactional processing)

**OLAP: Online analytical processing**

(Data warehouse operations; Drilling, slicing, dicing, etc)

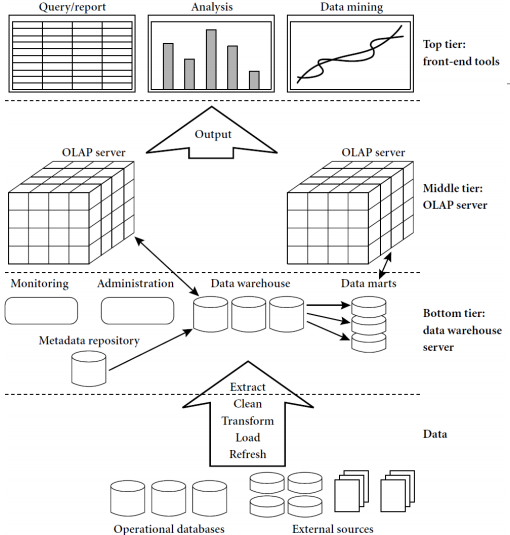
**Why a Separate Data Warehouse?**

**High performance for both systems** (DBMS— tuned for OLTP: access methods, indexing, concurrency control, recovery ; Warehouse—tuned for OLAP: complex OLAP queries, multidimensional view, consolidation)

**Different functions and different data:** (**missing data**: Decision support requires historical data which operational DBs do not typically maintain ; **data consolidation**: DS requires consolidation (aggregation, summarization) of data from heterogeneous sources **; data quality**: different sources typically use inconsistent data representations, codes and formats which have to be reconciled)

**Note: There are more and more systems which perform OLAP analysis directly on relational databases**

**Data Warehouse: A Multi-Tiered Architecture**



**Three Data Warehouse Models**

**Enterprise warehouse** (Collects all of the information about subjects spanning the entire organization)

**Data Mart** (A subset of corporate-wide data that is of value to a specific groups of users; Its scope is confined to specific, selected groups, such as marketing data mart (Independent vs. dependent (directly from warehouse) data mart))

**Virtual warehouse** (A set of views over operational databases; Only some of the possible summary views may be materialized)

**Extraction, Transformation, and Loading (ETL)**

**Data extraction** (get data from multiple, heterogeneous, and external sources)

**Data cleaning** (detect errors in the data and rectify them when possible)

**Data transformation** (convert data from legacy or host format to warehouse format)

**Load** (sort, summarize, consolidate, compute views, check integrity, and build indicies and partitions)

**Refresh** (propagate the updates from the data sources to the warehouse)

**Metadata Repository**

**Meta data is the data defining warehouse objects. It stores:**

**Description of the structure of the data warehouse** (schema, view, dimensions, hierarchies, derived data defn, data mart locations and contents)

**Operational meta-data** (data lineage (history of migrated data and transformation path), currency of data (active, archived, or purged), monitoring information (warehouse usage statistics, error reports, audit trails)

**The algorithms used for summarization**

**The mapping from operational environment to the data warehouse**

**Data related to system performance** (warehouse schema, view and derived data definitions)

**Business data** (business terms and definitions, ownership of data, charging policies)

**From Tables and Spreadsheets to Data Cubes**

**A data warehouse is based on a multidimensional data model which views data in the form of a data cube**

**A data cube, such as sales, allows data to be modeled and viewed in multiple dimensions** (**Dimension tables**, such as item (item\_name, brand, type), or time(day, week, month, quarter, year); **Fact table** contains **measures** (such as dollars\_sold) and keys to each of the related dimension tables)

**Data cube**: A lattice of cuboids (In data warehousing literature, an n-D base cube is called a base cuboid ; The top most 0-D cuboid, which holds the highest-level of summarization, is called the apex cuboid ; The lattice of cuboids forms a data cube)

**Data Cube: A Lattice of Cuboids**



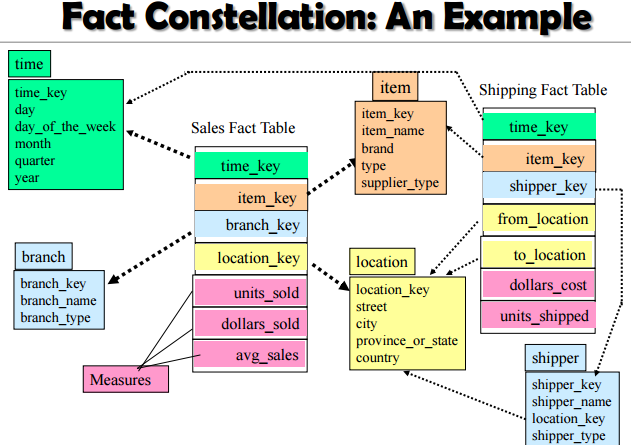
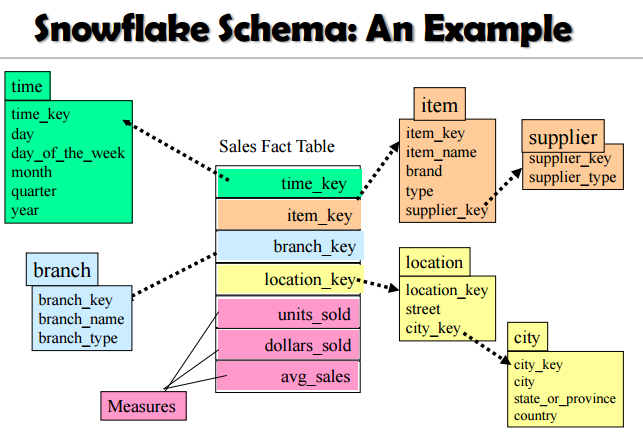
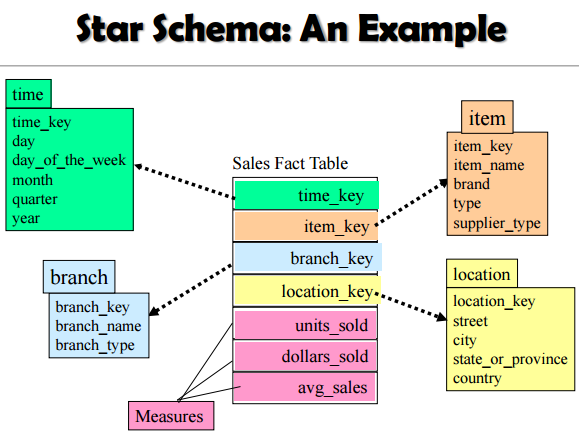
**Conceptual Modeling of Data Warehouses**

**Modeling data warehouses: dimensions & measures :**

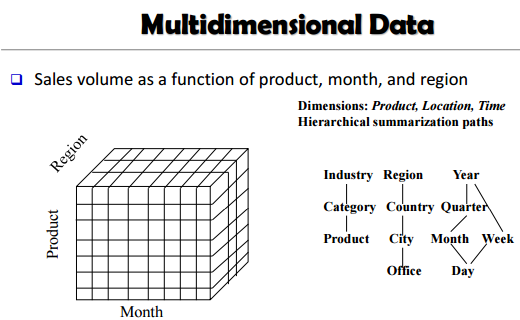
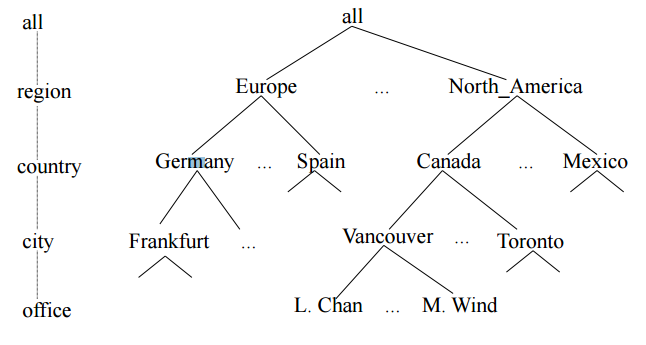
**Star schema**: A fact table in the middle connected to a set of dimension tables

**Snowflake schema**: A refinement of star schema where some dimensional hierarchy is normalized into a set of smaller dimension tables, forming a shape similar to snowflake

**Fact constellations**: Multiple fact tables share dimension tables, viewed as a collection of stars, therefore called galaxy schema or fact constellation



**A Concept Hierarchy for a Dimension (location)**



**Data Cube Measures: Three Categories**

**Distributive**: if the result derived by applying the function to n aggregate values is the same as that derived by applying the function on all the data without partitioning (E.g., count(), sum(), min(), max())

**Algebraic**: if it can be computed by an algebraic function with M arguments (where M is a bounded integer), each of which is obtained by applying a distributive aggregate function (avg(x) = sum(x) / count(x); Is min\_N() an algebraic measure? How about standard\_deviation()?)

**Holistic**: if there is no constant bound on the storage size needed to describe a subaggregate. (E.g., median(), mode(), rank())

**Typical OLAP Operations**

**Roll up (drill-up):** summarize data (by climbing up hierarchy or by dimension reduction)

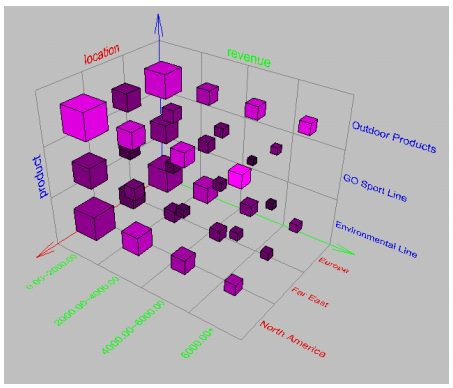
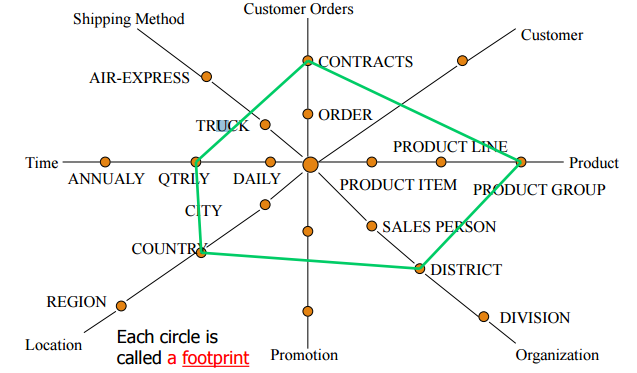
**Drill down (roll down):** reverse of roll-up (from higher level summary to lower level summary or detailed data, or introducing new dimensions)

**Slice and dice:** project and select

**Pivot (rotate):** (reorient the cube, visualization, 3D to series of 2D planes)

**Other operations** (**Drill across**: involving (across) more than one fact table; **Drill through**: through the bottom level of the cube to its back-end relational tables (using SQL))

**A Star-Net Query Model Browsing a Data Cube**



**Design of Data Warehouse: A Business Analysis Framework**

**Four views regarding the design of a data warehouse**

**Top-down view** (allows selection of the relevant information necessary for the data warehouse)

**Data source view** (exposes the information being captured, stored, and managed by operational systems)

**Data warehouse view** (consists of fact tables and dimension tables)

**Business query view** (sees the perspectives of data in the warehouse from the view of end-user)

**Data Warehouse Design Process**

**Top-down, bottom-up approaches or a combination of both** (Top-down: Starts with overall design and planning (mature); Bottom-up: Starts with experiments and prototypes (rapid))

**From software engineering point of view** (Waterfall: structured and systematic analysis at each step before proceeding to the next; Spiral: rapid generation of increasingly functional systems, short turn around time, quick turn around)

**Typical data warehouse design process** (Choose a business process to model, e.g., orders, invoices, etc; Choose the grain (atomic level of data) of the business process; Choose the dimensions that will apply to each fact table record; Choose the measure that will populate each fact table record)

**Data Warehouse Usage**

**Three kinds of data warehouse applications:**

**Information processing** (supports querying, basic statistical analysis, and reporting using crosstabs, tables, charts and graphs)

**Analytical processing** (multidimensional analysis of data warehouse data; supports basic OLAP operations, slice-dice, drilling, pivoting)

**Data mining** (knowledge discovery from hidden patterns; supports associations, constructing analytical models, performing classification and prediction, and presenting the mining results using visualization tools)

**Efficient Data Cube Computation**

**Data cube can be viewed as a lattice of cuboids** (The bottom-most cuboid is the base cuboid; The top-most cuboid (apex) contains only one cell; How many cuboids in an n-dimensional cube with L levels?)

