H

Handhelds Interfaces

▶ Mobile Interfaces

Handwritten Text

▶ Electronic Ink Indexing

Hanf-Locality

▶ Locality of Queries

Hard Disk

▶ Disk

Hardware-Conscious Database System

▶ Architecture-Conscious Database System

Harmonic Mean of Recall and Precision

▶ F-Measure

Hash File

▶ Hash-based Indexing

Hash Filter

▶ Bloom Filters

Hash Filter Join

▶ Semijoin

Hash Functions

Marina Blanton
University of Notre Dame, Notre Dame, IN, USA

Synonyms

Cryptographic hash functions; One-way hash functions

Definition

A hash function is a well-defined deterministic algorithm that takes as input data of arbitrary length and produces a short fixed-length digital representation of the data, or a digest, as its output. The output of a hash function can serve the role of a digital "fingerprint" of the input data, as an important design property of hash functions is that of collision resilience: two hashes produced on different inputs are very unlikely to result in the same value. Furthermore, given a hash function output, it is normally infeasible to find a (previously unseen) input that matches that output (this property is called preimage resistance).

Key Points

Hash functions have many uses, and in cryptography they are widely used because of their collusion resistance and preimage resistance properties. In particular, hash functions are used to verify integrity of messages and can be used to construct *message authentication codes*. They are also used in many cryptographic protocols and to construct cryptographic primitives (e.g., to construct symmetric encryption schemes and in digital signatures) [4].

The two most commonly used hash functions are MD5 [3] and SHA-1 [1], but a variety of other algorithms is also available. Both MD5 and SHA-1, however, should be used with caution because a recent development of attacks indicate that MD5 is too weak to meet the necessary cryptographic properties and it may become possible to also attack SHA-1 in the near future. SHA-2 [2] is a collection of new hash functions from the SHA family (such as SHA-256, SHA-224, SHA-384, and SHA-512) which were designed by the National Security Agency (NSA) to be a new standard.

Cross-references

► Digital Signatures
► Merkle Hash Trees
► Message Authentication Codes

Recommended Reading

1. Eastlake D. and Jones P. US Secure Hash Algorithm 1 (SHA1). IETF RFC 3174, 2001. http://www.ietf.org/rfc/rfc3174.txt.
2. National Institute of Standards and Technology (NIST). FIPS 180-2: Secure Hash Standard (SHS), Current version of the Secure Hash Standard (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512), 2004.
3. Rivest R. The MD5 message-digest algorithm. IETF RFC 1321, 1992. http://www.ietf.org/rfc/rfc1321.txt.
4. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code (2nd edn.). Wiley, New York, NY, 1996.

Hash Join

JINGREN ZHOU
Microsoft Research, Redmond, WA, USA

Synonyms

Hash join

Definition

The hash join is a common join algorithm in database systems using hashing. The join predicate needs to be an equality join predicate. The classic algorithm consists of two phases: the "build" phase and the "probe" phase. In the "build" phase, the algorithm builds a hash table on the smaller relation, say *R*, by applying a hash function to the join attribute of each tuple. In the "probe" phase, the algorithm probes the hash table using tuples of the larger relation, say *S*, to find matches.

Key Points

The classic algorithm is simple, but it requires that the smaller join relation fits into memory. If there is no enough memory to hold all the tuples in *R*, an additional "partition" phase is required. There are several variants of the classic hash join algorithm. They differ in terms of utilizing memory and handling overflow.

Grace Hash Join The idea behind grace hash join is to hash partition both relations on the join attribute, using the *same* hash function. As the result, each relation is hashed into *k* partitions, and these partitions are written to disk. The key observation is that *R* tuples in partition *i* can join only with *S* tuples in the same partition *i*. If any given partition of *R* can be hold in memory, the algorithm can read in and build a hash table on the partition of *R*, and then probe the hash table using tuples of the corresponding partition of *S* for matches.

If one or more of the partitions still does not fit into the available memory (for instance, due to data skewness), the algorithm is recursively applied. An additional orthogonal hash function is chosen to hash the large partition into sub-partitions, which are then processed as before.

Hybrid Hash Join The hybrid hash join algorithm is a refinement of the grace hash join algorithm which takes advantage of more available memory. To partition *R* (*S*) into *k* partitions, the grace hash join uses one input buffer for reading in the relation and *k* output buffers, one for each partitions.

```
Algorithm 1: Grace Hash Join: R ⋈_{r=s} S
// partition R into k partitions
foreach r ∈ R do
  read r and add it to buffer page $h_1$ (r);
  flush the page to disk if full;
end
// partition S into k partitions
foreach s ∈ S do
  read s and add it to buffer page $h_1$ (s);
  flush the page to disk if full;
end
// "build" and "probe" phases
for i ← 1 to k do
  foreach r ∈ partition $R_i$ do
    read r and insert into the hash table using $h_2$ (r);
  end
  foreach s ∈ partition $S_i$ do
    read s and probe the hash table using $h_2$(s);
    for matching r tuples, add {r, s} to result;
  end
  clear the hash table and release the memory;
end
```

Suppose there are enough extra memory to hold an in-memory hash table for one partition, say the first partition, of *R*, the hybrid hash join does not write the partition to disk. Instead, it builds an in-memory hash table for the first partition of *R* during the "partition" phase. Similarly, when partitioning *S*, for the tuples in the first partition of *S*, the algorithm directly probes the in-memory hash table and writes out the results. At the end of the "partition" phase, the algorithm completes the join between the first partitions of *R* and *S* while partitioning the two relations. The algorithm then joins the remaining partitions as the grace hash join algorithm does.

Compared with the grace hash join algorithm, the hybrid hash join algorithm avoids writing the first partitions of *R* and *S* to disk during the "partition" phase and reading them in again during the "build" and the "probe" phases.

Cross-references

▶ Hashing
▶ Parallel Join Algorithms
▶ Evaluation of Relational Operators

Recommended Reading

1. Mishra P. and Eich M.H. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.

Hash Trees

▶ Merkle Trees

Hash-based Indexing

Mirella M. Moro[1], Donghui Zhang[2], Vassilis J. Tsotras[3]
[1]Federal University of Minas Gerais, Belo Horizonte, Brazil
[2]Northeastern University, Boston, MA, USA
[3]University of California, Riverside, CA, USA

Synonyms

Hash file; External hashing; Extensible hashing; Linear hashing; Grid file

Definition

Consider a relation *R* with some attribute *A* taking values over domain *D*. A *membership* (or *equality*) *query* retrieves all tuples in *R* with $A = x$ ($x \in D$). To enable fast processing of such equality selection queries, an access method that can group records by their value on attribute *A* is needed. A hash-based scheme maps the search-key values on a collection of buckets. The bucket to which a value is assigned (mapped) is determined by a function, called the *hashing function*.

Key Points

A major performance goal of a database management system is to minimize the number of I/O's (i.e., blocks or pages transferred) between the disk and main memory when answering a query. To achieve such fast access, additional data structures called access methods (or indices) are designed per database file. There are two fundamental access methods, namely tree-based and hash-based indexing. They differ on the kind of queries that they can efficiently address.

Hash-based indexing does not maintain any ordering among the indexed values; rather it is based on mapping the search-key values on a collection of buckets. Therefore it can only address *equality* (or *membership*) queries. Tree-based indices maintain order and can thus also address range queries.

In database management systems hashing is implemented on disk-resident data (also termed as *external hashing*). Here instead of a slot the scheme uses a *bucket* which is typically the size of a page (or a fixed number of pages) and can thus hold many records. Consider a hashing scheme using *M* buckets. Assume a page can store *B* values. A *hashing function h* is a function that maps *D* to *M* (one example of such function is $h(x) = x \bmod M$). To insert a record with search-key x ($x \in D$), the hashing function $h(x)$ is computed which gives the bucket to store that record. In *static* external hashing, if this bucket has space the record is simply added. If *B* values have already been hashed on this bucket, an *overflow* occurs. An extra page is added (and chained to this bucket) and the new record is stored there.

Given that the number of possible values in domain *D* is typically much larger than *M*, different values can be hashed on the same bucket. Nevertheless, a given hashing function will always hash a given value on the same bucket. Searching for record(s) with a given search-key *y* (also called *hash-key*) is simple. The hashing function is computed on the *y* and *h*(*y*) will be the bucket accessed. The record(s) with the requested value will either be found on this bucket or there is no such record in the file. No other buckets need to be searched. Note that searching a hash-based scheme avoids the tree navigation that a tree-index implies. Rather, a hashing function is computed (in main memory) and the only I/O is for bringing in main memory the page(s) of a given bucket.

In static external hashing, the number of buckets is pre-allocated and does not change whether the file is small or large. In contrast, *dynamic* hashing schemes (like *extensible* and *linear* hashing) use a number of buckets that increases/decreases as the size of the indexed file changes by record insertions/deletions.

The cost for updating and querying a hash-based hashing scheme is constant in the expected case (but can be linear in the worst case, for example when most records are mapped to one bucket creating a long overflow chain of pages); its space requirements are linear to the size of the indexed file.

Another hashing related scheme is the Bloom filter, a space-efficient probabilistic data-structure that drastically reducing space by allowing false positives (but not false negatives). For multi-attribute search, the Grid-File has been proposed, which can be considered as an extension of hashing into many dimensions.

Cross-references

- ▶ Access Methods
- ▶ Extendible Hashing
- ▶ Grid File
- ▶ Indexing
- ▶ Linear Hashing

Recommended Reading

1. Manolopoulos Y., Theodoridis Y., and Tsotras V.J. Advanced Database Indexing. Kluwer, Dordrecht, 1999.
2. Elmasri R. and Navathe S.B. Fundamentals of Database Systems, 5th edn. Addisson-Wesley, Reading, MA, 2007.
3. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edn. McGraw-Hill, New York, 2003.

Hater's Model

▶ Two-Poisson model

HCC

▶ Human-centered Computing: Application to Multimedia

HCI

▶ Human-Computer Interaction

HCM

▶ Human-centered Computing: Application to Multimedia

Health Informatics

▶ Taxonomy: Biomedical Health Informatics

Healthcare Informatics

▶ Taxonomy: Biomedical Health Informatics

Heat Map

▶ Visualizing Clustering Results

Heavy Hitters

▶ Frequent Items on Streams

Heterogeneous Distributed Database Systems

▶ Distributed Database Systems

Heterogeneously Distributed Data

▶ Vertically Partitioned Data

HHH

▶ Hierarchical Heavy Hitter Mining on Streams

Hidden-Web Search

▶ Deep-Web Search

Hierarchial Clustering

Maria Halkidi
University of Piraeus, Piraeus, Greece

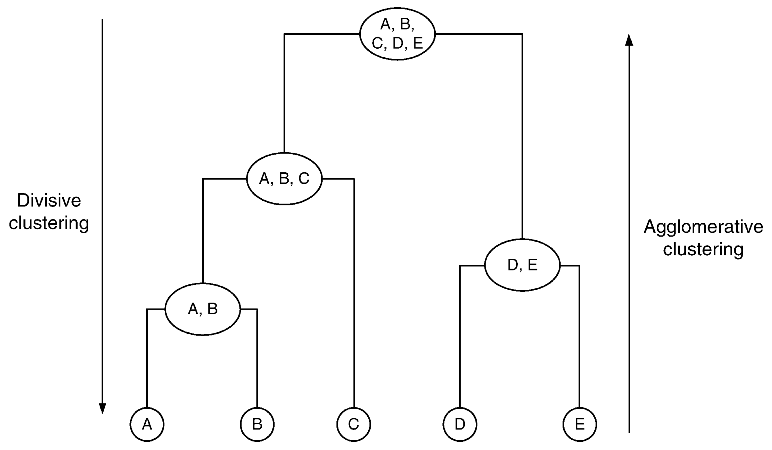
Definition

A *hierarchical clustering* method generates a sequence of partitions of data objects. It proceeds successively by either merging smaller clusters into larger ones, or by splitting larger clusters. The result of the algorithm is a tree of clusters, called dendrogram (see Fig. 1), which shows how the clusters are related. By cutting the dendrogram at a desired level, a clustering of the data items into disjointed groups is obtained.

Historical Background

Hierarchical clustering is one of the main methods used in data mining to partition a data collection. A number of hierarchical clustering algorithms [3] have been developed to deal with various types of data and application requirements. According to the method that the algorithms produce clusters, can be classified into the following categories:

1. *Agglomerative hierarchical clustering.* The algorithms of this category follow a bottom-up strategy and they produce sequence of data clusterings by decreasing number of clusters at each step (resulting in a different tree level). The algorithm starts by considering each data object as a separate cluster. Then, at each step it selects to merge the pair of two closest clusters defining thus a new clustering. To find the similarity of two clusters, one of the following criteria is typically used: *the minimum, the maximum,* or *the average distance* between objects of the two clusters. The merging procedure proceeds iteratively until all objects are in a single cluster or a termination condition is satisfied.
2. *Divisive hierarchical clustering.* These algorithms produce a sequence of clusterings increasing the number of clusters at each step. Contrary to the agglomerative, the divisive algorithms produce at each step a clustering from the previous one by splitting a cluster into two. The algorithm starts with all objects in a single cluster. Then it iteratively subdivides the clusters into smaller ones, until each object forms a separate cluster or a cetrain termination condition is satisfied.



Hierarchial Clustering. Figure 1. Dendogram.

One of the drawbacks of pure hierarchical clustering methods is their inability to perform adjustment when a merge or split decision has been executed. Once a group (cluster) of data is merged or split, the clustering process at the next step will operate on the newly generated clusters. Thus the quality of hierarchical clustering depends on the selection of merge or split point. To tackle this problem the recent studies have focused on the integration of hierarchical agglomerative with multiple phase clustering techniques and relocation methods.

Foundations

This section provides an overview of the main algorithms that are representatives of hierarchical clustering method.

BIRCH [6] uses a hierarchical data structure called CF-tree for partitioning the incoming data objects in an incremental and dynamic way. CF-tree is a height-balanced tree, which stores the clustering features. A clustering feature (CF) is a structure that summarizes statistics for the given sub-clusters. Considering a set $X = \{X_1,...,X_N\}$ of d-dimensional data objects assigned into a sub-cluster C_i, the CF of C_i is defined as $CF = \{N, LS, SS\}$, where $LS = \sum_{i=1}^{N} X_i$ is the linear sum on N objects and $SS = \sum_{i=1}^{N} X_i^2$ is the square sum of data objects.

The BIRCH algorithm is based on two parameters: *branching factor B* and *threshold T*, which refer to the diameter of a cluster (the diameter (or radius) of each cluster must be less than T). It consists of two phases:

1. Scan of the data collection to *build an initial in-memory CF tree*. It can be viewed as a multilevel compression of data that tries to preserve the inherent clustering structure of the data.
2. Application of a clustering algorithm to *cluster the leaf nodes of the CF tree.*

BIRCH adopts an incremental clustering method since the CF tree is built dynamically as new objects are inserted. It can typically find a good clustering with a single scan of the data and improve the quality further with a few additional scans. Thus using a multi-phase clustering BIRCH tries to produce the best clusters with the available resources. However, BIRCH does not always correspond to a natural cluster (as a user may consider it), since each node in CF-tree can hold a limited number of entries due to its size. Also it uses the notion of radius or diameter to control the boundaries of a cluster and thus it tends to favor spherical clusters. Moreover, it is order-sensitive as it may generate different clusters for different orders of the same input data. The computational complexity of BIRCH to cluster a set of n objects is $O(n)$.

CURE [1] is a hierarchical clustering algorithm that combines centroid-based and representative object based approaches. CURE represents each cluster by a certain number of objects that are generated by selecting well-scattered points and then shrinking them toward the cluster center by a specified fraction α. These representative points try to capture the natural schema and the geometry of a cluster. Additionally, moving the dispersed points toward the cluster center by a certain factor (further reffered to as α) the algorithm aims to remove the noise and eliminate the influence of outliers. Significant movements of outliers will eliminate the possibility of merging inappropriate clusters. A high value of α shrinks the representatives closer to the cluster center and thus it favors more compact clusters. On the other hand, a small value of α shrinks more slowly the representatives favoring elongated clusters. Thus CURE efficiently achieves to identify arbitrarily shaped clusters (i.e., non spherical) and it is robust to the presence of outliers.

CURE scales for large databases using a combination of random sampling and partition clustering. The data that are used as input to the algorithm, can be a sample randomly selected from the original data set. The selected data sample is partitioned into a certain number of partitions and then each of the defined partitions is partially clustered. A clustering step follows that aims to generate a hierarchy of partial clusters. The clustering starts considering each data object of input (partial clusters) as a separate cluster and then it iteratively merges the nearest pairs of clusters. The distance between two clusters is defined as the distance between their closest representatives. Thus only the representative points of clusters are used to measure the distance between them. Then the representative points falling in each of the newly defined clusters are moved toward the center of clusters by a shrinking factor α.

The time complexity of CURE is $O(n^2 \, log \, n)$ (where n is the number of data objects to be clustered), while it reduces to $O(n^2)$ in case of low-dimensional data.

ROCK [2], is a robust hierarchical clustering algorithm for Boolean and categorical data. It introduces two new concepts: (i) the *neighbors of a point* and (ii) *links*. The clustering algorithm is based on these concepts to measure the similarity/proximity between a pair of data points. Given a user-defined threshold $\theta \in [0,1]$, a pair of points p_i, p_j are defined to be neighbors if $sim(p_i, p_j) \geq \theta$. Also the term $link(p_i, p_j)$ is defined to be the number of common neighbors between p_i and p_j. The ROCK algorithm exploits the concept of *links* to make decisions about the clusters that will be merged at each step. The similarity between a pair of clusters (C_i, C_j) is measured based on the number of points from C_i and C_j that have neighbors in common. For a pair of clusters C_i and C_j, the algorithm measure the number of cross links between the clusters, that is, $\sum_{pk \in C_i \text{ and } pl \in C_j} links(p_k, p_l)$. Then the *goodness measure* for merging clusters C_i, C_j is defined as follows:

$$goodness(C_i, C_j) = \frac{link(C_i, C_j)}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

Then, at a given step, the algorithm selects to merge the pair of clusters for which the goodness measure is maximum.

CHAMELEON [4] is a clustering algorithm that explores characteristics of dynamic modeling in hierarchical clustering. It is an agglomerative hierarchical algorithm that measures the similarity of two clusters based on a dynamic model. Specifically, it defines the clusters in the data set by using a two-phase algorithm. During the first phase, CHAMELEON uses a graph-clustering algorithm to partition a data set into a large number of relatively small sub-clusters. During the second phase, it uses an agglomerative hierarchical clustering algorithm to find the clusters by repeatedly combining together these sub-clusters. The similarity between clusters is determined by looking at their relative *inter-connectivity* and *relative closeness*. The representation of the data objects is based on the widely used k-nearest neighbor graph approach. The vertices of the graph represents data objects, and there is an edge between two vertices if data object corresponding to one of the nodes is among the k-most similar data objects of the other node. Then the algorithm finds the initial sub-clusters using a graph-partitioning algorithm in order to partition the k-nearest neighbor graph of the considered data set into a large number of partitions. During the next phase CHAMELEON switches to an agglomerative hierarchical clustering that combines together these small sub-clusters. It measures the similarity between each pair of considered clusters based on their *relative inter-connectivity* and *closeness of the sub-clusters*:

1. The *relative interconnectivity* between a pair of clusters C_i and C_j is their absolute interconnectivity normalized with respect to their internal interconnectivities. It is given by

$$RI(C_i, C_j) = \frac{\left|EC_{\{C_i,C_j\}}\right|}{\frac{1}{2}(|EC_{C_i}| + |EC_{C_i}|)}$$

   where $EC_{\{C_i,C_j\}}$ is defined as the sum of weights of the edges that connect vertices in C_i to vertices in C_j and EC_{C_i} is defined as the weighted sum of the edges that partition the graph into two roughly equal parts.
2. The relative closeness between a pair of clusters C_i and C_j is the absolute closeness normalized with respect to the internal closeness of the two clusters:

$$RC(C_i, C_j) = \frac{\overline{S}_{EC_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC_{C_j}}}$$

   where $\overline{S}_{EC_{\{C_i,C_j\}}}$ is defined as the average weight of the edges that connect vertices in C_i to vertices in C_j, and $\overline{S}_{EC_{C_i}}$ is the average weight of edges that partition the graph into two roughly equal parts.

