F

Faceted Browsing

► Faceted Search

Faceted Classifications

► Lightweight Ontologies

Faceted Search

Susan Dumais
Microsoft Research, Redmond, WA, USA

Synonyms

Dynamic taxonomies; Faceted browsing; Hierarchical faceted metadata

Definition

The term *facet* means "little face" and is often used to describe one side of a many-sided object, especially a cut gemstone. In the context of information science, where the item being described is an information object, facets could refer to the object's author, date, topic, etc. Facets are used to describe both the organization of information (faceted classification), and to interface techniques that provide flexible access to that information (faceted search). The motivation for faceted classification and search is that any single organizational structure is too limiting to accommodate access to complex domains. Multiple independent facets provide alternative ways of getting to the same information, thus supporting a wider range of end-user tasks and knowledge. The fields of faceted classification, information architecture, and data modeling provide theory and methods for identifying and organizing facets. The user interface challenge for faceted systems is in managing this added complexity, especially when working with very large and diverse information collections. Most interfaces to faceted information provide support for structured browsing (faceted navigation or browsing). In addition, some systems offer search capabilities and, more generally, tightly coupled views of the same information. This entry covers both the organization of information using facets, and the design of user interfaces to support searchers in accessing the information since the two aspects are closely related and should be considered together in designing information systems.

The term facet is widely used in the information science community. In other disciplines attribute, dimension, metadata, property, or taxonomy are used to refer to similar concepts. Faceted search is used in this entry to refer to flexible access to faceted information, using both browsing and search. Other terms such as hierarchical faceted metadata, faceted search and browsing, and dynamic taxonomies refer to similar concepts.

Historical Background

Shiyali Ramamrita Ranganathan, an Indian mathematician, first introduced the term "facet" in information science in the 1930s when he developed a theory of facet analysis, culminating with the publication of his book *Colon Classification* in 1933 [7]. The name comes from the use of colons to separate different facets in writing composite class numbers. Ranganathan applied the principles of faceted classification to organize all of human knowledge in libraries using five main facets – personality, matter or property, energy, space, and time. The Colon Classification system is still used in libraries in India, and the principles and techniques of facet analysis have been more widely influential as reviewed by Vickery [13]. Faceted classification is also called analytico-synthetic classification, after the two main processes involved – analysis which breaks down each subject into its basic concepts or facets, and synthesis which combines concepts

to describe the subset of information of interest. Thus, in faceted classification, new classes are created by a searcher from the logical combination of facets and facet values. This is often contrasted with enumerative classification in which all classes of interest are specified by the indexer at the time of creation. Most well-known library classification systems (Dewey Decimal Classification (DDC), Universal Decimal Classification (UDC) and Library of Congress Classification (LCC)) are enumerative systems, although DDC and UDC have elements of faceted classifications as well.

In the 1950s, Calvin Mooers (who coined the term *information retrieval*) and Mortimer Taube developed new methods for searching information. Before this time, most access was based on pre-coordinated index terms. Index terms were assigned to objects and only those specific index terms could be used to retrieve items. This is very much the way that back-of-the-book indices work, with the index creator anticipating the compound subjects that people will want to search for. In contrast, in Mooers' Zatocoding system [5], there was one punch card per item and many index terms coded by notch positions on the cards. At retrieval time, a searcher used needles to isolate items sharing a common term, and multiple terms could be combined using Boolean logic. Taube, who coined the term *coordinate indexing*, developed a system in which each card represented a term (the so-called uniterm system), along with a coding of the documents to which it referred [12]. Again, flexible post-coordination of terms at retrieval time was supported. Although the early coordinate indexing systems focused on index terms, the emphasis on analyzing items of information so that retrieval could be performed by the Boolean operations on simple index entries made these ideas widely applicable to online faceted systems. The first online retrieval systems appeared in the 1960s, many containing quite sophisticated retrieval functionality. The 1980s and 1990s saw the development of much richer systems that combined browsing and searching in more flexible ways to support non-professional end users in finding information. Some early systems showing key capabilities of faceted search include Pollitt [6] and Remde et al. [8]. As described in more detail below, many of the ideas explored in these systems are seen today in faceted search systems and on some web sites, notably e-commerce sites.

Foundations

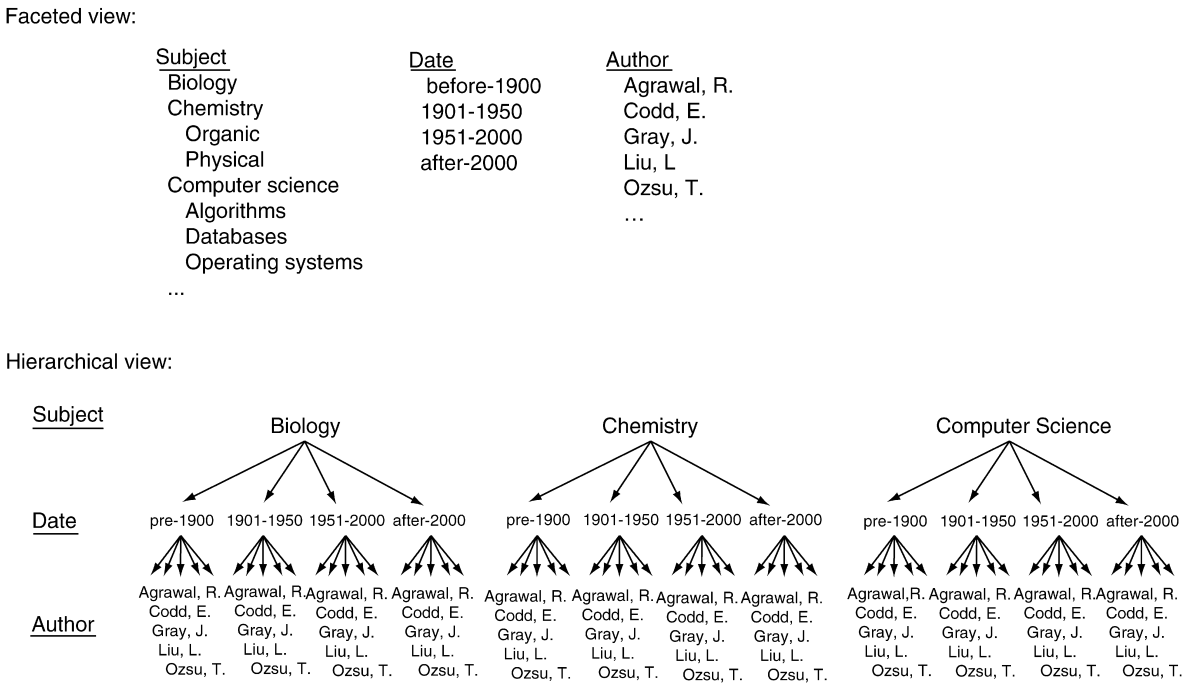
An Example

As a concrete example, consider organizing and providing access to a large collection of diverse technical publications. One way to do so is by general topic or subject area, much as the entries in this encyclopedia are organized. But there are many other facets in addition to Subject that can be used to characterize each publication – for example: Publication date, Author, Author's institution, Publication, Genre, Language, Length, Price, Tags, Download frequency, Citations by other publications, etc. In a faceted organization, each item is characterized by a value on each facet, and there are no explicit relationships among the facets. (There are a wide range of implicit relationships based on which items share facet values.) A single hierarchical structure can, of course, represent these same facets, with each level representing one of the facets. There are added storage costs since each facet needs to be replicated for every subcategory. More importantly, a single hierarchy imposes one fixed structure for navigation; each facet must be visited in the order provided by the hierarchy. Facets, on the other hand, provide many alternative ways of getting to the same information and can be combined in any order, thus providing much greater flexibility for searchers to find information. (See Fig. 1.)

Faceted Organization

Ranganathan pioneered principles and methods for identifying facets which Vickery and others extended and summarized. More recently the field of information architecture has developed these and related ideas in the context of large electronically available collections. Two important tasks are to identify the most appropriate facets for a collection and to assign a value to each item on each facet. Identifying facets is a challenging task and depends on the details of the collection, anticipated user tasks, etc. and will not be discussed in detail here. Rosenfeld and Morville [9] provide an overview of recent work on information architecture and data modeling.

The principles of faceted organization are widely applicable and flexible. Each individual *facet* can be organized in many ways, e.g., hierarchically, as a flat list, or using other relations. Each facet has a name (e.g., Subject) as well as values or labels (e.g., Biology, Chemistry, Computer Science). Each *item* in the collection is assigned one or more values on every facet.



Faceted Search. Figure 1. Comparison of faceted and hierarchical views.

For example, a technical publication will usually have only one Length or Date, but may have many Subjects, Authors or Tags. A probability or confidence can be associated with each value, as often happens when values are assigned algorithmically, although interfaces that expose this are rare. Any collection of items can be represented by a faceted organization – documents, web pages, images, videos, products, people, etc.

Once the facets for a collection have been determined, each item must be assigned values on every facet. In early library systems, all facets were assigned by trained human indexers, and many systems today still depend on human assignment of values for at least some of their facets. Machine learning techniques can also be used to provide facet values for items. Typically this involves a supervised learning phase in which items with known labels are used to build a predictive model, and an evaluation phase in which labels are assigned to new items; text classification systems are a widely-used example of this process. Sometimes values are captured implicitly via an application or use. For example, in email the sender and recipient are required for successful use of the application and are thus captured. Facets that reflect interaction with items (e.g., Download frequency, User-generated tags) are also captured implicitly as part of ongoing interactions with applications.

A key motivation for faceted systems is that any single organizational structure is too limiting to accommodate access to complex domains. Multiple facets provide many different ways of organizing information, thus enabling flexible end-user access to the information. Facets are a way of conveniently referring to a subset of items in the collection. Facets can be specified in any order, and the resulting sets can be combined using Boolean logic. The most common method for doing so (in interfaces) is to use AND to combine values from different facets and OR to combine values from within the same facet. Thus the resulting query is a conjunct of disjuncts over the selected facet labels. Other combination methods are possible, although there are interesting challenges in providing interfaces to do so. The ability to specify facets in any order and to combine them using Boolean logic illustrates the power and flexibility of faceted organization.

Compared to a single hierarchical organization of information, faceted systems provide many advantages. First, facets provide an efficient way of discriminating among sets of items. A single hierarchy with 100,000 nodes would be required to represent all the combinations that can be instantiated by 5 facets each with 10 values. Second, the ability to combine

facets in any order provides users with more flexibility to specify their interests using whatever facets they are confident about for a particular search (as described in more detail below). Finally, facets are easier to manage and update, and efficient to use as input to machine learning algorithms.

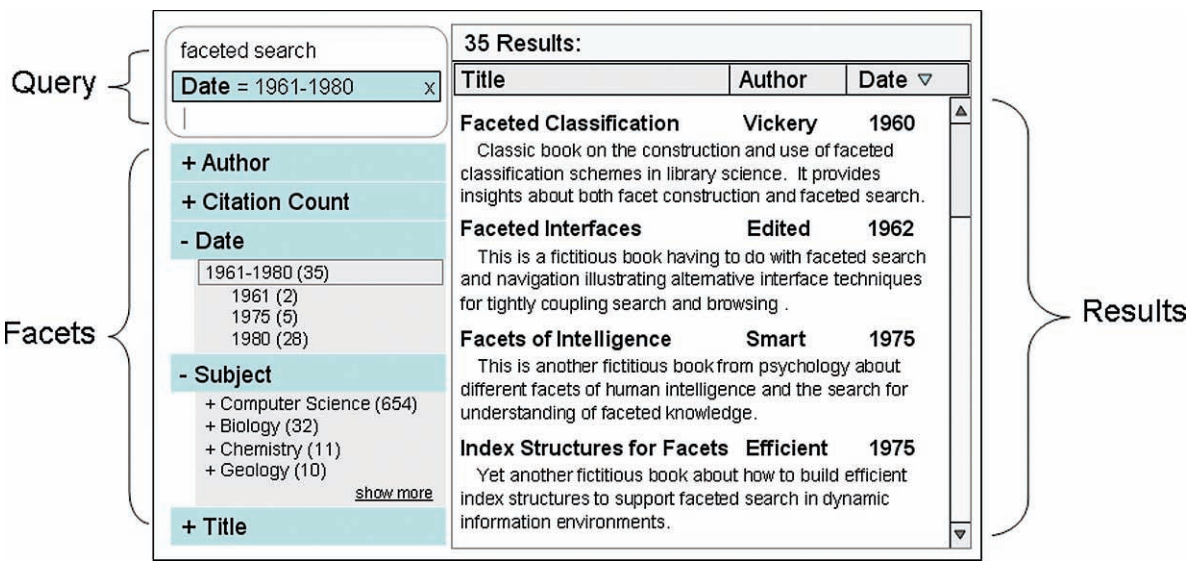
Faceted Access

Developing a faceted organization of information is only the first step in any application. Using this representation to support end-users in finding information or in understanding large multi-dimensional information spaces is critical for any successful application. In this entry, the term faceted access is used to refer to faceted browsing, search, and the integration of the two.

Several faceted search systems have been designed and deployed during the last two decades. The success of any system in supporting end-users is highly dependent on details of the domain of interest (e.g., searcher's tasks, their familiarity with the facets, etc.). A brief summary of the key components is outlined below. Most systems show the query, the facet structure, the subset of results currently specified, and sometimes a detailed view of an individual item. (See Fig. 2.)

Browsing The most common access technique supported by facets is navigation. Faceted navigation is a way to browse information, guided by the facet structure. Selecting a facet label (e.g., Subject=Computer Science) results in a list of all items described by that label. Additional facet values can be selected at any time and in any order (e.g., Date=2000; Author= Dumais; Subject=Computer Science/Databases). Faceted browsing allows searchers to refine a long list of results, using multiple dimensions, in whatever order they choose. From the user experience perspective, the big challenge is in managing the scale and complexity of multiple facets, each of which may contain thousands of values organized in either a hierarchy or flat list. Consider how cumbersome it can be to navigate using the file explorer when there are hundreds of thousands of files. And this is only a single facet (based on file location); now add other facets such as file type, size, creation date, access patterns, etc. and the design issues become apparent.

One challenge is to provide compressed views of very large information structures in ways that balance overview (context) and detailed (focus) information. A common technique is to progressively expose details for hierarchical facets, guided by the user. Initially only the top level values for each facet are shown. Once a facet is selected the next level of labels are exposed, and previous levels may be collapsed to save space with only a breadcrumb remaining. Yee et al. [14] show several examples of this. Another technique is to truncate long lists of values by showing a few labels with an option to "see more." The values can be selected in any



Faceted Search. Figure 2. Example faceted search interface, illustrating facet browsing, searching, and tight coupling of the two.

number of ways. In some cases an ordered list may be binned (e.g., instead of showing all unique dates, the dates can be grouped: before 1960; 1961–1980; 1981–2000; after 2000). In other cases the values are chosen alphabetically or numerically, to span the space, or based on the most common user tasks. And, in some cases, search over facet values is provided as a means of quickly winnowing down a long list of possible values (e.g., [2]).

Another challenge is to select labels that are understandable to users, thus enabling them to know which facet to select to get to the desired information. This is relatively easy for some facets, like Date or Language or Author, but more difficult for others like Subject (e.g., is a paper describing a user study of faceted navigation in Subject=Computer Science or Subject=Psychology or both or neither?). In cases like this, facet labels can be expanded by including examples of labels at the next level (e.g., Subject=Computer Science (AIgorithms, Databases, Human-Computer Interaction), thus improving the "information scent" of the description. Many web directories like the Open Directory (http://www.dmoz.org) label their categories in this manner.