**Materialization of data cube** (Full materialization: Materialize every (cuboid); No materialization: Materialize none (cuboid); Partial materialization: Materialize some cuboids;)

**The “Compute Cube” Opera**

Cube definition and computation in DMQL define cube sales [item, city, year]: sum (sales\_in\_dollars) compute cube sales

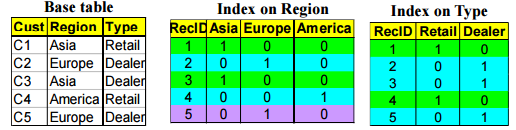
**Transform** it into a **SQL-like language** (with a new operator cube by, introduced by Gray et al.’96) SELECT item, city, year, SUM (amount) FROM SALES CUBE BY item, city, year

Need compute the following **Group-Bys** (date, product, customer), (date, product),(date, customer), (product, customer), (date), (product), (customer) ()

**Indexing OLAP Data: Bitmap Index**

**Index on a particular column** (Each value in the column has a bit vector: bit-op is fast; The length of the bit vector: # of records in the base table; The i-th bit is set if the i-th row of the base table has the value for the indexed column; not suitable for high cardinality domains)

**A recent bit compression technique, Word-Aligned Hybrid (WAH), makes it work for high cardinality domain as well [Wu, et al. TODS’06]**



Indexing OLAP Data: Join Indices

Join index: JI(R-id, S-id) where R (R-id, …) >< S (S-id, …)

Traditional indices map the values to a list of record ids (It materializes relational join in JI file and speeds up relational join)

In data warehouses, join index relates the values of the dimensions of a start schema to rows in the fact table. (E.g., fact table: Sales and two dimensions city and product : (A join index on city maintains for each distinct city a list of R-IDs of the tuples recording the Sales in the city) ; Join indices can span multiple dimensions)

**Efficient Processing OLAP Queries**

**Determine which operations** should be performed on the available cuboids (Transform drill, roll, etc. into corresponding SQL and/or OLAP operations, e.g., dice = selection + projection)

**OLAP Server Architectures**

**Relational OLAP (ROLAP)** (Use relational or extended-relational DBMS to store and manage warehouse data and OLAP middle ware ; Include optimization of DBMS backend, implementation of aggregation navigation logic, and additional tools and services ; Greater scalability);

**Multidimensional OLAP (MOLAP)** (Sparse array-based multidimensional storage engine; Fast indexing to pre-computed summarized data)

**Hybrid OLAP (HOLAP)** (e.g., Microsoft SQLServer) (Flexibility, e.g., low level: relational, high-level: array)

**Specialized SQL servers (e.g., Redbricks)** (Specialized support for SQL queries over star/snowflake schemas)

**Data Cube Technology**

**Cube Materialization: Full Cube vs. Iceberg Cube**

Full cube vs. iceberg cube (having count(\*) >= min support)

**Compute only the cells whose measure satisfies the iceberg condition**

**Only a small portion of cells may be “above the water’’ in a sparse cube**

**Why Iceberg Cube?**

**Advantages of computing iceberg cubes** (No need to save nor show those cells whose value is below the threshold (iceberg condition); Efficient methods may even avoid computing the un-needed, intermediate cells; Avoid explosive growth)

**Example:** A cube with 100 dimensions (Suppose it contains only 2 base cells: {(a1, a2, a3, …., a100), (a1, a2, b3, …, b100)} ; How many aggregate cells if “having count >= 1”? Answer: 2101 ─ 4 (Why?!) What about the iceberg cells, (i,e., with condition: “having count >= 2”)? Answer: 4 (Why?!))

**Close cube:A cell c is closed** if there exists no cell d, such that d is a descendant of c, and d has the same measure value as c

Ex. The same cube P has only 3 closed cells: {(a1, a2, \*, …, \*): 20, (a1, a2, a3 . . . , a100): 10, (a1, a2, b3, . . . , b100): 10}

**A closed cube is a cube consisting of only closed cells**

**Cube Shell:** The cuboids involving only a small # of dimensions, e.g., 2 θ Idea: Only compute cube shells, other dimension combinations can be computed on the fly

**Roadmap for Efficient Computation**

**General computation heuristics (Agarwal et al.’96)**

**Computing full/iceberg cubes: 3 methodologies:**

**Bottom-Up**: Multi-Way array aggregation (Zhao, Deshpande & Naughton, SIGMOD’97

**Top-down**: BUC (Beyer & Ramarkrishnan, SIGMOD’99)

**Integrating Top-Down and Bottom-Up**: Star-cubing algorithm (Xin, Han, Li & Wah: VLDB’03)

**High-dimensional OLAP**: A Shell-Fragment Approach (Li, et al. VLDB’04)

**Computing alternative kinds of cubes:** Partial cube, closed cube, approximate cube,

**Efficient Data Cube Computation: General Heuristics**

**Sorting, hashing, and grouping operations are applied to the dimension attributes in order to reorder and cluster related tuples**

**Aggregates may be computed from previously computed aggregates, rather than from the base fact table** (**Smallest-child**: computing a cuboid from the smallest, previously computed cuboid; **Cache-results**: caching results of a cuboid from which other cuboids are computed to reduce disk I/Os; **Amortize-scans**: computing as many as possible cuboids at the same time to amortize disk reads; **Share-sorts**: sharing sorting costs cross multiple cuboids when sort-based method is used; **Share-partitions**: sharing the partitioning cost across multiple cuboids when hash-based algorithms are used)

**Multi-Way Array Aggregation**

Array-based “bottom-up” algorithm (from ABC to AB,…)

Using multi-dimensional chunks

Simultaneous aggregation on multiple dimensions

Intermediate aggregate values are re-used for computing ancestor cuboids

Cannot do Apriori pruning: No iceberg optimization

Comments on the method:

Efficient for computing the full cube for a **small number** of dimensions

If there are a large number of dimensions, “top-down” computation and iceberg cube computation methods (e.g., BUC) should be used

**Cube Computation: Multi-Way Array Aggregation (MOLAP)**

Partition arrays into chunks (a small subcube which fits in memory).

Compressed sparse array addressing: (chunk\_id, offset)

Compute aggregates in “multiway” by visiting cube cells in the order which minimizes the # of times to visit each cell, and reduces memory access and storage cost

**Cube Computation: Computing in Reverse Order**

BUC (Beyer & Ramakrishnan, SIGMOD’99) BUC: acronym of Bottom-Up (cube) Computation (Note: It is “top-down” in our view since we put Apex cuboid on the top!)

Divides dimensions into partitions and facilitates iceberg pruning (If a partition does not satisfy min\_sup, its descendants can be pruned; If minsup = 1 Þ compute full CUBE!)

No simultaneous aggregation

**BUC: Partitioning and Aggregating**

Usually, entire data set cannot fit in main memory

Sort distinct values (partition into blocks that fit)

Continue processing

Optimizations (Partitioning θ External Sorting, Hashing, Counting Sort; Ordering dimensions to encourage pruning θ Cardinality, Skew, Correlation; Collapsing duplicates θ Cannot do holistic aggregates anymore!)

**High-Dimensional OLAP?—The Curse of Dimensionality**

**High-D OLAP: Needed in many applications** (Science and engineering analysis; Bio-data analysis: thousands of genes; Statistical surveys: hundreds of variables)

**None of the previous cubing method can handle high dimensionality!** (Iceberg cube and compressed cubes: only delay the inevitable explosion; Full materialization: still significant overhead in accessing results on disk)

**A shell-fragment approach: X. Li, J. Han, and H. Gonzalez, High-Dimensional OLAP: A Minimal Cubing Approach, VLDB'04**

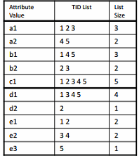
**Fast High-D OLAP with Minimal Cubing**

**Observation**: OLAP occurs only on a small subset of dimensions at a time

**Semi-Online Computational Model** (Partition the set of dimensions into **shell fragments**; Compute data cubes for each shell fragment while retaining **inverted indices** or **value-list indices**; Given the pre-computed **fragment cubes**, dynamically compute cube cells of the high-dimensional data cube online)

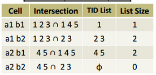
**Major idea:** Tradeoff between the amount of pre-computation and the speed of online computation (Reducing computing high-dimensional cube into precomputing a set of lower dimensional cubes; Online re-construction of original high-dimensional space; Lossless reduction)

**Shell Fragment Cubes: Size and Design**

**Given a database of T tuples, D dimensions, and F shell fragment size, the fragment cubes’ space requirement is:**

For F<5, the growth is sub-linear

**Shell fragments do not have to be disjoint**

**Fragment groupings can be arbitrary to allow for maximum online performance** (Known common combinations (e.g.,) should be grouped together)

**Shell fragment sizes can be adjusted for optimal balance between offline and online computation**

**Data Mining in Cube Space**

**Data cube greatly increases the analysis bandwidth**

**Four ways to interact OLAP-styled analysis and data mining** (Using cube space to define data space for mining; Using OLAP queries to generate features and targets for mining, e.g., multi-feature cube; Using data-mining models as building blocks in a multi-step mining process, e.g.,

**Complex Aggregation at Multiple** prediction cube; Using data-cube computation techniques to speed up repeated model construction: (Cube-space data mining may require building a model for each candidate data space; Sharing computation across model-construction for different candidates may lead to efficient mining))**Granularities: Multi-Feature Cubes**

**Multi-feature cubes** (Ross, et al. 1998): Compute complex queries involving multiple dependent aggregates at multiple granularities

Ex. Grouping by all subsets of {item, region, month}, find the maximum price in 2010 for each group, and the total sales among all maximum price tuples (select item, region, month, max(price), sum(R.sales) from purchases where year = 2010 cube by item, region, month: R such that R.price = max(price))

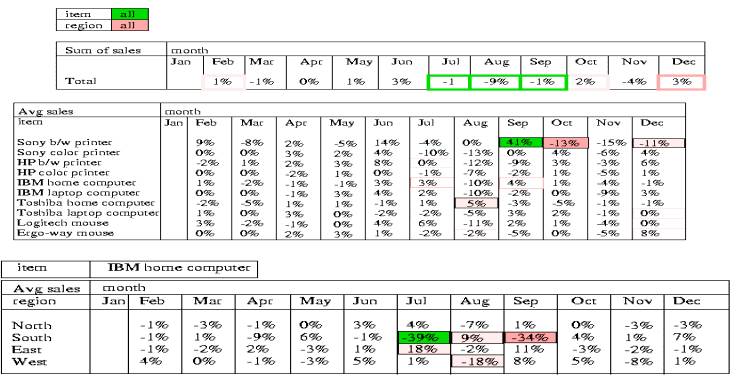
**Discovery-Driven Exploration of Data Cubes**

**Discovery-driven exploration of huge cube space (Sarawagi, et al.’98)** (Effective navigation of large OLAP data cubes; pre-compute measures indicating exceptions, guide user in the data analysis, at all levels of aggregation; Exception: significantly different from the value anticipated, based on a statistical mode; Visual cues such as background color are used to reflect the degree of exception of each cell)

**Kinds of** **exceptions** (SelfExp: surprise of cell relative to other cells at same level of aggregation; InExp: surprise beneath the cell; PathExp: surprise beneath cell for each drill-down path

**Computation of exception indicator can be overlapped with cube construction** (Exceptions can be stored, indexed and retrieved like precomputed aggregates)

**Examples: Discovery-Driven Data Cubes**



**What Is Pattern Discovery?**

**What are patterns?**

**Patterns**: A set of items, subsequences, or substructures that occur frequently together (or strongly correlated) in a data set

Patterns represent **intrinsic** and **important properties** of datasets

**Pattern discovery**: Uncovering patterns from massive data sets

**Motivation examples**: (What products were often purchased together?; What are the subsequent purchases after buying an iPad?; What code segments likely contain copy-and-paste bugs?; What word sequences likely form phrases in this corpus?)

**Pattern Discovery: Why Is It Important?**

**Finding inherent regularities in a data set**

**Foundation for many essential data mining tasks :(**

Association, correlation, and causality analysis;

Mining sequential, structural (e.g., sub-graph) patterns;

Pattern analysis in spatiotemporal, multimedia, time-series, and stream data;

**Classification**: Discriminative pattern-based analysis;

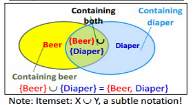
**Cluster analysis**: Pattern-based subspace clustering)

**Broad applications** (Market basket analysis, cross-marketing, catalog design, sale campaign analysis, Web log analysis, biological sequence analysis)

**Basic Concepts: Frequent Itemsets (Patterns)**

**Itemset**: A set of one or more items

**k-itemset**: X = {x1 , …, xk }

**(absolute) support (count) of X**: Frequency or the number of occurrences of an itemset X

**(relative) support, s**: The fraction of transactions that contains X (i.e., the probability that a transaction contains X)

An itemset X is frequent if the support of X is no less than a minsup threshold (denoted as σ)

**From Frequent Itemsets to Association Rules**

**Association rules: X -> Y (s, c)**

Support, s: The probability that a transaction contains X ∪ Y

Confidence, c: The conditional probability that a transaction containing X also contains Y

c = sup(X ∪ Y) / sup(X)

**Association rule** **mining**: Find all of the rules, X -> Y, with minimum support and confidence

**Frequent itemsets**: Let **minsup = 50%** (**Freq. 1-itemsets**: Beer: 3, Nuts: 3, Diaper: 4, Eggs: 3 ; **Freq. 2-itemsets**: {Beer, Diaper}: 3)

Association rules: Let **minconf = 50%** (Beer -> Diaper (60%, 100%); Diaper -> Beer (60%, 75%))

**Challenge: There Are Too Many Frequent Patterns!**

**A long pattern contains a combinatorial number of sub-patterns**

**How many frequent itemsets does the following TDB1 contain?**

TDB1: T1 : {a1 , …, a50}; T2 : {a1 , …, a100}

Assuming (absolute) minsup = 1

Let’s have a try

1-itemsets: {a1 }: 2, {a2 }: 2, …, {a50}: 2, {a51}: 1, …, {a100}: 1,

2-itemsets: {a1 , a2 }: 2, …, {a1 , a50}: 2, {a1 , a51}: 1 …, …, {a99, a100}: 1, …, …, …, …

99-itemsets: {a1 , a2 , …, a99}: 1, …, {a2 , a3 , …, a100}: 1

100-itemset: {a1 , a2 , …, a100}: 1

In total: (100 1 ) + (100 2 ) + … + (100 100) = 2^100 – 1 sub-patterns!

**Expressing Patterns in Compressed Form: Closed Patterns**

How to handle such a challenge?

Solution 1: **Closed patterns**: A pattern (itemset) X is closed if X is frequent, and there exists no super-pattern Y כ X, with the same support as X

Let Transaction DB TDB1 : T1 : {a1 , …, a50}; T2 : {a1 , …, a100}

Suppose minsup = 1. How many closed patterns does TDB1 contain? **(Two: P1 : “{a1 , …, a50}: 2”; P2 : “{a1 , …, a100}: 1”)**

**Closed pattern is a lossless compression of frequent patterns** (Reduces the # of patterns but does not lose the support information!; You will still be able to say: “{a2 , …, a40}: 2”, “{a5 , a51}: 1”)

**Expressing Patterns in Compressed Form: Max-Patterns**

Solution 2: **Max-patterns**: A pattern X is a max-pattern if X is frequent and there exists no frequent super-pattern Y כ X

Difference from close-patterns?

Do not care the real support of the sub-patterns of a max-pattern

Let Transaction DB TDB1 : T1 : {a1 , …, a50}; T2 : {a1 , …, a100}

Suppose minsup = 1. How many max-patterns does TDB1 contain?