According to the above definition CHAMELEON makes merging decisions. Those pairs of clusters whose relative inter-connectivity and closeness are above some user-specified threshold are merged. CHAMELEON has been proved to be more powerful at discovering arbitrarily shaped clusters than CURE. However, the processing cost for high dimensional data may require quadratic time to the number of processed objects ($O(n^2)$).

The most recent clustering algorithm that combines characteristics of hierarchical clustering and graph-theory is C^2P [5]. It exploits index structures, and the processing of the *Closest-Pair* queries in spatial databases. The algorithm considers that the data points can be organized in an R-tree data structure to facilitate the searching of the nearest representative point. Then the Closest-Pair Query (CPQ) is used to find the closest pair of points from two datasets indexed with two R-tree data structures.

The C^2P algorithm consists of two main phases. The first phase (Phase I) organizes a set of objects into a number of sub-clusters, which are an effective representation of the final clusters. The *Self Semi-Closest-Pair* Query (Self-Semi-CPQ) is used to find pairs of objects (p, p') that belong to a data set S such as $dist(p, p') = min_{\forall x \in S}\{dist(p, x)\}$. The algorithm uses a graph representation that organizes the proximity information computed by the CPQ. Using the Depth-First Search the algorithm can efficiently find the c connected components of the graph, which also comprises the sub-clusters of the data set. All objects that belong to the same connected component can be considered as a sub-cluster. When the number of defined sub-clusters, is equal to the required number of sub-clusters the Phase I terminates. Otherwise, the algorithm finds the center of each sub-cluster to represent it. Then the previously described procedure is iteratively applied to the set of c cluster centers until the required number of sub-clusters is defined. The second phase is a specialization of the first phase using a different cluster representation so as to produce the finer final clustering. The second phase (Phase II) has as input the centers of sub-clusters defined in Phase I. At each iteration of Phase II, Self-CPQ finds the closest pair of clusters by finding the closest pair among their representatives. Then these two clusters are merged and the r data objects, among all the objects of the merged clusters, that are closest to the cluster center are selected as representatives of the new cluster. Using multi-representatives instead of the center, C^2P can effectively capture the shape and size of the clusters. The procedure terminates when the required number of clusters is reached. The above description shows that Phase II operates in a fashion analogous to a hierarchical agglomerative clustering algorithm.

The C^2P algorithm is shown to scale well to large databases. Its time complexity for a dataset with n objects is $O(n \log n)$.

Key Applications

Hierarchical clustering has applications in various fields of real world. In biology, it can be used to define taxonomies, categorize genes with similar functionality and gain insights into structures inherent in populations. Clustering may help to automate the process of analyzing and understanding spatial data. It is used to identify and extract interesting characteristics and patterns that may exist in large spatial databases. Also hierarchical clustering is used to discover significant groups of documents on the Web's huge collection of semi-structured documents. This classification of Web documents assists in information discovery.

Cross-references

- ▶ Balanced Trees
- ▶ Clustering Methods
- ▶ Graph Clustering
- ▶ Indexing
- ▶ k-Nearest Neighbors
- ▶ R-Tree

Recommended Reading

1. Guha S., Rastogi R., and Shim K. CURE: an efficient clustering algorithm for large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 73–84.
2. Guha S., Rastogi R., and Shim K. ROCK: a robust clustering algorithm for categorical attributes. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 512–521.
3. Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2001.
4. Karypis G., Han E.-H., and Kumar V. CHAMELEON: a hierarchical clustering algorithm using dynamic modeling. IEEE Comput., 32(8):68–75, 1999.
5. Nanopoulos A., Theodoridis Y., and Manolopoulos Y. *C^2P:* clustering based on closest pairs. In Proc. 27th Int. Conf. on Very Large Daa Bases, 2001, pp. 331–340.
6. Zhang T., Ramakrishnman R., and Livny M. BIRCH: an efficient method for very large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 103–114.

Hierarchical Data Model

JEAN-LUC HAINAUT
University of Namur, Namur, Belgium

Synonyms

IMS data model

Definition

The hierarchical data model is based on a view of the application domain (i.e., the *world*) as a hierarchical arrangement of concepts where some concepts exist on their own while the others depend on the former. According to this conceptual model, data are organized into records that are recursively composed of other records. Though this paradigm is fairly common in data structures, the term hierarchical model most generally refers to the *IMS model*, a proprietary DBMS developed by IBM from the sixties that is still widely used.

The IMS model organizes data in tree structures of records augmented with additional links that compensate for the weaknesses of this base model. Data processing itself is hierarchical, starting from the root of a record tree then parsing the dependent records in depth-first, left-to-right, traversal order.

Historical Background

In 1966, IBM started the development of ICS (Information Control System), a data management software for, and with the collaboration of, American Rockwell, then in charge of building the Apollo spacecraft which was to send men on the moon. IBM commercialized ICS in 1969 under the name IMS (Information Management System) [1].

The first version offered storage and processing facilities for data records organized in trees and linearized according to the depth-first traversal order. The data could be stored on tapes and on disks and were processed sequentially. Thanks to the addressability of magnetic discs, IMS was later given direct access (through hashing and B-tree techniques) to root records and therefore was able to support transaction processing.

At that time, tree structures were strictly independent and not connected, a fact that led to much redundancy (a record cannot be stored in two different trees without duplication). IBM then introduced logical relationship types which explicitly linked records from different tree structures. A record could then be shared by several trees.

Later on, IMS was added secondary indexes that provided direct access to non-root records. IMS is now a complex and powerful data management and data communication environment mostly used by data-intensive batch and On-Line Transaction Processing (OLTP) applications.

Though System 2000 (SAS Institute), XML document structures (DTD and XML Schema) and standard file structures can legitimately claim to belong to the hierarchical family, the presentation will focus on the IMS model. Data description and manipulation languages, API and implementation techniques will not be addressed in this chapter. They can be found in references [2, 4–6].

Foundations

Due to historical reasons, in particular the incremental development of IMS, the description of its data structures is generally intricate. In this entry, they will be described by way of a simpler approach based on graph theory. Of course, some very specific details will be ignored.

Graphs and Hierarchies

This section relates database schemas with various kinds of graphs. Due to the limited scope of this chapter, only an intuitive view of the equivalence principles will be developed. Is-a hierarchies, in particular, will be ignored.

Preliminary Definitions

A database schema can be seen as a *multigraph*, whose nodes denote entity types and whose labeled edges represent binary relationship types (*rel-types* for short). This applies to any Entity-relationship schema, provided n-ary rel-types have been replaced by relationship entity types through some kind of *reification* transformation (such as T1 in Fig. 1). In this graph, edges are given a pictorial representation that specifies the functional dependencies that hold in the rel-type according to the usual arrow convention. A single arrow denotes a N:1 rel-type, a double arrow a 1:1 rel-type and a plain edge a N:N rel-type. N:1 and 1:1 rel-types are called *functional* as well as the edges that represent them.

A *hierarchical graph* is a binary graph such that (i) the edges are functional and (ii) the edges define a partial order on the nodes. In practical words, there is no chain of successive N:1 edges the starting and arrival nodes of which are the same node (in short, *no circuit*). Considering a directed edge drawn from *B* (the *many* side) to *A* (the *one* side), *A* is called a *parent* of *B* while *B* is a *child* of *A*. *A* is called a *k-node* if it is the child of *k* parents. A graph is a *n-hierarchy* if none of its nodes has more than n parents. A 0-node is a *root* of its hierarchy. A hierarchy comprises at least one root. A node that is not a parent is a leaf node. A *1-hierarchy* is a set of trees, that is, a forest.

Properties of Hierarchies

1. *Any multigraph can be losslessly transformed into a hierarchy.* Three patterns can prevent an arbitrary schema from defining a hierarchy, namely complex rel-types (n-ary, with attributes), N:N rel-types

and rel-types that form a circuit. N-ary and N:N rel-types can be processed by transformations T1 and T2 respectively (Fig. 1). Any functional rel-type of a circuit transformed through T3 or T4 (Fig. 1) destroys this circuit. All these entity-generating transformations have been proved to preserve the semantics of the schema [3]. A large number of hierarchies can be derived from a definite multigraph.

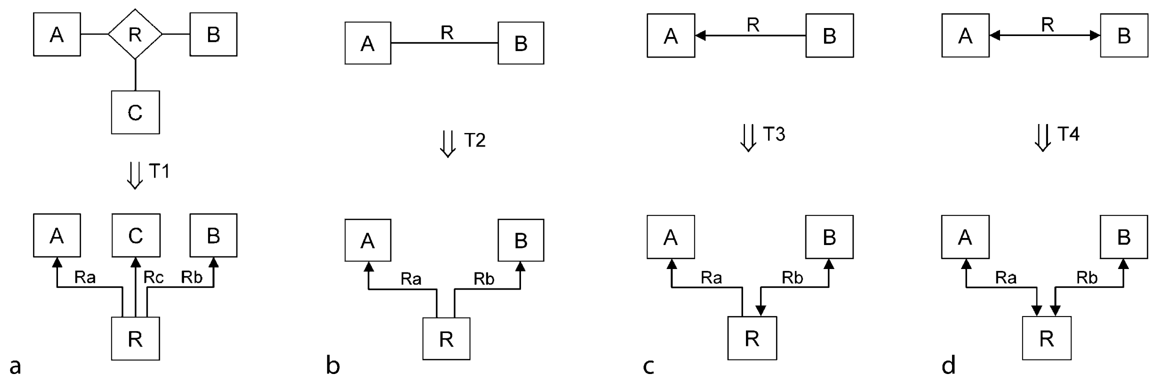
2. *Any hierarchy can be losslessly transformed into a 2-hierarchy.* Transformation T3 shows that *B*, which is the child of *A* in the source schema looses its parent (which becomes its "spouse") in the target schema. A similar transformation exists for 1:1 rel-types (T4). Iteratively applying this transformation produces an equivalent 2-hierarchy from an arbitrary hierarchy (Fig. 2). This process is not unique, so that many 2-hierarchies can be derived from a given hierarchy.
3. *Any 2-hierarchy can be built by superimposing two forests on the same set of nodes.* For each child node of a 2-hierarchy *H* with two parents, one of the parental edges is colored in black and the other one in gray. Each of the remaining edges is arbitrarily colored in black or in gray. The set of nodes together with black (resp. gray) edges form the black (resp. gray) subgraph(s). Each of them is a forest and their union is *H*. There are many ways to color the edges of a given 2-hierarchy (Fig. 3).
4. *Any arbitrary Entity-relationship schema is equivalent to the superimposition of two forests.* Properties 1 and 2 show that any Entity-relationship schema is equivalent to a 2-hierarchy, which in turn can be decomposed into two forests.

IMS Data Structures

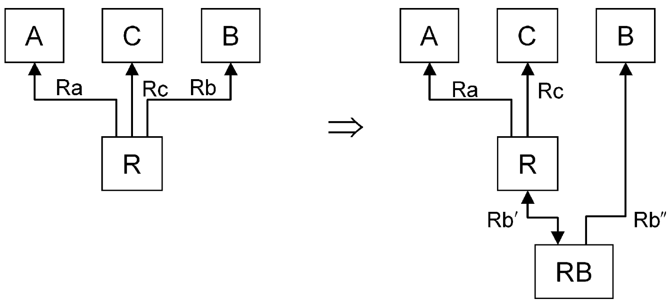
IMS is by far the most popular DBMS based on the hierarchical data model. This section describes its main data structures.

The IMS Global Schema

The global schema of an IMS database can be modeled by a 2-hierarchy formed by two distinct forests. The nodes represent *segment types* (the IMS name for record types) and the edges N:1 *parent-child rel-types*. The black rel-types are called *physical rel-types* and the



Hierarchical Data Model. Figure 1. Transforming arbitrary relationship types into functional relationship types.



Hierarchical Data Model. Figure 2. Reduction of a *n*-hierarchy to a *n-1*-hierarchy by application of transformation T3.

gray ones *logical rel-types* (this naming is purely historical and bears little meaning at this level). According to the *color* of the rel-type that links them, two segment types will be called respectively physical/logical child and physical/logical parent. A rel-type bears no name. If two rel-types link the same pair of segment types, one of them is physical and the other one is logical. IMS schemas use graphical conventions that are close to those of Figs. 1–3 but their edges have no arrow heads (Fig. 4).

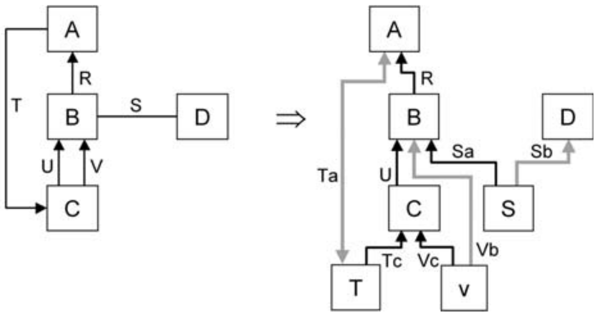
IMS imposes additional constraints of this double tree structure:

1. a logical child must be a physical child; therefore a root segment type cannot be a logical child;
2. a logical child cannot be a logical parent;
3. a logical child cannot have a physical child that is also a logical child.

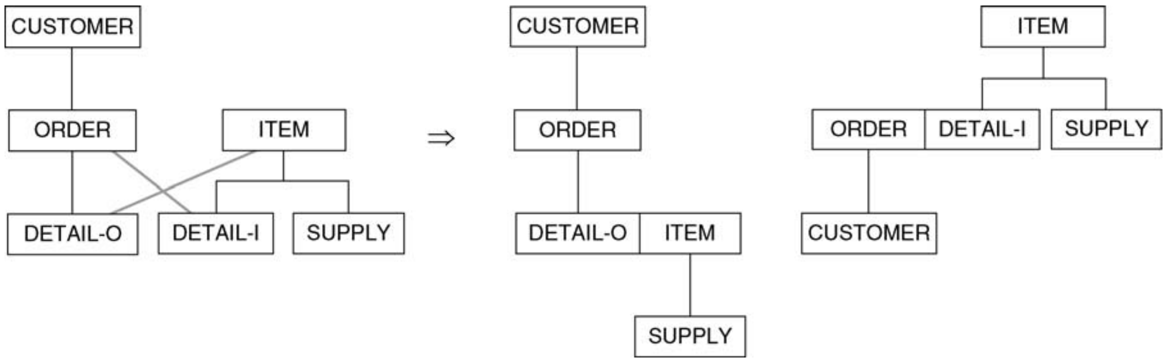
As a consequence, many logical children are leaf physical segment types (Fig. 3 – right).

A segment type has a name and a length. It is made up of fields. A field is an untyped byte string with which a name, a starting position and a length are associated. Some parts of the segment type may be left undefined, while some fields may overlap, which is a way of describing compound fields. In summary, a segment type comprises single-valued mandatory fields that can be atomic or compound.

Now only the physical (black) rel-types are considered. Each 0-node denotes a *physical root segment type*. Each physical root segment type, together with all the segment types to which it is directly or transitively connected through a physical rel-type, form a *physical database*. A root segment followed by all its direct and indirect child segments, in the depth-first order traversal (or preorder), is a *physical record* of this database. A physical database contains a sequence of physical records. One of the fields of the root segment type is the *sequence field* or *key*. No two records can have the same key value and the records are sorted on this key. Direct access is allowed to a record based on its key value. A field of a dependent segment type can also be declared a key. In this case, and unless otherwise stated, the children segments of a definite



Hierarchical Data Model. Figure 3. Any multigraph is equivalent to the superimposition of two forests.



Hierarchical Data Model. Figure 4. Two examples of logical database schemas.

parent have distinct values of their key. The key value of a segment prefixed by the concatenated key of its parent forms its *concatenated key*. The latter identifies this segment in the physical database. The implicit goal of a physical database is twofold:

- historically, a physical record collects all the data that are needed by an application program (process-oriented approach);
- according to the modern way of perceiving a database, a physical record collects the proper data of a major entity of the application domain, independent of the programs that will use it (domain-oriented approach).

An *IMS database* comprises one or several physical databases together with all their logical rel-types. Logical rel-types can be defined between two segment types pertaining to one or two physical databases.

The IMS Logical Database

A *logical database* is a virtual tree data structure derived from an IMS global database schema and tailored to the specific needs of an application program. Its role is similar to that of CODASYL subschemas and of relational views. The main derivation rules are the following:

1. the logical database schema *L* comprises a connected subset of the physical database schema *PH*;
2. the root of *L* is the root of *PH* (secondary indexes allow more flexible structures);
3. the logical parent *P* of a logical child *C* of *L* can be concatenated with *C*, giving fictitious segment type *CP*;
4. all or some of the direct and transitive physical children of *P* can be attached to *CP*;
5. all or some of the direct and transitive physical parents of *P* can be attached to *CP* as fictitious children.

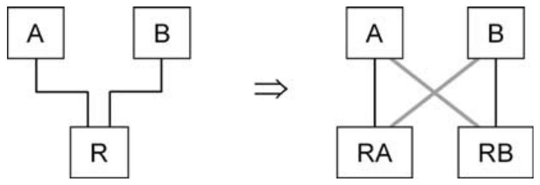
Fig. 4 shows some examples of logical database schemas derived from the same IMS database.

Additional Constructs

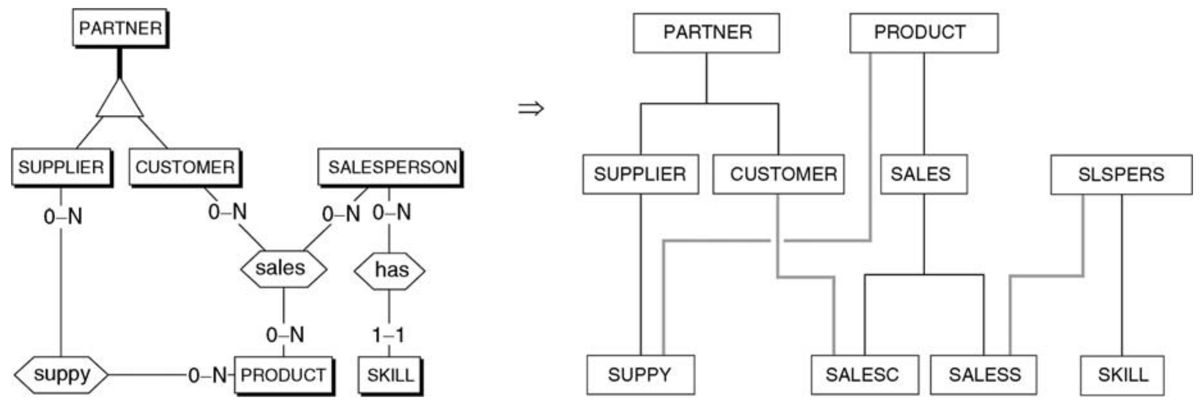
Several other concepts are part of the IMS model and are devoted large sections in usual IMS presentations. They have not been included in the preceding sections inasmuch as they do not contribute to the understanding of the data model but rather appear, according to today's conceptual standards, as minor idiosyncrasies. Two of them are briefly described, namely *segment pairing* and *secondary indexes*.

Segment pairing. Consider the hierarchical schema fragment of Fig. 5 (left), in which *relationship* segment type *R* materializes a N:N link between *A* and *B*. Though this pattern is intuitive from the conceptual and physical points of view, it violates the spirit of tree structures that underlies the hierarchical model of IMS: for bidirectional access, both *A* and *B* must have a normal (physical) child of their own. So, each of them is given a physical child, namely *RA* and *RB*, and it is linked to the other parent through a logical (gray) rel-type. However, since *RA* and *RB* are just clones of each other, they are declared *paired* (Fig. 5 – right). Physically, this pattern can be implemented according to two techniques: with a single child segment type (virtual pairing) or with two redundant child segment types whose contents are synchronized automatically (physical pairing).

Secondary indexes. Physical (or logical) records can be directly accessed through indexing techniques applied to the root sequence field. A secondary index allows additional access based on other fields of the root segment type or access to non root segment types. It is implemented as a special-purpose physical database, the *index database*, made up of one segment type (the *pointer* segment type), that collects all the values of the key fields together with the addresses of the segments that include these values. In fact, this technique is based on two segment types in the indexed database. The indexed values are extracted from the *source segment type* while the access is performed on the *target segment type*. These segment types are generally the same but the former can be the child of the latter. For instance, a secondary index can allow access to *CUSTOMER* segments that have at least one *ORDER* child segment with a given date of order. In addition, a logical database can be built on an index database in such a way



Hierarchical Data Model. Figure 5. Normalization of a *relationship segment type* according to the IMS model.