An important technique for providing additional guidance about what will be found by selecting a facet value is called "dynamic query previews" (e.g., Greene et al. [3] Shneiderman [14]). The basic idea is to show some preview of the results that each query will generate. For facet browsing, this is commonly done by showing the number of results for each facet label next to the label (e.g., Biology (32); Chemistry (1); Computer Science (654); Geology (0)). Showing previews prevents also prevents users from selecting facets that will return no results. There are interesting challenges in efficiently providing such previews in distributed networked environments, and in cases where the set of items has been generated by some other means like a search.

Facet labels and query previews aid searchers in selecting the right facets. Interfaces should also allow for *both* specialization (narrowing a search) and generalization (broadening a search). Techniques like breadcrumbs are used to represent the current state of the "query" as it evolves over time through interaction. This provides a visible and actionable representation – searchers are able to understand the current set of items, and to quickly generalize as well as specialize.

In addition to supporting facet navigation, the interface should also show some representation of the current set of results. A simple list is one method for doing so. Richer list views in which the searcher can control the sort order along one or more facets are also used. Results sets can also be grouped by facet value to enable a greater diversity of results to be shown and to further reinforce the facet structure. Grouping is typically done using the next level for hierarchically organized facets [14], or the facet used for sorting [2]. In addition, some interfaces allow users to specify queries by example. Given a result or set of results, a new query can be generated to find other items that share one or more facets.

Searching (and Integrated Browsing and Searching) Search is often contrasted with browsing, and suggested as a means for providing access to large, unstructured collections of information. In web search engines, for example, a query results in a long unstructured list of results. Many faceted browsing systems also provide a search capability that is largely independent of the facet structure.

An improved user experience can be achieved by tightly coupling faceted browsing and searching capabilities. The facet structure can be leveraged in several ways to do this. A simple integration is to use the facets to navigate to a subset of items of interest and then to search within that. The converse is to start with search and to show the distribution of search results in terms of their corresponding facet structure. For example, a query for "faceted search" returns results that have a distribution of values on the Subject facet – e.g., Computer Science (654); Biology (32); Chemistry (1); Geology (0).

The tightest coupling is to view both search and navigation as techniques for specifying subsets of information, and to allow either method to be used at all times. Thus search and navigation can be combined in any order at any time, while maintaining a consistent representation and interface. The Flamenco [14] and Phlat [2] systems support this kind of seamless integration between search and navigation. Effective data structures and algorithms have been developed to support quickly finding sets of results and summarizing the distributions of their facet values [1,10].

Scale and Details An important design challenge in faceted interfaces is handling scale – the overall size of the collection, the number of different facets, and the number of values per facet (especially for multi-valued facets like names or user-generated tags). Displaying all values for all facets is not possible in most realistic

applications without requiring users to spend most of their time scrolling through very long lists or managing multiple windows. Methods for progressively exposing more detail for hierarchically organized facets, or for showing a limited set of key values for each facet, with more alternatives available on demand, are important techniques for managing scale. Using the facet structure to tightly couple browsing and searching is increasingly being used in e-commerce applications. This provides the searcher with tools that support a wide variety of tasks and states of knowledge.

There are still some capabilities that are difficult to implement in a usable and discoverable fashion. Searching over facet values when they are many values (e.g., Author names, Tags), multi-select operations within a facet, and using the same facet structure to support both access and tagging are all still challenging design issues. Decisions about which facets to display and how to do so depend critically on the application domain – e.g., price is very important in shopping but less so in finding technical materials; supporting range queries makes sense for Price, but probably not for Subject. Hearst et al. [4] provide a number of examples of how usability tests guided important interface design choices. These evaluations examine outcome measures such as task completion time and accuracy, subjective preferences, interaction trajectories, etc., and are used to iterate on the design until user performance goals are achieved.

Key Applications

Faceted search systems have been developed for a many applications in e-commerce (music, books, e-bay), bibliographic databases, image collections, and for finding files and email in the Windows and Macintosh operating systems. The following is a small list of applications available on the web, covering different domains.

- EBay Express, http://www.express.ebay.com/–general product information
- Flamenco, http://flamenco.berkeley.edu/demos.html – prototype applications for art images, architecture images and nobel prize winners
- NCSU Libraries http://www2.lib.ncsu.edu/catalog/–bibliographic records
- Tower Records, http://www.tower.com/–music, videos and books

These applications all in fairly narrow vertical domains and have readily available facet information. In the Flamenco prototypes, hand-generated metadata was available for each collection. In the NCSU application, the facets were populated using bibliographic records. And, in the e-commerce applications, many of the facets displayed are required for business purposes (e.g., price, manufacturer, etc.) and thus readily available.

Future Directions

New interface techniques for managing scale in existing applications will continue to be a fruitful area for innovation. Algorithms and architectures that can provide real-time dynamic previews of results in the context of the facet structure are required for distributed or networked applications. Developing methods for representing and displaying a confidence score for each facet value is another new direction.

Finally, it is interesting to consider why faceted interfaces are not widely available for general web search. There have been some attempts to structure the web using a topic hierarchy like Open Directory (http://www.dmoz.org) or Yahoo! in its early days. However this represents only a single facet (topic), and facet values are available for only a small subset of web content. Facet values could be assigned automatically using text classification techniques (e.g., topic, genre, commerciality), and Web pages already have some structure that could support faceted access (e.g., site names, creation date, modification date, language, usage history). Understanding which facets are most important to support the variety of information needs that people use the web for, and handling large-scale dynamic collections will be key in determining the success of faceted search methods for web content.

Cross-references

- ▶ Browsing in Digital Libraries
- ▶ Cataloging in Digital Libraries
- ▶ Cross-Modal Multimedia Information Retrieval
- ▶ Digital Libraries
- ▶ Human-Computer Interaction
- ▶ Information Retrieval
- ▶ Ontology
- ▶ Presenting Structured Text Retrieval Results
- ▶ Searching Digital Libraries
- ▶ Text Categorization
- ▶ Text Indexing and Retrieval
- ▶ Web Information Retrieval Models

Recommended Reading

1. Bast H. and Weber I. The complete search engine: interactive, efficient and towards IR & DB integration. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 88–95.
2. Cutrell E., Robbins D.C., Dumais S.T., and Sarin R. Fast, flexible filtering with Phlat – personal search and organization made easy. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2006, pp. 261–270.
3. Greene S., Marchionini G., Plaisant C., and Shneiderman B. Previews and overviews in digital libraries: designing surrogates to support visual information-seeking. J. Am. Soc. Inform. Sci., 51(3):380–393, 2000.
4. Hearst M., Smalley P., and Chander C. Faceted metadata for information architecture and search. Course at SIGCHI Conf. on Human Factors in Computing Systems, 2006.
5. Mooers C.N. Zatocoding applied to mechanical organization of knowledge. Am. Doc., 2(1):20–32, 1951.
6. Pollitt A.S. A common query interface using MenUSE – A menu-based user search engine. In Proc. 12th Int. Online Information Meeting, 1988, pp. 445–457.
7. Ranganathan S.R. Colon classification. The Madras Library Association, Madras, India and Goldston, London, 1933.
8. Remde J.R., Gomez L., and Landauer T.K. SuperBook: An automatic tool for information exploration. In Proc. Hypertext'87, Conf. Chapel Hill, NC, 1987, pp. 175–188.
9. Rosenfeld L. and Morville P. Information Architecture for the World Wide Web. O'Reilly Media, Sebastapol, CA, 2006.
10. Sacco G.M. Dynamic taxonomies: a model for large information bases. IEEE Trans. Knowl. Data Eng., 12(2):468–479, 2000.
11. Shneiderman B. Dynamic queries for visual information seeking. IEEE Softw., 11(6): 70–77, 1994.
12. Taube M. et al. Studies in Coordinate Indexing (vols. 1–5). Documentation Incorporation, Washington, DC, 1953–1959.
13. Vickery B.C. Faceted Classification: A Guide to the Construction and Use of Special Schemes. ASLIB, London, 1960.
14. Yee K., Swearingen K., Li K., and Hearst M. Faceted metadata for image search and browsing. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2003, pp. 401–408.

Facility-Location Problem

► Resource Allocation Problems in Spatial Databases

Fact-Oriented Modeling

► Object-Role Modeling

Failure Handling

► Crash Recovery
► Logging and Recovery

False Negative Rate

► Precision and Recall

Fault Tolerant Applications

► Application Recovery

F

Fault-Tolerance

► Replication for High Availability

Fault-Tolerance and High Availability in Data Stream Management Systems

Magdalena Balazinska[1], Jeong-Hyon Hwang[2], Mehul A. Shah[3]
[1]University of Washington, Seattle, WA, USA
[2]Brown University, Providence, RI, USA
[3]HP Labs, Palo Alto, CA, USA

Definition

Just like any other software system, a data stream management system (DSMS) can experience failures of its different components. Failures are especially common in *distributed* DSMSs, where query operators are spread across multiple processing nodes, i.e., independent processes typically running on different physical machines in a local-area network (LAN) or in a wide-area network (WAN). Failures of processing nodes or failures in the underlying communication network can cause continuous queries (CQ) in a DSMS to stall or produce erroneous results. These failures can adversely affect critical client applications relying on these queries.

Traditionally, availability has been defined as the fraction of time that a system remains operational and properly services requests. In DSMSs, however, availability often also incorporates end-to-end latencies as applications need to quickly react to real-time events and thus can tolerate only small delays. A DSMS can handle failures using a variety of techniques that offer different levels of availability depending on application needs.

All fault-tolerance methods rely on some form of replication, where the volatile state is stored in independent locations to protect against failures. This entry describes several such methods for DSMSs that offer different trade-offs between availability and runtime overhead while maintaining consistency. For cases of network partitions, it outlines techniques that avoid stalling the query at the cost of temporary inconsistency, thereby providing the highest availability. This entry focuses on failures within a DSMS and does not discuss failures of the data sources or client applications.

Historical Background

Recently, DSMSs have been developed to support critical applications that must quickly and continuously process data as soon as they become available. Example applications include financial stream analysis and network intrusion detection (see Key Applications for more). Fault-tolerance and high availability are important for these applications because faults can lead to quantifiable losses. To support such applications, a DSMS must be equipped with techniques to handle both node and network failures.

All basic techniques for coping with failures involve some kind of replication. Typically, a system replicates the state of its computation onto independently failing nodes. It must then coordinate the replicas in order to recover properly from failures. Fault-tolerance techniques are usually designed to tolerate up to a predefined number, k, of simultaneous failures. Using such methods, the system is then said to be k-fault tolerant.

There are two general approaches for replication and coordination. Both approaches assume that the computation can be modeled as a deterministic state-machine [4,11]. This assumption implies that two non-faulty computations that receive the same input in the same order will produce the same output in the same order. Hereafter, two computations are called *consistent* if they generate the same output in the same order.

The first approach, known as the *state-machine* approach, replicates the computation on $k + 1 \geq 2$ independent nodes and coordinates the replicas by sending the same input in the same order to all [11]. The details of how to deliver the same input define the various techniques. Later sections in this entry describe variants that are specific to DSMSs. The state-machine approach requires $k + 1$ times the resources of a single replica, but allows for quick fail-over, so a failure causes little disruption to the output stream. This property is important for critical monitoring tasks such as intrusion detection that require low-latency results at all times.

The second general approach is known as *rollback recovery* [4]. In this approach, a system periodically packages the state of its computation into a *checkpoint*, and copies the checkpoint to an independent node or a non-volatile location such as disk. Between checkpoints, the system logs the input to the computation. Since disks have high latencies, existing fault-tolerance methods for DSMSs copy the checkpointed state to other nodes and maintain logs in memory. Upon failure, the system reconstructs the state from the most recent checkpoint, and replays the log to recover the exact pre-failure state of the computation. This approach has much lower runtime overhead at the expense of longer recovery times. It is useful in situations where resources are limited, the state of the computation is small, fault-tolerance is important, but rare moderate latencies are acceptable. An example application is fabrication-line monitoring using a server cluster with limited resources.

In some cases, users are willing to tolerate temporary inconsistencies to maintain availability at all times. One example is in the wide-area where network partitions are likely (e.g., large-scale network and system monitoring). To maintain availability in face of network partitions, the system must move forward with the computation ignoring the disconnected members. In this case, however, replicas that process different inputs will have inconsistent states. There are two general approaches for recovering from such inconsistencies after the network partition heals. One approach is to propagate all updates to all members and apply various rules for reconciling conflicting updates [6,9]. The other approach is to undo all changes performed during the partition and redo the correct ones [6,15].

This entry presents how these general approaches can be adapted to distributed DSMSs. The main challenge is to ensure that applications receive low-latency results during both normal processing and failures. To do so, the methods presented leverage the structure of continuous queries (CQs) in DSMSs.

This entry makes the following assumptions. A CQ is a connected directed-acyclic graph of query operators. The operators can be distributed among many processing nodes with possibly multiple operators per node. A processing node is the unit of failure. For simplicity of exposition, this entry focuses on the

case of $k = 1$ (i.e., two query replicas) although all shown techniques can handle any k. This entry describes methods to tolerate node crash failures and temporary network partitions. Roughly speaking, a node has crashed if it visibly halts or simply becomes unresponsive [12]. A network partition splits nodes into two or more groups where nodes from one group cannot communicate with nodes in another.

Foundations

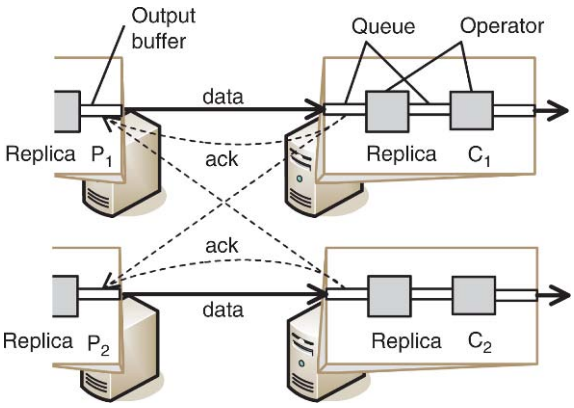
Techniques for Handling Crash Failures

This section describes new fault-tolerance techniques devised by applying the general fault-tolerance methods to continuous queries in DSMSs.

Active Replicas Active replicas are an application of the state-machine approach in which query operators are replicated and run on independently failing nodes. A simple variant of the active replicas approach uses the traditional process-pair technique to coordinate the replicas. The process-pair technique runs two copies of the query and specifies one to be the primary and the other to be the secondary. In this approach, the primary forwards all input, in the same order, to the secondary and works in lock-step with the secondary [5].

A DSMS can rely on a looser synchronization between the replicas by taking advantage of the structure of CQ dataflows. In a CQ dataflow, the operators obey a producer-consumer relationship. To provide high availability, the system replicates both the producer and consumer as illustrated in Fig. 1. In this model, there is no notion of a primary or secondary. Instead, each producer logs its output and forwards the output to its current consumer(s). Each consumer sends periodic acknowledgments to all producers to indicate that it has received the input stream up-to a certain point. An acknowledgment indicates that the input need not be resent in case of failure, so producers can truncate their output logs. Use of reliable, in-order network delivery (e.g., TCP) or checkpoints allows important optimizations where consumers send application-level acknowledgments to only a subset of producers [7,14].

The symmetric design of active replicas has some benefits. The normal-case behavior has few cases, so it is simple to implement and verify. Additionally, with sufficient buffering, each pipeline can operate at its own pace, in looser synchronization with the other.