**One: P: “{a1 , …, a100}: 1”**

**Max-pattern is a lossy compression!** (We only know {a1 , …, a40} is frequent; But we do not know the real support of {a1 , …, a40}, …, any more!)

Thus in many applications, mining close-patterns is more desirable than mining max-patterns

**Efficient Pattern Mining Methods**

**The Downward Closure Property of Frequent Patterns**

**The Apriori Algorithm**

**Extensions or Improvements of Apriori**

**Mining Frequent Patterns by Exploring Vertical Data Format**

**FPGrowth: A Frequent Pattern-Growth Approach**

**Mining Closed Patterns**

**The Downward Closure Property of Frequent Patterns**

**Observation: From TDB1: T1 : {a1 , …, a50}; T2 : {a1 , …, a100}** (We get a frequent itemset: {a1 , …, a50}; Also, its subsets are all frequent: {a1 }, {a2 }, …, {a50}, {a1 , a2 }, …, {a1 , …, a49}, …; There must be some hidden relationships among frequent patterns!)

**The downward closure (also called “Apriori”) property of frequent patterns**(If {beer, diaper, nuts} is frequent, so is {beer, diaper}; Every transaction containing {beer, diaper, nuts} also contains {beer, diaper}; Apriori: Any subset of a frequent itemset must be frequent)

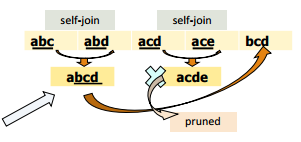
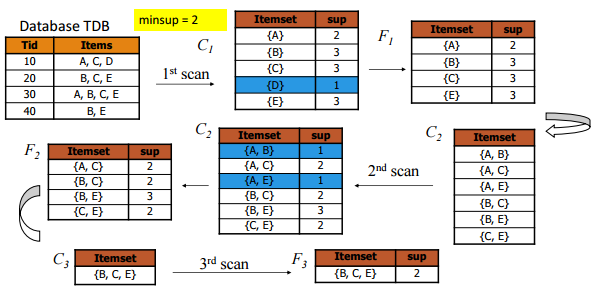
**Efficient mining methodology** (If any subset of an itemset S is infrequent, then there is no chance for S to be frequent—why do we even have to consider S!?)

**Apriori Pruning and Scalable Mining Methods**

Apriori pruning principle: If there is any itemset which is infrequent, its superset should not even be generated! (Agrawal & Srikant @VLDB’94, Mannila, et al. @ KDD’ 94)

**Scalable mining Methods: Three major approaches** (Level-wise, join-based approach: Apriori (Agrawal & Srikant@VLDB’94; Vertical data format approach: Eclat (Zaki, Parthasarathy, Ogihara, Li @KDD’97); Frequent pattern projection and growth: FPgrowth (Han, Pei, Yin @SIGMOD’00))

**The Apriori Algorithm—An Example**



**Apriori: Implementation Tricks**

**How to generate candidates?** (Step 1: self-joining Fk ;Step2: pruning)

**Example of candidate-generation** (F3 = {abc, abd, acd, ace, bcd}; Self-joining: F3\*F3 (abcd from abc and abd; acde from acd and ace); Pruning: (acde is removed because ade is not in F3); C4 = {abcd})

**Apriori: Improvements and Alternatives**

**Reduce passes of transaction database scans** (Partitioning (e.g., Savasere, et al., 1995); Dynamic itemset counting (Brin, et al., 1997))

**Shrink the number of candidates** (Hashing (e.g., DHP: Park, et al., 1995); Pruning by support lower bounding (e.g., Bayardo 1998); Sampling (e.g., Toivonen, 1996))

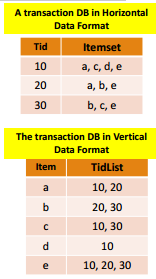
**Exploring special data structures** (Tree projection (Agarwal, et al., 2001); H-miner (Pei, et al., 2001); Hypecube decomposition (e.g., LCM: Uno, et al., 2004))

**Theorem: Any itemset that is potentially frequent in TDB must be frequent in at least one of the partitions of TDB**

**Direct Hashing and Pruning (DHP)**

**DHP** (Direct Hashing and Pruning): Reduce the number of candidates (J. Park, M. Chen, and P. Yu, SIGMOD’95)

**Observation**: A k-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent (Candidates: a, b, c, d, e; Hash entries ({ab, ad, ae}; {bd, be, de}); Frequent 1-itemset: a, b, d, e; ab is not a candidate 2-itemset if the sum of count of {ab, ad, ae} is below support threshold)

**Exploring Vertical Data Format: ECLAT**

**ECLAT** (Equivalence Class Transformation): A depth-first search algorithm using set intersection [Zaki et al. @KDD’97]

Tid-List: List of transaction-ids containing an itemset

Vertical format: t(e) = {T10, T20, T30}; t(a) = {T10, T20}; t(ae) = {T10, T20}

Properties of Tid-Lists(t(X) = t(Y): X and Y always happen together (e.g., t(ac} = t(d}); t(X) = t(Y): X and Y always happen together (e.g., t(ac} = t(d}))

Deriving frequent patterns based on vertical intersections

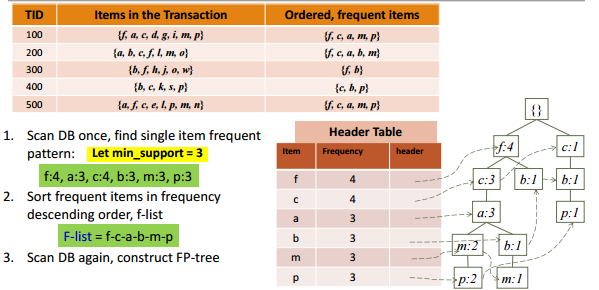
**Using diffset to accelerate mining** (Only keep track of differences of tids; t(e) = {T10, T20, T30}, t(ce) = {T10, T30} → Diffset (ce, e) = {T20})

**FPGrowth: Mining Frequent Patterns by Pattern Growth**

**Idea**: Frequent pattern growth (FPGrowth) (Find frequent single items and partition the database based on each such item; Recursively grow frequent patterns by doing the above for each partitioned database (also called conditional database); To facilitate efficient processing, an efficient data structure, FP-tree, can be constructed)

**Mining becomes** (Recursively construct and mine (conditional) FP-trees; Until the resulting FP-tree is empty, or until it contains only one path— single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)

**Example: Construct FP-tree from a Transational DB**



**Mine Each Conditional Pattern-Base Recursively**

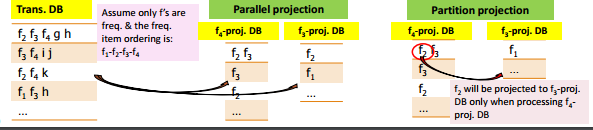


**Scaling FP-growth by Database Projection**

**Parallel projection vs. partition projection**

Parallel projection: Project the DB on each frequent item (Space costly, all partitions can be processed in parallel)

Partition projection: Partition the DB in order (Passing the unprocessed parts to subsequent partitions)



**CLOSET+: Mining Closed Itemsets by Pattern-Growth**

Efficient, direct mining of closed itemsets

Ex. Itemset merging: If Y appears in every occurrence of X, then Y is merged with X (d-proj. db: {acef, acf} ∨ acfd-proj. db: {e}, thus we get: acfd:2)

Many other tricks (but not detailed here), such as (Hybrid tree projection: Bottom-up physical tree-projection; Top-down pseudo tree-projection); Sub-itemset pruning; Item skipping; Efficient subset checking)

For details, see J. Wang, et al., “CLOSET+: ……”, KDD'03

**How to Judge if a Rule/Pattern Is Interesting?**

Pattern-mining will generate a large set of patterns/rules (Not all the generated patterns/rules are interesting)

Interestingness measures: Objective vs. subjective

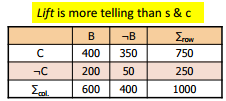
Objective interestingness measures (Support, confidence, correlation)

Subjective interestingness measures: One man’s trash could be another man’s treasure (Query-based: Relevant to a user’s particular request; Against one’s knowledge-base: unexpected, freshness, timeliness; Visualization tools: Multi-dimensional, interactive examination)

**Interestingness Measure: Lift**

**Measure of dependent/correlated events: lift**

=

**Lift(B, C) may tell how B and C are correlated** (Lift(B, C) = 1: B and C are independent; > 1: positively correlated; < 1: negatively correlated)

**For example**

Lift(B,C) = (400/1000)/ (600/1000 \* 750/1000) = 0.89

Lift(B,-C) = (200/1000)/ (600/1000 \* 250/1000) = 1.33

**Thus**,

B and C are negatively correlated since lift(B, C) < 1;

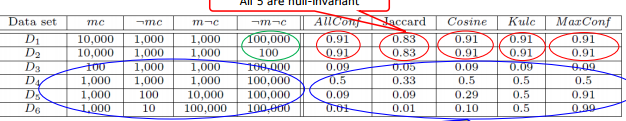
B and ¬C are positively correlated since lift(B, ¬C) > 1

**Interestingness Measures & Null-Invariance**

|  |  |  |  |
| --- | --- | --- | --- |
| Measure | Definition | Range | N I |
|  |  | [0, inf] | No |
| Lift(A,B) |  | [0, inf] | No |
| Allconf(A,B) |  | [0, 1] | Yes |
| Jaccard(A,B) |  | [0, 1] | Yes |
| Cosine(A,B) |  | [0, 1] | Yes |
| Kul (A,B) |  | [0, 1] | Yes |
| MaxConf(A,B) |  | [0, 1] | Yes |

Null invariance: Value does not change with the # of null-transactions

A few interestingness measures: Some are null invariant



**Imbalance Ratio with Kulczynski Measure**

IR (Imbalance Ratio): measure the imbalance of two itemsets A and B in rule implications:

**Mining Diverse Patterns**

**Mining Multiple-Level Associations**

**Mining Multi-Dimensional Associations**

**Mining Quantitative Associations**

**Mining Negative Correlations**

**Mining Compressed and Redundancy-Aware Patterns**

**Mining Multiple-Level Frequent Patterns**

Items often form hierarchies (Ex.: Dairyland 2% milk; Wonder wheat bread)

**How to set min-support thresholds?** (Uniform min-support across multiple levels (reasonable?); Level-reduced min-support: Items at the lower level are expected to have lower support)

**Efficient mining**: Shared multi-level mining (Use the lowest min-support to pass down the set of candidates)

**Redundancy Filtering at Mining Multi-Level Associations**

Multi-level association mining may generate many redundant rules

**Redundancy filtering: Some rules may be redundant due to “ancestor” relationships between items** (Suppose the 2% milk sold is about ¼ of milk sold in gallons) (milk ⇒ wheat bread [support = 8%, confidence = 70%] (1); 2% milk ⇒ wheat bread [support = 2%, confidence = 72%] (2) )

**A rule is redundant if its support is close to the “expected” value, according to its “ancestor” rule, and it has a similar confidence as its “ancestor”** (Rule (1) is an ancestor of rule (2), which one to prune?)

**Mining Multi-Dimensional Associations**

**Single-dimensional rules** (e.g., items are all in “product” dimension) (buys(X, “milk”) ⇒ buys(X, “bread”));

**Multi-dimensional rules** (i.e., items in ≥ 2 dimensions or predicates) (Inter-dimension association rules (no repeated predicates) : age(X, “18-25”) ∧ occupation(X, “student”) ⇒ buys(X, “coke”); Hybrid-dimension association rules (repeated predicates): age(X, “18-25”) ∧ buys(X, “popcorn”) ⇒ buys(X, “coke”))

**Attributes can be categorical or numerical** (Categorical Attributes (e.g., profession, product: no ordering among values): Data cube for inter-dimension association; Quantitative Attributes: Numeric, implicit ordering among values— discretization, clustering, and gradient approaches)

**Mining Quantitative Associations**

**Mining associations with numerical attributes** (Ex.: Numerical attributes: age and salary)

**Methods:**

Static discretization based on predefined concept hierarchies: (Data cube-based aggregation)

Dynamic discretization based on data distribution

Clustering: Distance-based association (First one-dimensional clustering, then association)

Deviation analysis: (Gender = female ⇒ Wage: mean=$7/hr (overall mean = $9))

**Mining Extraordinary Phenomena in Quantitative Association Mining**

**Mining extraordinary** (i.e., interesting) phenomena(Ex.: Gender = female ⇒ Wage: mean=$7/hr (overall mean = $9); LHS: a subset of the population; RHS: an extraordinary behavior of this subset)

The rule is accepted only if a statistical test (e.g., Z-test) confirms the inference with high confidence

Subrule: Highlights the extraordinary behavior of a subset of the population of the super rule (Ex.: (Gender = female) ^ (South = yes) ⇒ mean wage = $6.3/hr)

Rule condition can be categorical or numerical (quantitative rules) (Ex.: Education in [14-18] (yrs) ⇒ mean wage = $11.64/hr)

Efficient methods have been developed for mining such extraordinary rules (e.g., Aumann and Lindell@KDD’99)

**Rare Patterns vs. Negative Patterns**

**Rare patterns** (Very low support but interesting (e.g., buying Rolex watches); How to mine them? Setting individualized, group-based min-support thresholds for different groups of items)

**Negative patterns** (Negatively correlated: Unlikely to happen together; Ex.: Since it is unlikely that the same customer buys both a Ford Expedition (an SUV car) and a Ford Fusion (a hybrid car), buying a Ford Expedition and buying a Ford Fusion are likely negatively correlated patterns)

**Defining Negative Correlation: Need Null-Invariance in Definition**

**A good definition on negative correlation should take care of the nullinvariance problem** (Whether two itemsets A and B are negatively correlated should not be influenced by the number of null-transactions)

**A Kulczynski measure-based definition** (If itemsets A and B are frequent but (P(A|B) + P(B|A))/2 < є, where є is a negative pattern threshold, then A and B are negatively correlated)

For the same needle package problem: (No matter there are in total 200 or 10^5 transactions; If є = 0.01, we have (P(A|B) + P(B|A))/2 = (0.01 + 0.01)/2 < є)

**Mining Compressed Patterns**

Pattern distance measure:

δ-clustering: For each pattern P, find all patterns which can be expressed by P and whose distance to P is within δ (δ-cover)

All patterns in the cluster can be represented by P

Method for efficient, direct mining of compressed frequent patterns (e.g., D. Xin, J. Han, X. Yan, H. Cheng, "On Compressing Frequent Patterns", Knowledge and Data Engineering, 60:5-29, 2007)

**Redundancy-Aware Top-k Patterns**

**Desired patterns**: high significance & low redundancy

**Method**: Use MMS (Maximal Marginal Significance) for measuring the combined significance of a pattern set

Xin et al., Extracting Redundancy-Aware Top-K Patterns, KDD’06

**Why Constraint-Based Mining?**

Finding all the patterns in a dataset autonomously? — unrealistic! (Too many patterns but not necessarily user-interested!)