Hierarchical Data Model. Figure 6. Partial translation of a representative Entity-relationship schema (left) into a hierarchical schema (right).

that its logical root segment type is any, possibly non-root, segment type of the indexed database.

Entity-Relationship to Hierarchical Mapping

As shown above, an Entity-relationship schema can always be translated into an equivalent 2-hierarchy, a structure that is close to the IMS data model. The translation is mostly based on a unique technique, that is, the transformation of a rel-type into a relationship entity type together with two or several functional rel-types. The procedure of deriving an IMS database schema from an Entity-relationship schema can be sketched as follows.

1. Each entity type is represented by a segment type, each attribute by a field and each functional rel-type by a parent-child rel-type (still *uncolored*).
2. A non functional rel-type is transformed into a segment type by techniques T1 and T2.
3. Is-a hierarchies are best transformed through the *one segment type per entity type* technique; subtypes are represented by physical children of their supertype segment type.
4. A circuit is opened by the transformation of one of its links by technique T3 and T4.
5. 1:1 rel-types are implemented as standard 1:N rel-types and controlled by the application programs.
6. The schema is then transformed into a 2-hierarchy by the technique illustrated in Fig. 3. When a segment type has two parents, two techniques can be used. The first one consists in marking one rel-type as *logical*. The second one applies transformations T3 or T4. The latter will be preferred when IMS structural rules concerning logical rel-types are violated or to make the schema more balanced (rel-type *RA* of the schema of Fig. 3 could be processed in the same way as *RB*).

Figure 6 illustrates some of these principles. The IMS global schema comprises three physical databases and three logical relationship types. For readability, segment pairing, according to which each logical child is duplicated as a dependent of its respective logical parent, is not shown.

Key Applications

IMS is a major legacy technology in which many large corporate databases are still implemented. It is mainly used for stable, slowly evolving, batch and OLTP applications, notably in banking companies. The complexity of the hierarchical model and its lack of flexibility in evolving domains make IMS technology less attractive for decisional applications, such as data warehouses.

Cross-references

▶ Database Management System
▶ Entity-Relationship Model
▶ Network Data Model
▶ Relational Model

Recommended Reading

1. Blackman K. IMS celebrates thirty years as an IBM product. IBM Syst. J., 37(4):596–603, 1998.
2. Elmasri R. and Navathe S. Fundamentals of Database Systems (3rd edn.). Addison-Wesley, 2000. (The appendix on the hierarchical data model has been removed from later editions but is now available on the authors' site.)

3. Hainaut J-L. The transformational approach to database engineering. In Generative and Transformational Techniques in Software Engineering, R. Lämmel, J. Saraiva, J. Visser (eds.). Springer, New York, NY, 2006, pp. 89–138.
4. Long R., Harrington M., Hain R., and Nicholls G. IMS Primer – IBM Redbooks, 2000.
5. Meltz D., Long R., Harrington M., Hain R., and Nichols G. An Introduction to IMS. IBM Press, Armonk, NY, 2005.
6. Tsichritzis D. and Lochovsky F. Hierarchical data-base management: a survey. ACM Comput. Surv.(Special Issue: Data-Base Management Systems), 8(1):105–124, 1976.

Hierarchical Data Organization

► Indexing and Similarity Search

Hierarchical Data Summarization

Egemen Tanin
University of Melbourne, Melbourne, VIC, Australia

Synonyms

Hierarchical data summarization

Definition

Given a set of records data summaries on different attributes are frequently produced in data management systems. Commonly used examples are the number of records that fall into a set of ranges of an attribute or the minimum values in these ranges. To improve the efficiency in accessing summaries at different resolutions or due to a direct need for investigating a hierarchy that is inherent to the data type, such as dates, hierarchical versions of data summaries can be used. A data structure or algorithm is labelled as hierarchical if that structure or algorithm uses the concept of subcomponents to systematically obtain conceptually larger components. The method of obtaining a larger component is regularly induced by the user's understanding of the domain, such as dates in a year, as well as the fact that hierarchies can also be created automatically by a set of rules embedded into the system. Thus, rules used in a data structure's creation, e.g., B+-trees, are also considered as a means for hierarchical data summarization. In fact, different variants of popular data structures are used in hierarchical data summarization. Various algorithms for data reduction and aggregation have also adopted hierarchical processing techniques.

Historical Background

From a data structures point of view, foundations of hierarchical data summarization (HDS) techniques can be found in indexing literature for databases. Although many of the indexing techniques, e.g., B+-trees, are used for efficiently selecting records stored on a disk, they can also be considered as hierarchical summaries on large amounts of data. For multidimensional and spatial data, indices such as R-trees and quadtrees can be used for HDS.

Today, many versions of popular indexing techniques that directly target retrieval of summary information exist. Some indices are also used in query optimization due to their HDS capabilities, e.g., using a space decomposition one can guess the number of records in a certain region of data before a join operation can take place. More recently, spatial indexing techniques, for example quadtrees, were developed for distributed settings such as sensor networks for HDS.

Historically, histograms are the most basic structures that could be used for data summarization. They are frequently utilized in query optimization decisions. They are also used in data warehousing. Hierarchical versions of histograms were recently built and are of interest for HDS.

From an algorithmic point of view, techniques such as wavelet transformations, sketches, and data clustering with aggregation, when run in a hierarchical fashion, can be considered as HDS techniques. These techniques are extensively deployed in data management as well as in other fields of computer science over many years.

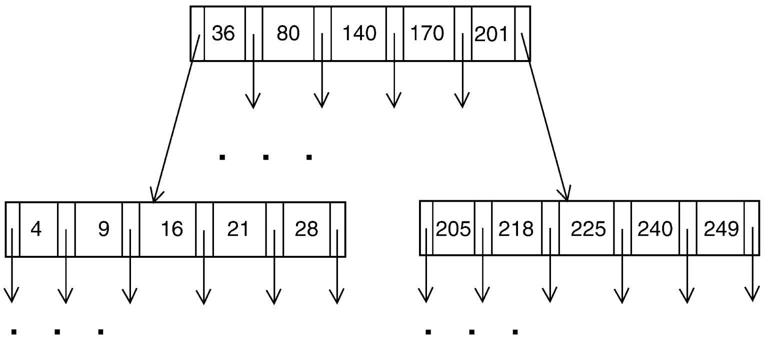
In recent years, for distributed data processing, variants of known algorithms have become popular in HDS. For example, researchers have introduced data aggregation techniques on sensor networks that can be considered as HDS techniques that rely on sketches. In this context, random-tree-based data aggregation algorithms in sensor networks can also be considered as basic HDS techniques. All of these different roots and aspects of hierarchical data summarization are visited in this article.

Foundations

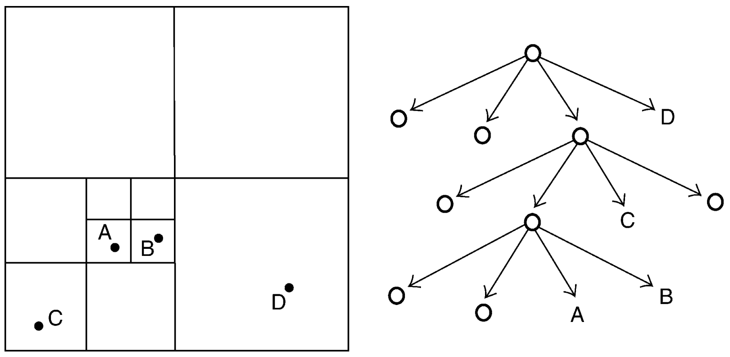
B+-trees are frequently used in databases. A B+-tree is given in Fig. 1 (only some parts of the tree are shown to simplify the presentation). B+-trees are hierarchical structures where internal nodes store keys and the leaf nodes contain the records attached to these keys. Due to their high fanout they are commonly shallow as well as balanced, i.e., in comparison to binary trees. They are used for efficient selection of a range of records from disks. The lowest level contains links between neighboring nodes to allow for sequential access to consecutive data items.

The B+-tree and related data structures can be considered as basic means of keeping hierarchical summary information. Given a fanout for a B+-tree, upper levels of the tree can be easily used for approximate HDS. One can refer to these levels to find the approximate number of items in a range. To make this HDS method more accurate, extra information should be maintained within the tree structure. For example, counts can be kept with each link in this hierarchy [10]. This requires extra space and maintenance costs as each count needs to be stored and updated with insertion and deletion operations. This can cause problems if many levels and nodes need to be maintained per update operation and if updates occur frequently for a given tree. If small errors in counters are tolerated then these overheads can be significantly reduced [3]. Counts form only one form of data summaries. Thus, the idea of counts is extended to other types of data summaries in [4,9].

For a set of queries and objects in space, such as range queries and a set of waterways in a country, spatial data structures can be used to efficiently store the data and answer queries on this data. For example, quadtrees are well-known, space-partitioning based structures. They are used with many different types of spatial data and thus many quadtree variants can be found in the literature. For example, a PR quadtree is given in Fig. 2. In this example, the space is recursively divided into four quadrants until a single data item is left in each quadrant. In Fig. 2, the space partitioning is shown (on the left) with its mapping tree structure (on the right). The positions of all the data items are also stored in the structure (not shown in the figure). Another related space partitioning method is the k-d tree. For k-d trees different dimensions of the



Hierarchical Data Summarization. Figure 1. An example B+-tree.



Hierarchical Data Summarization. Figure 2. An example quadtree with point objects.

underlying space is partitioned in turns at different levels of the tree. (Note that some of these methods are better named as tries, however, due to historical reasons, they are referred to as trees.)

If spatial objects are grouped together using bounding boxes and then a hierarchy of these bounding boxes are created, one can obtain an index called the R-tree. R-trees also have many variants. In comparison to quadtrees, they are commonly more balanced indexing schemes. However, many variants suffer from the fact that multiple bounding boxes, defining the tree nodes, can overlap in space. This nature of R-trees can reduce the pruning capability of this structure as a query may have to investigate multiple branches for the same space. Although disjoint-bounding-box based versions of R-trees exist, these variants could partition the data items into multiple boxes. There are many other spatial indices that are not presented in this article for the brevity of the presentation. The techniques mentioned are used to present HDS methods based on spatial indexing.

Similar to the case in B+-trees, spatial indices can also be viewed as HDS techniques. Moreover, data summaries can be explicitly maintained with these spatial data structures. This information can then be used for query processing, e.g., aggregate queries. Recently, spatial indexing is used in distributed settings for data summarization. For example, [7] introduces fractional cascading in sensor networks. In this approach, each sensor maintains detailed readings that it has obtained as well as data from its nearby neighbors. Information regarding other sensors are not kept as accurately. The space of sensors is partitioned using a distributed quadtree that is overlayed onto the sensor network. The partitioning is done in a similar manner to the PR quadtree example in Fig. 2. With increasing distance to the rest of the sensors in the network (i.e., to faraway quadrants) the amount of data collected from them drops with a function, e.g., a logarithmic function. This paradigm utilized the fact that data and queries in sensor networks are spatially and temporally correlated. Thus, this structure can be used to efficiently serve routing requests using locally summarized data as well as to answer queries. The distributed structure can be seen as a multi-rooted HDS technique as each sensor uses the same summarization scheme independently.

Similar to fractional cascading, [6] introduces the DIMENSIONS system that uses a pyramid-based space decomposition to aggregate and summarize data in a sensor network. Each quadrant finds a "local" leader node for building a distributed pyramid of nodes with their data. Other similar systems are DIM and DIFS systems [8,11] that use spatial indices on sensor networks for processing selection queries as well as resorting to summaries for user interest elimination. In [8] a k-d tree based structure is introduced while [11] introduces a multi-rooted quadtree type for avoiding bottlenecks from having a single root node.

Indices such as quadtrees and R-trees are also utilized in query optimization (e.g., [1]). As they represent summary information about the space they cover, they can easily be used in estimating the runtime costs of a query before it is executed.

In comparison to sophisticated indexing methods, a simple technique for summarizing data is the histogram. Histograms have long been employed in query optimization as they are compact and easy to maintain. With the emergence of data warehousing and On-line Analytical Processing (OLAP) technologies, they have also become crucial components from a new angle in data management. For data sets that explicitly contain hierarchies, e.g., years-months-weeks, histograms can easily be used.

From a processing cost estimation and query optimization point of view, Bruno et al. [5] introduced the concept of nested buckets with histograms. This can be seen as the first form of HDS using histograms. Later, Reiss et al. [14] built on this concept for distributed settings for bandwidth usage reduction. Reiss et al. present hierarchical histograms for aggregate query processing on identification data, i.e., RFIDs.

From an algorithmic point of view many methods that have long been used in approximating and summarizing data can be considered as hierarchical approaches to summarization. For example, wavelet transformations are well-established techniques in signal processing that can be used and considered as HDS methods. The following considers wavelets in the context of spatial data to give a simple example.

A three dimensional (3D) object in space can be approximated using a triangular mesh. One can use different sets of triangles, i.e., small or large, to give a more or less detailed approximation of the surface of the 3D object. Thus, an object can be represented in different resolutions using different meshes. If these meshes are related to each other geometrically, one can easily progressively update the details for this object on demand. Therefore, if M^I denotes a triangular mesh at resolution I, one can then represent an object as a series of meshes, $M^0, M^1, \ldots, M^J$, where, M^0 is the base mesh and M^J is the final mesh. Figure 3a shows a triangular mesh with one triangle, M^0, (1,2,3), for a 2D object. The triangle is the coarse approximation for the surface of the given circle.

Consider a simple transformation. To obtain a higher resolution approximation of the given surface in the figure, the triangle (1,2,3) is divided into four sub-faces by introducing new vertices (4′,5′,6′), Fig. 3b. The new set of vertices are now displaced to make the mesh better fit to the surface of the circle. The new, finer resolution mesh M^1, is shown in Fig. 3c. This operation can be done recursively and can be represented with a simple transformation function. The coefficients that represent the difference between M^0 and M^1 are $d_4{}^0$, $d_5{}^0$, and $d_6{}^0$. In this simple wavelet transformation, for example, the wavelet coefficient $d_4{}^0$ is obtained by $v_4^1 - \frac{v_1^0+v_2^0}{2} = v_4^1 - v_{4'}^1$. Thus, the wavelet-based decomposition of a mesh M^J produces a base mesh M^0 and the sets, $\{W_0, W_1, \ldots, W_{J-1}\}$, of coefficients. From an HDS point of view, the recursive execution of the above mentioned method can be seen as a hierarchical summarization of a detailed polygonal representation of a complex data set. Various further HDS methods can be derived using the base concept of wavelets. For example, recently, [2] uses wavelets with R-trees to progressively retrieve and refine spatial data from a remote database.

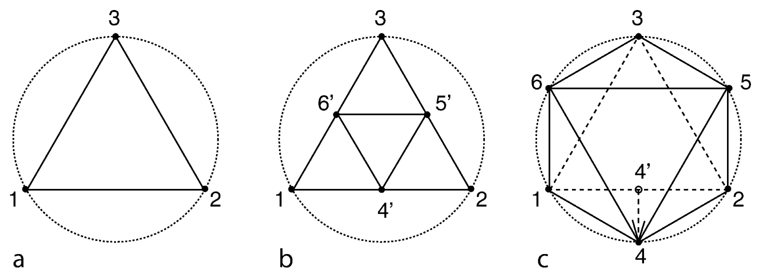
With the emergence of distributed systems such as sensor networks, aggregate query processing itself can now also be considered as a HDS technique. For example, to process an aggregate query in sensor networks, [12] uses a random-tree with in-network aggregation. Each node in this tree can compute an aggregate from its sub-trees, such as a minimum, and then pass this information to the higher-levels of the tree along the data collection path. The base-station, root, can then present a summary of the sensor data to the user.

Random-trees, however, are not robust. A single failure can cause significant problems (especially when a node that is close to the root fails). To address these problems, researchers have been working on multi-path data aggregation methods. In this scheme, multiple reports, due to the wireless coverage advantage in sensor networks, can be sent through different routes for increasing the robustness of the data collection method. However, this can cause deviations in certain aggregation operations, e.g., counts, as the same data is incorporated to the result multiple times. In [13], Nath et al. introduce a sketch theory-based HDS method to address this issue. They map aggregate functions, e.g., counts, to a set of order and duplicate insensitive synopsis generation and fusion functions.

Data clustering forms another area of research that, when applied using data aggregates and hierarchies, can be considered as a source of HDS techniques. For example, [15] introduces the STING system which uses a hierarchy of cells that contain aggregate information about the individual data items. Thus, for many query types, they can resort to these cells, rather than items, to answer queries efficiently. For queries that cannot be answered using summary data, individual data items can still be used as a backup strategy.

Key Applications

Key applications of HDS techniques are aggregate query processing and query optimization. If many queries are interested in retrieving summary data, e.g., aggregate queries, then maintaining a hierarchical summary would be efficient. For example, for data warehousing applications with hierarchical data, the benefits for maintaining a hierarchical summary could be significant. Data summaries have also long been used in query optimization. For example, an optimizer can use HDS techniques for selectivity estimation on attributes.



Hierarchical Data Summarization. Figure 3. A wavelet-based approximation.

Future Directions

There is a significant amount of activity in using data summaries in distributed settings and especially in sensor networks. In addition, with the emerging research directions in location-based services and VANETs, readers may expect to see the use of spatial HDS techniques more frequently. In distributed settings, bandwidth savings on the communication optimization front using HDS could be significant.

Url to Code

Demos for many of the spatial indices that are mentioned in this article can be found at http://www.cs.umd.edu/~hjs/quadtree/index.html.

Cross-references

▶ Aggregate Queries in P2P Systems
▶ B+-Tree
▶ Histogram
▶ Indexing
▶ Quadtrees (and Family)
▶ Query Processing and Optimization in Object Relational Databases
▶ Rtree
▶ Sensor Networks
▶ Sketch
▶ Spatial Network Databases
▶ Wavelets on Streams

Recommended Reading

1. Aboulnaga A. and Aref W.G. Window query processing in linear quadtrees. Distrib. Parallel Dat., 10(10):111–126, 2001.
2. Ali M.E., Zhang R., Tanin E., and Kulik L. A motion-aware approach to continuous retrieval of 3D objects. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 843–852.
3. Antoshenkov G. Query processing in DEC RDB: major issues and future challenges. IEEE Data Eng. Bull., 16(4):42–45, 1993.
4. Aoki P.M. Generalizing "search" in generalized search trees. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 380–389.
5. Bruno N., Chaudhuri S., and Gravano L. STHoles: a multidimensional workload-aware histogram. ACM SIGMOD Rec., 30 (2):211–222, 2001.
6. Ganesan D., Estrin D., and Heidemann J. DIMENSIONS: why do we need a new data handling architecture for sensor networks? In Proc. ACM Workshop on Hot Topics in Networks, 2002.
7. Gao J., Guibas L.J., Hershberger J., and Zhang L. Fractionally cascaded information in a sensor network. In Proc. 3rd Int. Symp. Inf. Proc. in Sensor Networks, 2004, pp. 311–319.
8. Greenstein B., Estrin D., Govindan R., Ratnasamy S., and Shenker S. DIFS: a distributed index for features in sensor networks. In Proc. IEEE Int. Workshop on Sensor Network Protocols and Applications, 2003, pp. 163–173.
9. Hellerstein J.M., Naughton J.F., and Pfeffer A. Generalized search trees for database systems. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 562–573.
10. Knuth D.E. Sorting and Searching, The Art of Computer Programming, vol. 3. Addison Wesley, Redwood City, CA, 1973.
11. Li X., Kim Y.J., Govindan R., and Hong W. Multi-dimensional range queries in sensor networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003, pp. 5–7.
12. Madden S.R., Franklin M.J., Hellerstein J.M., and Hong W. TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst., 30(1):122–173, 2005.
13. Nath S., Gibbons P.B., Seshan S., and Anderson Z.R. Synopsis diffusion for robust aggregation in sensor networks. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 250–262.
14. Reiss F., Garofalakis M., and Hellerstein J.M. Compact histograms for hierarchical identifiers. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 870–881.
15. Wang W., Yang J., and Muntz R. STING: a statistical information grid approach to spatial data mining. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997, pp. 186–195.