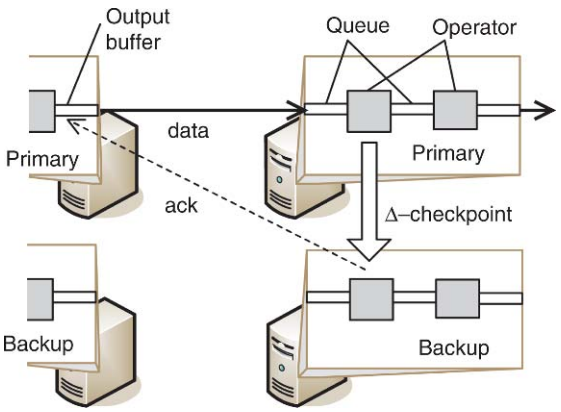
Fault-Tolerance and High Availability in Data Stream Management Systems. Figure 1. Active replicas. The operators on replicas P_1 and P_2 are the producers. The operators on C_1 and C_2 are the consumers.

The Flux [14] approach was the first to investigate this looser synchronization between replicated queries. Flux is an opaque operator that can be interposed between any two operators in a CQ. Flux implements a simple variant of this protocol and assists in recovery. The Borealis "Delay, Process, and Correct" (DPC) protocol [1,2] also uses the above coordination scheme, but differs from Flux in its recovery, as discussed later. The Flux and DPC approaches both ensure strict consistency in the face of crash failures: no duplicate output is produced and no output is lost.

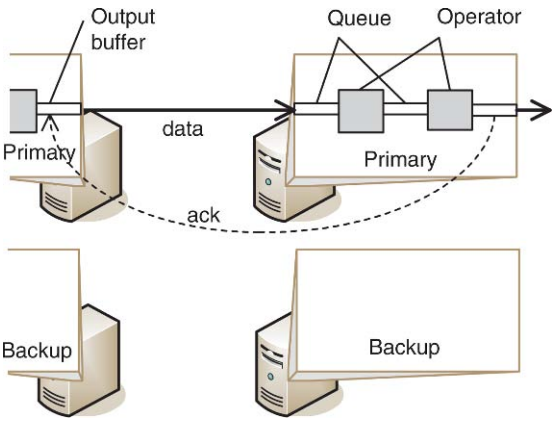
Passive Replicas There have been two applications of the rollback recovery approach to CQs [7,8]. The first, called *passive standby*, handles all types of operators. The second, called *upstream backup*, is optimized for more specific bounded-history operators that frequently arise in CQs.

In the passive standby approach, a primary node periodically checkpoints its state and sends that checkpoint to a backup. The state includes any data maintained by the operators and tuples stored in queues between operators. In practice, sending the entire state at every checkpoint is not necessary. Instead, each primary periodically performs only a *delta-checkpoint* as illustrated in Fig. 2. During a delta-checkpoint, the primary updates the backup by copying only the difference between its current state and the state at the time of the previous checkpoint.

Because of these periodic checkpoints, a backup always has its primary's state as of the last checkpoint.



Fault-Tolerance and High Availability in Data Stream Management Systems. Figure 2. Passive standby.



Fault-Tolerance and High Availability in Data Stream Management Systems. Figure 3. Upstream backup.

If the primary fails, the backup recovers by restarting from that state and reprocessing all the input tuples that the primary had processed since the last checkpoint. To enable backups to reprocess such input tuples, all primaries log their output tuples. If a downstream primary fails, each upstream primary re-sends its output tuples to the downstream backup. In a CQ, because the output of an operator can be connected to more than one downstream consumer operator, primaries discard logged output tuples only after *all* downstream backups have acknowledged a checkpoint.

For many important CQ operators, the internal state often depends only on a small amount of recent input. Examples of such operators include joins and aggregates with windows that span a short time-period or a small number of tuples. For such operators, DSMSs can use the upstream backup method to avoid any checkpointing overhead. In this approach, primaries log their output tuples, but backups remain idle as illustrated in Fig. 3. The primaries trim their logs based on notifications from operators 1-level (or more) downstream, indicating that the states of consuming operators no longer depend on the logged input. To generate these notifications, downstream operators determine, from their output, what logged input tuples can be safely discarded. If a primary fails, an empty backup rebuilds the latest state of the primary using the logs kept at upstream primaries.

Failure Recovery When a failure occurs, a DSMS must first detect and then recover from that failure. DSMSs detect failures using timeouts and, in general, rely on standard group membership mechanisms to keep consistent track of nodes entering and leaving the system [10,13]. Recovery ensues after failure detection.

There are two parts to failure recovery. The first part involves masking the failure by using the remaining replica to continue processing. For active replicas, this part is called *fail-over*. In both Flux and DPC, fail-over is straightforward. Consumers and producers adjust their connections to receive input data from or send output data to the remaining live copy of the failed node. To avoid stalls in the output stream, it is safe for active replicas to proceed with fail-over without waiting for group membership consensus [1,13]. For passive standby and upstream backup, this first part also involves bringing the state of the backup to the pre-failure state of the failed primary, as described earlier, before the backup starts sending data to downstream consumers.

The second part of recovery, called *repair*, allows the query to repair its failed pieces and regain its original level of fault-tolerance. In upstream backup, the system regains its normal fault-tolerance level when the new replica fills its output log with enough data to rebuild the states of downstream nodes.

For both active replicas and passive standby, repair can cause significant disruptions in the result stream depending on the granularity of coordination in the query. For example, if a system uses active replica coordination only at the input(s) and output(s) of a distributed query, the system must destroy the *entire* query affected by the failure, stall the *entire* remaining query, checkpoint its state, copy that state onto independent nodes, and reintegrate the new copy with the remaining query. The system must repair a query at a time because it has no control over inflight data in the

network between nodes in a query. If the query state is large, e.g., tens of gigabytes, repair can take minutes, causing significant latencies in the result stream. Similarly, coarse coordination in passive standby would cause the first checkpoint after recovery to stall processing for a long time.

To remedy this problem, most high-availability CQ schemes (e.g., Flux [14,13], Borealis DPC [1,2], Active Standby [7], Passive Standby [7,8]) coordinate and repair in smaller chunks: between nodes (containing groups of operators), between operators, or even finer. Then, after failure, they can repair the lost pieces one at time, allowing the remaining pieces to continue processing and reduce the impact of stalls. In the presence of $k + 1 > 2$ replicas, DSMSs can use the extra replicas to further smooth the impact of stalls during repair. Also with finer coordination, DSMSs need to repair only the lost pieces, thereby reducing mean-time-to-recovery and improving system reliability, i.e., mean-time-to-failure [14].

Trade-Offs Among Crash Failure Techniques The above techniques provide different trade-offs between runtime overhead and recovery performance. Active replicas provide quick fail-over because replicas are always "up-to-date". With this approach, however, the runtime overhead is directly proportional to the degree of replication. Passive standby provides a flexible trade-off between runtime overhead and recovery speed through the configurable checkpoint interval. As the checkpoint interval decreases, the runtime computation and network overheads increase because the primaries copy more intermediate changes to the backups. However, recovery speed improves because the backups are in general more up-to-date when they take over. Finally, upstream backup incurs the lowest overhead because backups remain idle in the absence of failures. For upstream backup, recovery time is proportional to the size of the upstream buffers. The size of these buffers, in turn, depends on how much history is necessary to rebuild the state of downstream nodes. Thus, upstream backup is practical in small history settings.

Techniques for Handling Network Partitions

The previous techniques can mask crash failures and a limited set of network failures (by converting disconnected nodes into crashed nodes), but cannot handle network partitions in which data sources, processing nodes, and clients are split into groups that cannot communicate with each other.

In the presence of network partitions, a DSMS, like all distributed systems, has two choices. It can either suspend processing to ensure consistency, or continue processing the remaining streams with best-effort results to provide availability [3]. Existing work on fault-tolerance in distributed DSMSs has explored both options. The Flux protocol [13], originally set in the local-area where network partitions are rare, favors consistency. The Borealis's DPC protocol, designed for wide-area monitoring applications where partitions are more frequent, favors availability. During partitions, DPC generates best-effort result tuples which are labeled as *tentative*. Further, DPC allows applications to specify a maximum tolerable latency for flexibly controlling the tradeoff between consistency and availability [1,2].

Once a network partition heals, a stalled CQ node can simply resume. A node that continued processing with missing input, however, might be in a diverged state, i.e., a state different from that of a failure-free execution. To reconcile a node's diverged state, a DSMS can take two approaches. The system can revert the node to a consistent, checkpointed state and replay the subsequent, complete input, or the system can undo and redo the processing of all tuples since the network partition. To avoid stalling the output during reconciliation, a DSMS must take care not to reconcile all replicas of a node simultaneously. Moreover, nodes must correct their previous output tentative tuples to enable downstream nodes to correct their states, in turn, and to allow applications to ultimately receive the correct and complete output. The Borealis DPC protocol supports these techniques [1,2].

Optimizations

Flux: Integrating Fault Tolerance and Load Balancing A large-scale cluster is a dynamic environment in which DSMSs face not only failures but also load imbalances which reduce availability. In this setting, DSMSs typically split the state of operators into *partitions* and spread them across a cluster for scalability. A single overloaded machine, in this setup, can severely slow down an entire CQ. The Flux operator can be interposed between producer–consumer partitions to coordinate their communication. To absorb short delays and imbalances, Flux allows out-of-order processing of partition input. For long-term imbalances, it supports fine-grained, online partition state movement. The Flux operators interact with a global

controller that coordinates repair and rebalancing, and uses the same state movement mechanism for both [13]. Integrating load balancing with fault tolerance allows a system to better utilize available resources as nodes enter and leave the system. These features allow smooth hardware refresh and system growth, and are essential for administering DSMSs in highly dynamic and heterogeneous environments.

Leveraging Replication for Availability and Performance in Wide-Area Networks In previous active replicas approaches, a consumer replica receives the output stream of only one of many producer replicas. In wide area networks, the connection to any single producer is likely to slow down or fail, thereby disrupting the subsequent processing until fail-over completes. To avoid such disruptions, each consumer replica in Hwang et al.'s method [7] merges streams from multiple producer replicas into a single duplicate-free stream. This scheme allows each consumer, at any instant, to use the fastest of its replicated input streams. To further reduce latency, they redesign operators to avoid blocking by processing input out-of-order when possible, while ensuring that applications receive the same results as in the non-replicated, failure-free case. Moreover, their scheme continually adjusts the replication level and placement of operator replicas to optimize global query performance in a dynamically changing environment.

Passive Standby: Distributed Checkpointing and Parallel Recovery The basic passive standby approach has two drawbacks: it introduces extra latencies due to checkpoints and has a slower recovery speed than active replicas. Hwang et al.'s distributed checkpointing technique overcomes both problems [8]. This approach groups nodes into logical clusters and backs up each node using the others in the same cluster. Because different operators on a single node are checkpointed onto separate nodes, they can be recovered in parallel. This approach dynamically assigns the backup node for each operator and schedules checkpoints in a manner that maximizes the recovery speed. To reduce disruptions, this approach checkpoints a few operators at a time. Such checkpoints also begin only at idle times.

Key Applications

There are a number of critical, online monitoring tasks that require 24×7 operation. For example, IT administrators often want to monitor their networks for intrusions. Brokerage firms want to analyze quotes from various exchanges in search for arbitrage opportunities. Phone companies want to process call-records for correct billing. Web site owners want to analyze and monitor click-streams to improve targeted advertising and to identify malicious users. These applications, and more, can benefit from fault-tolerant and highly available CQ systems.

Future Directions

Key open problems in the area of fault-tolerance and high availability in DSMSs include handling Byzantine failures, integrating different fault-tolerance mechanisms, and leveraging persistent storage. Techniques for handling failures of data sources or dirty data produced by data sources (e.g., sensors) are also areas for future work.

Experimental Results

See [1,2,7,8,13,14] for detailed evaluations of the different fault-tolerance algorithms.

URL to Code

Borealis is available at: http://www.cs.brown.edu/research/borealis/public/

Cross-references

- ▶ Continuous Query
- ▶ Data Stream
- ▶ Distributed Data Streams
- ▶ Stream Processing

Recommended Reading

1. Balazinska M. Fault-Tolerance and Load Management in a Distributed Stream Processing System. Ph.D. thesis, Massachusetts Institute of Technology, 2006.
2. Balazinska M., Balakrishnan H., Madden S., and Stonebraker M. Fault-Tolerance in the Borealis Distributed Stream Processing System. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 13–24.
3. Brewer E.A. Lessons from giant-scale services. IEEE Internet Comput., 5(4):46–55, 2001.
4. Elnozahy E.N.M., Alvisi L., Wang Y.M., and Johnson D.B. A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv., 34(3):375–408, 2002.
5. Gray J. Why do computers stop and what can be done about it? Technical Report 85.7, Tandem Computers, 1985.
6. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.

7. Hwang J.H., Balazinska M., Rasin A., Çetintemel U., Stonebraker M., and Zdonik S. High-Availability Algorithms for Distributed Stream Processing. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 779–790.
8. Hwang J.H., Xing Y., Çetintemel U., and Zdonik S. A cooperative, self-configuring high-availability solution for stream processing. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 176–185.
9. Kawell L., Beckhardt S., Halvorsen T., Ozzie R., and Greif I. Replicated Document Management in a Group Communication System. In Proc. ACM Conf. on Computer-Supported Cooperative Work, 1988.
10. Schiper A. and Toueg S. From set membership to group membership: a separation of concerns. IEEE Trans. Dependable Secure Comput., 3(1):2–12, 2006.
11. Schneider F.B. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv., 22(4): 299–319, 1990.
12. Schneider F.B. What good are models and what models are good? In Distributed Systems. ACM/Addison-Wesley Publishing Co, 2nd edn., 1993, pp. 17–26.
13. Shah M.A. Flux: A Mechanism for Building Robust, Scalable Dataflows. Ph.D. thesis, University of California, Berkeley, 2004.
14. Shah M., Hellerstein J., and Brewer E. Highly-Available, Fault-Tolerant, Parallel Dataflows. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 827–838.
15. Terry D.B., Theimer M., Petersen K., Demers A.J., Spreitzer M., and Hauser C. Managing update conflicts in Bayou, a weakly connected replicated storage system. In Proc. 15th ACM Symp. on Operating System Principles, 1995, pp. 172–183.

FCP

► Storage Protocols

FD

► Functional Dependency

Feature Extraction for Content-Based Image Retrieval

Raimondo Schettini[1], Gianluigi Ciocca[1], Isabella Gagliardi[2]
[1]University of Milano-Bicocca, Milan, Italy
[2]National Research Council (CNR), Milan, Italy

Synonyms

Image indexing

Definition

Feature extraction for content-based image retrieval is the process of automatically computing a compact representation (numerical or alphanumerical) of some attribute of digital images, to be used to derive information about the image contents. It can be seen as a case of dimensionality reduction. A feature, or attribute, can be related to a visual characteristic, but it may also be related to an interpretative response to an image or to a spatial, symbolic, semantic, or emotional characteristic. A feature may relate to a single attribute or be a composite representation of different attributes. Features can be classified as general purpose or domain-dependent. The general purpose features can be used in any context, while the domain-dependent features are designed specifically for a given application. Every feature is intimately tied with the kind of information that it captures. The choice of a particular feature over another depends on the given application, and the kind (level) of information required.

Historical Background

The interest of the scientific community in the problem of image retrieval can be dated back to the early 1990s. One of the pioneering examples on image feature extraction is the work by Swain and Ballard [6] where the concept of color histogram is introduced for image indexing. Since then, color (in many forms and evolutions) has been one of the most widely used features exploited for indexing the image contents. This is mainly due to its power to visually capture perceptual properties, robustness with respect to many image transformations, and computation efficiency. While more complex and sophisticated general purpose and domain dependent features were being developed, two important issues became evident: the sensory gap and the semantic gap.

The sensory gap is the gap between the information of the real world, and the information in a computational description derived from a digital recording of a scene of the world. The semantic gap is the lack of coincidence between the information that one can extract from the visual data and the interpretation that the same data have for a user in a given application [5].