**Pattern mining should be an interactive process** (User directs what to be mined using a data mining query language (or a graphical user interface))

**Constraint-based mining** (User flexibility: provides constraints on what to be mined; Optimization: explores such constraints for efficient mining (Constraint-based mining: Constraint-pushing, similar to push selection first in DB query processing))

**Constraints in General Data Mining**

**Knowledge type constraint**: (Ex.: classification, association, clustering, outlier finding, ....)

**Data constrain**t — using SQL-like queries (Ex.: find products sold together in NY stores this year)

**Dimension/level constraint** (Ex.: in relevance to region, price, brand, customer category)

**Rule (or pattern) constraint** (Ex.: small sales (price < $10) triggers big sales (sum > $200))

**Interestingness constraint** (Ex.: strong rules: min\_sup ≥ 0.02, min\_conf ≥ 0.6, min\_correlation ≥ 0.7)

**Meta-Rule Guided Mining**

**A meta-rule can contain partially instantiated predicates & constants** (P1 (X, Y) ^ P2 (X, W) ⇒ buys(X, “iPad”))

**The resulting mined rule can be** (age(X, “15-25”) ^ profession(X, “student”) ⇒ buys(X, “iPad”))

**In general, (meta) rules can be in the form of** (P1 ^ P2 ^ … ^ Pl ⇒ Q1 ^ Q2 ^ … ^ Qr)

**Method to find meta-rules** (Find frequent (l + r) predicates (based on min-support); Push constants deeply when possible into the mining process: Using constraint-push techniques introduced in this lecture; Also, push min\_conf, min\_correlation, and other measures as early as possible (measures acting as constraints))

**Different Kinds of Constraints Lead to Different Pruning Strategies**

**Constraints can be categorized as:** Pattern space pruning constraints vs. data space pruning constraints

**Pattern space pruning constraints:**

Anti-monotonic: If constraint c is violated, its further mining can be terminated

Monotonic: If c is satisfied, no need to check c again

Succinct: if the constraint c can be enforced by directly manipulating the data

Convertible: c can be converted to monotonic or anti-monotonic if items can be properly ordered in processing

**Data space pruning constraints**

Data succinct: Data space can be pruned at the initial pattern mining process

Data anti-monotonic: If a transaction t does not satisfy c, then t can be pruned to reduce data processing effort

**Pattern Space Pruning with Pattern Anti-Monotonicity**

Constraint c is anti-monotone (If an itemset S violates constraint c, so does any of its superset; That is, mining on itemset S can be terminated)

Ex. 1: c1 : sum(S.price) ≤ v is anti-monotone

Ex. 2: c2 : range(S.profit) ≤ 15 is anti-monotone (Itemset ab violates c2 (range(ab) = 40), So does every superset of ab)

Ex. 3. c3 : sum(S.Price) ≥ v is not anti-monotone

Ex. 4. Is c4 : support(S) ≥ σ anti-monotone? (Yes! Apriori pruning is essentially pruning with an anti-monotonic constraint!)

**Pattern Monotonicity and Its Roles**

A constraint c is monotone: if an itemset S satisfies the constraint c, so does any of its superset (That is, we do not need to check c in subsequent mining)

Ex. 1: c1 : sum(S.Price) ≥ v is monotone

Ex. 2: c2 : min(S.Price) ≤ v is monotone

Ex. 3: c3 : range(S.profit) ≥ 15 is monotone (Itemset ab satisfies c; So does every superset of ab)

**Data Space Pruning with Data Anti-Monotonicity**

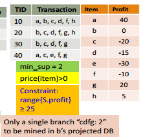
A constraint c is data anti-monotone: In the mining process, if a data entry t cannot satisfy a pattern p under c, t cannot satisfy p’s superset either (Data space pruning: Data entry t can be pruned)

Ex. 1: c1 : sum(S.Profit) ≥ v is data anti-monotone (Let constraint c1 be: sum{S.Profit} ≥ 25: T30: {b, c, d, f, g} can be removed since none of their combinations can make an S whose sum of the profit is ≥ 25)

Ex. 2: c2 : min(S.Price) ≤ v is data anti-monotone (Consider v = 5 but every item in transaction T50 has a price higher than 10)

Ex. 3: c3 : range(S.Profit) ≥ 25 is data anti-monotone

**Data Space Pruning Should Be Explored Recursively**

Example. c3 : range(S.Profit) > 25:

We check b’s projected database (But item “a” is infrequent (sup = 1))

After removing “a (40)” from T10 (T10 cannot satisfy c3 any more : since “b (0)” and “c (-20), d (-15), f (-10), h (5)”; By removing T10, we can also prune “h” in T20) 

**Succinctness: Pruning Both Data and Pattern Spaces**

**Succinctness**: if the constraint c can be enforced by directly manipulating the data

Ex. 1: To find those patterns without item I (Remove i from DB and then mine (pattern space pruning))

Ex. 2: To find those patterns containing item I (Mine only i-projected DB (data space pruning))

Ex. 3: c3 : min(S.Price) ≤ v is succinct (Start with only items whose price ≤ v (pattern space pruning) and remove transactions with high-price items only (data space pruning))

Ex. 4: c4 : sum(S.Price) ≥ v is not succinct (It cannot be determined beforehand since sum of the price of itemset S keeps increasing)

**Convertible Constraints: Ordering Data in Transactions**

Convert tough constraints into (anti-)monotone by proper ordering of items in transactions

**Examine c1 : avg(S.profit) > 20**

Order items in value-descending order ( <a,g,f,h,b,d,c,e>)

An itemset ab violates c1 (avg(ab) = 20) (So does ab\* (i.e., ab-projected DB); C1 : anti-monotone if patterns grow in the right order!)

Can item-reordering work for Apriori? (Does not work for level-wise candidate generation!; avg(agf) = 23.3 > 20, but avg(gf) = 15 < 20)

**How to Handle Multiple Constraints?**

**It is beneficial to use multiple constraints in pattern mining**

**But different constraints may require potentially conflicting item-ordering** (If there exists an order R making both c1 and c2 convertible, try to sort items in the order that benefits pruning most; If there exists conflict ordering between c1 and c2 : (Try to sort data and enforce one constraint first (which one?); Then enforce the other when mining the projected databases)

**Ex. c1 : avg(S.profit) > 20, and c2 : avg(S.price) < 50** (Sorted in profit descending order and use c1 first(assuming c1 has more pruning power); For each project DB, sort trans. in price ascending order and use c2 at mining)

**Mining Long Patterns: Challenges**

**Mining long patterns is needed in bioinformatics, social network analysis, software engineering, …** But the methods introduced so far mine only short patterns (e.g., length < 10)

**Challenges of mining long patterns**

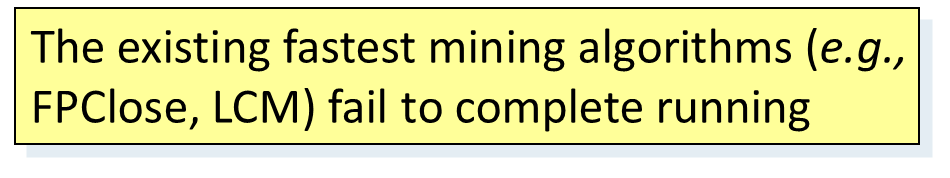
- The curse of “downward closure” property of frequent patterns

--- Any sub-pattern of a frequent pattern is frequent

--- If {a1, a2, …, a100} is frequent, then {a1}, {a2}, …, {a100}, {a1, a2}, {a1, a3}, …, {a1, a100}, {a1, a2, a3}, … are all frequent! There are about 2100 such frequent itemsets!

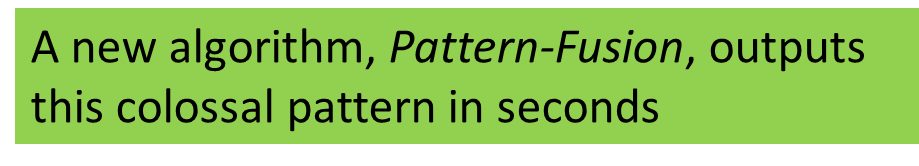
- No matter searching in breadth-first (e.g., Apriori) or depth-first (e.g., FPgrowth), if we still adopt the “small to large” step-by-step growing paradigm, we have to examine so many patterns, which leads to combinatorial explosion!

**Colossal Patterns: A Motivating Example**

**T1 = 2 3 4 ….. 39 40**

**T2 = 1 3 4 ….. 39 40**

**: .**

**: .**

**: .**

**: .**

**T40=1 2 3 4 …… 39**

**T41= 41 42 43 ….. 79**

**T42= 41 42 43 ….. 79**

**: .**

**: .**

**T60= 41 42 43 … 79**

- Let min-support σ= 20

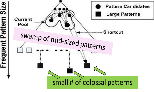
- # of closed/maximal patterns of size 20: about

- But there is only one pattern with size close to 40 (i.e., long or colossal) ---α= {41,42,…,79} of size 39

- Q: How to find it without generating an exponential number of size-20 patterns?

**What Is Pattern-Fusion?**

- Not strive for completeness (why?)

- Jump out of the swamp of the mid-sized intermediate “results”

- Strive for mining almost complete and representative colossal patterns: identify “short-cuts” and take “leaps”

- Key observation

--- The larger the pattern or the more distinct the pattern, the greater chance it will be generated from small ones

- Philosophy: Collection of small patterns hints at the larger patterns

- Pattern fusion strategy (“not crawl but jump”): Fuse small patterns together in one step to generate new pattern candidates of significant sizes

**Observation: Colossal Patterns and Core Patterns**

- Suppose dataset D contains 4 colossal patterns (below) plus many small patterns ({a1, a2, …, a50}: 40, {a3, a6, …, a99}: 60, {a5, a10, …, a95}: 80, {a10, a20, …, a100}: 100)

- If you check the pattern pool of size-3, you may likely find

({a2, a4, a45}: ~40; {a3, a34, a39}: ~40; …, {a5, a15, a85}: ~80, …, {a20, a40, a85}: ~80, …)

- If you merge the patterns with similar support, you may obtain candidates of much bigger size and easily validate whether they are true patterns

- Core patterns of a colossal pattern α: A set of subpatterns of α that cluster around α by sharing a similar support

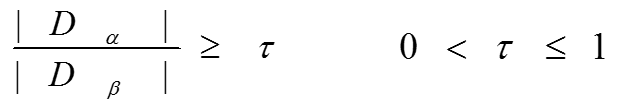
- A colossal pattern has far more core patterns than a small-sized pattern

- A random draw from a complete set of pattern of size c would be more likely to pick a core pattern (or its descendant) of a colossal pattern

- A colossal pattern can be generated by merging a set of core patterns

**Robustness of Colossal Patterns**

- Core Patterns: For a frequent pattern α, a subpattern β is a τ-core pattern of α if β shares a similar support set with α, i.e.,

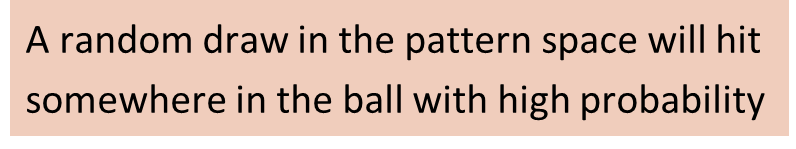


where τ is called the core ratio

- (d, τ)-robustness: A pattern α is (d, τ)-robust if d is the maximum number of items that can be removed from α for the resulting pattern to remain a τ-core pattern of α

- For a (d, τ)-robust pattern α, it has  core patterns

- Robustness of Colossal Patterns: A colossal pattern tends to have much more core patterns than small patterns

- Such core patterns can be clustered together to form “dense balls” based on pattern distance defined by 

**The Pattern-Fusion Algorithm**

- Initialization (Creating initial pool): Use an existing algorithm to mine all frequent patterns up to a small size, e.g., 3

- Iteration (Iterative Pattern Fusion):

--- At each iteration, K seed patterns are randomly picked from the current pattern pool

--- For each seed pattern thus picked, we find all the patterns within a bounding ball centered at the seed pattern

--- All these patterns found are fused together to generate a set of super-patterns

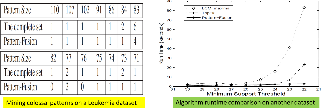
--- All the super-patterns thus generated form a new pool for the next iteration

- Termination: when the current pool contains no more than K patterns at the beginning of an iteration

**Experimental Results on Data Set: ALL**

- ALL: A popular gene expression clinical data set on ALL-AML leukemia, with 38 transactions, each with 866 columns. There are 1736 items in total.

--- When minimum support is high (e.g., 30), Pattern-Fusion gets all the largest colossal patterns with size greater than 85



**Sequence Databases & Sequential Patterns**

- **Sequential pattern mining has broad applications**

--- Customer shopping sequences

----- Purchase a laptop first, then a digital camera, and then a smartphone, within 6 months

--- Medical treatments, natural disasters (e.g., earthquakes), science & engineering processes, stocks and markets, ...