Hierarchical Entity-Relationship Model

▶ Extended Entity-Relationship Model

Hierarchical Faceted Metadata

▶ Faceted Search

Hierarchical Graph Layout

▶ Visualizing Hierarchical Data

Hierarchical Heavy Hitter Mining on Streams

Flip R. Korn
AT&T Labs–Research, Florham Park, NJ, USA

Synonyms

HHH

Definition

Given a multiset S of N elements from a hierarchical domain D and a count thres hold $\phi \in (0,1)$, Hierarchical Heavy Hitters (HHH) summarize the distribution of S projected along the hierarchy of D as a set of prefixes $P \subseteq D$, and are defined inductively as the nodes in the hierarchy such that their "HHH count" exceeds φN, where the HHH count is the sum of all descendant nodes having no HHH ancestors. The approximate HHH problem over a data stream of elements e is defined with an additional error parameter $\varepsilon \in (0,\phi)$, where a set of prefixes $P \subseteq D$ and estimates of their associated frequencies, with accuracy bounds on the frequency of each $p \in P$, f_{min} and f_{max}, is output with $f_{min}(p) \leq f^*(p) \leq f_{max}(p)$ such that $f^*(p)$ is the true frequency of p in S (i.e., $f^*(p) = \sum_{e \preceq p} f(e)$) and $f_{max}(p) - f_{min}(p) \leq \varepsilon N$. Additionally, there is a coverage guarantee that, for all prefixes $q \notin P$, $\phi N > \sum f(e) : (e \preceq q) \wedge (e \not\preceq P)$, with $\preceq$ denoting prefix containment and $(e \preceq P)$ denoting $(\exists p \in P : e \preceq p)$. Algorithms for maintaining HHHs on data streams aim to minimize space usage as well as update time and possibly query reporting time, while maintaining these accuracy and coverage guarantees.

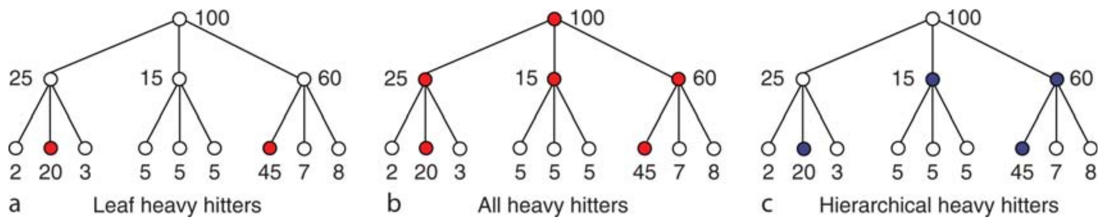
Historical Background

The Hierarchical Heavy Hitters problem was defined in [2] for one-dimensional hierarchical domains, and algorithms for finding online HHHs were given. The multi-dimensional version of the problem was defined and studied in [3,5]. HHHs extend the concept of Heavy Hitters (frequent items), which are all items with frequency at least ϕN, to hierarchical domains; the problem of finding Heavy Hitters on data streams was studied in [6,10,11].

Foundations

Figure 1 shows an example distribution of $N = 100$ items over a simple hierarchy in one dimension, with the counts for each internal node representing the total number of items at leaves of the corresponding subtree. Figure 1a shows that setting $\phi = 0.1$ yields two heavy hitters, that is, items with frequency above 10. However, this does not adequately cover the full distribution, and so one seeks a definition which also considers heavy hitters at points in the hierarchy other than the leaves. A natural approach is to apply the heavy hitters definition at each level of generalization: at the leaves, but also for each internal node. The effect of this definition is shown in Fig. 1b. But this fails to convey the complexity of the distribution: is a node marked as significant merely because it contains a child which is significant, or because the aggregation of its children makes it significant? This leads to the definition of HHHs given a fraction ϕ: find nodes in the hierarchy such that their HHH count exceeds ϕN, where the HHH count is the sum of all descendant nodes which have no HHH ancestors. This is best seen through an example, as shown in Fig. 1c. Observe that the node with total count 25 is not an HHH, since its HHH count is only 5 (less than the threshold of 10): the child node with count 20 is an HHH, and so does not contribute. But the node with total count 60 is an HHH, since its HHH count is 15. Thus the set of HHHs forms a superset of the heavy hitters consisting of only data stream elements, but a subset of the heavy hitters over all prefixes of all elements in the data stream.

One approach to computing HHHs extends the `LossyCounting` algorithm for frequency estimates on streams proposed in [10], which involves maintaining a set of samples from the input stream and associating a small amount of auxiliary information with each sampled item e used for determining the lower- and upper-bounds on the frequency of e. The space usage of `LossyCounting` is $O(\frac{1}{\epsilon}\log(\epsilon N))$, where N is the length of the stream (Better asymptotic space bounds for the heavy hitter problem were obtained in [12], but it was reported in [10] that `LossyCounting` often



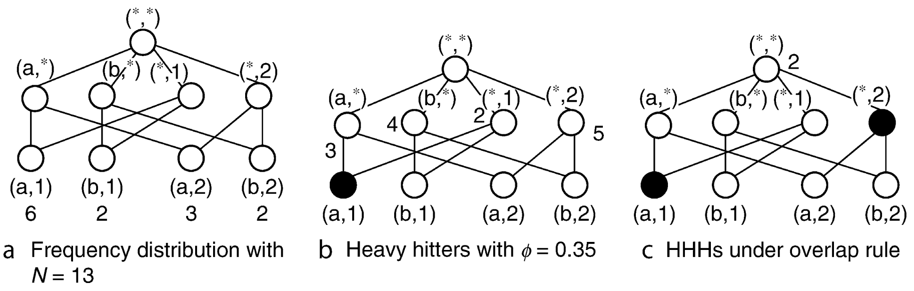
Hierarchical Heavy Hitter Mining on Streams. Figure 1. Illustration of HHH concept ($N = 100$, $\phi = 0.1$).

uses less space in practice). Online algorithms for finding HHHs similarly sample the input stream, but maintain a trie data structure T consisting of a set of tuples which correspond to samples from the input stream and their prefix relationships. There are two classes of these algorithms, which are described as follows.

Using the "Full Ancestry" strategy for finding HHHs, the data structure tracks information about a set of nodes that vary over time, but which always form a subtree of the full hierarchy. It enforces the property that if information about a node in the hierarchy is stored, then all of its ancestors are also stored. When a new node is inserted, information stored by its ancestors is used to give more accurate information about the possible frequency count of the node. This has the twin benefits of yielding more accurate answers and keeping fewer nodes in the data structure (since one can more quickly determine if a node cannot be frequent and so does not need to be stored). Thus, it can be shown that the algorithm requirements are no worse than that required for find heavy hitters at all prefix lengths of maximum length h, which is $O(\frac{h}{\epsilon}\log(\epsilon N))$ based on `LossyCounting`.

Observe that the previous strategy can be wasteful of space, since it retains all ancestors of the fringe nodes, even when these have low (even zero) count. In the "Partial Ancestry" strategy, such low count nodes can be removed from the data structure, thus potentially using less space. Thus, it is no longer the case that every prefix that is stored has all its ancestors stored as well. Frequency bounds are obtained from the closest existing ancestor of the newly inserted element. While this strategy does not come with space and time guarantees, its performance can be superior to the Full Ancestry strategy in practice [2].

The *Multi-Dimensional Hierarchical Heavy* Hitters problem is to find all items whose count exceeds a given fraction, ϕ, of the total count of all items, after discounting the appropriate descendants that are themselves HHHs. This still needs further refinement, since in this setting nodes can have multiple parents (i.e., the prefix containment forms a lattice rather than a tree), so it is not immediately clear how to compute the count of items at various nodes. In the one-dimensional HHH problem, the semantics of what to do with the count of a single element when it was rolled up was clear: simply add the count of the rolled up element to that of its (unique) parent. In the d-dimensional case, adding the count of the rolled up element to all its d parents runs the risk of overcounting. This can be rectified via careful bookkeeping, as illustrated in the example in Fig. 2. Consider a two-dimensional domain, where the first attribute can take on values a and b, and the second 1 and 2. Figure 2a shows a distribution where $(a, 1)$ has count 6, $(a, 2)$ has count 3, $(b, 1)$ has count 2, and $(b, 2)$ has count 2. Moving up the hierarchy on one or other of the dimensions yields internal nodes: $(a, *)$ covers both $(a, 1)$ and $(a, 2)$ and has count 9; $(*, 2)$ covers both $(a, 2)$ and $(b, 2)$, and has count 5. Setting $\phi = 0.35$ means that a count of 5 or higher suffices, thus there is only one Heavy Hitter over the leaves of the domain, as shown in Fig. 2b. In the multi-dimensional case, since one input item may contribute to multiple ancestors becoming HHHs, each node therefore counts the contributions of all its non-HHH descendants. Figure 2c shows the result on the example: the node $(*, 2)$ becomes an HHH, since it covers a count of 5. $(*, 1)$ is not an HHH, because the count of its non-HHH descendants is only 2. Note that the root node is not a HHH since, after subtracting off the contributions from $(a, 1)$ and $(*, 2)$, its remaining count is only 2. Online algorithms for finding multi-dimensional HHHs in data streams were first proposed and



Hierarchical Heavy Hitter Mining on Streams. Figure 2. Illustration of HHH in two dimensions.

analyzed in [3]; a complete analyis of space and time requirements can be found in [5].

Note that the bounds on the frequency estimates given by online HHH algorithms are for the total frequencies, not the (discounted) HHH counts. By appropriate rescaling of ε, one could find the discounted counts accurately; however, this comes at a high price for the required space, multiplying by a factor proportional to the largest possible number of HHH descendants. It is shown in [9] that such a factor is essentially unavoidable: any approximation guarantee on discounted counts of ϕ-HHHs requires $\Omega(1/\phi^{d+1})$ space. Hence, only guarantees on the total frequencies, not HHH counts, are provided by online algorithms.

Related follow-up work considered the problem of HHH detection without compensating for the count of HHH descendants [13]. The problem therefore simplifies to finding all nodes in the lattice whose count is above the threshold, that is, the heavy hitters over all prefixes. In [1], a sketch-based approach was considered for finding one-dimensional HHHs requiring $O(\frac{h}{\epsilon}\ln(1/\delta))$ space and $O(h\ln(1/\delta))$ time per update.

Key Applications

Hierarchical heavy hitters were implicitly studied in [7,8], to find patterns of traffic (offline) over a multidimensional hierarchy of source and destination ports and addresses in what the authors call "compressed traffic clusters." Online HHHs were used for realtime anomaly detection on IP data streams, again based on source and destination ports and addresses in [13]. A DDos detection system based on HHHs was proposed in [12].

Experimental Results

Experiments reported in [5], based on an implementation of HHHs as a User Defined Aggregate Function (UDAF) in the Gigascope data stream system (See [4] for a description of Gigascope UDAFs), show that the proposed online algorithms yielded outputs that were very similar to their offline counterparts and significantly better than that of heavy hitters on all prefixes. The algorithms also require an order of magnitude less space usage while giving competitive performance compared with the heavy hitters approach. The Partial Ancestry strategy is better when space usage is of importance whereas the Full Ancestry strategy is better when update time and output size is more crucial.

Cross-references

▶ Heavy Hitters
▶ Quantiles on Streams

Recommended Reading

1. Cheung-Mon-Chan P. and Clerot F. Finding hierarchical heavy hitters with the count min sketch. In Proc. Int. Workshop on Internet Rent, Simulation, Monitoring, Measurement, 2006.
2. Cormode G., Korn F., Muthukrishnan S., and Srivastava D. Finding hierarchical heavy hitters in data streams. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 464–475.
3. Cormode G., Korn F., Muthukrishnan S., and Srivastava D. Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 155–166.
4. Cormode G., Korn F., Muthukrishnan S., Johnson T., Spatscheck O., and Srivastava D. Holistic UDAFs at streaming speeds. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 35–46.
5. Cormode G., Korn F., Muthukrishnan S., and Srivastava D. Finding hierarchical heavy hitters in streaming data. ACM Trans. Knowl. Discov. Data, 1(4), 2008.
6. Demaine E., López-Ortiz A., and Munro J.I. Frequency estimation of internet packet streams with limited space. In Proc. European Symp. on Algorithms, 2002, pp. 348–360.
7. Estan C., Savage S., and Varghese G. Automatically inferring patterns of resource consumption in network traffic. In Proc. ACM Int. Conf. of the on Data Communication, 2003, pp. 137–148.
8. Estan C. and Magin G. Interactive traffic analysis and visualization with Wisconsin netpy. In Proc. Int. Conf. on Large Installation System Administration, 2005, pp. 177–184.
9. Hershberger J., Shrivastava N., Suri S., and Toth C. Space complexity of hierarchical heavy hitters in multi-dimensional data streams. In Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 2005, pp. 338–347.
10. Manku G.S. and Motwani R. Approximate frequency counts over data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 346–357.
11. Misra J. and Gries D. Finding repeated elements. Sci. Comput. Program., 2:143–152, 1982.
12. Sekar V., Duffield N., Spatscheck O., van der Merwe J., and Zhang H. LADS: large-scale automated DDoS detection system. In Proc. USENIX Annual Technical Conf., General Track, 2006, pp. 171–184.
13. Zhang Y., Singh S., Sen S., Duffield N., and Lund C. Online identification of hieararchical heavy hitters: algorithms, evaluation and applications. In Proc. Internet Measurement Conference. Taormina, 2004, pp. 135–148.

Hierarchical Memory System

▶ Memory Hierarchy

Hierarchical Regular-Decomposition Structures

▶ Quadtrees (and Family)

Hierarchical Spatial Indexes

▶ Quadtrees (and Family)

Hierarchical Storage Management

▶ Storage of Large Scale Multidimensional Data

Hierarchical Visualization

▶ Visualizing Hierarchical Data

Hierarchies

▶ Specialization and Generalization

Hierarchy

Torben Bach Pedersen
Aalborg University, Aalborg, Denmark

Definition

A *hierarchy* is a structure specifying the containment relationships of a number of *values* (or *nodes* or *elements*). A hierarchy has a single *root* (or *top* or *ALL*) value. A value in a hierarchy may be linked to one or more *children* and a non-top value is linked to one or more *parents*. The links between values thus specify the containment structure. Hierarchies are essential for specifying *dimensions* in multidimensional *cubes*.

Key Points

The notion of a dimension is an essential and distinguishing concept for multidimensional cubes. Dimensions are used for two purposes: the *selection* of data and the *grouping* of data at a desired level of detail. As an example, a three-dimensional cube for capturing product sales may have a Product dimension, a Time dimension, and a Store dimension. The Product dimension captures information about the product sold, such as textual description, color, weight, etc., as well as groupings of products (product groups, product families, departments, etc.). The root value, representing "All products" then has store departments such as "Food" and "Electronics" as children, "Food" has product families such as "Dairy," "Meat," etc., as children and so on.

A hierarchy is typically organized into a number of *levels*, each of which represents a level of detail that is of interest to the analyses to be performed. In the example above, the levels could be ALL Products, Departments, Product Families, Product Groups and Products. The instances of the dimension, e.g., "Food" are typically called *dimension values*. Each such value then belongs to a particular level. The hierarchy is used intensively, e.g., when performing *On-Line Analytical Processing (OLAP)*. Here, data is explored by either moving up in the hierarchy to get a better overview (*rollup*) or moving down in the hierarchy to get more details (*drilldown*).

In some cases, it is advantageous for a dimension to have *multiple hierarchies* defined on it. For example, a Time dimension may have hierarchies for both *Fiscal Year* and *Calendar Year* defined on it. Multiple hierarchies share one or more common lowest level(s), e.g., Day and Month, and then group these into multiple levels higher up, e.g., Fiscal Quarter and Calendar Quarter to allow for easy reference to several ways of grouping. Most multidimensional models allow multiple hierarchies. A dimension hierarchy is defined in the metadata of the cube, or the metadata of the multidimensional database, if dimensions can be shared.

Most models require dimension hierarchies to form *balanced trees*. This means that the dimension hierarchy must have uniform height everywhere, e.g., all departments, even small ones, must be subdivided into Product families (and so on, all the way down to Products). If the hierarchy is not balanced like this, it is referred to as a *non-onto* or *unbalanced* hierarchy [2,3]. Additionally, direct links between dimension values can only go between immediate parent-child levels, and not jump two or more levels. For example, all cities are first grouped into states and then into countries, so cities cannot be grouped directly under countries (as is the case in Denmark which has no states). If such non-immediate links occur in the hierarchy, it is called a *non-covering* or *ragged* hierarchy [2,3], Finally, each non-top value has precisely one parent, e.g., a product must belong to exactly one product group. This may not always be desirable, e.g., it would be natural to put skimmed milk into both the "Diet" and "Dairy" product groups. If the hierarchies do not form balanced trees, this affects the so-called *summarizability* of the data, which means that special care must be taken to obtain correct aggregation results [1].

Cross-references

▶ Dimension
▶ Multidimensional Modeling
▶ On-Line Analytical Processing
▶ Summarizability

Recommended Reading

1. Lenz H. and Shoshani A. Summarizability in OLAP and statistical data bases. In Proc. 9th Int. Conf. on Scientific and Statistical Database Management, 1997, pp. 39–48.
2. Pedersen T.B., Jensen C.S., and Dyreson C.E. A foundation for capturing and querying complex multidimensional data. Inf. Syst., 26(5):383–423, 2001.
3. Thomsen E. OLAP Solutions: Building Multidimensional Information Systems. Wiley, New York, NY, 1997.

High Dimensional Indexing

Christian Böhm, Claudia Plant
University of Munich, Munich, Germany

Synonyms

Indexing for similarity search

Definition

The term High Dimensional Indexing [6,9] subsumes all techniques for indexing vector spaces addressing problems which are specific in the context of high dimensional data spaces, and all optimization techniques to improve index structures, and the algorithms for various variants of similarity search (nearest neighbor, reverse nearest neighbor queries, range queries, similarity joins etc.) for high dimensional spaces. The well-known Curse of Dimensionality leads to a worsening of the index selectivity with increasing dimensionality of the data space, an effect which already starts at dimensions of 10–15, also depending on the size of the database and the data distribution (clustering, attribute dependencies). During query processing, large parts of conventional hierarchical indexes (e.g., R-tree) need to be randomly accessed, which is by a factor of up to 20 more expensive than sequential reading operations. Therefore, specialized indexing techniques for high dimensional spaces include e.g., ideas to scale up sequential scans, hybrid approaches combining elements of hierarchical and scan-based indexes, dimensionality reduction and data compression.

Historical Background

One of the first indexing techniques which is nowadays wide spread in commercial databases is the B-tree. One dimensional data items, so-called keys, are stored in a hierarchical balanced search tree. Since the height of a B-tree is bounded above by $O(\log N)$, the index provides retrieval in logarithmic time complexity. Approaches to extend the B-tree to higher dimensions include the R-tree [12], which has been originally designed for two dimensional spatial objects (polygons). The R-tree consists of two types of nodes: directory and data pages. The directory pages contain rectangles (or hyper-rectangles for higher dimensional data) which are the minimum bounding rectangles (MBR) for the underlaying sub-trees, all the way down to the data pages at the leaves. The hyper-rectangles may overlap, and also the directory does not need to cover the whole data space. This implies that the search is not guaranteed to be restricted to one single path from the root to a leave as in the B-tree. However, since the R-tree is a balanced search tree and algorithms for tree construction and restructuring are designed to minimize overlap, search operations on low dimensional data can be performed in almost logarithmic time. Various variants of the R-tree, such as the R*-tree have been proposed.

Foundations

The central problem of similarity search in high dimensional spaces is the deterioration of the index selectivity with increasing dimensionality of the data space, an effect which is essentially inherent to any kind of index structure. Conventional hierarchical index structures achieve impressive performance gains over sequential scan on low to medium dimensional data by excluding large parts of the data contained in sub-trees which do not need to be visited. Special properties of high dimensional spaces, often subsumed by the term Curse of Dimensionality cause that conventional hierarchical indexes break down in performance on high dimensional data. With increasing dimensionality, more points are situated at the boundaries of the data space and the distances between points assimilate. During retrieval, large parts of the index have to be accessed. In the extreme case of very high dimensionality ($d \rightarrow \infty$) it is therefore obvious that no indexing technique is able to outperform the simplest processing technique for similarity queries, the sequential scan. This fact has been discussed in the context of cost models and of various indexing methods [3,7,19] and has also solicited a scientific discussion of the usefulness of similarity queries per se [6]. However, when, and to which extent the dimensionality curse occurs, depends on the size of the database and various other parameters such as the statistical distribution of the data, correlations between the single attributes (i.e., whether the complete data space is covered or the data reside in an arbitrarily complex subspace), and clustering (i.e., if there are clearly distinguishable groups of similar objects in the database). Many indexing methods can be successfully applied in moderately high dimensions, and many dedicated indexing methods are able to index data spaces of a dimensionality which is considerably higher than expected. In order to achieve the goal of efficiently indexing (moderately) high dimensional spaces, the well-known proposals to high-dimensional indexing all apply a combination of tricks which can be categorized into the following classes:

1. Dimensionality reduction
2. Data compression
3. Optimized i/o schedules (page size optimization and fast index scan)
4. Hierarchy flattening
5. Optimizing of the shape of page regions
6. Clustering

In the following, the most relevant approaches to High Dimensional Indexing are described in chronological order.