The problems of the sensory and semantic gaps are less relevant for very specific applications (e.g., medical imaging) where the domain may be so narrow that high knowledge can be embedded in few features. However, for general purpose applications, sensory and semantic

gaps cannot be bridged by a few features alone. To cope with the sensory gap problem, current approaches try to combine different features (mostly at low or middle levels of information), and different representations of the same feature. One way to cope with the semantic gap problem is not to tackle it at the feature level but at the system-user interaction level with human-adaptive paradigms. For these paradigms, the user represents a central role for a content-based image retrieval system. User interaction allows the system to automatically adapt to the changing requests; for example, in the way the images are organized, evaluated, or retrieved [7].

Foundations

Since an image conveys information at different levels, the use of different features at the same time is a necessary requisite to develop effective indexing algorithms for content-based image retrieval. For example, any visual property can have different feature representations describing it. Visual properties can be low level properties such as color, shape or texture; middle level properties such as regions and spatial relationships; high level properties such as the localization and identification of objects or image categorization. The process of choosing the best features for a particular application is denoted as feature selection.

The selected features should ideally present the following basic properties:

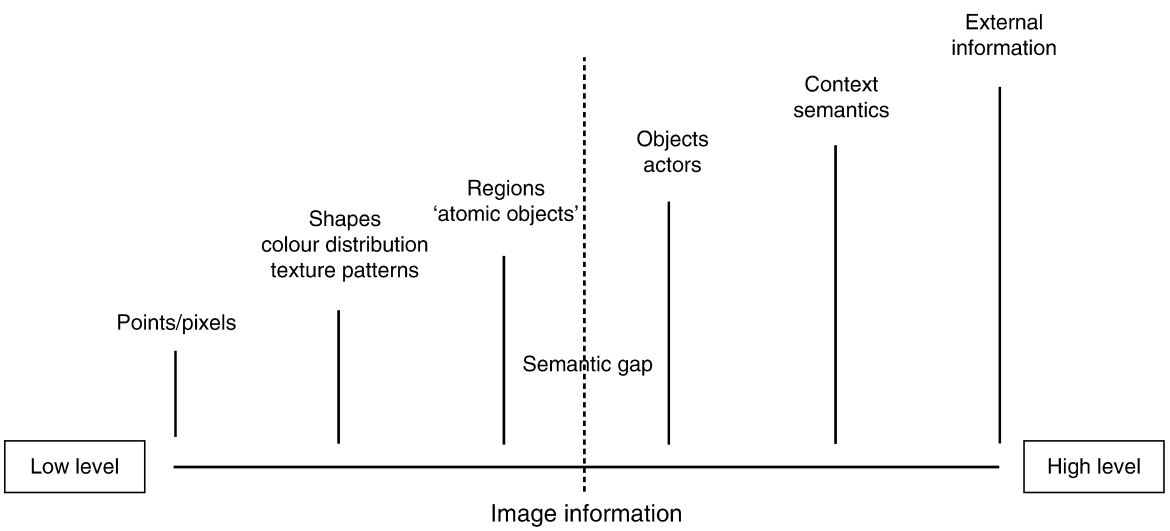
1. Perceptual similarity: the feature distance between two images is great only if the images are not "similar";
2. Efficiency: they can be rapidly computed;
3. Economy: their dimensions are small in order not to affect retrieval efficiency;
4. Sscalability: the performance of the system is not influenced by the size of the database;
5. Rrobustness: changes in the imaging conditions (i.e., illuminations, geometric transformations,. . .) of the database images should not affect retrieval.

Once the features have been selected, a similarity measure (one for each feature and/or a global one) is chosen in order to evaluate the user query against the images in the database, and rank the retrieved images. Ideally a similarity measure should be correlated with the user's perception of image matching or *image similarity*.

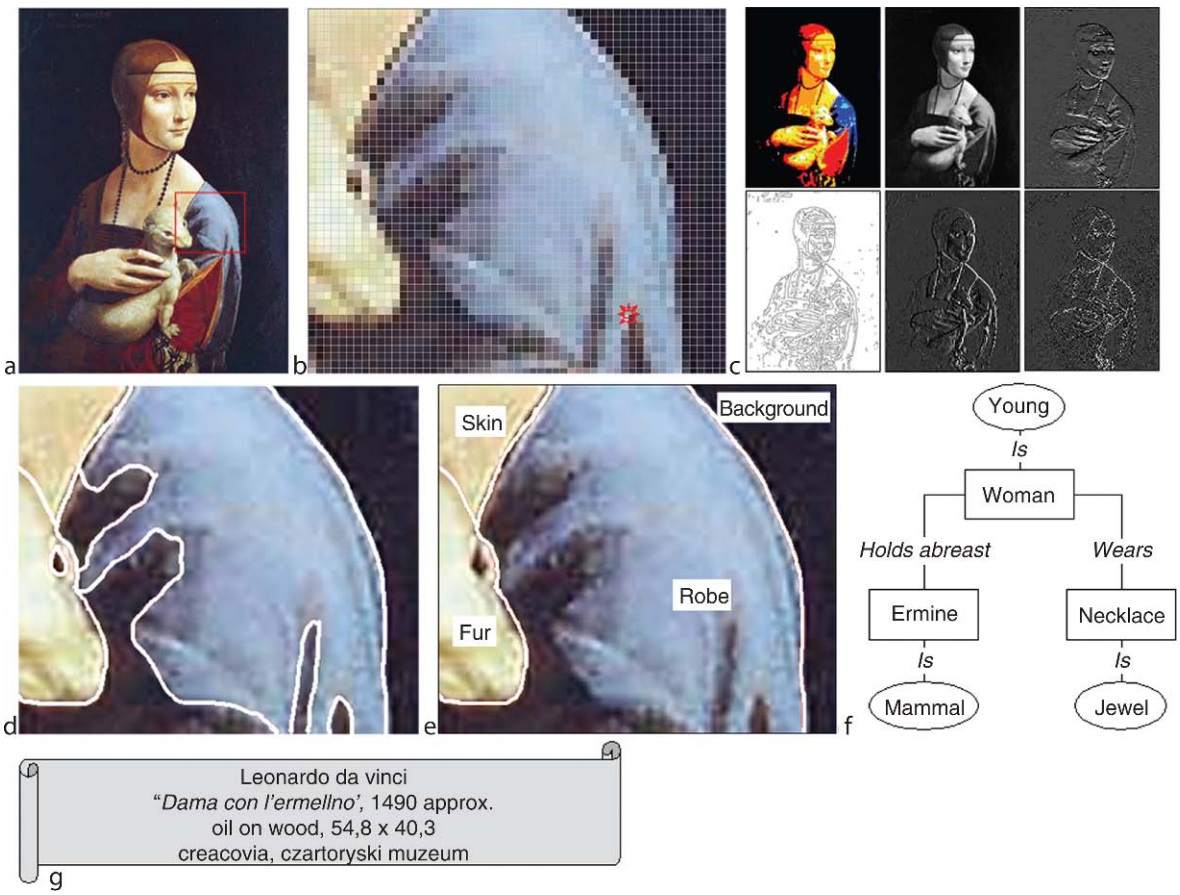
Figure 1 shows the different levels of information that can be derived from an image: the dashed line indicates where the semantic gap starts to influence the feature extraction algorithms.

Figure 2 is an example of the different information that can be obtained from an image. At each level of information more complex features are required to capture that kind of information. Details of each level are as follows:

1. Digital image of a painting.
2. Pixels. The image is coded using a numerical representation of physical properties derived from the input device with implicit spatial relationships. This representation may capture color as well as pressure, range, heath, non visible wavelengths and



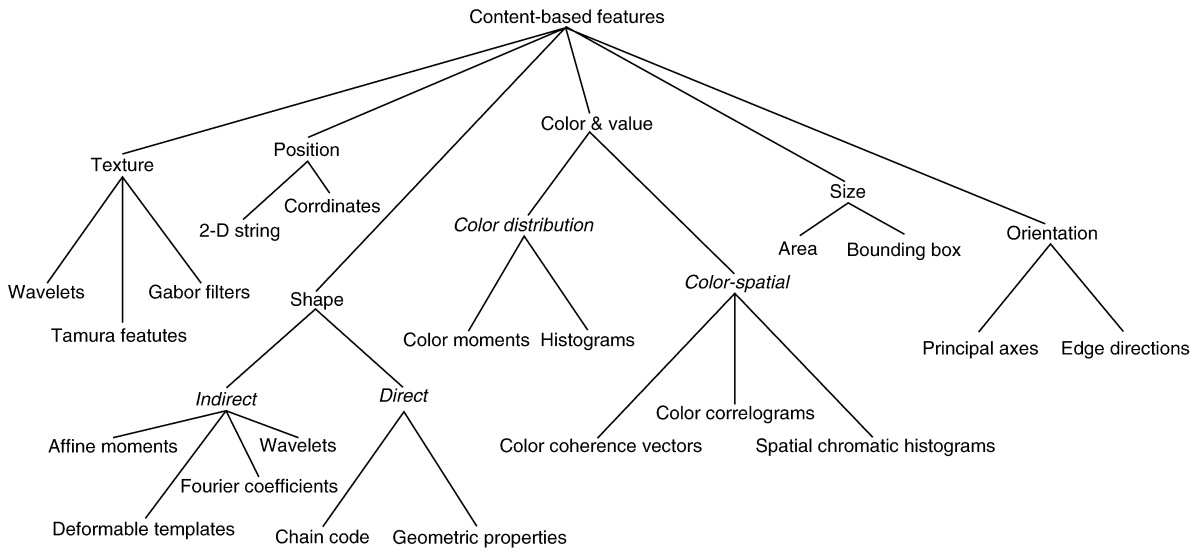
Feature Extraction for Content-Based Image Retrieval. Figure 1. The levels of information, and the semantic gap.



Feature Extraction for Content-Based Image Retrieval. Figure 2. Example of features corresponding to different levels of information. (a) the example image; (b) pixel level; (c) global low level features; (d) local low level features; (e) semantic regions; (f) semantic context; (g) external information.

so on. Each of these pieces of information physical information can be described using different numerical representations.

3. Low level information. These are global features that can describe the image contents from different perspectives in a compact way. The examples shown are: color contents in a quantized color space (top left), global edges computed with the Canny edge detector (bottom left), the gray level image (top middle), and horizontal, vertical, and diagonal edges computed from a wavelet transformation (to right, bottom middle, and bottom right respectively). This information is usually coded in terms of simple statistics (mean, variance,...), histograms, or signatures to improve their robustness to image variations, and reduce their dimensions for retrieval efficiency. The schema below shows an incomplete and non-exhaustive categorization of some low level visual properties, and their possible feature representations. Many representations and sub-representations are available for every feature (see Fig. 3). For further information interested readers may refer to [1,2,3,4].
4. Regions or "atomic objects." These are visually homogeneous regions that can be identified starting from the combination of different low level attributes (color, texture,...). *Data clustering* algorithms can be exploited for this task. Domain-dependent knowledge (for example specific spatial relationships between the areas) can be exploited to drive or refine the identification of these regions.
5. Objects/Actors. Identification of semantic objects or categories may be performed by the analysis of logically and spatially related regions. At this level features should cope with the problem of the semantic gap. The recognition of simple objects or



Feature Extraction for Content-Based Image Retrieval. Figure 3. Example of features, their corresponding representations and sub-representations.

categories is very complex, and is currently fully possible only for very narrow domains. Identification of semantic objects can be done exploiting pattern matching, *classification and decision tree* algorithms. These algorithms give a semantic tag or name to each candidate region (such as face, people, skin, sky, car, building, water, . . .). Generally one tag for each region is assigned, although multiple tags are sometimes allowed. This process is generally known as image annotation. Semantic tags can be expressed numerically or with more meaningful textual strings. Textual strings can be stored in traditional database systems to be used for image retrieval based on keywords.

6. Context semantics. This information relates to mutual spatial relationships as well as to specific actors/objects characteristics that may allow reconstruction/guessing of the overall image context. Some characteristics of the actors/objects may be derived using term taxonomies, synonyms, and dictionaries. Spatial relationships can also be described using textual strings by creating an ad-hoc language grammar such as A < B meaning that object A is at the left of object B or A ^ B meaning that object A is above object B, and so on. It should be noted that the classification and categorization of an image as a whole may involve information at the objects/actors level (the image itself is an object that can be annotated using low level features), and information at the semantic level (the presence of objects/actors, and some related characteristics can be used as clues in guessing the context of the image).
7. External information. This information does not describe some visual properties or intrinsic properties of the image. This auxiliary information cannot be derived or inferred directly from the image itself: external knowledge belonging to some entities (e.g., experts, common knowledge, etc. . .) is required. No automatic algorithm can be exploited with this kind of information: human intervention is the only viable approach in creating textual annotations usually given in free text form. The EXIF data also belong to this information category. These data, automatically embedded into the image by the hardware (e.g., camera), describe information about the device, device settings and scene information.

Key Applications

Feature extraction is a fundamental task for any application requiring the analysis of images. In particular, content-based image retrieval is important in many areas. The following is a sampling of possible content-based image retrieval applications:

1. Architecture and design (finding the right appearance)
2. Biochemical application (finding molecules)

3. Cultural heritage services (museum, art galleries)
4. Digital catalogs (browsing and searching)
5. Entertainment (image, film and video archive indexing)
6. Journalism (research and past exploration)
7. Medicine (pathology comparisons)
8. Remote sensing (localization and target finding)
9. Surveillance (identification and recognition)

Cross-references

- ▶ Automatic Image Annotation
- ▶ Decision Tree Classification
- ▶ Image Content Modeling
- ▶ Image Metadata
- ▶ Image Representation
- ▶ Indexing and Similarity Search

Recommended Reading

1. Antani S., Kasturi R., and Jain R. Survey on the use of pattern recognition methods for abstraction, indexing and retrieval of images and video. Pattern recognit., 35:945–965, 2002.
2. Eakins J.P. Towards intelligent image retrieval. Pattern Recognit., 35:3–14, 2002.
3. Schettini R., Ciocca G., and Zuffi S. Indexing and retrieval in color image databases. In Color Imaging Science: Exploiting Digital Media, R. Luo, L. MacDonald (eds.). Wiley, New york, 2002, pp. 183–211.
4. Sikora T. The MPEG-7 visual standard for content description – An overview. IEEE Trans. Circuits Syst. Video Technol., 11 (6):696–702, 2001.
5. Smeulders A.W.M., Worring M., Santini S., Gupta A., and Jain R. Content-based image retrieval at the end of the early years. IEEE Trans. Pattern Anal. Mach. Intell., 2(2):1349–1380, 2000.
6. Swain M.J. and Ballard D.H. Indexing via color histograms. In Proc. 3rd IEEE Conf. Computer Vision, 1990, pp. 390–393.
7. Zhou X.S. and Huang T.S. Relevance feedback in image retrieval: a comprehensive review. Multimed. Syst., 8(6):536–544, 2003.

Feature Selection for Clustering

Manoranjan Dash, Poon Wei Koot
Nanyang Technological University, Singapore, Singapore

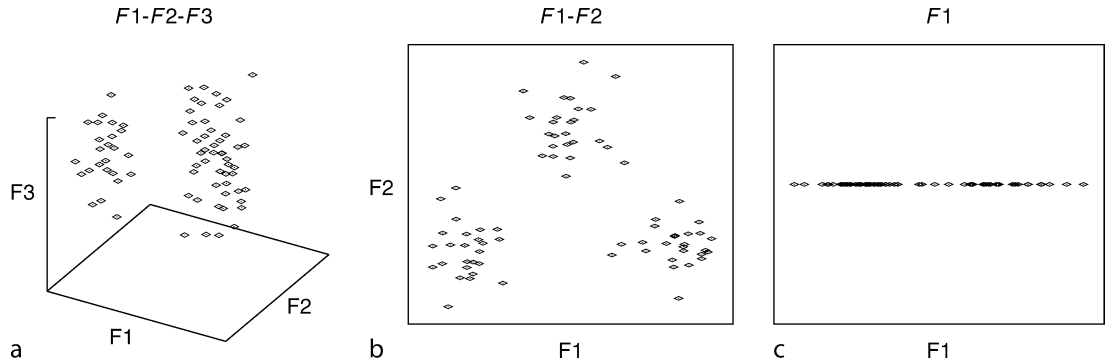
Definition

The problem of feature selection originates from the fact that while collecting data, one tends to collect all possible data. But for a specific learning task such as clustering not all the attributes or features are important. Feature selection is popular in supervised learning or for the classification task because the class labels are given and it is easier to select those features that lead to these classes. But for unsupervised data without class labels, or for the clustering task, it is not so obvious which features are to be selected. Some of the features may be redundant, some are irrelevant, and others may be "weakly relevant". The task of feature selection for clustering is to select "best" set of relevant features that helps to uncover the natural clusters from data according to the chosen criterion.