--- Weblog click streams, calling patterns, …

--- Software engineering: Program execution sequences, …

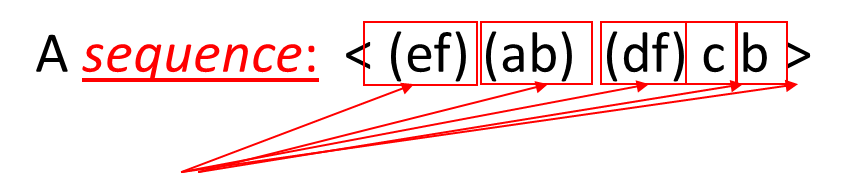
--- Biological sequences: DNA, protein, …

- **Transaction DB, sequence DB vs. time-series DB**

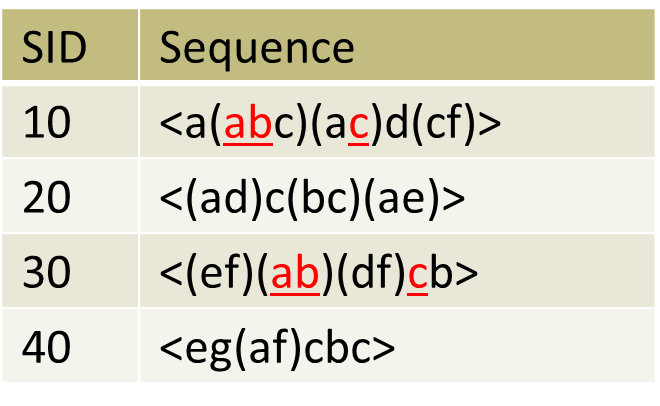
- **Gapped vs. non-gapped sequential patterns**

--- Shopping sequences, clicking streams vs. biological sequences

**Sequential Pattern and Sequential Pattern Mining**

- Sequential pattern mining: Given a set of sequences, find the complete set of frequent subsequences (i.e., satisfying the min\_sup threshold)

A sequence database

An element may contain a set of items (also called events)

Items within an element are unordered and we list them alphabetically

Given support threshold min\_sup = 2, <(ab)c> is a sequential pattern

**Sequential Pattern Mining Algorithms**

- Algorithm requirement: Efficient, scalable, finding complete set, incorporating various kinds of user-specific constraints

- The Apriori property still holds: If a subsequence s1 is infrequent, none of s1’s super-sequences can be frequent

- Representative algorithms

--- GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT’96)

--- Vertical format-based mining: SPADE (Zaki@Machine Leanining’00)

--- Pattern-growth methods: PrefixSpan (Pei, et al. @TKDE’04)

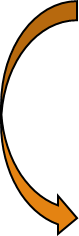
- Mining closed sequential patterns: CloSpan (Yan, et al. @SDM’03)

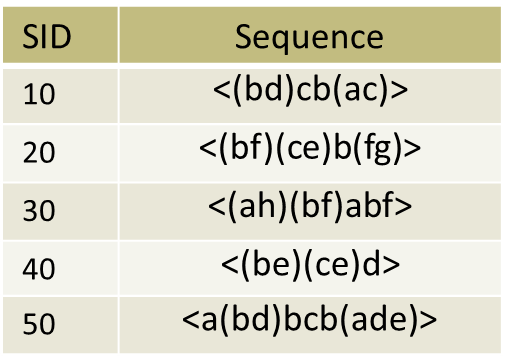
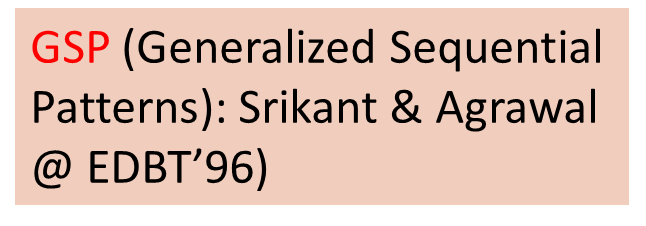
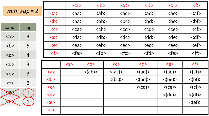
- Constraint-based sequential pattern mining

**GSP: Apriori-Based Sequential Pattern Mining**

- Initial candidates: All singleton sequences

--- <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>

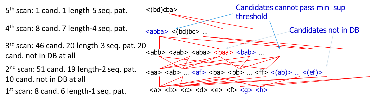
- Scan DB once, count support for each candidate

- Generate length-2 candidate sequences

- Length-2 candidates: 36 + 15= 51

- Without Apriori pruning: 8\*8+8\*7/2=92 candidates

**GSP Mining and Pruning**



- Repeat (for each level (i.e., length-k))

--- Scan DB to find length-k frequent sequences

--- Generate length-(k+1) candidate sequences from length-k frequent sequences using Apriori

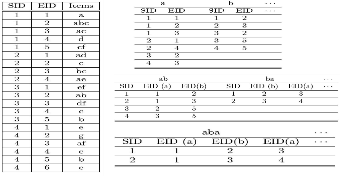
--- set k = k+1

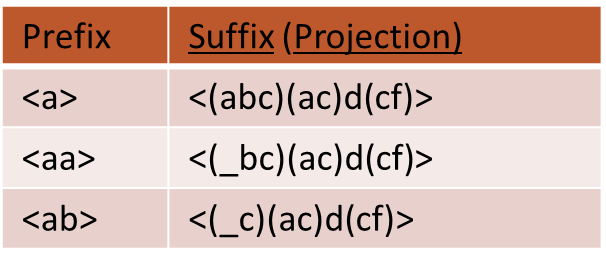
- Until no frequent sequence or no candidate can be found

**Sequential Pattern Mining in Vertical Data Format: The SPADE Algorithm**

- A sequence database is mapped to: <SID, EID>

- Grow the subsequences (patterns) one item at a time by Apriori candidate generation



**PrefixSpan: A Pattern-Growth Approach**

- Prefix and suffix

--- Given <a(abc)(ac)d(cf)>

--- Prefixes: <a>, <aa>, <a(ab)>, <a(abc)>, …

--- Suffix: Prefixes-based projection

- PrefixSpan Mining: Prefix Projections

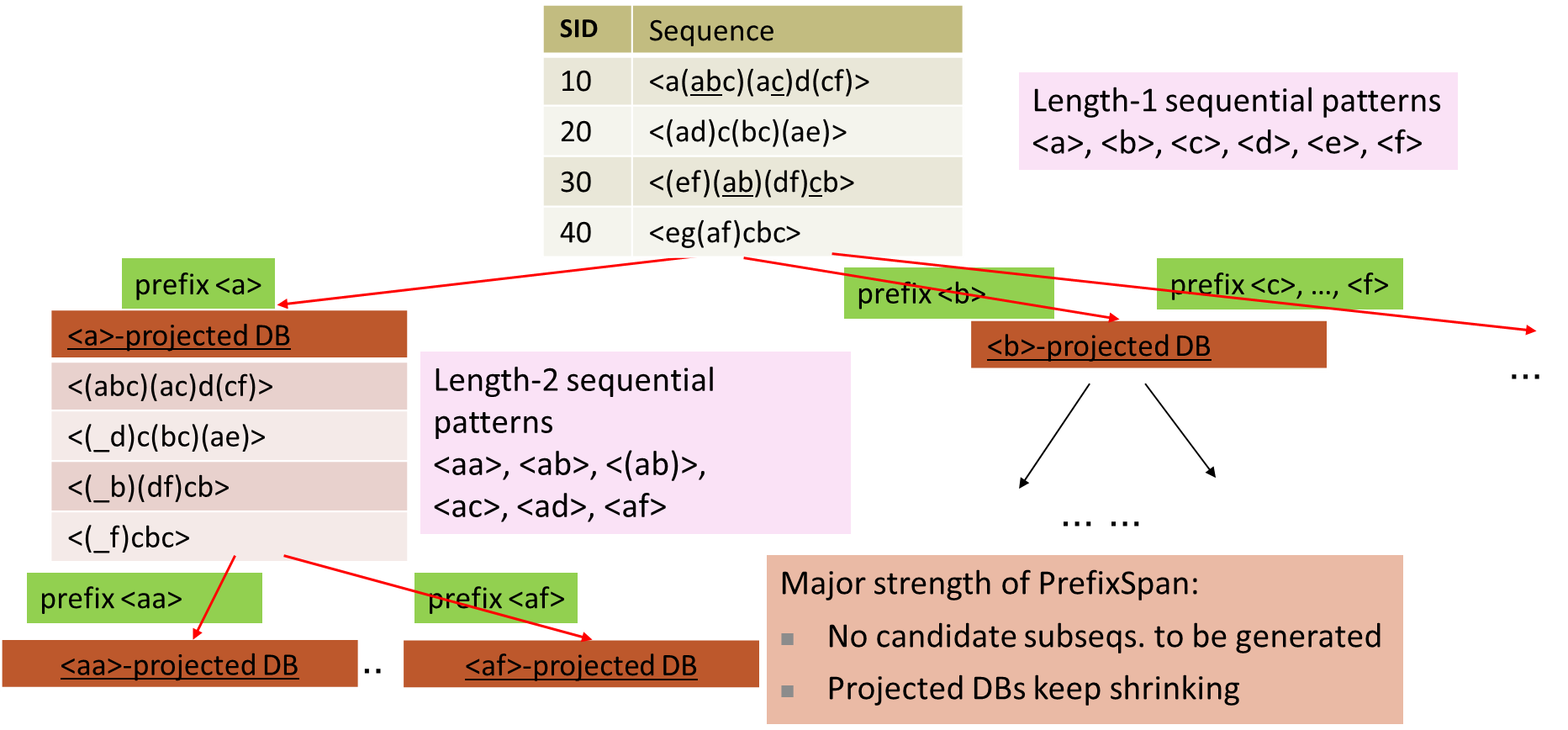
--- Step 1: Find length-1 sequential patterns

----- <a>, <b>, <c>, <d>, <e>, <f>

--- Step 2: Divide search space and mine each projected DB

----- <a>-projected DB, <b>-projected DB, … <f>-projected DB, …

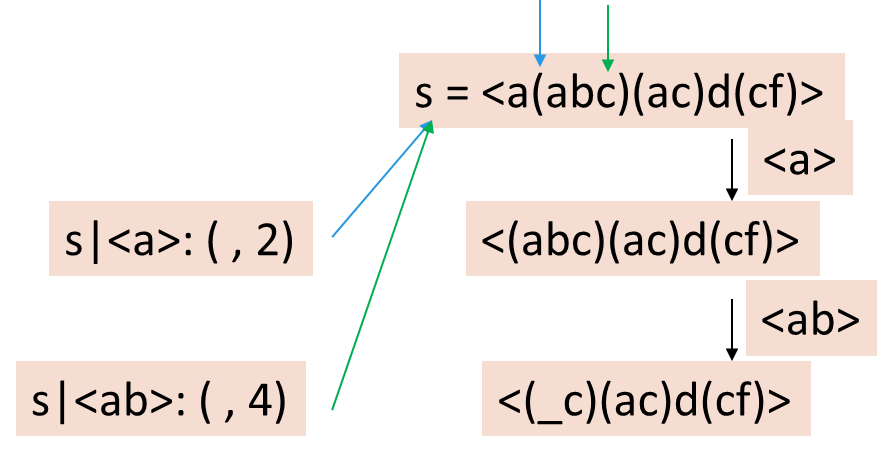
**PrefixSpan: Mining Prefix-Projected DBs**

****

**Implementation Consideration: Pseudo-Projection vs. Physical Projection**

- Major cost of PrefixSpan: Constructing projected DBs

--- Suffixes largely repeating in recursive projected DBs

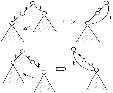
- When DB can be held in main memory, use pseudo projection --- No physically copying suffixes --- Pointer to the sequence --- Offset of the suffix - But if it does not fit in memory --- Physical projection

- Suggested approach: --- Integration of physical and pseudo-projection

--- Swapping to pseudo-projection when the data fits in memory

**CloSpan: Mining Closed Sequential Patterns**

- A closed sequential pattern s: There exists no superpattern s’ such that s’ כ s, and s’ and s have the same support - Which ones are closed? <abc>: 20, <abcd>:20, <abcde>: 15 - Why directly mine closed sequential patterns? --- Reduce # of (redundant) patterns --- Attain the same expressive power

- Property P1: If s כ s1, s is closed iff two project DBs have the same size

- Explore Backward Subpattern and Backward Superpattern pruning to prune redundant search space

- Greatly enhances efficiency (Yan, et al., SDM’03)

**Constraint-Based Sequential-Pattern Mining**

- Share many similarities with constraint-based itemset mining

- Anti-monotonic: If S violates c, the super-sequences of S also violate c --- sum(S.price) < 150; min(S.value) > 10 - Monotonic: If S satisfies c, the super-sequences of S also do so --- element\_count (S) > 5; S ⊇{PC, digital\_camera} - Data anti-monotonic: If a sequence s1 with respect to S violates c3, s1 can be removed --- c3: sum(S.price) ≥ v - Succinct: Enforce constraint c by explicitly manipulating data --- S ⊇ {i-phone, MacAir} - Convertible: Projection based on the sorted value not sequence order --- value\_avg(S) < 25; profit\_sum (S) > 160