TV-Tree

High dimensional real data often exhibits a much lower intrinsic dimensionality. In this case, principal component analysis leads to a few highly loaded components almost totally covering the variance in the data. The TV-tree [16] exploits the fact that those top ranked components are highly selective dimensions for similarity search, whereas the remaining dimensions are of minor importance. Therefore, only the most selective components are used for pruning during query processing. Since irrelevant sub-trees should be excluded as early as possible, these components, called *active dimensions* are placed at the topmost levels of the index. A region of the TV-tree is described by a sphere in the active dimensions. The remaining dimensions may be active at lower levels of the index or not selective enough for query processing. The authors report a good speed-up in comparison with the R*-tree if the data can be effectively reduced by PCA, Fourier transform etc. On uniform or other real data not amenable to PCA, the X-tree outperforms the TV-tree.

The main contribution to High Dimensional Indexing is the implicit dimensionality reduction. Depending on the depth of the tree, only the first few dimensions are considered in the directory structure. Further, in these considered dimensions, the pages are approximated by bounding spheres which are more suitable for Euclidean queries.

SS-Tree

The SS-tree [20] also uses spheres instead of bounding rectangles as page regions. For efficiency, the spheres are not minimum bounding spheres. Rather, the centroid of the stored points is used as center for the sphere and the radius is chosen such that all objects are included. The region description consists of the centroid and the radius and thus allows efficient pruning. The SS-tree is suitable for all kinds of data distributions and outperforms the $R^{\star}$-tree by a factor of 2, which is mainly due to the fact that spherical page regions are, as mentioned, more suitable to support Euclidean queries. In addition, the algorithms for tree construction and maintenance are highly efficient and also very effective for low to medium dimensional data. On high dimensional data, the SS-tree encounters similar problems as the R-tree family. Compared to the minimum bounding rectangles (MBR) of R-trees, spherical page regions are even more difficult to split in an overlap-free way. This problem is tackled by the SR-tree [15] which uses a combination of a rectangle and a sphere as page region.

X-Tree

In contrast to the TV-tree and the SS-tree, the X-tree [5] is a high-dimensional index structure which is more closely related to the R-tree, and, in particular, to the R*-tree. It is the first technique which introduces the ideas of flattening the hierarchies of the directory and of enlarging the sizes of directory pages, but in contrast to the techniques described in the next section, this idea was not inspired by a cost analysis but simply from the observation that it can be difficult to split directory nodes (particularly of the higher directory levels) in an overlap-free way. The regions of the child nodes of such directory nodes are axis-parallel rectangles, which have, themselves, been created in a recursive process of splitting. Typically, only the first split in the split history of the child nodes is a good split decision for the parent node. Therefore, each directory node stores this split history of its children and, thus, tries to imitate the splits of its children. If this imitation would result in an unbalanced tree, the split is avoided and, instead, a so-called supernode is created, i.e., a node with an enlarged size and capacity.

Cost Model Based Optimization Techniques

In [3,7], a cost model for vector space index structures with particular emphasis on high dimensional spaces was proposed. The main idea is to estimate the average size of a page region and of a query and to form the Minkowski sum of these two geometric objetcs which has been shown to be a measure of the probability of a page access. To take effects in high-dimensional spaces into account, two concepts were additionally included into the cost model, *the boundary effect* and *the fractal dimension*. The boundary effect means that for high-dimensional spaces typically large parts of a query sphere and of the Minkowski sum are outside the covered data space. The fractal dimension covers the sparsity of the data space. Typically, a high dimensional space is not uniformly and independently in all dimensions covered by data objects. Rather, some of the attributes are dependent on each other, and, therefore, only subspaces of lower dimensionality (not necessarily linear subspaces) are populated. The intrinsic dimensionality of the data can be formalized by the fractal dimension.

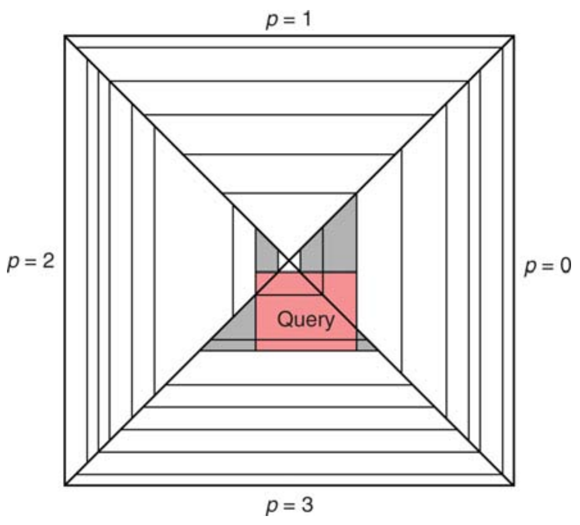
It is important to optimize various parameters of an index using a cost model in order to make it competitive. For instance, in [8], it was proposed to optimize the page size of data pages of the index according to the cost model. To do this automatically is particularly important because data sets with a large intrinsic dimensionality cause scanning of large parts of the index. If small page sizes (such as 4 KB) are used in this case, the random accesses cause a large I/O load, and an unoptimized index is much slower than sequential data processing. In contrast, if the intrinsic dimensionality is small, then large page sizes lead to unnecessarily large reading operations. Carefully optimized page sizes outperform sequential scanning and non-optimized indexes for most of the data sets and tie with the competitors in the worst case. The dynamic page size optimization allows the automatic adaptation of the page size even if the data distribution of the data set changes over time. Moreover, it is possible to use different page sizes for different parts of the index structure, if the index stores groups of data with different characteristics.

Another example of the successful application of a cost model is tree striping [4], where the data objects are vertically decomposed into sub-vectors which are stored in separate search trees. The dimensionality of the sub-vectors (and, inversely, the number of index structures to store them) is an optimization task which is important, because the unnecessary decomposition in too many, small sub-vectors causes a large overhead in the final merging of the results, whereas no decomposition or an insufficient decomposition the query processing inside a single index is too expensive due to the curse of dimensionality. Again, a cost model can decide an optimal dimensionality.

Pyramid Technique

The Pyramid Technique [1] is a *one-dimensional embedding technique*, that means, it transforms the high-dimensional objects into a single-dimensional space which can be efficiently indexed using B-trees, for instance. In contrast to most previous embedding techniques which are based on space-filling curves, the Pyramid Technique does not rely on a *recursive* schema of space partitioning. In contrast, the data space is partitioned into $2 \cdot d$ hyper-pyramids which share the origin of the data space (which can be chosen as the center point of the data set) as top point and have each an individual $(d-1)$-dimensional basis area (cf. the four pyramids in Fig. 1 $p = 0..3$). The pyramids are systematically numbered which forms the first part of the embedding key (a natural number p). The second part is the distance (with respect to the maximum metric) from the origin (a positive real number r). The embedding key can be formed as an ordered pair $k = (p, r)$, or, equivalently, if the maximum of all r-values (r_{max}) is known, one single embedding key

$k' = r_{max} \cdot p + r$ can be formed. In both cases, a d-dimensional range query can be translated into a set of search intervals on the search keys. The number of intervals is at most $2 \cdot d$ because the query object can at most have one intersection with each of the pyramids. Since nearest neighbor queries can be transformed into range queries (which requires a set of at most two one-dimensional ranking processes per pyramid) it is also possible to evaluate nearest neighbor queries. The Pyramid Technique is in general not limited to a particular metric, but the schema of space partitioning makes it particularly suited for queries using the maximum metric. The Pyramid Technique has inspired a number of other techniques such as the Onion Technique [10] or Concentric Hyperspaces [11] and many others which focus on different metrics including the Euclidean metric.



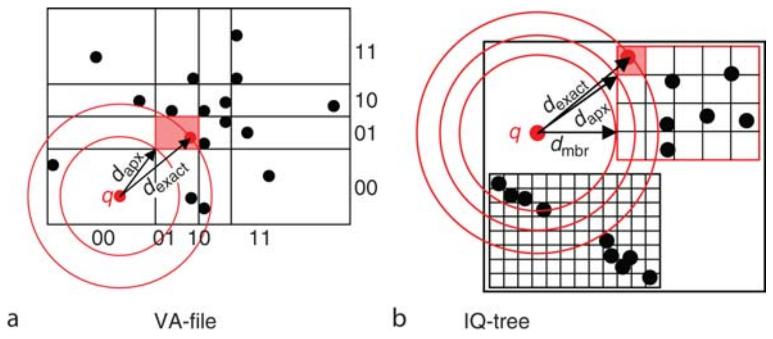
High Dimensional Indexing. Figure 1. Pyramid technique.

VA-File

As an alternative to tree based indexing techniques, the VA-file (Vector Approximation File [6]) has been designed to speed up the linear scan. The basic idea is to store compact approximations of the high dimensional feature vectors in the so-called *approximation file* which is small enough to enable a fast linear scan during query processing. The approximations are derived by dividing each dimension of the data space into equally populated intervals. Thus, the data space is divided into hyper-rectangular cells. Each cell is allocated by a bit string composed of a fixed number of bits per dimension. As approximation for all contained points, this bit string is stored in the approximation file. The principle of vector quantization is visualized in Fig. 2a. In this example, two bits are assigned to each dimension. While the approximation file is scanned, upper and lower bounds on the distances from the query to the approximations d_{apx} are derived and refined. As a result, only very few approximations have to be further checked for answer candidates, which requires random accesses. Extensions to the VA-file e.g., include the parallel VA-file [18] originally designed for parallel nearest neighbor search in large image collections and the kernel-VA-file [13] for complex, kernel supported distance metrics. Due to global quantization, the VA-file is especially suitable for uniformly distributed data. For clustered data, hierarchical techniques show superior performance.

IQ-Tree

With the IQ-tree [2], the idea of quantizing the data using a grid has been integrated into a hierarchical indexing method. In the IQ-tree, every data page has two versions, one which contains the data points in a compressed way, and one which contains the exact



High Dimensional Indexing. Figure 2. Schematic view of vector quantization.

information. In contrast to the VA-file, each page is quantized independently, and this independent quantization gives the structure its name. The quantization grid is not based on quantiles but is a regular grid partition of the rectangular page region. In the IQ-tree, it was shown that quantiles are not necessary because the page regions already serve a sufficient adaptation to the actual data distribution, and storing individual quantiles for each page would result in a prohibitive storage overhead. In contrast to the VA-file, the resolution of the grid is not determined experimentally but dynamically optimized using a cost model. In the IQ-tree, the actual directory is non-hierarchic and contains only one level. Taken together, the IQ-tree consists of three levels, the directory level, the compressed data level and the uncompressed data level. The vector quantization principle is illustrated in Fig. 2b. When considering a query object q and an arbitrary database object, there are two lower bounding approximations of the exact distance d_{exact}, the distance to the minimum bounding rectangle d_{mbr} which can be determined from the directory and the distance to the grid cell (d_{apx}) which can be derived from the compressed data level.

Apart from the idea of independent quantization, the IQ tree contains the idea of the Fast Index Scan (FIS). After scanning of the directory, it can be decided for every page, whether it is certainly needed, certainly excluded, or has some probability to be needed for a given nearest neighbor query. The probability can again be estimated using a cost model. From this probability, a query processing algorithm can derive optimal schedules of pages, i.e., pages which have neighboring disk addresses, can be called in together in a single I/O operation if they have both high probabilities. Depending on data and query characteristics, the pages of the compressed data level can either be accessed by random accesses or in a more sequential style. Another related approach, also designed to allow a flexible adaptation of the height of the directory to the current data distribution is the A-tree [17].

iDistance

iDistance [21] combines a one-dimensional embedding technique with clustering. The embedding is obtained by expressing the similarity ratios in high dimensional space in terms of distances to reference points which can be stored in a single dimensional index. More precisely, the data space is first split into partitions. As a second step, a reference point is selected for each partition. For all data objects the distances to their reference points are stored in a B^+-tree. The performance of the index strongly depends on an appropriate partitioning and on the strategy how to select the reference points. In the original paper, the authors proposed two variants of partitioning: The straightforward approach of equal data space partitioning is especially suitable for uniformly distributed data. For clustered data the partitions can be determined by an arbitrary clustering algorithm, a simple sampling based method is proposed in the paper. Often it is favorable to select the centroid of a partition as reference point, however selecting an edge point may help to reduce overlap.

The most important strategies to cope with the problems of high dimensional spaces can be subsumed by clustering and mapping to one dimensional space. During query processing, the maximum distance between the query point and the points contained in each partition is used for pruning. The single dimensional space of distances can be very efficiently searched supported by the B^+-tree. However, in high dimensional data, the distances to reference points are often not selective and there are no clusters in the full dimensional space such that large parts of the index need to be accessed. But many real world data sets exhibit more selective subspaces. Therefore in [14] a subspace clustering step to identify local correlations in the data is applied before indexing.

Key Applications

High dimensional indexing is important for similarity search systems in various application areas such as multimedia, CAD, systems biology, medical image analysis, time sequence analysis and many others. Complex objects are typically transformed into vectors of a high-dimensional space (feature vectors), and the similarity search thereby translates into a range or nearest neighbor query on the feature vectors. High-dimensional feature vectors are also required for more advanced data analysis tasks such as cluster analysis or classification.

Cross-references

- ▶ Curse of Dimensionality
- ▶ Dimensionality Reduction
- ▶ Indexing Metric Spaces
- ▶ Nearest Neighbor Query

Recommended Reading

1. Berchtold S., Böhm C., and Kriegel H.-P. The pyramid-technique: towards breaking the curse of dimensionality. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 142–153.
2. Berchtold S., Böhm C., Jagadish H.V., Kriegel H.-P., and Sander J. Independent quantization: an index compression technique for high-dimensional data spaces. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 577–588.
3. Berchtold S., Böhm C., Keim D.A., and Kriegel H.-P. A cost model for nearest neighbor search in high-dimensional data space. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 78–86.
4. Berchtold S., Böhm C., Keim D.A., Kriegel H.-P., and Xu X. Optimal multidimensional query processing using tree striping. In Proc. 2nd Int. Conf. Data Warehousing and Knowledge Discovery, 2000, pp. 244–257.
5. Berchtold S., Keim D.A., and Kriegel H.-P. The x-tree : an index structure for high-dimensional data. In Proc. 22nd Int. Conf. on Very Large Data Bases, 1996, pp. 28–39.
6. Beyer K.S., Goldstein J., Ramakrishnan R., and Shaft U. When is "nearest neighbor" meaningful? In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 217–235.
7. Böhm C. A cost model for query processing in high dimensional data spaces. ACM Trans. Database Syst., 25(2):129–178, 2000.
8. Böhm C. and Kriegel H.-P. Dynamically optimizing high-dimensional index structures. In Advances in Database Technology, Proc. 7th Int Conf on Extending Database Technology, 2000, pp. 36–50.
9. Böhm C., Berchtold S., and Keim D.A. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv., 33(3):322–373, 2001.
10. Chang Y.-C., Bergman L.D., Castelli V., Li C.-S., Lo M.-L., and Smith J.R. The onion technique: indexing for linear optimization queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 391–402.
11. Ferhatosmanoglu H., Agrawal D., and Abbadi A.E. Concentric hyperspaces and disk allocation for fast parallel range searching. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 608–615.
12. Guttman A. R-trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
13. Heisterkamp D.R. and Peng J. Kernel vector approximation files for relevance feedback retrieval in large image databases. Multimed. Tools Appl., 26(2):175–189, 2005.
14. Jin H., Ooi B.C., Shen H.T., Yu C., and Zhou A. An adaptive and efficient dimensionality reduction algorithm for high-dimensional indexing. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 87–98.
15. Katayama N. and Satoh S. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 369–380.
16. Lin K.-I., Jagadish H.V., and Faloutsos C. The tv-tree: an index structure for high-dimensional data. VLDB J., 3(4):517–542, 1994.
17. Sakurai Y., Yoshikawa M., Uemura S., and Kojima H. The A-tree: an index structure for high-dimensional spaces using relative approximation. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 516–526.
18. Weber R., Böhm K., and Schek H.-J. Interactive-time similarity search for large image collections using parallel VA-files. In Proc. 4th European Conf. Research and Advanced Tech. for Digital Libraries. Springer, 2000, pp. 83–92.
19. Weber R., Schek H.-J., and Blott S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 194–205.
20. White D.A. and Jain R. Similarity indexing with the ss-tree. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 516–523.
21. Yu C., Ooi B.C., Tan K.-L., and Jagadish H.V. Indexing the distance: an efficient method to KNN processing. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 421–430.

High-Dimensional Clustering

► Document Clustering

Higher-Order Entity-Relationship Model

► Extended Entity-Relationship Model

Histogram

Qing Zhang
CSIRO ICT Centre, Herston, QLD, Australia

Definition

Given a relation R and an attribute X of R, the domain D of X is the set of all possible values of X, and a finite set $V(\subseteq D)$ denotes the distinct values of X in an instance r of R. Let V be ordered, that is: $V = \{v_i : 1 \leq i \leq n\}$, where $v_i < v_j$ if $i < j$. The instance r of R restricted to X is denoted by T, and can be represented as: $T = \{(v_1, f_1), \cdots (v_n, f_n)\}$. In T, each v_i is distinct and is called a *value* of T; and f_i is the occurrence of v_i in T and is called the *frequency* of v_i, T is called the *data distribution*. A *histogram* on data distribution T is constructed by the following two steps.

1. Partitioning the values of T into $\beta(\geq 1)$ disjoint intervals (called buckets) – $\{B_i : 1 \leq i \leq \beta\}$, such that each value in B_i is smaller than that in B_j if $i < j$.

2. Approximately representing the frequencies and values in each bucket.

Key Points

Histogram, as a summarization of the data distribution, has been successfully used by the query optimizer in RDBMS for the past few decades. Due to its simplicity and straightforwardness, it is also the most popular data reduction technique for approximately processing range aggregates. However, to find a good histogram, which occupies a reasonable small storage space yet can provide accurate approximations, is not an easy work. The key issues in constructing a histogram are: (i) how to partition the original data into buckets, and (ii) how to approximate the original data in each bucket.

Since a histogram only occupies limited storage space in the system catalog, partition methods study how to effectively partition the original data distribution, under a fixed bucket number, to construct a histogram with good performance. Different partition strategies can greatly influence the histogram's performance. Four basic partition methods, based on different partition goals, are: *Equi-width, Equi-sum, Maxdiff, V-optimal* [3]. Empirical results indicate that in most applications, Maxdiff and V-optimal outperform Equi-width and Equi-sum. V-optimal usually leads to more accurate approximation than Maxdiff does. However, the time complexity on constructing a V-optimal histogram seriously impedes its further application.

Techniques on approximately representing the values and frequencies in each histogram bucket usually adopt the *uniform spread* and *average* frequency assumptions. That is distinct values in a same bucket are assumed to evenly span the bucket and every value has a same frequency, the average frequency of the bucket. More accurate approximation techniques on the values and frequencies have also been proposed in recent years, such as *curve-fitting histogram* [2] and *4-level tree index* [1].

There exist other types of histograms in the literature, such as spatio-temporal histogram, multi-dimensional histogram, etc.

Cross-references

▶ Approximate Query Processing
▶ Data Reduction
▶ Hierarchical Data Summarization
▶ Quantiles on Streams

Recommended Reading

1. Buccafurri F., Rosaci D., Doutieri L., and Sacca D. Improving range query estimation on histograms. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp 628–638.
2. Konig A.C. and Weikum G. Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999.
3. Poosala V., Ioannidis Y.E., Haas P.J., and Shekita E.J. Improved histograms for selectivity estimation of range predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 1996, pp. 294–305.