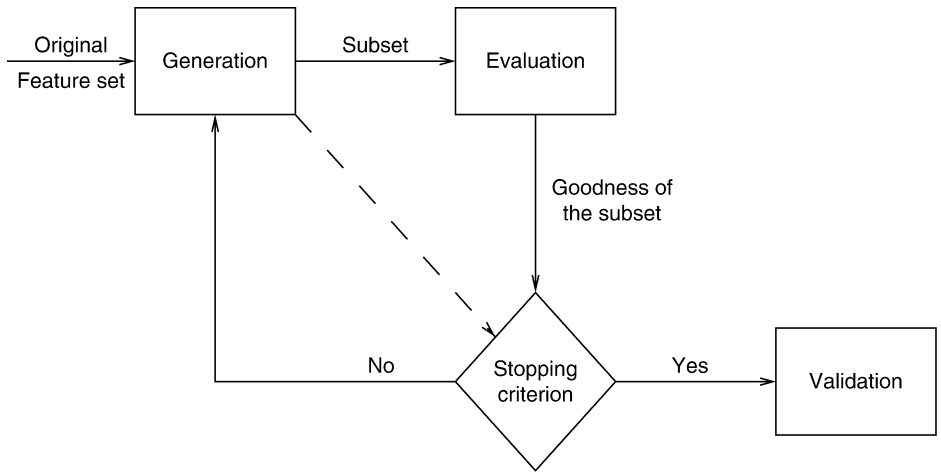
Figure 1 shows an example using a synthetic data. There are three clusters in $F1$–$F2$ dimensions which follow Gaussian distribution whereas $F3$, which does not define any cluster, follows a uniform distribution. When all three features are used the clusters are unnecessarily complex (see Fig. 1a), whereas no clusters can be found when one visualizes using only one feature $F1$ (Fig. 1c). Figure 1b with $F1$–$F2$ features shows three well-formed clusters. Selecting features $F1$ and $F2$ reduces the dimensionality of the data while forming well separated clusters. The goal of feature selection for clustering is to select important original features for clustering thus reducing the data size (and the computation time of other subsequent data mining tasks), solving the problem of curse of high-dimensionality and improving the knowledge discovery performance and comprehensibility. There are four basic steps in a typical feature selection method: a method to generate candidate subsets, an evaluation function, a stopping criterion, and a validation procedure (Fig. 2).

Historical Background

The literature for feature selection for classification, which is a supervised learning task, is very vast (see a review in [6]). On the other hand, there are only a few, mostly recent, feature selection methods for clustering, which is a unsupervised task. The reason behind this gap in research is it is easier to validate the selected features for classification (e.g., by accuracy of the classification) than to validate the selected features for clustering. Although there are many cluster validation techniques, the techniques have their limitations, and thus there is lack of unanimity among the researchers [19]. An issue that is yet to be resolved is how many clusters are there in a data set. Particularly in real-world



Feature Selection for Clustering. Figure 1. Effect of features on clustering.



Feature Selection for Clustering. Figure 2. Feature selection process with validation.

data sets often one encounters high-dimensional data sets with unknown number of cluster. It is shown that number of clusters and number of features have interrelation thus complicating the matter further [10].

Feature selection for clustering is the task of selecting important features for the underlying clusters. These methods can be divided using different categorization such as: global vs. local and wrapper (i.e., with feedback) vs. filter (i.e., without feedback – blind). Global methods select features for the whole data set whereas local methods select features for each individual cluster.

Examples of global methods are [5,12,17,22]. In [5] was proposed a filter method. A filter method is independent from the clustering algorithm to be used, whereas wrapper method uses the clustering method itself to select features. In this paper, the authors we proposed to measure entropy of the data set using point-to-point distances. Entropy is independent of number of features. Lower the entropy better the clustering. Features were ranked from most important to least important and a forward selection algorithm is used to select the relevant features. In [12] Dy and Brodley proposed a wrapper criterion for clustering. A candidate subset of features is used to cluster the data and the quality of clusters is evaluated using normalized cluster separability (for *K*-means) or normalized likelihood (for EM – expectation maximization – clustering). The bias on the feature subsets with respect to dimensionality is ameliorated by cross-projection normalization. In [11] the same authors used EM and trace measure (which is invariant for varying number of dimensions [9]) are used for evaluation. Visual aids to decide the optimal number of features were also proposed. The method described in [17] uses

K-means for evaluation of subsets of features. In [22] authors proposed an objective function for choosing the feature subset and finding the optimal number of clusters for a document clustering problem using a Bayesian statistical estimation framework.

Examples of local methods are [2,14,18,21]. In [14] authors proposed a distance measure. Using this measure with usual distance based clustering algorithms encourages the detection of subgroups of objects that preferentially cluster on subsets of features. The relevant feature subsets for each individual cluster can be different and partially or completely overlap with those of other clusters. In [18], feature saliency is integrated in EM algorithm so that feature selection is performed simultaneously with clustering process. Projected clustering (ProClus [1]) finds subsets of features defining (or important for) each cluster. ProClus first finds clusters using K-medoid considering all features and then finds the most important features for each cluster using Manhattan distance. The algorithm called CLIQUE in [2] divides each dimension into a user given divisions. It starts with finding dense regions (or clusters) in one-dimensional data and works upward to find higher-dimensional dense regions using candidate generation algorithm Apriori. In [20,21] Talavera used category utility to select features and these features are used to construct COBWEB [13] (COBWEB is a hierarchical clustering algorithm for categorical data.).

Foundations

This section first gives some insight into feature selection for clustering using an example method and then briefly discusses other methods and techniques used.

An Example of a Filter-Global Method

Figure 3 shows the histogram of point-to-point distances for two datasets: one with clusters and the other without clusters. The point-to-point distances are computed, normalized and are used to populate the bins of the histograms. An important distinction between the two histograms is that histogram for data without clusters has a predictable shape similar to bell shape. But the histogram for data with clusters has very different distribution. The smaller buckets (or distances) are mostly intra-cluster while the latter ones are mostly inter-cluster. *Typically, if the dataset consists of some clusters then, the majority of the intra-cluster distances will be smaller than the majority of the inter-cluster distances.* This observation is true for a wide range of data types. When clusters are very distinct this separation between intra-cluster and inter-cluster distances is quite distinguishable. In [5] a method is proposed which, without doing clustering, can distinguish between data with clusters and data without clusters.

Distance-based Entropy Measure From entropy theory it is known that entropy of a system can measure the amount of disorder in the system. Mathematically entropy of a dataset is given as:

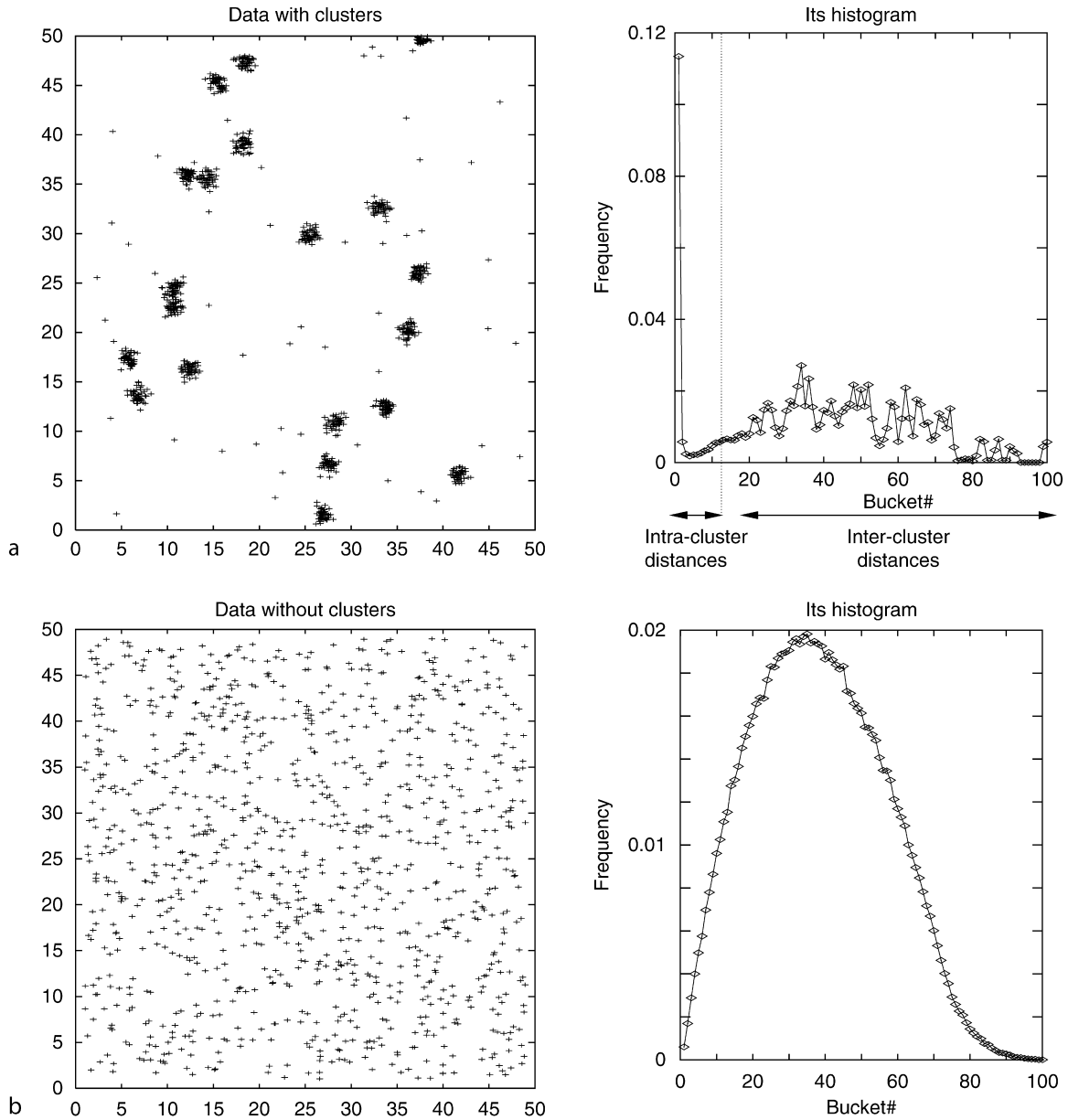
$$E = -\sum_{X_i} p(x_{i_1},\ldots,x_{i_M}) \log p(x_{i_1},\ldots,x_{i_M}) + (1 - p(x_{i_1},\ldots,x_{i_M})) \log(1 - p(x_{i_1},\ldots,x_{i_M})) \quad (1)$$

where $p(X_{i_1},\ldots,X_{i_M})$ is the probability or density at the point $(X_{i_1},\ldots,X_{i_M})$ where M is the dimensionality. The second term in the expression inside summation is used to make the expression symmetric. If the probability of each point is equal one is most uncertain about the outcome, and entropy is the maximum. This will happen when the data points are uniformly distributed in the feature space. On the other hand, when the data has well-formed clusters the uncertainty is low and so also the entropy. So, the entropy can be used to distinguish between data with clusters and data without clusters in different subspaces.

A straightforward method to compute the probability at each point is by substituting probability with distance in the following way:

$$E = -\sum_{X_i}\sum_{X_j} D_{ij} \log D_{ij} + \big(1 - D_{ij}\big) \log\big(1 - D_{ij}\big) \quad (2)$$

where D_{ij} is the normalized distance (This uses Euclidean (L_2) measure although other distances such as Manhattan (L_1) can be used.) in the range [0.0–1.0] between instances X_i and X_j. Figure 4a shows the relationship between entropy and distance after normalizing E to the range [0.0–1.0]. This measure assigns lowest entropy (0.0) for the minimum (0.0) or the maximum (1.0) distance and assigns highest entropy (1.0) for the mean distance (0.5). Although, to some extent, it works well in distinguishing data with clusters from data without clusters, it suffers from the following two drawbacks. (i) The mean distance of 0.5 can be an inter-cluster distance, but still it assigns the highest entropy. The reason is the meeting point



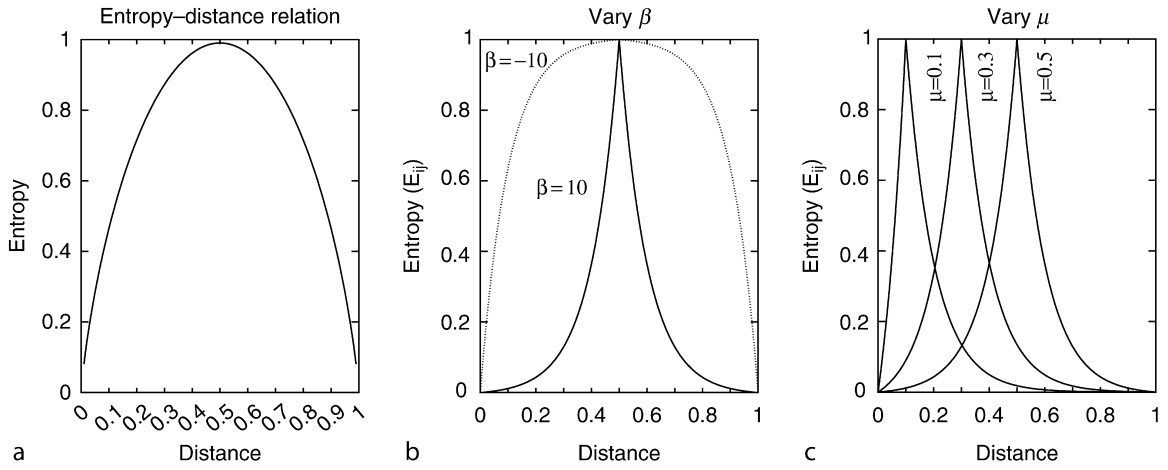
Feature Selection for Clustering. Figure 3. Distance histograms of data *with* and *without* clusters.

(μ) of the two sides (i.e., left and right) of the plot is fixed at distance 0.5. (ii) Entropy increases rapidly for very small distances thus assigning very different entropy values for intra-cluster distances. In summary, this measure does not work very well in assigning small entropy for both intra-cluster and inter-cluster distances. The second drawback can be easily overcome by incorporating a coefficient (β) in the equation and by using an exponential function in place of logarithmic function. Regarding the first drawback, the meeting point (μ) can be set so as to separate the intra-cluster and inter-cluster distances. Considering all these the following method was proposed:

$$E = \sum_{x_i} \sum_{x_j} E_{ij} \tag{3}$$

$$E_{ij} = \begin{cases} \frac{exp(\beta * D_{ij}) - exp(0)}{exp(\beta * \mu) - exp(0)} & : \quad 0 \le D_{ij} \le \mu \\ \frac{\exp(\beta * (1.0 - D_{ij} - \exp(0)}{\exp(\beta * (1.0 - \mu)) - \exp(0)} & : \quad \mu \le D_{ij} \le 1.0 \end{cases} \tag{4}$$

where E_{ij} is normalized to the range [0.01-1.0].