--- max(S)/avg(S) < 2; median(S) – min(S) > 5

**Timing-Based Constraints in Seq.-Pattern Mining**

- Order constraint: Some items must happen before the other

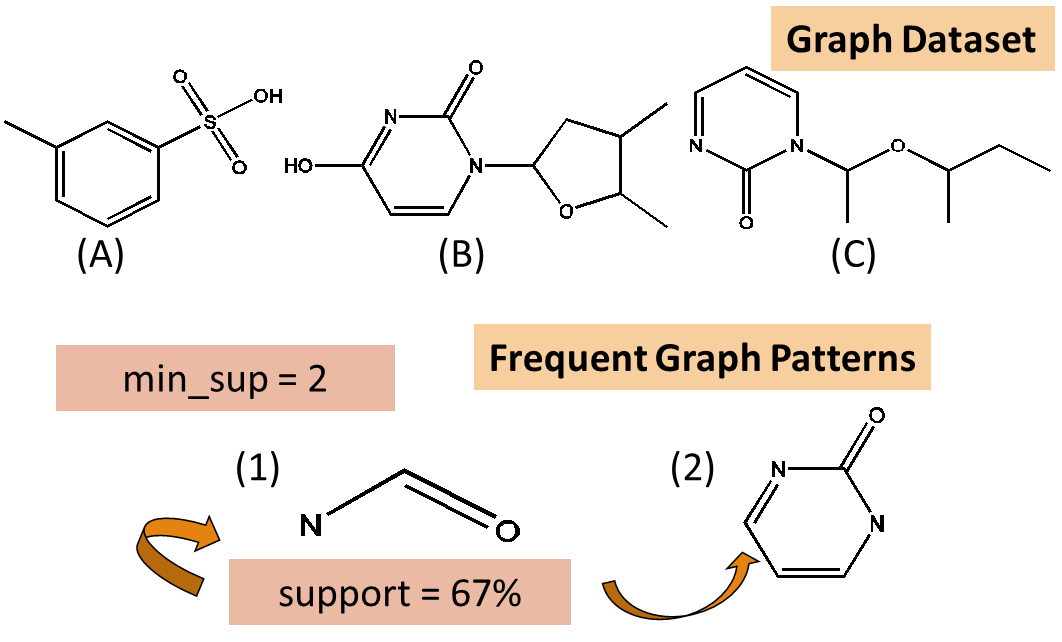
--- {algebra, geometry} → {calculus} (where “→” indicates ordering) --- Anti-monotonic: Constraint-violating sub-patterns pruned - Min-gap/max-gap constraint: Confines two elements in a pattern --- E.g., mingap = 1, maxgap = 4 --- Succinct: Enforced directly during pattern growth - Max-span constraint: Maximum allowed time difference between the 1st and the last elements in the pattern --- E.g., maxspan (S) = 60 (days) --- Succinct: Enforced directly when the 1st element is determined - Window size constraint: Events in an element do not have to occur at the same time: Enforce max allowed time difference --- E.g., window-size = 2: Various ways to merge events into elements

**Episodes and Episode Pattern Mining**

- Episodes and regular expressions: Alternative to seq. patterns --- Serial episodes: A → B --- Parallel episodes: A | B (Indicating partial order relationships) --- Regular expressions: (A|B)C\*(D → E)

- Methods for episode pattern mining --- Variations of Apriori/GSP-like algorithms --- Projection-based pattern growth ----- Q1: Can you work out the details? ----- Q2: What are the differences between mining episodes and constraint-based pattern mining?

**Frequent (Sub)Graph Patterns**

- Given a labeled graph dataset D = {G1, G2, …, Gn), the supporting graph set of a subgraph g is Dg = {Gi | g  Gi, Gi D}.

--- support(g) = |Dg|/ |D|

- A (sub)graph g is frequent if support(g) ≥ min\_sup

- Ex.: Chemical structures

- Alternative:

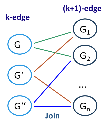
--- Mining frequent subgraph patterns from a single large graph or network

**Applications of Graph Pattern Mining**

- Bioinformatics --- Gene networks, protein interactions, metabolic pathways - Chem-informatics: Mining chemical compound structures - Social networks, web communities, tweets, …

- Cell phone networks, computer networks, … - Web graphs, XML structures, semantic Web, information networks - Software engineering: program execution flow analysis - Building blocks for graph classification, clustering, compression, comparison, and correlation analysis - Graph indexing and graph similarity search

**Graph Pattern Mining Algorithms: Different Methodologies**

****- Generation of candidate subgraphs --- Apriori vs. pattern growth (e.g., FSG vs. gSpan) - Search order --- Breadth vs. depth - Elimination of duplicate subgraphs --- Passive vs. active (e.g., gSpan (Yan&Han’02)) - Support calculation --- Store embeddings (e.g., GASTON (Nijssen&Kok’04, FFSM (Huan, et al.’03), MoFa (Borgelt and Berthold ICDM’02)) - Order of pattern discovery

---Path 🡪 tree 🡪 graph (e.g., GASTON (Nijssen&Kok’04)

**Apriori-Based Approach**

- The Apriori property (anti-monotonicity): A size-*k* subgraph is frequent if and only if all of its subgraphs are frequent

- A candidate size-(*k*+1) edge/vertex subgraph is generated if its corresponding two *k*-edge/vertex subgraphs are frequent

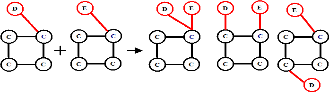
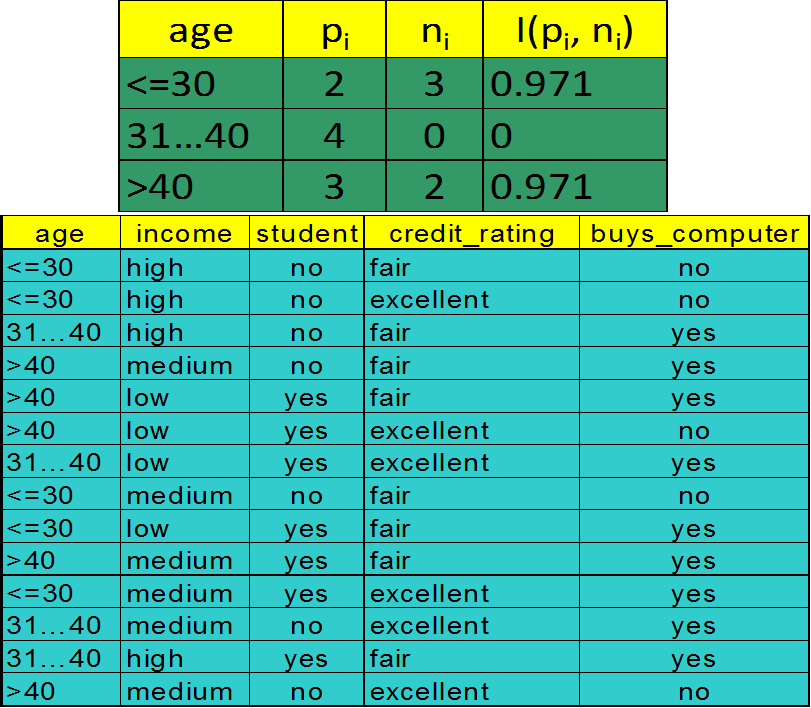
- Iterative mining process:

--- Candidate-generation 🡪 candidate pruning 🡪 support counting 🡪 candidate elimination

**Candidate Generation: Vertex Growing vs. Edge Growing**

- Methodology: breadth-search, Apriori joining two size-*k* graphs

--- Many possibilities at generating size-(*k*+1) candidate graphs

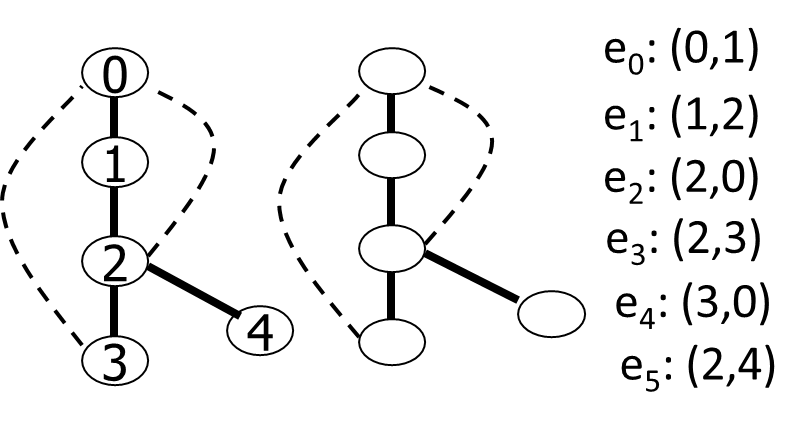


- Generating new graphs with one more vertex --- AGM (Inokuchi, et al., PKDD’00) - Generating new graphs with one more edge

--- FSG (Kuramochi and Karypis, ICDM’01)

- Performance shows via edge growing is more efficient

**gSPAN: Graph Pattern Growth in Order**

- Right-most path extension in subgraph pattern growth

--- Right-most path: The path from root to the right-most leaf (choose the vertex w. the smallest index at each step)

--- Reduce generation of duplicate subgraphs

- Completeness: The Enumeration of graphs using right-most path extension is complete

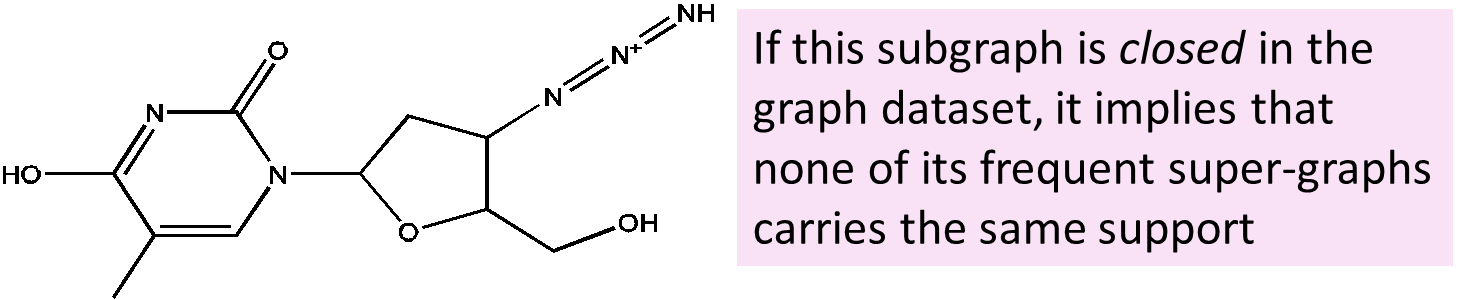
- DFS Code: Flatten a graph into a sequence using depth-first search

**Why Mining Closed Graph Patterns?**

Challenge: An n-edge frequent graph may have 2n subgraphs

Motivation: Explore closed frequent subgraphs to handle graph pattern explosion problem

A frequent graph G is closed if there exists no supergraph of G that carries the same support as G

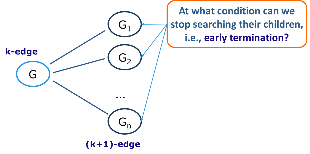


Lossless compression: Does not contain non-closed graphs, but still ensures that the mining result is complete

Algorithm CloseGraph: Mines closed graph patterns directly

**CLOSEGRAPH: Directly Mining Closed Graph Patterns**

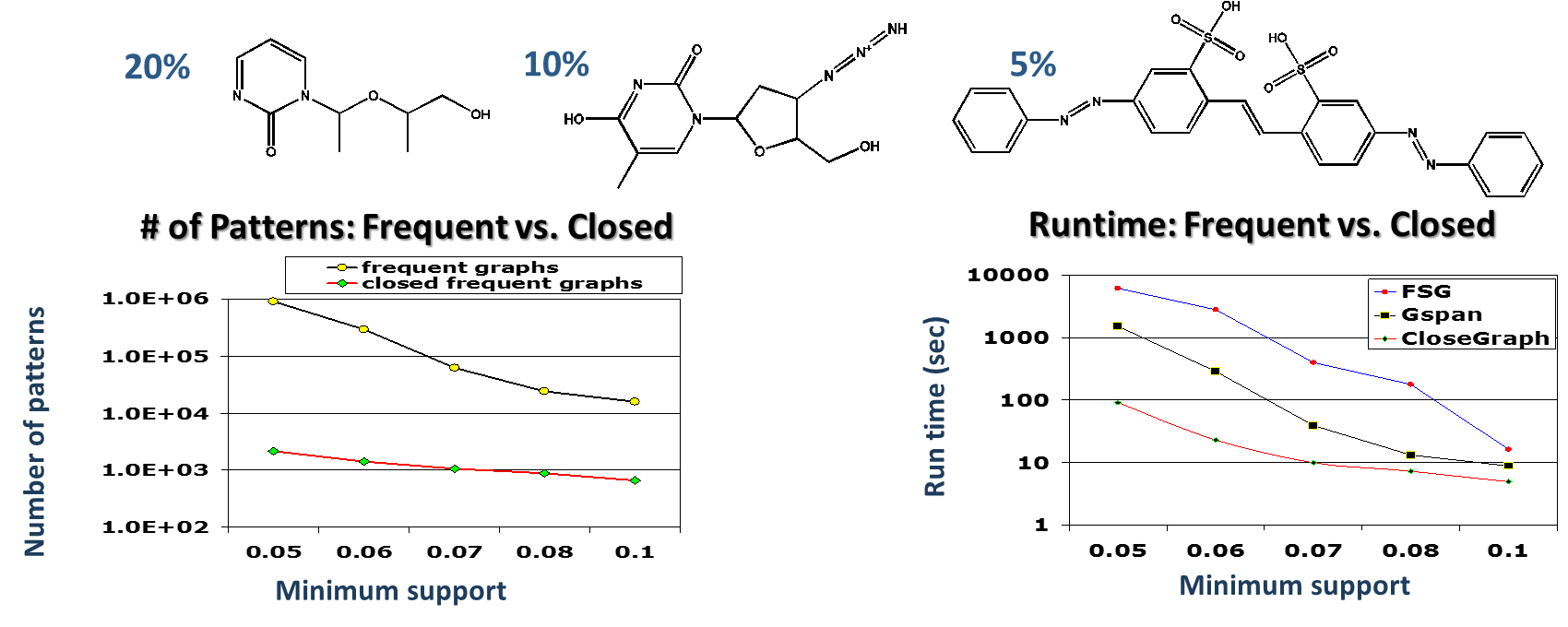
CloseGraph: Mining closed graph patterns by extending gSpan (Yan & Han, KDD’03)

****- Suppose G and G1 are frequent, and G is a subgraph of G1

- If in any part of the graph in the dataset where G occurs, G1 also occurs, then we need not grow G (except some special, subtle cases), since none of G’s children will be closed except those of G1

**Experiment and Performance Comparison**

- The AIDS antiviral screen compound dataset from NCI/NIH - The dataset contains 43,905 chemical compounds - Discovered Patterns: The smaller minimum support, the bigger and more interesting subgraph patterns discovered