Histograms on Streams

MARTIN J. STRAUSS
University of Michigan, Ann Arbor, MI, USA

Synonyms

Piecewise-constant approximations

Definition

A B-bucket histogram of length N is a partition of the set $[0,N)$ of N integers into intervals $[b_0,b_1) \cup [b_1,b_2) \cup ... \cup [b_{B-1},b_B)$, where $b_0 = 0$ and $b_B = N$, together with a collection of B heights h_j, for $0 \leq j < B$, one for each bucket. On point query i, the histogram answer is h_j, where j is the index of the interval (or "bucket") containing i; that is, the unique j with $b_j \leq i < b_{j+1}$. In vector notation, χ_S is the vector that is 1 on the set S and zero elsewhere and the answer vector of a histogram is $\vec{H} = \sum_{0 \leq j < B} h_j \chi_{[b_j,b_{j+1})}$.

A histogram, $\vec{H}$, is often used to approximate some other function, $\vec{A}$, on $[0,N)$. In building a B-bucket histogram, it is desirable to choose $B-1$ boundaries b_j and B heights h_j that tend to minimize some distance, e.g., the sum square error $\left\|\vec{A}-\vec{H}\right\|^2 = \sum_i |\vec{A}[i]-\vec{H}[i]|^2$.

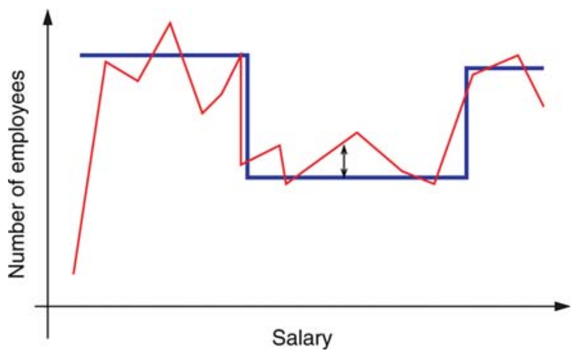
The function $\vec{A}$ may be presented in a stream in one of several ways. In many of these cases, it is infeasible to produce a best histogram; instead, algorithms trade off the representational accuracy of the histogram, the time to process stream items, and the space used.

Historical Background

Fix a universe $[0,N) = \{0,1,2,...,N-1\}$ of N item types, e.g., salaries. The goal is to summarize a vector $\vec{A} = \langle \vec{A}[0], \vec{A}[1], ..., \vec{A}[N-1] \rangle$ over $[0,N)$, in which $\vec{A}[i]$ represents, e.g., the number of employees earning salary i. To summarize $\vec{A}$ by a histogram, partition the interval of indices (salaries) into B subintervals and represent $\vec{A}$ on interval j as a single number, h_j, according to some specification. For example, h_j might be the total number $\vec{A}[b_j,b_{j+1})$ of items in the range (the total number of employees whose salary i falls in the range $[b_j,b_{j+1})$). Alternatively, h_j might be the normalized expression $\frac{\vec{A}[b_j,b_{j+1})}{b_{j+1}-b_j}$ (the *average height* of $\vec{A}$), which conveys the same information but is more natural below to analyze, because the average height is the optimal value of h_j from the perspective of sum-square-error; it will be the focus of this article. See Fig. 1.

Classical histograms include equi-width histograms, in which the jth boundary b_j, for $0 \le j \le B$, is fixed to be (the integer nearest to) jN/B. In many circumstances, average height of all buckets can be found efficiently and straightforwardly when the bucket boundaries are fixed in advance.

Letting the bucket boundaries vary, however, can lead to more accurate histograms. This generalization can be handled [5] by dynamic programming when time and space polynomial in N is allowed; i.e., for non-streaming data. For each $j \le N$ and each $k \le B$, find the best k-bucket histogram on $[0,j)$; this clearly suffices to find the best B-bucket histogram on $[0,N)$. Observe that the best k-bucket histogram on $[0,j)$ consists of a $(k-1)$-bucket histogram on $[0,i)$ for some $i < j$ followed by a single bucket $[i,j)$ with height $\frac{\vec{A}[i,j)}{j-i}$; this naturally leads to an algorithm that uses time $O(N^2B)$, assuming one can find $\vec{A}[i,j)$ in constant time. The latter task may be accomplished by computing, in time and space $O(N)$ during preprocessing, the *prefix array* $\vec{P}$ for $\vec{A}$ defined by $\vec{P}[0] = 0$ and $\vec{P}[j] = \vec{P}[j-1] + \vec{A}[j-1]$ for $j > 0$. Thus $\vec{P}[j] = \vec{A}[0,j) = \vec{A}[0] + \vec{A}[1] + \ldots + \vec{A}[j-1]$ so that $\vec{A}[i,j) = \vec{P}[j] - \vec{P}[i]$ can be computed in constant time.



Histograms on Streams. Figure 1. A histogram of three buckets for a frequency vector of data (jagged polygon). Often the goal is to minimize the sum of square between the data vector and histogram by choosing the bucket boundaries and heights for the histogram.

Many variations of histograms are possible, including equi-depth histograms, also known as quantile summaries, in which bucket boundaries are fixed so that the total number of items in each bucket is equal. For example, a 4-bucket equi-depth histogram has boundaries at the 25th, 50th, and 75th percentiles. Neither quantiles nor other variations will be discussed further in this article.

A fuller history of histogram research is available [4].

Foundations

There are several ways in which a data vector $\vec{A}$ can be presented in a stream; this article discusses two. In the *ordered aggregate* (or *time-series*) presentation, the algorithm receives $\vec{A}[0],\vec{A}[1],\vec{A}[2],\ldots$, in order. In the *turnstile* (or *dynamic*) presentation, $\vec{A}$ is implicit and initially zero. The algorithm receives update commands of the form (i, u), whose semantics is "add u to $\vec{A}[i]$." In general, the update u may be negative or positive. More efficient algorithms are possible if it is known that the updates are positive (or merely that each $\vec{A}[i]$ is never negative); that will not be discussed further here.

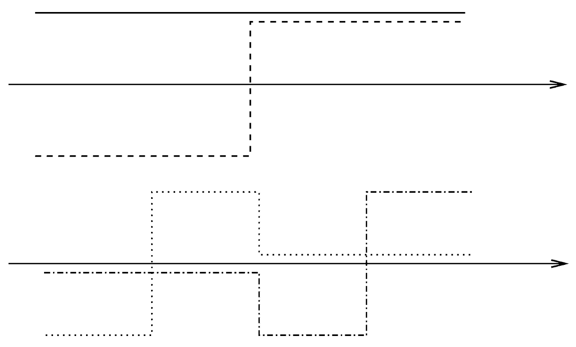
A class of algorithms for building near-optimal histograms relies on the theory of *Haar wavelets*. Assume N is a power of 2 (which can be achieved, without loss of generality, by padding $\vec{A}$ implicitly with trailing zeros). A Haar wavelet is either the constant vector $\phi = N^{-1/2}\chi_{[0,N)}$ or a vector of the form

$$\psi_{j,k}(i) = \begin{cases} -\sqrt{2^j/N}, & kN2^{-(j+1)} \le i < (k+1)N2^{-(j+1)}; \\ +\sqrt{2^j/N}, & (k+1)N2^{-(j+1)} \le i < (k+2)N2^{-(j+1)}, \end{cases}$$

where $0 \le j < \log_2(N)$ and $0 \le k < 2^j$ are integers. That is, for some scale j, divide $[0,N)$ into 2^j subintervals of length $N2^{-j}$. Then $\psi_{j,k}$ takes the value $-\sqrt{2^j/N}$ on the left half of the kth interval and takes the value $+\sqrt{2^j/N}$ on the right half. See Fig. 2.

One also writes ψ_ℓ to index all the $\psi_{j,k}$ and ϕ by a single index. A partial Haar wavelet representation is $\Sigma_{\ell\in\Lambda}\langle\vec{A}\psi_\ell\rangle\psi_\ell$, where $\langle .,.\rangle$ denotes the dot product and Λ is a subset of $[0, N)$.

In many streaming situations, it is not possible to find the best B-bucket histogram $\vec{H}_{\text{opt}}$ for $\vec{A}$. Instead,



Histograms on Streams. Figure 2. The four coarsest Haar wavelets ϕ,$\psi_{0,0}$,$\psi_{1,0}$, and $\psi_{0,1}$ with normalization modified for visibility.

the goal will be to find a near-best histogram, $\|\vec{H} - \vec{A}\| \leq (1+\varepsilon)\|\vec{H}_{\text{opt}} - \vec{A}\|$, where ε is a user-supplied accuracy parameter.

A class of algorithms for building near-optimal histograms efficiently from streaming data proceeds as follows.

1. Find the largest B_1 Haar wavelet terms, where $B_1 \leq (B\log(N)/\varepsilon)^{O(1)}$. Let $\vec{R}$ be the resulting vector. That is, let Λ_1 be a set of size B_1 consisting of the ℓ's that maximize $|\langle \vec{A}, \psi_\ell \rangle|$ and let $\vec{R}$ be $\sum_{\ell \in \Lambda_1} c_\ell \psi_\ell$, where c_ℓ is $\langle \vec{A}, \psi_\ell \rangle$ or a close enough approximation. How to find $\vec{R}$ depends on the way that $\vec{A}$ is presented. The representation $\vec{R}$ must be constructible with little space and little per-item time, so that its construction fits within the constraints of a streaming algorithm. Furthermore, a description of $\vec{R}$ has size $|\vec{R}|$ that is small enough so that polynomial time in $|\vec{R}|$ is also within the constraints of a streaming algorithm for the overall problem.
2. Let $\vec{H}_{rob} = \sum_{\ell \in \Lambda_2} c_\ell \psi_\ell$, where Λ_2 is a subset of Λ_1, chosen so that either $\|\vec{R} - \vec{H}_{\text{rob}}\|$ cannot be significantly improved by enlarging Λ_2 within Λ_1 by $\Theta(B \log(N))$ wavelet terms (if there is such a set Λ_2) or $\Lambda_2 = \Lambda_1$ (otherwise). This is done greedily. Let Λ_2 be initially empty. Let Λ^* be the set of $\Theta(B \log(N))$ indices $\ell \in \Lambda_2 \setminus \Lambda_1$ with maximal $|c_\ell|$; if $\|\Sigma_{\ell \in \Lambda_2 \cup \Lambda^*} c_\ell \psi_\ell - \vec{R}\|$ is significantly less than $\|\sum_{\ell \in \Lambda_2} c_\ell \psi_\ell - \vec{R}\|$, then put $\Lambda_2 \leftarrow \Lambda_2 \cup \Lambda^*$ and repeat; otherwise, halt and return the current Λ_2.
3. Let $\vec{H}$ be the best B-bucket histogram representation for $\vec{H}_{\text{rob}}$. This can be done efficiently using dynamic programming, as above, noting that the boundaries of $\vec{H}$ will be among the boundaries of $\vec{H}_{\text{rob}}$. Output $\vec{H}$ as a near-best B-bucket histogram for $\vec{A}$.

In the case of time-series data, the wavelet coefficients can be found efficiently using a structure based on a binary tree in which each leaf corresponds to an input item $\vec{A}[i]$ and each internal node receives inputs denoted x and y from its children, outputs $y - x$ as a wavelet coefficient (up to normalization) and passes $x + y$ as an input to the node's parent. Note that, while reading $\vec{A}[i]$, the algorithm only needs to instantiate nodes on the path from the ith leaf to the root, for a total of $O(\log(N))$ space. Each of $O(N)$ nodes performs $O(1)$ work, for a total of $O(N)$ time. The top B_1 wavelet coefficients can be collected from the tree's output stream in time $O(N)$. It follows that the above algorithm, on time-series data, takes total time $c_1 N + (B\log(N)/\epsilon)^{c_2}$ and space $(B\log(N)/\epsilon)^{c_3}$, for constants c_1,c_2, and c_3. Using suitable buffering (without significantly increasing the space requirements), the per-item time is $O(1)$.

In the case of dynamic data, updates to $\vec{A}$ positions are transformed into updates to wavelet coefficients of $\vec{A}$ and the large wavelet coefficients are tracked using a sketch-based structure. An update (i,u), with semantics "add u to $\vec{A}[i]$," is regarded for analysis as the vector $u\delta_i$ that takes the value u at position i and zero elsewhere. Using the tree structure of wavelets, it can be shown that, for all i, $\delta_i = \sum_{\ell \in \Lambda_i} \langle \delta_i, \psi_\ell \rangle \psi_\ell$, where Λ_i has size $O(\log(N))$ and can be constructed quickly from i. Given update (u,i) to $\vec{A}$, the algorithm quickly forms updates $(u\langle \delta_i, \psi_\ell \rangle, \ell)$ to the vector $\hat{\vec{A}}$ of wavelet coefficients for each $\ell \in \Lambda_i$. This algorithm uses a synopsis data structure, called an "L^2 count sketch," parametrized by N and η, that efficiently supports the following operations on an unordered set S of size N:

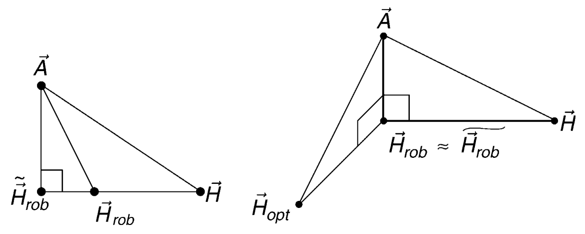
1. UPDATES of the form "add x to a count $\mathbf{C}[i]$ for item $i \in S$."
2. Unparametrized "FIND" queries. The answer is a list containing all j where $C[j]^2 \geq \eta \Sigma_i C[i]^2$.
3. "COUNT" queries about item i, for which the answer is $\tilde{C}[i]$ such that $\left|\tilde{C}[i] - C[i]\right|^2 \leq \eta \|C\|^2$.

There are randomized implementations of L^2 count sketches [1,2] that need space $(\log(N)/\eta)^{O(1)}$ and time $(\log(N)/\eta)^{O(1)}$ for either query. Such an L^2 count sketch with appropriate η at least $(B\log(N)/\varepsilon)^{-O(1)}$ suffices to track the large wavelet terms needed for our algorithm. With appropriate implementations, the algorithm will take space and per-item time $(B\log(N)/\varepsilon)^{O(1)}$, including time to build the histogram completely after each update. (The algorithm depends on a randomized data structure. There is also a cost factor of $O(log(1/\delta))$ to achieve success probability $1 - \delta$.)

Correctness of the algorithm depends on which of the two conditions is in force when constructing $\vec{H}_{\text{rob}}$. If $\vec{H}_{\text{rob}} = \vec{R}$, then one can show that the wavelet coefficient magnitudes of $\vec{A}$ decay essentially exponentially. It follows that $\vec{H}_{\text{rob}}$ is a very good approximation to $\vec{A}$, say, $\|\vec{H}_{rob} - A\| \leq \frac{\epsilon}{2}\|\vec{H}_{opt} - A\|$, where $\vec{H}_{\text{opt}}$ is the optimal B-bucket histogram for $\vec{A}$. In that case, since $\vec{H}$ is the optimal B-bucket histogram for $\vec{H}_{\text{rob}}$,

$$\begin{aligned}\|\vec{H} - \vec{A}\| &\leq \|\vec{H} - \vec{H}_{rob}\| + \|\vec{H}_{rob} - \vec{A}\| \\ &\leq \|\vec{H}_{opt} - \vec{H}_{rob}\| + \|\vec{H}_{rob} - \vec{A}\| \\ &\leq \|\vec{H}_{opt} - \vec{A}\| + 2\|\vec{H}_{rob} - \vec{A}\| \\ &\leq (1+\epsilon)\|\vec{H}_{opt} - \vec{A}\|.\end{aligned}$$

Otherwise, if $\vec{H}_{\text{rob}} \neq \vec{R}$, it follows that $\vec{H}_{\text{rob}}$ can not be much improved as an approximation to $\vec{R}$ whence it cannot be much improved as an approximation to $\vec{A}$ by refinements of $B - 1$ more bucket boundaries and bucket heights that are optimal or close enough to optimal so that the effect of approximation of bucket heights can be ignored. In particular, the best linear combination of $\vec{H}_{\text{rob}}$ and any B-bucket histogram is a refinement of $\vec{H}_{\text{rob}}$ by at most $B - 1$ boundaries and so is not much of an improvement over $\vec{H}_{\text{rob}}$. Thus, as in Fig. 3, there are two (near-) right angles at $\vec{H}_{\text{rob}}$ and so $\|\vec{H} - \vec{A}\|$ is (nearly) no worse than $\|\vec{H}_{\text{opt}} - \vec{A}\|$, i.e., $\|\vec{H} - \vec{A}\| \leq (1+\in)\|\vec{H}_{\text{opt}} - \vec{A}\|$, as desired.



Histograms on Streams. Figure 3. Illustration of histograms in correctness discussion. On the left, $\tilde{\vec{H}}_{\text{rob}}$ is the best histogram for $\vec{A}$ in the linear span of $\vec{H}_{\text{rob}}$ and $\vec{H}$; there is a right angle as indicated. On the right, since $\vec{H}_{\text{rob}} \approx \vec{H}_{\text{rob}}$, there are two near right angles at $\vec{H}_{\text{rob}}$. Since $\vec{H}$ is no farther from $\vec{H}_{\text{rob}}$ than $\vec{H}_{\text{opt}}$ is, it follows that $\vec{H}$ is almost as close to $\vec{A}$ as $\vec{H}_{\text{opt}}$ is. (Because the angles at $\vec{H}_{\text{rob}}$ are not quite right angles, it does not follow that $\|\vec{A} - \vec{H}\| \leq \|\vec{A} - \vec{H}_{\text{opt}}\|$, only that $\|\vec{A} - \vec{H}\| \leq (1+\in)\|\vec{A} - \vec{H}_{\text{opt}}\|$.)

Key Applications

Histograms on streaming data can be used for selectivity estimation as part of query execution optimization. They can also be used at the user level, e.g., for visualization of data. Many of the techniques developed for stream processing also support the processing of distributed databases.

Cross-references

▶ Heavy Hitters
▶ Sketch
▶ Wavelets on Streams

Recommended Reading

1. Cormode G. and Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications. In Proc. 6th Latin American Symp. Theoretical Informatics, 2004, pp. 29–38.
2. Gilbert A., Guha S., Indyk P., Kotidis Y., Muthukrishnan S., and Strauss M. Fast, small-space algorithms for approximate histogram maintenance. In Proc. 34th Annual ACM Symp. on Theory of Computing, 2002, pp. 389–398.
3. Guha S., Koudas N., and Shim K. Approximation and streaming algorithms for histogram construction problems. ACM Trans. Database Sys., 31(1):396–438, March 2006.
4. Ioannidis Y. The history of histograms (abridged). In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 19–30.
5. Jagadish H., Koudas N., Muthukrishnan S., Poosala V., Sevcik K., and Suel T. Optimal histograms with quality guarantees. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 275–286.
6. Muthukrishnan S. and Strauss M. Approximate histogram and wavelet summaries of streaming data. In Data-Stream Management – Processing High-Speed Data Streams. Springer, New York (Data-Centric Systems and Applications Series), 2009.

Historical Algebras

▶ Temporal Algebras

Historical Data Model

▶ Temporal Data Models

Historical Data Models

▶ Temporal Logical Models

Historical Database

▶ Temporal Database

Historical Query Languages

▶ Abstract Versus Concrete Temporal Query Languages
▶ Temporal Query Languages

Historical Spatio-Temporal Access Methods

▶ Indexing Historical Spatio-Temporal Data

History

▶ Provenance
▶ Provenance in Scientific Databases

History in Temporal Databases

Christian S. Jensen[1], Richard Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Synonyms

Time sequence; Time series; Temporal value; Temporal evolution

Definition

The *history* of an "object" of the real world or of a database is the temporal representation of that object. Depending on the specific object, one can have *attribute histories, entity histories, relationship histories, schema histories, transaction histories*, etc.

Key Points

"History" is a general concept, intended in the sense of "train of events connected with a person or thing."

In the realm of temporal databases, the concept of history is intended to include multiple time dimensions as well as multiple data models. Thus one can have, e.g., valid-time histories, transaction-time histories, and bitemporal histories. Also multi-dimensional histories can be defined from mono-dimensional ones (e.g., a bitemporal history can be seen as the transaction-time history of a valid-time history).

The term "history," defined formally or informally, has been used in many temporal database papers, also to explain other terms. For instance, salary history, object history, and transaction history are all expressions used in this respect.

Although "history" usually has to do with past events, its use for the future—as introduced by, e.g., prophecies, science fiction, and scientific forecasts—does not seem to present comprehension difficulties (The adjective "historical" seems more problematic for some). Talking about future history requires the same extension of meaning as required by talking about future data.