Feature Selection for Clustering. Figure 4. Relationship between entropy and distance with varying β and μ values.

Other Methods

Trace Measure In the wrapper and global methods feature subsets are evaluated by a clustering algorithm and the quality of clustering is evaluated using trace measure [12,7]. The Trace measure $(tr(S_W^{-1}S_B))$ is based on within and between cluster distances, where *tr* is *trace* os a matrix which is the sum of its diagonal elements; S_W is a within-cluster scatter matrix given as: $S_W = \sum_{i=1}^{c}\sum_{X\in\chi_i}(\mathbf{x}-\mathbf{m_i})(\mathbf{x}-\mathbf{m_i})^t$, and S_B is a between-cluster scatter matrix given as: $S_B = \sum_{i=1}^{c}(\mathbf{m_i}-\mathbf{m})(\mathbf{m_i}-\mathbf{m})^t$ where $\mathbf{x}$ is an instance vector, χ_i is the set of instances in *i*th cluster, $\mathbf{m_i}$ is the mean vector of *i*th cluster, $\mathbf{m}$ is the mean vector of the data. The higher the $tr(S_W{}^{-1}S_B)$, the higher the ratio of between-cluster to within-cluster scatter. If clustering is proper, then between cluster distance will be high and within cluster distance is low. Although there are many cluster validation techniques (see [16]), trace measure is especially selected because it is independent of number of features, i.e., irrespective of number of features in the subspace, trace measure can evaluate the clustering quality and compare among clusterings in different subspaces of equal or different dimensionality.

EM Method In [18], Law et al. cast the feature selection for clustering as an estimation problem rather than a search problem thus they avoided the combinatorial search through the feature space. Instead of selecting a subset of features they estimated a set of real-valued ([0,1]) quantities for each feature. They call it *feature saliency*. This estimation is carried out by an EM (Expectation Maximization) algorithm derived for the task. They avoided the situation where all the saliencies take the maximum possible value by adopting a minimum message length (MML) penalty. The MML criterion encourages the saliencies of the irrelevant features to go to zero. They combined the feature selection with EM clustering.

Conceptual Clustering Conceptual clustering is applicable for feature selection for clustering for data having only categorical features (e.g., COBWEB [13]). It creates a hierarchy or tree of clusters. As each data object is input to the system, the system categorizes the object by sorting it through hierarchy from the root node down to the leaves. At each level it determines whether (i) to create a new cluster, or (ii) to place the object in an existing cluster, and whether to restructure the hierarchy by (iii) merging two sibling clusters which were identified as the two best hosts for the new object or (iv) splitting the host cluster.

In [20,21] Talavera proposed two local methods for selecting features for conceptual clustering. In [21] Talavera used salience measure (this is different from the saliency measure in [18]) to select features. The higher the salience the greater is its relevance for the clustering. Salience measure originates from category utility measure.

Devaney and Ram [8] also used COBWEB to select features for clustering but they used it in a different way. The basic idea is to employ a wrapper approach with the average predictive accuracy over all features replacing the predictive accuracy of class labels. They used COBWEB as an evaluation function. The search through the feature space is either forward or backward. COBWEB is executed for each of these subsets

and category utility is computed of the first partition (children of the root) of the resulting concept hierarchy, retaining the highest score.

Important Applications

Feature selection has gained a wider audience in the past few years due to the high-dimensionality of databases. Due to the high number of low level features, a lot of the current methods of clustering in low dimensionality will fail miserably. Thus the only way is to identify the most important features and reduce dimensionality. In terms of applications, there are numerous possibilities.

An important application of feature selection is in the area of bioinformatics, using gene expression microarray data. Due to the nature of high dimensionality (thousands of genes) and sparsity of the data in feature space, clustering is exceptionally difficult. In [23] this issue is addressed using CLIFF, an algorithm based on normalized cut with their feature selection process. In [24] Malik et al. paper, he proposed another method for gene expression using recursive cluster elimination (RCE) with SVM, and claimed to have improved accuracy compared to other methods. In [14] an application of local feature selection for clustering is shown over Yeast gene expression data set with 6,141 genes (features) measured on 213 samples (instances).

Bekkerman et al. [4] applied feature selection with SVM to the problem of text categorization to yield high performance accuracy. Datasets used are: 20-Newgroups, Reuters-21578 and WebKB. The 20-Newsgroups contains 19,997 articles from the Usenet newsgroup collection, while Reuters-21578 corpus contains 21,578 articles from the Reuters newswire, and WebKB is a collection 8,282 web pages from four academic domain.

Bach et al. [15] applied feature selection to the reduction of attribute in image face recognition for male/female classification. Datasets consist of 1,450 images (1,000 train, and 450 for test) with 5,100 features.

In [3], feature selection is applied to PrimeClub, a game designed to teach prime numbers to sixth and seventh grade students. The objective is to keep the students continue learning, maintaining a high level of engagement as the game progresses. Biometric devices are attached to each student to detect the mental and emotional state of their expressions. In it, 28 features are recorded, and of those, only 4 are deemed important for clustering.

While not going into details, some of the other applications of feature selection are: customer relationship management, image retrieval, text mining, protein classification and intrusion detection.

URL

While not many author published their programs or code in the internet, there exist a few that do make theirs available for others to download and compare. "RCE classification and feature selection" as described in [24] is available for download at "http://showelab.wistar.upenn.edu/"

Mark Hall wrote a feature selection program for Weka, described in his PhD dissertation "Correlation-based Feature Subset Selection for Machine Learning."

CLOP is a Matlab package developed on top of the Spider for the WCCI 2006 performance prediction challenge, and it has produced some very good result in the NIPS 2003 feature selection challenge.

RapidMiner, a freely available open source for data mining and machine learning, has functionality for feature selection. It is written in Java and works on all major operating systems, available for download at "http://rapid-i.com/content/blogcategory/38/69/"

INTERACT is a feature selection method using inconsistency and symmetrical uncertainty measurements for finding interacting features. It is describe in "http://www.public.asu.edu/~huanliu/INTERACT/INTERACTsoftware.html," and available for download from the site.

Cross-references

- ▶ Cluster and Distance Measure
- ▶ Clustering Overview and Applications
- ▶ Curse of Dimensionality
- ▶ Data Cleaning
- ▶ Dimensionality Reduction
- ▶ Dimensionality Reduction Techniques for Clustering

Recommended Reading

1. Aggarwal C.C., Procopiuc C., Wolf J.L., Yu P.S., and Park J.S. Fast algorithms for projected clustering. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 61–72.
2. Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 94–105.
3. Amershi S., Conati C., and Maclaren H. Using feature selection and unsupervised clustering to identify affective expressions in educational games. In Proc. Workshop on Motivational and Affective Issues in ITS, 8th Int. Conf. on ITS, 2006, pp. 21–28.

4. Bekkerman R., El-Yaniv R., Tishby N., and Winter Y. Distributional Word Clusters vs Words for Text Categorization. J. Machine Learning Res. 3:1183–1208, 2008.
5. Dash M., Choi K., Scheuermann P., and Liu H. Feature selection for clustering – A filter solution. In Proc. 2002 IEEE Int. Conf. on Data Mining, 2002, pp. 115–122.
6. Dash M. and Liu H. Feature selection for classification. Int. J. Intell. Data Analy., 1(3):131–156, 1997.
7. Dash M. and Liu H. Handling large unsupervised data via dimensionality reduction. In Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 1999.
8. Devaney M. and Ram A. Efficient feature selection in conceptual clustering. In Proc. 14th Int. Conf. on Machine Learning, 1997, pp. 92–97.
9. Duda R.O. and Hart P.E. Pattern Classification and Scene Analysis, chap. Unsupervised learning and clustering. Wiley, New York, 1973.
10. Dy J.G. and Brodley C.E. Feature subset selection and order identification for unsupervised learning. In Proc. 17th Int. Conf. on Machine Learning, 2000, pp. 247–254.
11. Dy J.G. and Brodley C.E. Visualization and interactive feature selection for unsupervised data. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 360–364.
12. Dy J.G. and Brodley E. Feature Selection for Unsupervised Learning. J. of Machine Learning Res., 5:845–889, 2004.
13. Fisher D.H. Knowledge acquisition via incremental conceptual clustering. Mach. Learn., 2:139–172, 1987.
14. Friedman J. and Meulman J. Clustering objects on subsets of attributes. J. Royal Stat. Soc. B, 66(4):1–25, 2004.
15. Gilad-Bachrach R., Navot A., and Tishby N. Margin based feature selection – theory and algorithms. In Proc. 21st Int. Conf. on Machine Learning, 2004, pp. 43.
16. Jain A.K. and Dubes R.C. Algorithm for Clustering Data, chap. Clustering Methods and Algorithms. Prentice-Hall Advanced Reference Series, 1988.
17. Kim Y.S., Street W.N., and Menczer F. Feature selection in unsupervised learning via evolutionary search. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 365–369.
18. Law M.H.C., Figueiredo M.A.T., and Jain A.K. Simultaneous Feature Selection and Clustering Using Mixture Models. IEEE Trans. Pattern Analy. Mach. Intell., 26(9):1154–1166, 2004.
19. Milligan G.W. A monte carlo study of thirty internal criterion measures for cluster analysis. Psychometrika, 46(2):187–198, 1981.
20. Talavera L. Feature selection as a preprocessing step for hierarchical clustering. In Proc. 16th Int. Conf. on Machine Learning, 1999, pp 389–397.
21. Talavera L. Feature selection and incremental learning of probabilistic concept hierarchies. In Proc. 17th Int. Conf. on Machine Learning, 2000, pp 951–958.
22. Vaithyanathan S. and Dom B. Model selection in unsupervised learning with applications to document clustering. In Proc. 16th Int. Conf. on Machine Learning, 1999, pp. 433–443.
23. Xing E.P. and Karp R.M. CLIFF: clustering of high-dimensional microarray data via iterative feature filtering using normalized cuts. In Proc. 9th Int. Conf. on Intelligent Systems for Molecular Biology, 2001, pp. 306–315.
24. Yousef M., Jung S., Showe L.C., and Showe M.K. Recursive Cluster Elimination (RCE) for classification and feature selection from gene expression data. BMC Bioinformatics, 8:144, 2009.

Feature-Based 3D Object Retrieval

Benjamin Bustos[1], Tobias Schreck[2]
[1]Department of Computer Science, University of Chile, Santiago, Chile
[2]Darmstadt University of Technology, Darmstadt, Germany

Synonyms

Three-Dimensional similarity search; Shape descriptors

Definition

3D objects are an important type of data with many applications in domains such as engineering and computer aided design, science, simulation, visualization, cultural heritage, and entertainment. Technological progress in acquisition, modeling, processing, and dissemination of 3D geometry leads to the accumulation of large repositories of 3D objects. Consequently, there is a strong need to research and develop technology to support the effective retrieval of 3D object data from 3D repositories.

The feature-based approach is a prominent technique to implement content-based retrieval functionality for 3D object databases. It relies on extracting characteristic numerical attributes (so-called features) from a 3D object, usually forming high-dimensional vectors which represent the 3D object, or parts of it. The 3D feature vectors in turn are used to estimate object similarity for content-based retrieval, and can also be used for multidimensional indexing of 3D database content. There exist several degrees of freedom in obtaining 3D features. Important specifications to be made include the type of 3D characteristics or its level of detail considered, or invariance properties required, among others. Finding efficient and effective features for a given 3D repository is usually addressed by benchmarking.

Historical Background

The development of 3D object retrieval methods can be regarded as part of the larger multimedia retrieval

research area. The availability of increasing volumes of multimedia data such as digital images, digital video, or digital audio induced the need to develop content-based retrieval methods supporting these data types. Image retrieval has roots in image processing and database research of the 1980s, and was joined in the 1990s by similar efforts in the video and audio domains. Roughly, beginning by 2000, 3D objects increasingly came into focus of multimedia retrieval research. Driving motivation in the research and development of 3D retrieval methods are the increasing use of 3D object data in a range of application areas.

While retrieval of 3D objects is a prominent topic in multimedia database research, the definition of similarity notions for 3D data is also considered in related disciplines. In geometry processing, the registration or alignment of geometry is of interest, using certain definitions of geometric similarity. Computer vision is concerned with the recognition of objects in images taken by a camera or other scanner device, requiring appropriate segmentation and description methods for the objects in the scene under concern. In shape analysis, shapes and scenes are often analyzed for certain structural features, supporting e.g., classification and compression.

From the multimedia database research perspective, the focus of 3D retrieval implementations not only concerns the effectiveness of the retrieval, but also, their efficiency, demanding for real-time query processing on large repositories. The feature vector approach is therefore especially suited, as it allows the efficient evaluation of object similarity, usually by calculating a Minkowski distance between feature vectors representing underlying 3D objects.

Foundations

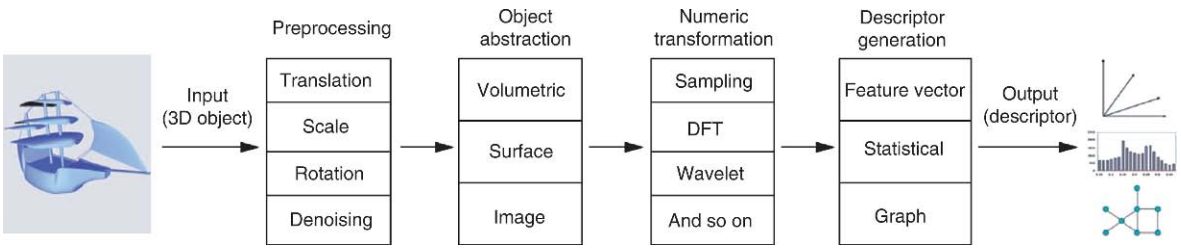
To represent 3D object data as points in feature vector space, it is necessary to find characteristics that describe the objects in a meaningful, discriminating way. A suitable feature extraction function calculates characteristic features from the 3D objects, thereby mapping them into d-dimensional feature vector space. With these feature vector representations, a similarity query in the original 3D object space is reduced to a search for close points in d-dimensional feature vector space.

Common Requirements of 3D Feature Extraction

For 3D object data, based on the given retrieval application, certain properties of the features extracted can be deemed desirable. The features may be required to be invariant with respect to changes in rotation, translation, and scale of the 3D models in their reference coordinate frame. Ideally, an arbitrary combination of translation, rotation and scale applied to one object should not affect its similarity measure with respect to another object. Another desirable property is robustness with respect to variation of the level-of-detail in which the 3D objects are given, and to small geometry and topology variations of the models. These invariance and robustness properties are especially important if the retrieval is expected to support 3D objects from heterogeneous data sources. This is because in such cases, the reference frames or levels-of-detail in which the models are represented may differ, and it cannot be assumed that respective meta data is available from all possible object sources.

3D Feature Extraction Process Model

A process model of 3D feature extraction is depicted in Fig. 1 and can be described as follows. Firstly, if required by the application, a preprocessing step normalizes the 3D object to approximate invariance to rotation, translation, scaling, and reflection [11]. A second step abstracts the 3D object according to a selected shape characteristic. For example, one can abstract a 3D object as a volume, or as an infinitely thin surface with precisely defined properties of differentiability, or as a set of 2D images formed by



Feature-Based 3D Object Retrieval. Figure 1. Feature extraction process for 3D objects.

projections from different perspectives. The third step captures the main features of the 3D object under the selected abstraction by means of a numeric transformation. As a result of this step, a numerical representation of the original 3D object is obtained. The last step of the feature extraction process model produces the final descriptor of the object from the numerical description. Generally, the descriptor may be a vector of numerical features, but it may also be a histogram of the measured characteristics, or a graph-based representation of the analyzed 3D object. Feature-based methods for 3D model retrieval usually are efficient, robust, and easy to implement. This does not imply, however, that statistical or graph-based methods should be disregarded. In fact, most of those methods have their particular strengths and may well be the ideal candidate for a specific application.