****

**Chapter 8. Classification: Basic Concepts**

**Supervised vs. Unsupervised Learning**

- Supervised learning (classification)

--- Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations

--- New data is classified based on the training set

- Unsupervised learning (clustering)

--- The class labels of training data is unknown

--- Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

**Prediction Problems: Classification vs. Numeric Prediction**

- Classification

--- predicts categorical class labels (discrete or nominal)

--- classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data

- Numeric Prediction

--- models continuous-valued functions, i.e., predicts unknown or missing values

- Typical applications

--- Credit/loan approval:

--- Medical diagnosis: if a tumor is cancerous or benign

--- Fraud detection: if a transaction is fraudulent

--- Web page categorization: which category it is

**Classification—A Two-Step Process**

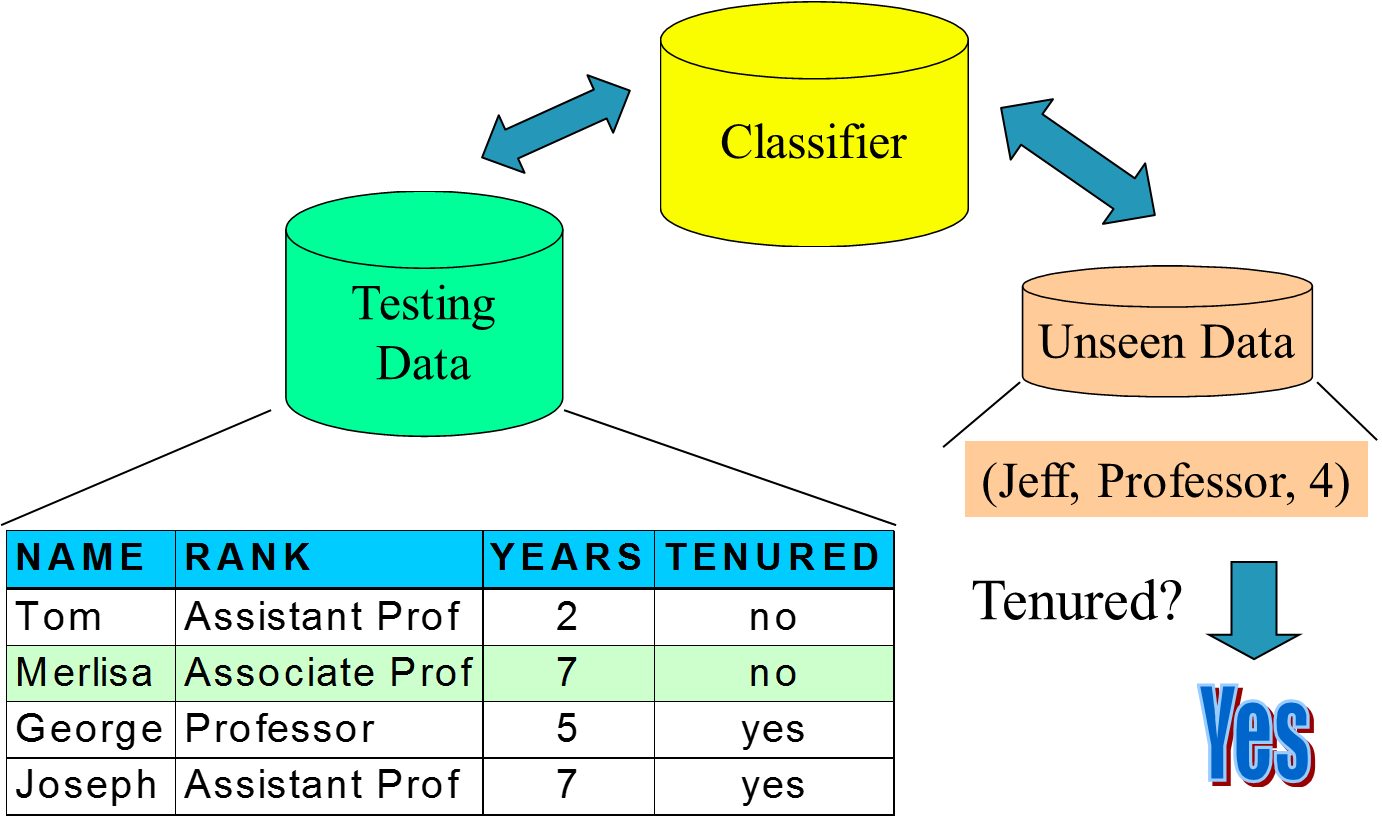
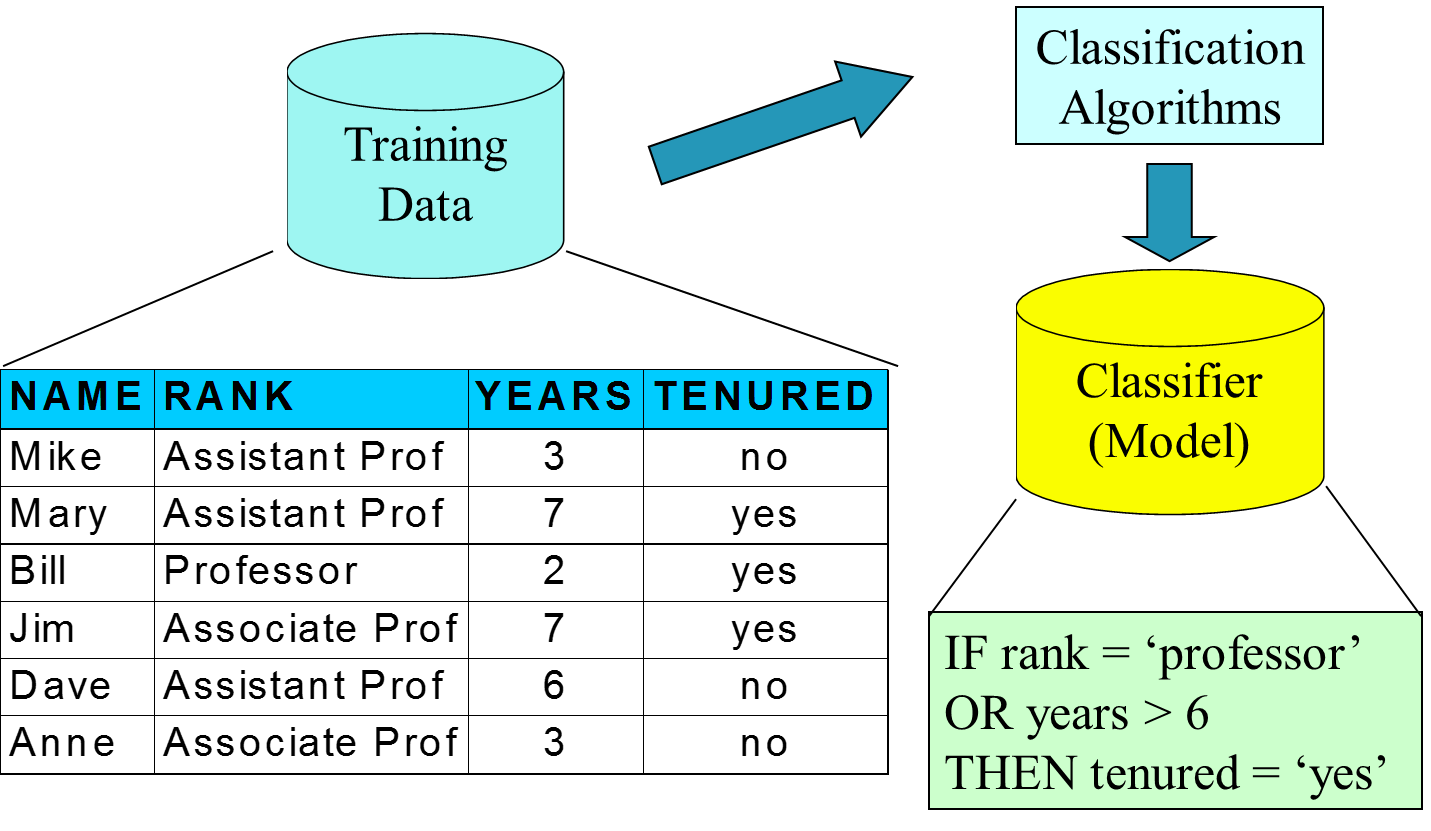
- Model construction: describing a set of predetermined classes --- Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute

--- The set of tuples used for model construction is training set --- Model: represented as classification rules, decision trees, or mathematical formulae

- Model usage: for classifying future or unknown objects --- Estimate accuracy of the model ----- The known label of test sample is compared with the classified result from the model ----- Accuracy: % of test set samples that are correctly classified by the model ----- Test set is independent of training set (otherwise overfitting) --- If the accuracy is acceptable, use the model to classify new data

- Note: If the test set is used to select/refine models, it is called validation (test) set or development test set

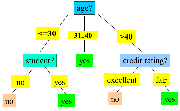
**Process (1): Model Construction && Process (2): Using the Model in Prediction**

****

**Decision Tree Induction: An Example**

- Training data set: Buys\_computer

- The data set follows an example of Quinlan’s ID3 (Playing Tennis) - Resulting tree:

**Algorithm for Decision Tree Induction**

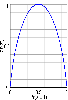
- Basic algorithm (a greedy algorithm) --- Tree is constructed in a top-down recursive divide-and-conquer manner --- At start, all the training examples are at the root --- Attributes are categorical (if continuous-valued, they are discretized in advance) --- Examples are partitioned recursively based on selected attributes --- Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)

- Conditions for stopping partitioning --- All samples for a given node belong to the same class --- There are no remaining attributes for further partitioning—majority voting is employed for classifying the leaf ---There are no samples left

**Brief Review of Entropy**

- Entropy (Information Theory)

--- A measure of uncertainty associated with a random number

--- Calculation: For a discrete random variable Y taking m distinct values {y1, y2, …, ym}



--- Interpretation ----- Higher entropy → higher uncertainty

----- Lower entropy → lower uncertainty

- Conditional entropy



**Attribute Selection Measure: Information Gain (ID3/C4.5)**

- Select the attribute with the highest information gain

- Let pi be the probability that an arbitrary tuple in D belongs to class Ci, estimated by |Ci, D|/|D| - Expected information (entropy) needed to classify a tuple in D: Info(D) =

- Information needed (after using A to split D into v partitions) to classify D: InfoA(D) =

Information gained by branching on attribute A:

Gain(A) = Info(D) – InfoA(D)

**Attribute Selection: Information Gain**

Class P: buys\_computer = “yes”

Class N: buys\_computer = “no”

Info(D) = I(9,5) = - 9/14log2(9/14) - 5/14 log2(5/14) = 0.94

Infoage(D) = 5/14 I(2,3) + 4/14 I(4,0)+5/14 I(3,2) = 0.694

Hence Gain(age) = Info(D) - Infoage(D) = 0.246; Gain(income) = 0.029; Gain(student) = 0.151; Gain(credit\_rating) = 0.048

**Computing Information-Gain for Continuous-Valued Attributes**

- Let attribute A be a continuous-valued attribute

- Must determine the best split point for A --- Sort the value A in increasing order --- Typically, the midpoint between each pair of adjacent values is considered as a possible split point

----- (ai+ai+1)/2 is the midpoint between the values of ai and ai+1 --- The point with the minimum expected information requirement for A is selected as the split-point for A

- Split: --- D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

**Gain Ratio for Attribute Selection (C4.5)**

- Information gain measure is biased towards attributes with a large number of values

- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

SplitInfoA(D) =

--- GainRatio(A) = Gain(A)/SplitInfo(A) Ex. SplitInfoIncome(D) = -4/14\*log2(4/14) - -6/14\*log2(6/14) -4/14\*log2(4/14) = 1.557 --- gain\_ratio(income) = 0.029/1.557 = 0.019

- The attribute with the maximum gain ratio is selected as the splitting attribute

**Gini Index (CART, IBM IntelligentMiner)**

- If a data set D contains examples from n classes, gini index, gini(D) is defined as: gini(D) = where pj is the relative frequency of class j in D - If a data set D is split on A into two subsets D1 and D2, the gini index gini(D) is defined as - Reduction in Impurity:

- The attribute provides the smallest ginisplit(D) (or the largest reduction in impurity) is chosen to split the node (need to enumerate all the possible splitting points for each attribute)

**Computation of Gini Index**

- Ex. D has 9 tuples in buys\_computer = “yes” and 5 in “no”: gini(D) =

- Suppose the attribute income partitions D into 10 in D1: {low, medium} and 4 in D2

Giniincome{low,medium}(D) = (10/14)Gini(D1) + (4/14)Gini(D2) = = Giniincome{high}(D); Gini{low,high} is 0.458; Gini{medium,high} is 0.450. Thus, split on the {low,medium} (and {high}) since it has the lowest Gini index

- All attributes are assumed continuous-valued - May need other tools, e.g., clustering, to get the possible split values - Can be modified for categorical attributes

**Comparing Attribute Selection Measures**

- The three measures, in general, return good results but

--- Information gain: ----- biased towards multivalued attributes --- Gain ratio: -----tends to prefer unbalanced splits in which one partition is much smaller than the others ---Gini index: -----biased to multivalued attributes -----has difficulty when # of classes is large -----tends to favor tests that result in equal-sized partitions and purity in both partitions

**Other Attribute Selection Measures**

- CHAID: a popular decision tree algorithm, measure based on χ2 test for independence - C-SEP: performs better than info. gain and gini index in certain cases - G-statistic: has a close approximation to χ2 distribution - MDL (Minimal Description Length) principle (i.e., the simplest solution is preferred):

--- The best tree as the one that requires the fewest # of bits to both (1) encode the tree, and (2) encode the exceptions to the tree - Multivariate splits (partition based on multiple variable combinations) --- CART: finds multivariate splits based on a linear comb. of attrs. - Which attribute selection measure is the best? --- Most give good results, none is significantly superior than others

**Overfitting and Tree Pruning**

- Overfitting: An induced tree may overfit the training data

--- Too many branches, some may reflect anomalies due to noise or outliers --- Poor accuracy for unseen samples