The synonym "temporal value" appears less general than "history," since it applies when "history" specializes into attribute history (value history), and it suggests a single value rather than a succession of values across time. The concept of a history is a slightly more general than the concept of a time sequence. Therefore the definition of "history" does not prevent defining "time sequence."

Since "history" in itself implies the idea of time, the use of "history" does not require further qualifications as is needed in the case of "sequence" or "series." In particular, "history" lends itself well to be used as modifier, even though "time sequence" is an alternative consolidated term.

Cross-references

▶ Bitemporal Relation
▶ Event
▶ Temporal Database
▶ Temporal Data Models
▶ Temporal Element
▶ Transaction Time
▶ User-Defined Time
▶ Valid Time

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, 1998, pp. 367–405.

Homogeneous Distributed Database Systems

▶ Distributed Database Systems

Homogeneously Distributed Data

▶ Horizontally Partitioned Data

Homomorphic Encryption

Ninghui Li
Purdue University, West Lafayette, IN, USA

Definition

Homomorphic encryption is a form of public-key encryption where one can perform a specific algebraic operation on the plaintext by performing a (possibly different) algebraic operation on the ciphertext. For example, knowing only $C_1 = \varepsilon(x)$ and $C_2 = \varepsilon(y)$, but not x, y, one may be able to get $\varepsilon(x + y)$ by calculating $C_1 \times C_2$. A homomorphic encryption scheme allows a third party to operate on encrypted values without knowing the plaintext.

Key Points

Depending on the intended application, the homomorphic property can be seen as a positive or negative attribute of the cryptosystem. Homomorphic encryption schemes are malleable by design. On the other hand, homomorphic encryption algorithms have found applications in a number of cryptographic problems, e.g., for secure voting and private information retrieval. Some encryption schemes that have the homomorphic property are given below.

RSA

If the public key is n, e, then the encryption of a message x is given by $\varepsilon(x) = x^e \bmod n$. The homomorphic property is property is as follows.

$$\begin{aligned}\varepsilon(x_1)\cdot\varepsilon(x_2) &= x_1^e x_2^e \bmod n\\ &= (x_1x_2)^e \bmod n\\ &= \varepsilon(x_1\cdot x_2 \bmod n)\end{aligned}$$

El Gamal

If the public key is p, g, $h = g^a$, and a is the secret key, then the encryption of a message x is $\varepsilon(x) = (g^r, x\cdot h^r)$. The homomorphic property is property is as follows.

$$\begin{aligned}\varepsilon(x_1)\cdot\varepsilon(x_2) &= (g^{r_1}, x_1\cdot h^{r_1})(g^{r_2}, x_2\cdot h^{r_2})\\ &= (g^{r_1+r_2}, (x_1\cdot x_2)h^{r_1+r_2})\\ &= \varepsilon(x_1\cdot x_2 \bmod p)\end{aligned}$$

Goldwasser-Micali

If the public key is the modulus n, and x is a quadratic non-residue modulo n, then the encryption of a bit b is $\varepsilon(b) = r^2x^b \bmod n$. The homomorphic property is property is as follows.

$$\begin{aligned}\varepsilon(b_1)\cdot\varepsilon(b_2) &= r_1^2x^{b_1}r_2^2x^{b_2} = (r_1r_2)^2x^{b_1+b_2}\\ &= \varepsilon(b_1\oplus b_2)\end{aligned}$$

where $\bigoplus$ denotes addition modulo 2 (i.e., exclusive-or).

Paillier

If the public key is the modulus n and the base g, then the encryption of a message x is $\varepsilon(x) = g^x r^n \bmod n^2$. The homomorphic property is property is as follows.

$$\begin{aligned}\varepsilon(x_1)\cdot\varepsilon(x_2) &= (g^{x_1}r_1^n)(g^{x_2}r_2^n) = g^{x_1+x_2}(r_1r_2)^n\\ &= \varepsilon(x_1 + x_2 \bmod n)\end{aligned}$$

Cross-references

▶ Asymmetric Encryption
▶ Data Encryption

Recommended Reading

1. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In Advances in Cryptology: EUROCRYPT '99. LNCS vol.1592, Springer, 1999, pp. 223–238.

Horizontal Fragmentation

▶ Distributed Database Design

Horizontally Partitioned Data

Murat Kantarcioglu
University of Texas at Dallas, Dallas, TX, USA

Synonyms

Homogeneously distributed data

Definition

Data is said to be horizontally partitioned when several organizations own the same set of attributes for different sets of entities. More formally, horizontal partitioning of data can be defined as follows: given a dataset $DB = (E, I)$ (e.g., hospital discharge data for state of Texas) where E is the set of entities about whom the information is collected (e.g., the set of patients) and I is the set of attributes that is collected about entities (e.g., set of features collected about patients), DB is said to be horizontally partitioned among k sites where each site owns $DB_i = (E_i, I_i)$, $1 \leq i \leq k$ if $E = E_1 \cup E_2 ... \cup E_k$, $E_i \cap E_j = \emptyset$, $1 \leq i \neq j \leq \mathbf{k}$ and $I = I_1 = I_2 ... = I_n$. In relational terms, with horizontal partitioning, the relation to be mined is the union of the relations at the sites.

Historical Background

Cheap data storage and abundant network capacity have revolutionized data collection and data dissemination. At the same time, advances in data mining have made it possible to learn previously unknown and interesting knowledge from the collected data [4]. These developments have caused serious concerns related to privacy implications of data mining. Especially, privacy issues related to sharing data for data mining raised significant concerns [2]. To address these privacy issues for the horizontally partitioned data case, two different solution ideas were initially proposed. In [1], perturbation-based techniques were proposed to build decision trees in a privacy preserving way. In perturbation based techniques data noise is added to the data before it is revealed to the data miner to protect individual values. In [10], the secure multi-party computation based approach was proposed to build decision trees. The definition of privacy followed in this line of research is conceptually simple: anything learned during the data mining process must be derivable given one's own data and the final result. Following these initial works, many different privacy-preserving data mining algorithms have been proposed including association rule mining [10], naive bayes classification [6], k-NN Classification [7], support vector machine classification [11], and k-means and EM clustering [5,9].

Foundations

Most of the work in privacy preserving data mining on horizontally partitioned data deals with the privacy issues that arise in data collection and data sharing for data mining. Due to privacy concerns, individuals and organizations may not be willing to share their data. For example, individuals may not be willing to tell their incomes or companies may not be willing to reveal statistics related to their core businesses. Fortunately, in many cases, the information aggregated over many individuals may not be considered a privacy problem, and it can be assumed that the data mining result itself is not a privacy violation. Therefore the main goal is to learn the data mining results by disclosing as little as possible about the data sources. Two different solution ideas have been suggested for addressing this privacy challenge. One idea is to apply data perturbation techniques to privacy-sensitive data before data mining (e.g., [1]). The other solution idea is to use cryptographic techniques to learn only the final data mining result without revealing anything under some cryptographic assumptions (e.g., [10]).

Key Applications

Since the horizontally partitioned data model assumes that different sites collect the same set of information about different entities, the solution techniques developed could be applied for any application that satisfies this assumption. For example, it could be applied for building fraud detection models using the data that is collected by different credit card companies related to the credit card transactions of different individuals.

The health-care industry provides one of the key application areas for the methods developed for horizontally partitioned data. For example, suppose the Centers for Disease Control (CDC), a public agency, would like to mine health records to try to find ways to reduce the proliferation of antibiotic resistant bacteria. Insurance companies have data on patient diseases and prescriptions. CDC may try to mine association rules of the form $X \Rightarrow Y$ such that the $Pr(X\&Y)$ and $Pr(Y|X)$ are above thresholds. Mining this data for association rules would allow the discovery of rules such as *Augmentin&Summer* $\Rightarrow$ *Infection&Fall*, i.e., people taking Augmentin in the summer seem to have recurring infections.

The problem is that insurance companies will be concerned about sharing this data. Not only must the privacy of patient records be maintained, but insurers will be unwilling to release rules pertaining only to them. Imagine a rule indicating a high rate of complications with a particular medical procedure. If this rule doesn't hold globally, the insurer would like to know this; they can then try to pinpoint the problem with their policies and improve patient care. If the fact that the insurer's data supports this rule is revealed (say, under a Freedom of Information Act request to the CDC), the insurer could be exposed to significant public relations or liability problems. This potential risk could exceed their own perception of the benefit of participating in the CDC study. In many such cases, privacy preserving distributed data mining techniques could be used to learn the final data mining result without revealing anything other than the result itself.

Future Directions

Although the provably secure distributed data mining protocols that reveal nothing but the resulting data mining model. This work still leaves a privacy question open: Do the resulting data mining models inherently violate privacy? This question is important because the full impact of privacy-preserving data mining will only be realized when there are guarantees that the resulting models do not violate privacy as well. The long list of possible privacy violations due to data mining results given in [8] indicates that care must be taken in revealing data mining results. Recently, in [3], the authors gave a new decision tree learning algorithm which guarantees that the data mining result does not violate the k-anonymity of the individuals represented in the training data. Although current work in this area has made valuable contributions, more research is needed to understand the privacy implications of data mining results.

Cross-references

▶ Privacy-Preserving Data Mining
▶ Vertically Partitioned Data

Recommended Reading

1. Agrawal R. and Srikant R. Privacy-preserving data mining. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 439–450.
2. Clifton C. and Marks D. Security and privacy implications of data mining. In Proc. Workshop on Data Mining and Knowledge Discovery, 1996, pp. 15–19.
3. Friedman A., Wolff R., and Schuster A. Providing k-anonymity in data mining. VLDB J., 17(4):789–804, 2008.
4. Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco, California, April 2000.
5. Jagannathan G. and Wright R.N. Privacy-preserving distributed *k*-means clustering over arbitrarily partitioned data. In Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2005, pp. 593–599.
6. Kantarcioglu M. and Vaidya J. Privacy preserving naive bayes classifier for horizontally partitioned data. In Proc. Workshop on Privacy Preserving Data Mining, 2003.
7. Kantarcıoğlu M. and Clifton C. Privately computing a distributed *k*-nn classifier. In Principles of Data Mining and Knowledge Discovery, 8th European Conf, 2004, pp. 279–290.
8. Kantarcıoğlu M., Jin J., and Clifton C. When do data mining results violate privacy? In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 599–604.
9. Lin X., Clifton C., and Zhu M. Privacy preserving clustering with distributed EM mixture modeling. Knowl. Inform. Syst., 8(1):68–81, July 2005.
10. Lindell Y. and Pinkas B. Privacy preserving data mining. In Advances in cryptology – CRYPTO 2000. Springer, Berlin, 2000, pp. 36–54.
11. Yu H., Jiang X., and Vaidya J. Privacy-preserving svm using nonlinear kernels on horizontally partitioned data. In Proc. 2006 ACM Symp. on Applied Computing, 2006, pp. 603–610.

Horn Clause Query

▶ Conjunctive Query

Hot Items

▶ Frequent Items on Streams

Hotspots

▶ Spatial Data Mining

HSM

▶ Storage of Large Scale Multidimensional Data

HTML Fragment

▶ Web Views

Human Centered 38 H Human-Computer Interaction Computing

▶ Human-Computer Interaction

Human Factors

▶ Human-Computer Interaction

Human Interface

▶ Human-Computer Interaction

Human-centered Computing: Application to Multimedia

Nicu Sebe[1,2], Alejandro Jaimes[3]
[1]University of Amsterdam, The Netherlands
[2]University of Trento, Trento, Italy
[3]Telefonica R&D, Madrid, Spain

Synonyms

HCC; Human-centered multimedia; HCM

Definition

Human-centered computing (HCC) is an emerging, interdisciplinary academic field broadly concerned with computing and computational artifacts as they relate to the human condition.

HCC consists of a set of methodologies for designing, implementing, and deploying computing in any field that uses computing in any form. The definition of HCC is purposely broad, and this reflects the view that computing plays an active and crucial role in a huge number of human activities.

One important area of HCC is concerned with the design, implementation, and deployment of computing systems or devices *in everyday environments*. In such cases, it is desirable for HCC approaches to:

- Integrate input from different types of sensors and communicate through a combination of media as output.
- Act according to relevant social and cultural contexts.
- Be useful to diverse individuals.

In general, HCC methodologies guide the design of effective computer systems taking into account personal, social, and cultural aspects. The field of study, therefore, includes topics such as information design, human-information interaction, human-computer interaction, and human-human interaction. Additionally, it includes topics that are not directly related to interaction (algorithms, data collection and analysis, system design, etc.). Of particular interest in HCC is a critical analysis of the relationships between computing technology, society, and culture – so works in fields such as new media art also form part of HCC.

The distinction between computing and multimedia computing is blurry but in spite of this, the main human-centered activities in multimedia can be identified as follows [6]: media production, annotation, organization, archival, retrieval, sharing, analysis, and communication. The above activities can in turn be clustered into three large activity areas: production, analysis, and interaction.

Historical Background

Discussions about HCC have taken place over several years [5,7,16,20]. HCC draws from concepts and research in the humanities and social sciences (cognitive psychology, sociology, anthropology, philosophy, information sciences, and several others), as well as from established and emerging technical and interdisciplinary fields (HCI, computer-supported cooperative work (CSCW), user-centered design (UCD) [11], ubiquitous computing, artificial intelligence, new media art, etc.).

Although it is difficult to quantify the contributions of various fields to HCC, it is generally acknowledged that researchers and practitioners in HCI, UCD, and CSCW have made many important contributions to HCC.

User-centered design can be characterized as a multistage problem-solving process that requires designers not only to analyze and foresee how users are likely to use an interface but also to test the validity of their assumptions with regard to user behavior in the real world [15]. CSCW, on the other hand, combines the understanding of the way people work in groups with the enabling technologies of computer networking and associated hardware, software, services, and techniques [4].

While many of the topics covered by UCD and CSCW are relevant to HCC, the scope of Human-centered computing is much broader, covering more than the traditional areas of usability engineering, HCI, and human factors, which are primarily concerned with user interfaces or user interaction [8]. Compared to HCI, HCC has a twofold perspective [20]:

- HCC is "conceived as a theme that is important for all computer-related research, not as a field that overlaps or is a subdiscipline of computer science."
- HCC acknowledges that "computing connotes both concrete technologies (that facilitate various tasks) and a major social and economic force."

HCC is not to be confused with *human-based computation* [3], a technique also known as interactive evolutionary computation first developed in the early 1990s which puts the human at the center, but in a different way. In traditional computation, a human provides a formalized problem description to a computer and receives a solution to interpret. In human computation, the roles are reversed: the computer asks a person or a large number of people to solve a problem, then collects, interprets, and integrates their solutions. In other words, the computer "asks" the human to do the work it cannot do. This is done as a way to overcome technical difficulties: instead of trying to get computers to solve problems that are too complex, use human beings. In human computation the human helps the computer solve difficult problems. In HCC, the goal is to consider human abilities in creating, developing, and deploying systems or devices that use computing.

Although HCC and human-based computation approach computing from two different perspectives, they both try to maximize the synergy between human abilities and computing resources. Work in human computation can therefore be of significant importance to HCC. On one hand, data collected through human computation systems can be valuable for developing machine-learning models. On the other hand, it can help in better understanding of human behavior and abilities, again of direct use in HCC algorithm development and system design.

Foundations

A human-centered approach to multimedia starts with a deep understanding of human abilities and behavior in particular socio-cultural contexts. When human interaction is part of the computing process, HCM methodologies must consider how human beings understand and interpret multimedia signals at the perceptual, cognitive, and affective levels, and how they interact naturally – embedding the cultural and social contexts as well as personal factors such as emotion, attitude, and attention. The human-centered systems that humans directly interact with should be multimodal, processing inputs and outputs in a naturally rich communication channel; be proactive – understanding cultural and social contexts and responding accordingly; and be easily accessible outside the desktop to a wide range of users [6]. Because of this, HCM must consider work in fields such as neuroscience, psychology, cognitive science, and others. One of the key challenges, therefore, is incorporating what is known in those fields within computational frameworks that integrate different media.

Research on machine learning integrated with domain knowledge, data mining, sensor fusion, and multimodal interaction will play a key role [18]. Further research into quantifying human-related knowledge is necessary, which means developing new theories and mathematical models of multimedia integration at multiple levels.

Although many disciplines contribute to HCM (and implicitly HCC), the field is at an early stage so a research agenda will involve a non-exhaustive list of goals including the following:

- New human-centered methodologies for the design of models and algorithms and the development of diverse human-centered systems
- Focused research on the integration of multiple sensors, media, and human sciences that have people as the central point
- New interdisciplinary academic and industrial programs, initiatives, and meeting opportunities

- Discussions on the impact of computing technology that include the social, economic, and cultural contexts in which such technology is or might be deployed
- Research data that reflects human-centered approaches – for example, rich data collected from real multisensorial and culturally diverse social situations
- Common computing resources – for example, software tools and platforms
- Evaluation metrics for theories, design processes, implementations, and systems from a human-centered perspective
- Methodologies for privacy protection and the consideration of ethical and cultural issues

Clearly, an HCM and HCC research agenda should include a broad understanding and a multidisciplinary approach. This is of particular importance when considering computing in the context of developing regions [2].

Key Applications

The span of HCM application areas is very broad and as computing becomes more ubiquitous, practically every aspect of interaction with objects, and the environment, as well as human-human interaction (e.g., remote collaboration, etc.) will make use of HCM techniques. Several key application area descriptions follow:

Ambient Intelligence and Personal Spaces

Computing is being integrated with everyday objects in a variety of domains. This implies that the model of "user" in which a person sits in front of a "computer" is no longer the only model. Therefore, the actions or events to be recognized by the "interface" are not necessarily explicit commands and must respond to personal context in homes, offices, and other spaces, making ambient intelligence [1] a very important application area of HCM.

Public information kiosk applications [10] present many technical challenges in the design and implementation of natural multimodal interaction: kiosks are often intended to be used by a wide audience, thus there may be few assumptions about the types of users of the system. Thus, they present interesting opportunities for HCM approaches and must integrate context and socio-cultural considerations.

HCM also plays an important role in personal spaces (e.g., cars, office spaces, etc.). On one hand HCM techniques emphasize personalization, and on the other hand they emphasize natural interaction. Computing in vehicles, for example, can be used to monitor the driver [9] and to adjust conditions according to personal preferences. In addition, since the driver must focus on the driving task, traditional interfaces (e.g., GUIs) are not suitable, creating the need for natural, non-disruptive interaction techniques focused on maximizing human abilities while answering human needs in specific situations.

Ubiquitous Computing

One of the major challenges in ubiquitous computing is that while devices such as PDAs and mobile phones have become smaller and more powerful, there has been little progress in developing effective interfaces to access the increased computational and media resources they posses. Computing in ubiquitous devices constitute a very important area of opportunity for research in HCM: they create challenges in interaction for creation and access to content, as well as opportunities for contextual data collection and personalization.

Data Analysis and Data Interaction

HCM approaches can be used to process, analyze, and visualize various types of human-related data. The range of personal information collected in digital format is very wide, and can be used to support many human activities. Social networks, for example, have gained significant popularity and are based largely on connections of data about different individuals. HCM approaches to analyze and interact with data and information are crucial in such types of applications. Web search and recommender systems are two additional areas in which HCM techniques play a significant role because they can strongly impact not only the interaction aspects, but how the information is organized, indexed, and searched. Systems for analyzing, processing, and visualizing customer data for user modeling, profiling, and market segmentation can clearly have positive impacts in fields that use computing (e.g., marketing, advertising) to determine how and where products or services are provided.

Virtual Environments

Virtual and augmented reality have been very active research areas at the crossroads of computer graphics, computer vision, and human-computer interaction. One of the major difficulties of VR systems is the interaction component, and many researchers are currently exploring the use of interaction analysis techniques to enhance the user experience. One reason this is very attractive in VR environments is that it helps disambiguate communication between users and machines (in some cases virtual characters, the virtual environment, or even other users represented by virtual characters [14]). HCM techniques can be applied, however, to many aspects of VR environments: from the visual representation of characters and the environment, to the "physical" behavior of objects and how events and actions are perceived by human "users" of such systems.

Art

Perhaps one of the most exciting application areas of HCM is art, from interactive installation and interactive architecture to performance and audience participation. Multiple modalities (video, audio, pressure sensors) can be used to allow audience participation and influence a performance [19]. Mouth gestures can be interpreted [13] by a wearable camera pointing at the wearer's mouth to generate MIDI sounds (so a musician can play other instruments while generating sounds by moving his mouth). HCM technologies can also be used in museums to augment exhibitions [17] and in many other contexts.