3D Feature Types

As surveys indicate [2,7,10], there is a wealth of different features that so far have been used to build 3D retrieval systems. The situation is comparable to content-based image retrieval (CBIR), where also, many different features have been proposed over the recent years. It can be stated that many of the 3D features proposed were heuristically introduced, motivated by techniques and practices from computer graphics (e.g., projection-based features), geometry processing (e.g., features based on surface curvature statistics), or Signal Processing (e.g., features obtained by representing object samples in the frequency domain). Some of the most effective 3D feature vector extractors proposed to date rely on features extracted from 2D projections of 3D objects.

Usually, it is a priori unclear which of the potentially many different features should be preferred for addressing the 3D retrieval problem. Each of the many possible descriptors captures specific model information, and their suitability for effective retrieval in a given application domain needs to be experimentally evaluated. In practice, it often shows that the effectiveness of 3D retrieval systems can benefit from using not a single, but several different types of features in combination.

Efficient 3D Object Retrieval

Similarity queries in 3D object databases may be answered by performing a sequential scan on the database, comparing the query object with all 3D objects stored in the database. This naive method might be too slow for real-world applications. In feature-based 3D object retrieval, the search system may use an index structure (e.g., spatial access methods or metric access methods) for efficient retrieval if the distance function used to compute the (dis)similarity of two 3D objects holds the properties of a metric (strict positiveness, symmetry, and the triangle inequality).

Spatial access methods [1] (also known as multidimensional indices) are index structures especially designed for vector spaces which, together with the metric properties of the distance function, use geometric information to discard points from the search space. Usually, these indices are hierarchical data structures that use a balanced tree to index the database. Metric access methods [4] (also known as metric indices) are index structures that use the metric properties of the distance function (especially the triangle inequality) to filter out certain zones of the space, thus avoiding the sequential scan.

Key Applications

Content-based 3D retrieval methods are potentially useful in all applications involving 3D object repositories, from which elements need to be retrieved based on geometric similarity. Several exemplary applications are detailed in the following, more exist.

Industrial Applications

Engineering and industrial design, the animation, and the entertainment industry heavily rely on digitized models of products or parts thereof. Computer-aided design allows the digital modeling of 3D content. Given effective retrieval capabilities, the re-usage of content from existing repositories can be supported for more efficient production processes [5].

Medicine

In medical imaging applications, often 3D volume data is generated, e.g., using MRI scans. A possible application lies in automatic diagnosis support by analysis of organ deformations, by matching actual images with medical database of known deformations.

Molecular Biology

Structural classification is a basic task in molecular biology. This classification can be supported by geometric similarity search, where proteins and molecules are modeled as 3D objects, which can be compared

against bio-molecular reference databases using geometric similarity measures.

Future Directions

Feature-based 3D retrieval research is still in a rather early stage. Current approaches mostly consider features describing the geometry of whole models, that is, they support global similarity between objects. Recently, approaches also considering local features based on identification of salient object regions have been proposed. These are expected to support not only the retrieval of complete models, but also, be suited for retrieval based on local similarity. Future work will address 3D retrieval under additional similarity models, including similarity models invariant with regard to non-rigid and structural object deformations. It is also expected that application specific, specialized similarity notions will become increasingly important.

Experimental Results

The effectiveness of 3D features for retrieval is usually determined experimentally based on reference benchmarks, and measured by information retrieval metrics [3]. Well-known 3D retrieval benchmarks include the Princeton Shape Benchmark [9] and the Purdue Engineering Shape Benchmark [8]. The SHREC contest [6] is an International shape retrieval contest held regularly that involves different 3D retrieval challenges.

Cross-references

▶ Feature Extraction for Content-Based Image Retrieval
▶ Index Structures for Biological Sequences
▶ Information Retrieval
▶ Multimedia Information Retrieval Model
▶ Multimedia Retrieval Evaluation

Recommended Reading

1. Böhm C., Berchtold S., and Keim D. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. ACM Comput. Surv., 33(3):322–373, 2001.
2. Bustos B., Keim D., Saupe D., Schreck T., and Vranić D. Feature-based similarity search in 3D object databases. ACM Comput. Surv., 37(4):345–387, 2005.
3. Bustos B., Keim D., Saupe D., Schreck T., and Vranić D. An experimental effectiveness comparison of methods for 3D similarity search. Int. J. Digit. Lib., Special issue on Multimedia Contents and Management in Digital Libraries, 6(1):39–54, 2006.
4. Chávez E., Navarro G., Baeza-Yates R., and Marroquín J. Searching in metric spaces. ACM Comput. Surv., 33(3):273–321, 2001.
5. Funkhouser T., Kazhdan M., Shilane P., Min P., Kiefer W., Tal A., Rusinkiewicz S., and Dobkin D. Modeling by example. ACM Trans. on Graphics, 23(3):652–663.
6. http://www.aimatshape.net/event/SHREC S.I.S.R.C.
7. Iyer N., Jayanti S., Lou K., Kalyanaraman Y., and Ramani K. Three Dimensional Shape Searching: State-of-the-art Review and Future Trends. Comput. Aided Design, 37(5):509–530, 2005.
8. Jayanti S., Kalyanaraman Y., Iyer N., and Ramani K. Developing an engineering shape benchmark for CAD models. Comput. Aided Design, 38(9):939–953, 2006.
9. Shilane P., Min P., Kazhdan M., and Funkhouser T. The Princeton Shape Benchmark. In Proc. Int. Conf. on Shape Modeling and Applications, 2004, pp. 167–178.
10. Tangelder J. and Veltkamp R. A survey of content based 3D shape retrieval methods. In Proc. Int. Conf. on Shape Modeling and Applications, 2004, pp. 145–156.
11. Vranić D., Saupe D., and Richter J. Tools for 3D-Object Retrieval: Karhunen-Loeve Transform and Spherical Harmonics. In Proc. IEEE 4th Workshop on Multimedia Signal Processing, 2001, pp. 293–298.

Federated Database

▶ Distributed Architecture

Federated Database Systems

▶ Distributed Database Systems

Federated Search

▶ Searching Digital Libraries

Federated Search Engine

▶ Metasearch Engines

Feedback Systems

▶ Reputation and Trust
▶ Trust and Reputation in Peer-to-Peer Systems

Field-Based Information Retrieval Models

Vassilis Plachouras
Yahoo Reasearch Barcelona, Barcelona, Spain

Definition

A document D consists of a set of n document fields, and it is represented by a set of n vectors, where each vector corresponds to a document field. A field-based Information Retrieval (IR) model assigns a score or Retrieval Status Value (RSV) to a document D and a query Q by distinguishing the occurrences of query terms in the different field vectors, and by weighting the contribution of each field appropriately.

Historical Background

Textual documents, whether they are news wire items, scientific publications, or Web pages, are rich in structure. For example, depending on its length, a text can be organized in chapters, sections, paragraphs, and each of those can have a concise description in the form of a title. Shorter texts, such as emails, also consist of free text and formatted text. In information retrieval (IR), however, documents are usually represented as a single vector, the dimensions of which correspond to terms occurring in the document. Such a representation ignores the structure within a document because it does not distinguish between the occurrence of terms in different parts of a document, such as the document title, or the document abstract.

Field-based IR models overcome this limitation of representing the document as one vector, or as a bag-of-words, by incorporating in the document weighting process the fact that a term may appear in different document fields, and assigning different importance to the occurrences in each field. Information retrieval with document fields has a long history. Switzer [14] described a model where each index term is a vector image of the basic index terms. A document is represented by a title image, an author image, and citation images, constructed from bibliographic references. In such a representation, the title image is the average of vector images of the index terms appearing in the title. Fox [2] extended the boolean model and the vector space model with multiple concept types or fields. In the extended vector space model, a document is represented as a vector of subvectors. The ranking of documents in the extended vector space model is performed according to the weighted sum of the similarities between each of the subvectors and the queries. Fox also investigated the characteristics of several concept types related to bibliographic citations, as well as the weighting of the contribution of each of the subvectors [3]. Wilkinson studied the contribution of different fields, or document representations to retrieval effectiveness and found that improvements in early precision were obtained by using both the whole documents and their fields [16].

Foundations

There have been several proposed models to combine information from different document fields. One approach involves the use of structured document retrieval models, which allow for contained elements. Myaeng et al. [10] proposed a model in which documents and their elements or fields are represented as nodes in a network. Then, elements are ranked according to their support for the query, considering the contained elements as well. Lalmas [6] introduced a model where a document is represented as a tree. The leaf nodes of the tree correspond to the elements or fields of the document and the non-leaf nodes correspond to aggregates of its child nodes.

One other approach involves performing retrieval independently from each field and then, merging the ranked lists of results [1]. In this case, the combination of document fields takes place by merging the ranked lists of documents corresponding to each document field. This approach can be used to combine any source of evidence since it is based on the ranks of the retrieved documents, and not on their scores.

In the context of language modeling, the combination of fields, or different document representations, can be achieved with a linear combination of language models computed for each of the fields or document representations [11]. Similarly to the extended vector space model introduced by Fox [2], the combination of the different fields is also performed by computing a score or RSV for each of the document fields independently, and then performing a weighted sum of the computed scores.

The integration of information from the different fields in a retrieval model is not straightforward, because each document field may have different characteristics and the term frequency distribution can be different [5]. Thus, performing normalization and weighting independently for the various fields allows

to take into account the different characteristics of the fields, and to achieve their most effective combination.

Robertson et al. [13] suggested that it is more appropriate to weight and combine the frequencies of terms from different fields in a *pseudo-frequency*, before applying a term weighting model. They argue that it is more intuitive to combine term frequencies rather than scores or RSVs assigned by scoring functions, which are not necessarily linear with respect to the term frequencies. Indeed, the linear combination of scores computed for each of the fields independently can lead to an overestimation of the importance of a term in a document. Then, Robertson et al. [13] proposed extending the BM25 weighting model to perform a weighted combination of document fields. The BM25 weighting model is given as follows:

$$w(D,Q) = \sum_{t \in D \cap Q} \frac{(k_1+1)tf}{(k_1(1-b)+b(l/\bar{l}))+tf} \cdot \log \frac{N-n_t+0.5}{n_t+0.5} \quad (1)$$

where k_1, k_3, and b are free parameters, tf is the frequency of term t in D, l is the length of D, $\bar{l}$ is the average document length in the collection of documents, N and n are the number of documents and the document frequency of t in the collection. In the above equation, the frequency of terms in the query Q is not included. The proposed extension replaces tf in the above equation with a weighted sum of the term frequencies from each of the n document fields:

$$tf = \sum_{i=1}^{n} w_i \cdot tf_i \quad (2)$$

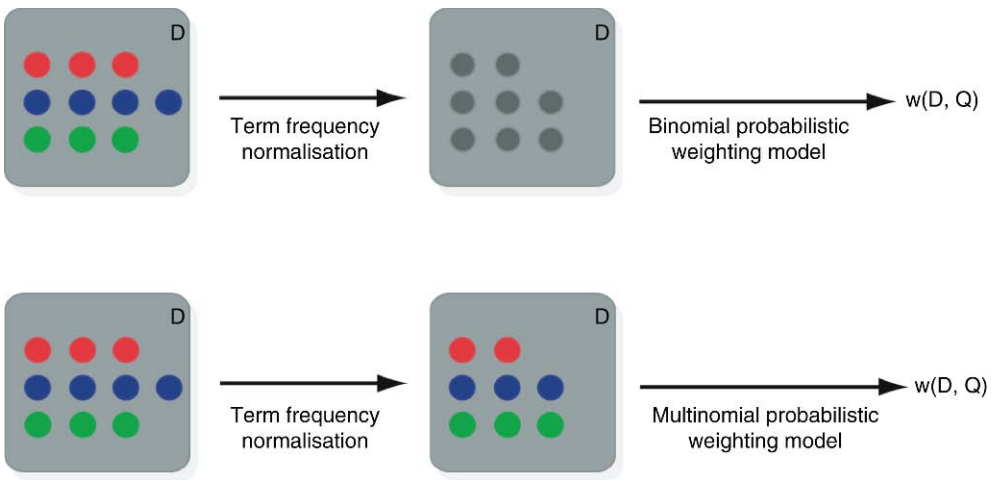
where tf_i is the frequency of term t in the i-th field and w_i is the weight of the i-th field. The described extension gives only a partial solution because it does address the combination of fields, but the term frequency normalization component is applied after computing the weighted sum of the term frequencies.

Zaragoza et al. [17] proposed BM25F, an extension of BM25 where length normalization is applied on a per-field basis. The formula of BM25F is given below:

$$w(D,Q) = \sum_{t \in Q} \frac{tfn}{k_1+tfn} \cdot \log \frac{N-n_t+0.5}{n_t+0.5}$$
$$\text{where } tfn = \sum_{i=1}^{n} w_i \cdot \frac{tf_i}{(1+b_i(l_i/\bar{l}_i))} \quad (3)$$

In the above equation, b_f is a field-dependent normalization parameter, similar to the parameter b of BM25, k_1 is a parameter that controls the saturation of tfn, similar to the parameter k_1 of BM25; $\bar{l}_i$ is the average length of the i-th field in the document collection; and l_i is the length of the i-th field in D. The parameter w_i is the weight of the i-th field.

The Divergence From Randomness (DFR) framework has also been extended to handle multiple document fields, and to apply per-field term frequency normalization and weighting. DFR is a framework for generating families of probabilistic retrieval models consisting of three components. The weighting



Field-Based Information Retrieval Models. Figure 1. An illustration of combination of document fields in the term frequency normalization component(top) and in the probabilistic retrieval model (bottom).

models of the Divergence From Randomness framework are based on combinations of three components: a randomness model $\mathcal{RM}$; an information gain model $\mathcal{GM}$; and a term frequency normalization model. Given a collection C of documents, the randomness model $\mathcal{RM}$ estimates the probability $P_{\mathcal{RM}}(t \in D|C)$ of having tf occurrences of a term t in D. The information gain model $\mathcal{GM}$ estimates the informative content $1 - P_{risk}$ of the probability P_{risk} that a term t is a good descriptor for a document. The third component of the DFR framework is the term frequency normalization model, which adjusts the frequency tf of the term t in d, given the length l of d and the average document length $\overline{l}$ in D.

There have been two ways proposed to extend the DFR weighting models with document fields. The first way to incorporate fields is by extending the term frequency normalization component of the DFR framework [12]. The frequency tf_i of term t in the i-th field is normalized and weighted independently of the other fields. Then, the normalized and weighted term frequencies are combined into one pseudo-frequency.

When combining the term frequencies in the length normalization component, as it happens in BM25F and in normalization 2F, it is implied that the term frequencies are drawn from the same distribution, even though the nature of each field may be different. The second way to incorporate fields in the DFR framework is by using multinomial randomness models [12]. Using the multinomial distribution, the probability that a term occurs tf_i times in the i-th field of D is given as follows:

$$P_{\mathcal{M}}(t \in D|C) = \binom{TF}{tf_1 \quad tf_2 \cdots tf_n \quad tf'} p_1^{tf_1} p_2^{tf_2} \cdots p_n^{tf_n} p'^{tf'} \tag{4}$$

In the above equation, TF is the frequency of t in the collection, $p_i = \frac{1}{n \cdot N}$ is the prior probability that a term occurs in a particular field of D, and N is the number of documents in the collection C. The frequency $t' = TF - \sum_{i=1}^{n} tf_i$ corresponds to the number of occurrences of t in documents other than D. The probability $p' = 1 - n\frac{1}{n \cdot N} = \frac{N-1}{N}$ corresponds to the probability that t does not appear in any of the fields of D. The use of the multinomial distribution as a randomness model requires the computation of several factorials, which can be expensive and also may introduce approximation errors. To overcome the need to compute these factorials, Plachouras and Ounis [12] also used an information theoretic approximation of the multinomial distribution.