- Two approaches to avoid overfitting

--- Prepruning: Halt tree construction early ̵ do not split a node if this would result in the goodness measure falling below a threshold ----- Difficult to choose an appropriate threshold

--- Postpruning: Remove branches from a “fully grown” tree—get a sequence of progressively pruned trees

----- Use a set of data different from the training data to decide which is the “best pruned tree”

**Classification in Large Databases**

- Classification—a classical problem extensively studied by statisticians and machine learning researchers

- Scalability: Classifying data sets with millions of examples and hundreds of attributes with reasonable speed

- Why is decision tree induction popular? --- relatively faster learning speed (than other classification methods) --- convertible to simple and easy to understand classification rules --- can use SQL queries for accessing databases --- comparable classification accuracy with other methods

- RainForest (VLDB’98 — Gehrke, Ramakrishnan & Ganti)

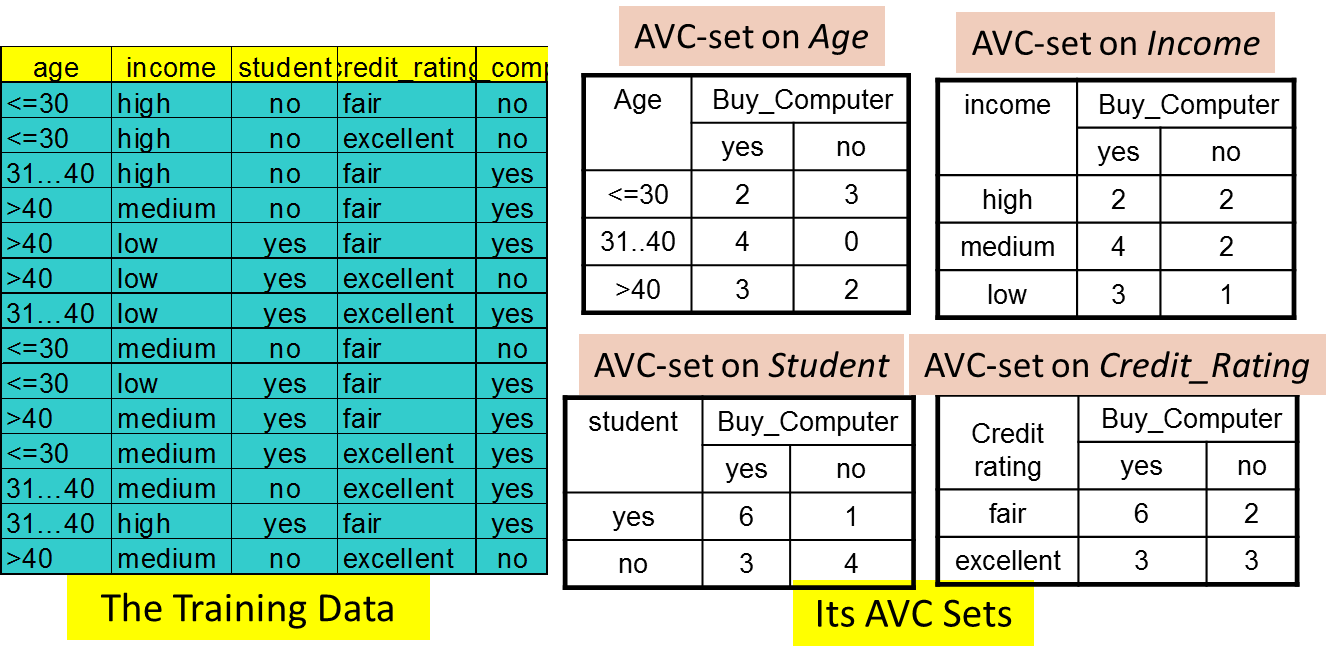
---Builds an AVC-list (attribute, value, class label)

**RainForest: A Scalable Classification Framework**

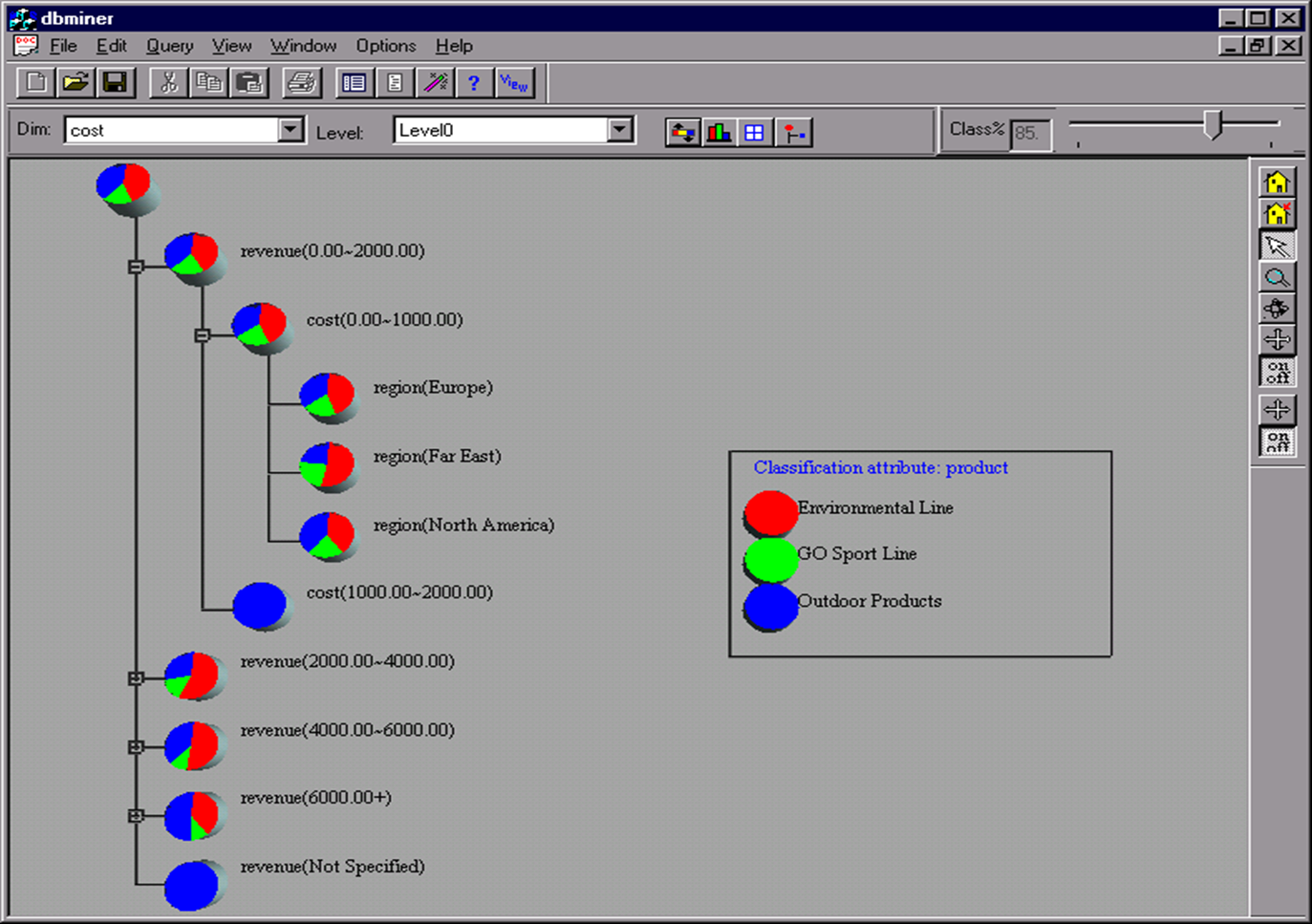
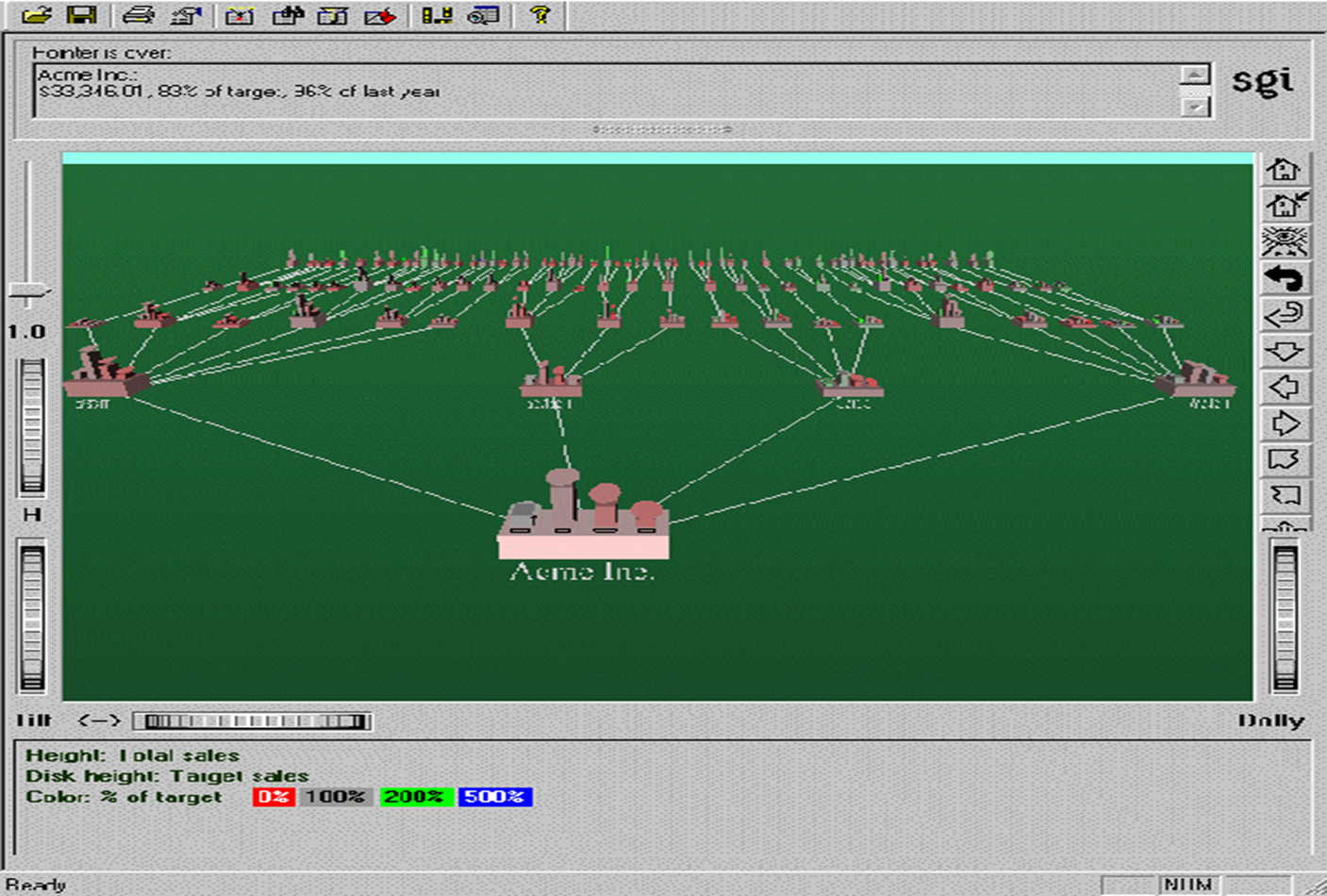
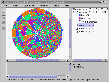
- The criteria that determine the quality of the tree can be computed separately --- Builds an AVC-list: AVC (Attribute, Value, Class\_label)

- AVC-set (of an attribute X ) --- Projection of training dataset onto the attribute X and class label where counts of individual class label are aggregated

- AVC-group (of a node n ) --- Set of AVC-sets of all predictor attributes at the node n



**Presentation of Classification Results/ Visualization of a Decision Tree in SGI/MineSet 3.0 / Interactive Visual Mining by Perception-Based Classification (PBC)**

**Bayesian Classification: Why?**

- A statistical classifier: performs probabilistic prediction, i.e., predicts class membership probabilities

- Foundation: Based on Bayes’ Theorem.

- Performance: A simple Bayesian classifier, naïve Bayesian classifier, has comparable performance with decision tree and selected neural network classifiers

- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data

- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

**Bayes’ Theorem: Basics**

- Total probability Theorem: P(B) =

- Bayes’ Theorem: P(H|X) = --- Let X be a data sample (“evidence”): class label is unknown --- Let H be a hypothesis that X belongs to class C --- Classification is to determine P(H|X), (i.e., posteriori probability): the probability that the hypothesis holds given the observed data sample X --- P(H) (prior probability): the initial probability ----- E.g., X will buy computer, regardless of age, income, … --- P(X): probability that sample data is observed --- P(X|H) (likelihood): the probability of observing the sample X, given that the hypothesis holds ---- E.g., Given that X will buy computer, the prob. that X is 31..40, medium income

**Prediction Based on Bayes’ Theorem**

- Given training data X, posteriori probability of a hypothesis H, P(H|X), follows the Bayes’ theorem :

P(H|X) =

- Informally, this can be viewed as : posteriori = likelihood x prior/evidence

- Predicts X belongs to Ci iff the probability P(Ci|X) is the highest among all the P(Ck|X) for all the k classes

- Practical difficulty: It requires initial knowledge of many probabilities, involving significant computational cost

**Classification Is to Derive the Maximum Posteriori**

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector X = (x1, x2, …, xn) - Suppose there are m classes C1, C2, …, Cm. - Classification is to derive the maximum posteriori, i.e., the maximal P(Ci|X) - This can be derived from Bayes’ theorem: - Since P(X) is constant for all classes, only needs to be maximized

**Naïve Bayes Classifier**

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

- This greatly reduces the computation cost: Only counts the class distribution

- If Ak is categorical, P(xk|Ci) is the # of tuples in Ci having value xk for Ak divided by |Ci, D| (# of tuples of Ci in D)

- If Ak is continous-valued, P(xk|Ci) is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

and P(xk|Ci) is