Persons with Disabilities

Persons with disabilities can benefit greatly from HCM techniques [12] because the goal is specifically to enhance or complement person's abilities. Solutions for smart wheel-chair systems, for example, integrate different types of sensors and consider different kinds of interaction such as using only eye blinks and eyebrow movements for navigation.

Other Applications

Other applications include education, medicine, remote collaboration, entertainment, robotics, surveillance, or biometrics. HCM also plays an important role in safety-critical applications (e.g., medicine, military, etc.) and in situations in which a lot of information from multiple sources has to be viewed in short periods of time (e.g., crisis management).

Cross-references

▶ Human-Computer Interaction
▶ Multimedia Data
▶ Multimodal Interfaces
▶ Personalized Web Search
▶ Social Networks

Recommended Reading

1. Arts E. Ambient intelligence: a multimedia perspective. IEEE Multimedia, 11(1):12–19, 2004.
2. Brewer E. et al. The case for technology in developing regions. Computer, 25–38, June 2005.
3. Gentry C., Ramzan Z., and Stubblebine S. Secure distributed human computation. In Proc. 6th ACM Conf. on Electronic Commerce, 2005, pp. 155–164.
4. Grudin J. Computer-supported cooperative work: Its history and participation. Computer, 27:19–26, 1994.
5. http://www.human-centered-computing.org/
6. Jaimes A. Human-centered multimedia: culture, deployment, and access. IEEE Multimedia, 13(1):12–19, 2006.
7. Jaimes A., Gatica-Perez D., Sebe N., and Huang T.S. Human-centered computing: toward a human revolution. Computer, 30–34, May 2007.
8. Jaimes A., Sebe N., and Gatica-Perez D. Human-centered computing: a multimedia perspective. In Proc. 14th ACM Int. Conf. on Multimedia, 2006, pp. 855–864.
9. Ji Q. and Yang X. Real-time eye, gaze, and face pose tracking for monitoring driver vigilance. Real-Time Imaging, 8:357–377, 2002.
10. Johnston M. and Bangalore S. Multimodal Applications from Mobile to Kiosk. In Proc. W3C Workshop on Multimodal Interaction, 2004.
11. Karat J. and Karat C.M. The evolution of User-centered focus in the Human-computer interaction field. IBM Syst. J., 42 (4):532–541, 2003.
12. Kuno Y., Shimada N., and Shirai Y. Look where you're going: a robotic wheelchair based on the integration of human and environmental observations. IEEE Robotics and Automation, 10(1):26–34, 2003.
13. Lyons M.J., Haehnel M., and Tetsutani N. Designing, playing, and performing, with a vision-based mouth Interface. In Proc. Conf. on New Interfaces for Musical Expression, 2003.
14. Nijholt A. and Heylen D. Multimodal communication in inhabited virtual environments. Int. J. of Speech Technol., 5:343–354, 2002.
15. Norman D.A. and Draper S.W. User-centered system design: New perspectives on Human-computer interaction. Lawrence Erlbaum, Hillsdale, NJ, 1986.
16. NSF Workshop On Human-Centered Systems: Information, Interactivity, and Intelligence (HCS) February 17–19, 1997 (http://www.ifp.uiuc.edu/nsfhcs/)
17. Paradiso J. and Sparacino F. Optical tracking for music and dance performance. In Proc. 4th Conf. on Optical 3-D Measurement Techniques IV, 1997, pp. 11–18.
18. Pentland A. Socially aware computation and communication. Computer, 38(3):33–40, 2005.
19. Wassermann K.C., Eng K., Verschure P.F.M.J., and Manzolli J. Live soundscape composition based on synthetic emotions. IEEE Multimedia Mag., 10(4), 2003.
20. www.cs.berkeley.edu/~jfc/hcc

Human-Centered Multimedia

▶ Human-centered Computing: Application to Multimedia

Human-Computer Interaction

Alan Dix
Lancaster University, Lancaster, UK

Synonyms

HCI; Computer human interaction (CHI); Human interface; Interaction design; Human centered computing; User-centred design; Human factors; Man-machine interaction (obsolete)

Definition

Human–Computer Interaction (HCI) is the study of the way in which computer technology influences human work and activities. The term "computer technology" now-a-days includes most technology from obvious computers with screens and keyboards to mobile phones, household appliances, in-car navigation systems and even embedded sensors and actuators such as automatic lighting. HCI has an associated design discipline, sometimes called *Interaction Design* or *User-Centered Design*, focused on how to design computer technology so that it is as easy and pleasant to use as possible. A key aspect of the design discipline is the notion of "usability," which is often defined in terms of efficiency, effectiveness and satisfaction. However, equally or more important in systems designed for personal use, such as internet shopping, is the idea of user experience – the way people feel about the system as they use it.

Historical Background

The roots of HCI can be traced back to the innovative work of Douglas Englebart at the Augmented Research Center at Stanford in the early 1960s [4] and more pragmatic work of Brain Shackel in the late 1950s [8]. However, it was with the rise of personal computing in the early 1980s that the discipline took shape and got its name(s). Notable was the founding of IFIP Technical Committee 13 on Human-Computer Interaction in 1981 and the first international INTERACT Conference in 1984. Several major national conferences started around the same period ACM CHI in the US, the British HCI conference and Vienna HCI. At this stage the main academic roots were in psychology, computing and ergonomics with much of the early work founded on strong experimental methods.

As in so many areas of computing XEROX PARC played a key role in HCI, developing the WIMP (window-icon-menu-pointer) interface style that was eventually popularized in the Macintosh in 1984. In the second half of the 1980s a number of books were published that shaped the development of the field. In particular the *User-Centered System Design* collection in 1986 [6] set the ground for what would be the dominant view of HCI for many years with a strong focus on cognitive modeling and theoretical accounts of *direct manipulation*. As a contrast to this, Suchman's *Plans and Situated Actions* [12] drew on anthropological traditions and in particular *ethnography* and was formative in the fledgling area of *Computer Supported Collaborative Work* (*CSCW*).

Towards the end of the 1980s and early 1990s technological advances as much as methodological and theoretical factors had a large impact on HCI. The introduction of computer networks both within and between offices was one of the factors that fueled the growth of CSCW as it became possible to develop systems that enabled remote groups of workers to collaborate electronically. Increased graphics and computational capabilities also meant that for the first time it was possible to have high-quality *interactive* visualization of large data sets.

The importance of HCI as a discipline has grown gradually over the years as industry became aware of the importance of the usability of its products in a competitive environment. This has intensified since the late 1990s due to the world-wide web and consumer electronics such as mobile phones. As individual consumer choice took over from corporate IT decisions not only usability but also user experience have become key selling points.

With this growth the field has also spawned subfields such as web usability and mobile HCI, which can be confusing to someone new to the area. HCI has also drawn in fresh theoretical input, for example *Activity Theory* (see [1], chapter 11), as well as developing its own methods, for example ways of studying users in real-world contexts.

Foundations

As noted, HCI is both an academic discipline studying the way technology impacts human activity and also a design discipline aimed at designing that technology for maximum effectiveness. It draws on many results from other disciplines such as psychology and sociology, but also has its own methods and techniques.

Usability

Usability is one of the core issues in HCI. There are various definitions, most notably in ISO 9241–11 it is defined as: "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

The key terms in this definition are:

Effectiveness–Can the user achieve their goals with accuracy?
Efficiency –Can this be achieved with minimal resources: time, physical, and mental effort?
Satisfaction–Do users feel comfortable or happy in doing this?

For many years satisfaction has been largely ignored, but more recently affective issues such as motivation, trust, enjoyment, and engagement have become increasingly important.

The concept of usability is quite broad. At the lowest level there is the visual layout of information and controls on a screen or on a physical appliance and their immediate behavior. At a higher level one also has to take into account the whole social and organizational context: the people who will use the system being designed, their beliefs and values, the purpose and constraints of the design.

Observation and Empirical Data

Various techniques are used to observe or evaluate technology in use (see also [3] chapter 9 and [9] chapters 12–14). These techniques may be used purely for research, early in the design cycle in order to understand a situation, or later when an artifact has been reduced and one desires to evaluate it and identify any usability problems.

These evaluation or observation methods fall into two main types:

1. *Laboratory experiments* – where users perform a task or interact with an application under some level of controlled conditions
2. *Field studies* – where users are "in the wild"; that is in their workplace, outside, in their homes: wherever they would normally interact with the technology being studied or evaluated

Laboratory studies are often criticized for lack of *ecological validity*, that is that they are too far from the real context of use. To counter this laboratory studies are often performed in semi-realistic situations and a *usability laboratory* may include features such as a one-way mirror to enable unobtrusive observation.

Field studies may include a level of control, for example, giving two groups of people different versions of an interface, or may be more open, simply observing people doing their day-to-day work or activities.

Both types of empirical study can be performed with a closed agenda to evaluate or improve a proposed or existing design, but can also be used in a more open or exploratory way to explore a design idea or attempt to learn unexpected things about an area.

Design and Methodology

It often requires too much time or skill to apply fundamental knowledge to a new design. Instead good practice is often captured in design guidelines, principles or heuristics, for example providing immediate feedback for user actions. There are many collections of detailed guidelines for example the US Department of Health and Human Services *Research-Based Web Design and Usability Guidelines* runs to 18 chapters [13], but also there are shorter lists of more generic rules such as Ben Shneiderman's Eight Golden Rules of Interface Design [10]:

- Strive for consistency
- Enable frequent users to use shortcuts
- Offer informative feedback
- Design dialog to yield closure
- Offer simple error handling
- Permit easy reversal of actions
- Support internal locus of control
- Reduce short-term memory load

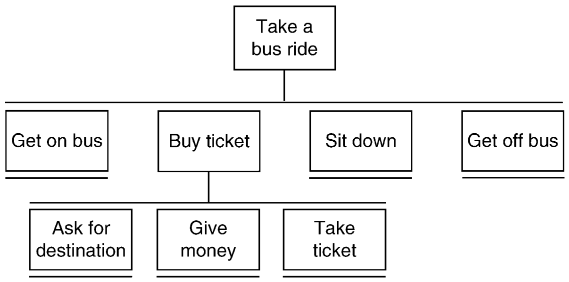
As important as design guidance are the design methods and processes. *User-centered design* refers to a number of practices that put the user first in thinking about the design of an application or product; this will typically involve some direct observation, interviewing of users or otherwise getting to know them. More radical are *participatory design* approaches where users are brought more deeply into the process acting as co-designers of the end product (see [7] Part VII).

Typically user-interface design processes are iterative involving some form of *prototype* or *storyboard* that enables users to see or experience early designs. This reflects the complexity of designing for real people and also the difficulty of even understanding what is wanted. Some level of iterative re-design is always required. However, this iterative design process typically needs a good start point so that more predictive and more iterative methods work together.

Representation and Analysis

Given the focus on users, HCI uses various forms of representation of users and their behavior. These may be informal such as personae or scenarios. A *persona* is a rich description of a real or fictitious user that can be used as a surrogate user in design. A *scenario* is a form of story describing a context in which a device or application is used and the way it gets used. These are both conveyed in text and pictures. More formal representations can also be used, for example, *hierarchical task analysis* (HTA, see [3] chapter 15) focuses on the decomposition of higher-level tasks (such as "take a bus ride" in Fig. 1) into lower evel tasks (such as "buy ticket").

Various forms of representations are also used to describe the behavior of interactive systems themselves; that is how the system responds to user inputs, the responses it produces, and the internal states insofar as they may affect future user interaction. This is often called a *dialogue specification*. These sometimes take the form of fairly informal diagrams, for example showing a collection of web pages and their



Human-Computer Interaction. Figure 1. Example of an hierarchical task analysis (HTA).

interconnections. However, more formal representations may be used, for example, using state-transition networks or Petri nets. The more formal representations can be analyzed to look for potential usability problems or even verified using forms of model checking to verify various forms of usability property. More commonly however, these are used to understand or communicate designs, or to specify them clearly for implementation.

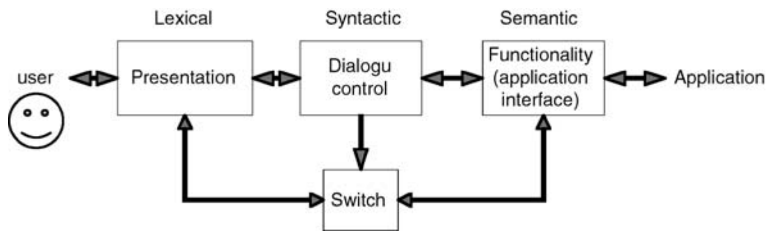
Implementation and User Interface Architecture

For the users of a system all that matters is its external behavior. However, in order to effectively construct such systems the internal structure or *software architecture* has to be in some way consonant with the external design (see also [3] chapter 8). The view that the user has of a system frequently cuts across what would be regarded as different functional areas, sometimes conflicting with what would otherwise be regarded as a suitable decomposition. It is also often desirable that more surface features of a user interface can be changed without a complete redesign of the entire system. Because of this, specific architectures have been developed that separate the parts that are closer to the user (e.g., choice of menu names or font color) from those about the underlying functionality. The most influential of these at a conceptual level and one of the earliest was the *Seeheim model*, which divided the system into presentation, dialogue and functionality (with some form of wrapper) (Fig. 2). At a practical level the *Model-View-Control* (MVC) model has been most widely adopted (for example in the *Java* Swing toolkit). This operates on the level of components or widgets (for example, the tree-view of a file system).

Key Applications

HCI is an issue whenever people use systems so the applications are very wide. Traditionally a substantial focus has been on office work where the desktop PC was the main computing device, although there has always been a strand focused on industrial interfaces and command and control situations (e.g., aircraft cockpits). As noted previously, domestic appliances and devices and systems for leisure and entertainment have become increasingly important.

Two of the biggest application areas over recent years have been in the web [2] and mobile computing [5]. However, these are really platforms and the range



Human-Computer Interaction. Figure 2. Seeheim model of user interface architecture.

applications supported on these have ranged from eCommerce to text messaging.

There has been some work focused specifically on user interfaces for databases to aid in areas such as query formulation or schema visualization. Related to this has been more recent work on usability issues for web search and user interfaces for various forms of ontologies and other semantic web structures. Applications in *information visualization* are also important allowing users to view structures including simple point data, hierarchies, trees, or geographic data [11].

Future Directions

The web will clearly be an important driver for some time to come. Whilst traditional web page usability has been well studied, the implications of *Web2.0* both as a social and a technological phenomenon are only just beginning to be understood. For example, users are increasingly making their own "mashups" and customizing shared web content, so that effectively they are taking on the role that used to be in the hands of developers – it is important that this does not become as confusing as cooking in someone else's kitchen! Developments in HCI aspects of mobile applications will continue to be important, especially as it is expected that mobile access to the Internet will outstrip desktop access in the next few years.

Looking slightly longer term, human aspects of *ubiquitous computing* are still only partially understood, but will be critical as homes, offices and public spaces become filled with sensors, points of interaction and numerous small and large screens. As these increasingly become wirelessly networked they will need to be designed in way that make their effects comprehensible and trustworthy.

In work environments technology is more stable and usability of desktop applications well understood. However, increasing availability of data and communications (e.g., email, instant messaging) is creating demands to manage large volumes of data. This may change the traditional desktop itself and is likely to involve more "intelligent" and often proactive applications, that will in turn need to be understood and controlled by their users.

Mobile and web applications are creating demands to re-target applications to multiple devices, and to share and mash-up data. For cost-effective construction and ease of maintenance it is likely that studies of user interface architecture, marginalized for a number of years, will become more important.

Web-based applications are leading to service-based solutions replacing many traditionally product-based ones. Greater choice and ease of movement between applications will demand changes in design practice away from "one size fits all" designs to more individual designs. This trend reflects similar diversification and shorter more dynamic product runs in traditional manufactured devices, which themselves are more likely to involve computation. This will clearly impact both design research and design practice.

Finally the demographics of computer and technology use is moving out from the relatively young and prosperous to include the increasingly older populations of developed countries, socially-deprived groups, and the growing markets of the developing world. While much of the understanding of human-computer interaction is universal, still many cultural and other assumptions are embedded in design practice and this will be an important issue for some years to come.

Cross-references

▶ Data Visualization
▶ Direct Manipulation
▶ GUIs for Web Data Extraction
▶ Human-Centered Computing: Application to Multimedia
▶ Icon
▶ Interface

▶ Mobile Interfaces
▶ Natural Interaction
▶ Usability
▶ Visual Interaction
▶ Visual Interfaces
▶ Visual Perception
▶ WIMP Interfaces

Recommended Reading

1. Carroll J. HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science. Morgan Kaufmann, San Francisco, CA, 2003.
2. Dix A. Human-computer interaction and web design, Chapter 3. In Handbook of Human Factors in Web Design, R.W, Proctor K.L. Vu (eds.). Lawrence Erlbaum, Mahwah, NJ, 2005, pp. 28–47.
3. Dix A., Finlay J., Abowd G., and Beale R. Human-Computer Interaction, 3rd edn. Prentice Hall, Upper Saddle River, NJ, 2004.
4. Engelbart D. Augmenting human intellect: a conceptual framework. Summary Report AFOSR-3223 under Contract AF 49 (638)-1024, SRI Project 3578 for Air Force Office of Scientific Research. Stanford Research Institute, Menlo Park, CA, 1962. Available online at: http://www.bootstrap.org/.
5. Jones M. and Marsden G. Mobile Interaction Design. Wiley, Chichester, UK, 2005.
6. Norman D.A. and Draper S.W. (eds.). User Centered System Design: New Perspectives on Human-Computer Interaction. Lawrence Erlbaum, Mahwah, NJ, 1986.
7. Sears A. and Jacko J. 1(eds). The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications, 2nd edn. CRC Press, West Palm Beach, FL, 2007.
8. Shackel B. Ergonomics for a Computer. Design, 120:36–39, 1959.
9. Sharp H., Rogers Y., and Preece J. Interaction Design: Beyond Human-Computer Interaction, 2nd edn. Wiley, Chichester, UK, 2007.
10. Shneiderman B. and Plaisant C. Designing the User Interface: Strategies for Effective Human-Computer Interaction, 4th edn. Addison-Wesley, Boston, MA, 2005.
11. Spence R. Information Visualization: Design for Interaction, 2nd edn. Prentice Hall, Uppersaddle River, NJ, 2007.
12. Suchman L. Plans and Situated Actions: The Problem of Human-Machine Communication. Cambridge University Press, New York, NY, 1987.
13. US Department of Health and Human Services. Research-Based Web Design and Usability Guidelines. Available online at: http://www.usability.gov/pdfs/guidelines.html (accessed on November 12, 2007).

Hypercube

▶ Cube

Hypermedia

▶ Hypertexts

Hypermedia Metadata

▶ Multimedia Metadata

Hypertexts

Frank Wm. Tompa
University of Waterloo, Waterloo, ON, Canada

Synonyms

Hypermedia

Definition

A hypertext is a collection of interconnected documents or document fragments. The idea of computer-based hypertexts is rooted in Vannevar Bush's vision, as described in his 1945 *Atlantic Monthly* article "As We May Think," of a personal document collection with "a provision whereby any item may be caused at will to select immediately and automatically another." Authors who wish to emphasize the multimedia nature of constituent documents and fragments prefer to use the term hypermedia when describing hypertexts.

Key Points

The term *hypertext* was introduced in the early 1960s by Ted Nelson, who advocated the power of non-linearity in organizing thoughts and discourses. Simultaneously, Doug Englebart demonstrated a system for "augmenting human intellect" that included a facility to expose inline fragments of text in response to users' request for finer detail. These early ideas have evolved to form the basis of HTML (the Hypertext Markup Language) and the World Wide Web.

Hypertexts have been used extensively as features of reference texts, collaborative design documents, and bibliographic systems, and as a structure for creative writing. Links between document fragments are used to represent various rhetorical relationships (support, rebuttal, elaboration, motivation, consequence, etc.).

Systems to support hypertexts include editors, browsers, and visualizers.

Cross-references

► Document Databases
► Digital Libraries
► Web Characteristics and Evolution

Recommended Reading

1. Ashman H. and Simpson R.M. Computing surveys' electronic symposium on hypertext and hypermedia: editorial. ACM Comput. Surv., 31(4):325–334, 1999.
2. Simpson R., Renear A., Mylonas E., and van Dam A. 50 years after "As we may think": the Brown/MIT Vannevar Bush symposium. Interactions, 3(2):47–67, 1996.

Hypothesis Generation and Exploration from Biological Resources

► Text Mining of Biological Resources