The models described above require additional information from the index to compute efficiently scores of documents. In both the cases of BM25F and the DFR field-based weighting models, the index must contain the frequency of a term in each of the document fields, rather than just the frequency of the term in the whole document. For each document, it is also required to have the length of each field to perform length normalization on a per-field basis.

Key Applications

There is a wide range of applications for which field-based IR models enhances the retrieval effectiveness. One of the most important and widely used ones is Web search, where useful fields are the title of Web documents as well as the anchor text of incoming hyperlinks. A different application in which retrieval effectiveness increases by using document fields is email search [17] where the email from, to and date, as well as the quoted text, can be considered as a different field. In general, field-based IR models can be readily applied to search tasks in which documents have metadata elements associated or in which parts of the documents are annotated.

Future Directions

The introduction of fields increases the complexity of the retrieval models by introducing parameters related to the weighting of each field, but also to other factors such as the length normalization of each field. As an example, BM25F [17] requires the tunning of $2n + 1$ parameters when using n different document fields. Similarly, PL2F [8] introduces $2n$ parameters when using n different fields. Since the fields are not necessarily independent, setting the parameters for one field depends also on the parameter setting for the other ones. As the number of fields and the number of parameters increases, performing an exhaustive search becomes prohibitive. The application of an extension of gradient descent has been proposed to set the document field parameters among other parameters [15]. Developing parameter-free field-based IR models, however, is still an open problem.

Data Sets

There are several available data sets that allow experimentation with field-based IR models. The most commonly

used ones are the standard TREC Web test collections WT10g, .GOV and .GOV2, which have been used for a range of search tasks, such as ad-hoc retrieval, named page and home page finding, as well as topic distillation. All these tasks have been evaluated in the context of Web track of Text REtrieval Conference (TREC) [4]. Another standard test collection that has been used for expert search finding and email search in mailing lists is the W3C collection, a crawl of the World Wide Web Consortium Web site available from TREC as well.

The INitiative for the Evaluation of XML Retrieval (INEX) [9], has also developed collections using XML. The first one consists of articles marked up in XML from a number of the IEEE Computer Society's publications. Other collections correspond to the contents of Wikipedia and to the contents of travel guides, marked up in XML.

Cross-references

- ▶ BM25
- ▶ Content-and-Structure-Query
- ▶ Divergence from Randomness Models
- ▶ Document Length Normalization
- ▶ Information Retrieval Model
- ▶ Vector Space Model

Recommended Reading

1. Fagin R., Kumar R., McCurley K.S., Novak J., Sivakumar D., Tomlin J.A., and Williamson D.P. Searching the workplace web. In Proc. 12th Int. World Wide Web Conference. 2003, pp. 366–375.
2. Fox E.A. Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types. Ph.D dissertation, Cornell University, 1983.
3. Fox E.A. Coefficients of combining concept classes in a collection. In Proc. 11th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1988, pp. 291–307.
4. Hawking D. and Craswell N. The very large collection and Web tracks. In TREC: Experiment and Evaluation in Information Retrieval, E. Voorhees, D. Harman (eds.). MIT, Cambridge, MA, USA, 2005, pp. 199–232.
5. Hawking D., Upstill T., and Craswell N. Toward better weighting of anchors. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 512–513.
6. Lalmas M. Uniform representation of content and structure for structured document retrieval. Technical report, Queen Mary University of London, 2000.
7. Macdonald C. and Ounis I. Combining fields in known-item email search. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 675–676.
8. Macdonald C., Plachouras V., He B., Lioma C., and Ounis I. University of Glasgow at WebCLEF 2005: experiments in per-field normalisation and language specific stemming. In Accessing Multilingual Information Repositories, Sixth Workshop of the Cross-Language Evaluation Forum, 2005, pp. 898–907.
9. Malik S., Trotman A., Lalmas M., and Fuhr N. Overview of INEX 2006. In Comparative Evaluation of XML Information Retrieval Systems. LNCS 4518, Springer, Berlin, 2007, pp. 1–11.
10. Myaeng S.H., Jang D.H., Kim M.S., and Zhoo Z.C. A flexible model for retrieval of SGML documents. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 138–145.
11. Ogilvie P. and Callan J. Combining document representations for known-item search. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 143–150.
12. Plachouras V. and Ounis I. Multinomial randomness models for retrieval with document fields. In Proc. 29th European Conf. on IR Research, 2007, pp. 28–39.
13. Robertson S., Zaragoza H., and Taylor M. Simple BM25 extension to multiple weighted fields. In Proc. Int. Conf. on Information and Knowledge Management, 2004, pp. 42–49.
14. Switzer P. Vector images in information retrieval. In Proc. Symp. on Statistical Association Methods for Mechanical Documentation,1965, pp. 163–171.
15. Taylor M., Zaragoza H., Craswell N., Robertson S., and Burges C. Optimisation methods for ranking functions with multiple parameters. In Proc. Int. Conf. on Information and Knowledge Management, 2006, pp. 585–593.
16. Wilkinson R. Effective retrieval of structured documents. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval. 1994, pp. 311–317.
17. Zaragoza H., Craswell N., Taylor M., Saria S., and Robertson S. Microsoft Cambridge at TREC-13: Web and HARD tracks. In Proc. 13th Text Retrieval Conf., 2004.

Field-Based Spatial Modeling

Michael F. Goodchild
University of California-Santa Barbara, Santa Barbara, CA, USA

Definition

A field (or continuous field) is defined as a mapping from location **x** to a function *f*. In modeling geographic phenomena the domain of **x** is most often the two dimensions of geographic space, but may include the third spatial dimension for applications that extend above or below the Earth's surface, and may include time for dynamic phenomena. Fields can also be defined on one-dimensional networks embedded in two- or three-dimensional space. Moreover, most applications are limited to a specified sub-domain of geographic

space, such as the limits of a country or county, or of a map sheet or arbitrarily defined study area. The domain of *f* includes scalar measurements on interval and ratio scales, nominal and ordinal classifications, and vectors describing such directional phenomena as wind or topographic gradient. Field-based spatial modeling can in principle be employed in the representation of any space, including the spaces of the human brain, the surfaces of other planets, or complex buildings.

Fields are one of two ways of conceptualizing the geographic world. By contrast, the discrete-object conceptualization imagines a world that is empty except where it is occupied by discrete, countable objects that maintain identity and geometric form through time. This conceptualization is more often adopted in the representation of biological organisms, manufactured objects, and man-made structures, whereas the continuous-field conceptualization is more appropriate for variables that can be defined at every location in space, such as air temperature, terrain height, soil moisture content, or wind speed.

Historical Background

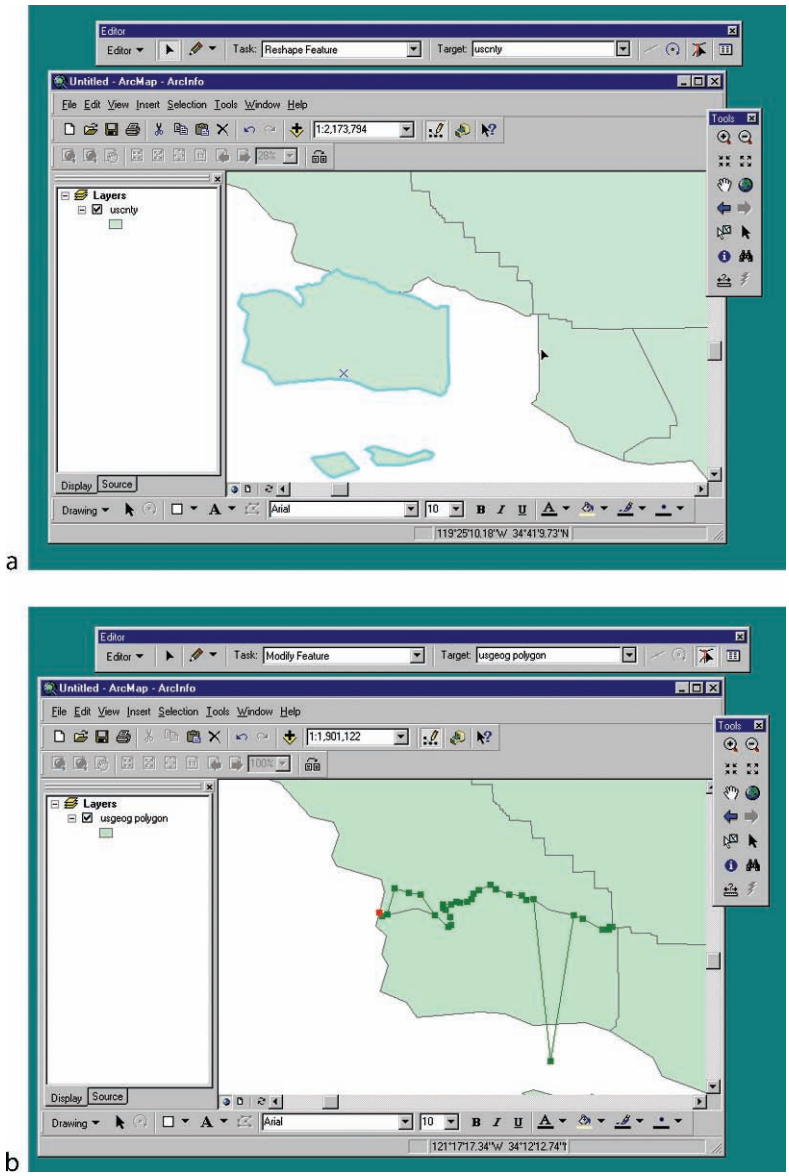
The field/object distinction has ancient roots. Its significance for geographic information science was first recognized in the late 1980s and early 1990s [1,3], and it has come to be acknowledged as the most important distinction underpinning the entire field of geographic data modeling [5,7]. Consider, for example, the section of the US county boundary map shown in Fig. 1. If counties are regarded as discrete objects then one should be able to move them around as pieces of a jigsaw, possibly overlapping and modifying the coastline of the US, as shown on the left. On the other hand if the variable *county* is regarded as a single-valued function of location **x**, defined everywhere inside the boundary of the US, then the boundaries merely indicate where the value of *county* changes. Edits that move common boundaries are feasible, but not edits that move the coastline, or edits such as that shown, which attempts to move vertices of the boundary outside the coastline. Conceptualizing the phenomenon as a field clearly results in a more consistent representation that is more appropriate to the nature of the phenomenon.

Figure 2 shows an area-class map, depicting the variation in vegetation cover class across a geographic domain. If such a map were conceptualized as a collection of discrete objects, then the issue of accuracy would come down to such questions as "Is the number of areas correct?," "Are the boundaries in the correct places?," and "Are the classes assigned to each area correct?" However uncertainty would be addressed very differently under a field conceptualization; one could ask only one form of question, "Is the value recorded at **x** correct?" It turns out that this second conceptualization is far more productive and tractable than the first [8].

Strong associations have been recognized between the two alternative conceptualizations and patterns of human thought. Many would argue that the discrete-object view is more compatible with human cognition – that human brains are in effect hard-wired to segment any visual image into a collection of discrete objects, and to track their movements through time. On the other hand the continuous-field view underlies many of the most significant advances in science, including electro-magnetism (the Maxwell equations), hydrodynamics (the Navier-Stokes equation), and quantum mechanics (the Schrödinger equation). Some of the most challenging problems in science are described in partial differential equations defined on fields, and solved using a variety of computational methods. The distinction is strongly linked to concepts of scale, as for example when the behavior of a group of ants is modeled at very detailed scale as interactions between discrete objects, or at a coarser scale as modifications to a continuous field of ant density. Very crudely, continuous-field conceptualizations tend to be more common in the natural and physical sciences, while discrete-object conceptualizations are more often found with reference to social phenomena.

Foundations

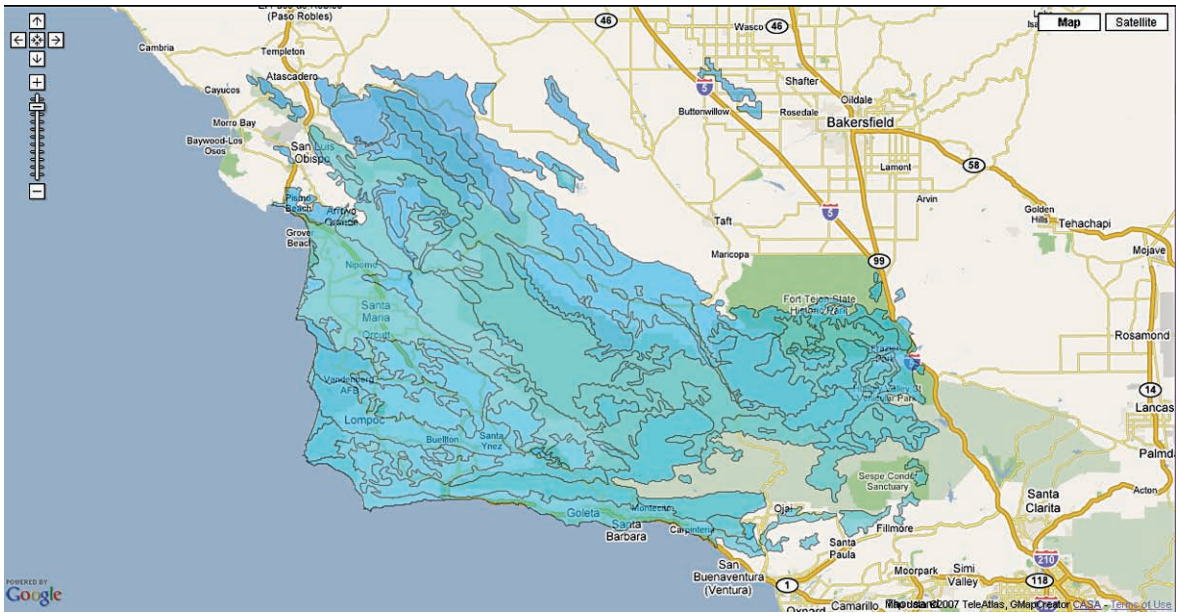
Computers are fundamentally discrete machines, founded on the representation of information in a two-valued alphabet, and as a result discrete-object conceptualizations are more readily implemented in spatial databases. There is little ambiguity, for example, in the representation of the current location of every aircraft in an airline's fleet as a point in space. However other phenomena are inherently continuous across space, including terrain, rivers, roads, and the tracks of moving objects, and their representation necessarily involves some form of discretization. A river, for example, may be broken into reaches, either at junctions or at points where the direction of the river changes significantly. Each reach will be represented using a simple mathematical function, most often a straight



Field-Based Spatial Modeling. Figure 1. Counties conceptualized as (a) discrete objects and (b) a continuous field. As a discrete object, any county is free to move independently of its neighbors and the coastline. As a continuous field, however, only the common boundaries where county values change can be moved, and not so far as to intersect the coastline (the edit shown would not be accepted).

line but sometimes as an arc of a circle or a spline function. Roads in a road network may be broken into segments at intersections, or at other points where direction changes. These stages necessarily modify the phenomena they are used to represent, since the discretized object will in most cases differ geometrically from the original, and discretization is essential if a geometric shape that is potentially infinitely complex is to be represented in a digital store of finite capacity. Thus in almost all cases the representation of real-world geometry requires the loss of some degree of detail.

This stage of discretization is necessary whenever an arbitrarily shaped geographic feature must be represented in digital form. But a further stage is needed when the characteristics of the phenomenon also vary continuously over space, in other words in the representation of phenomena conceptualized as continuous fields. Figure 3 shows the six methods most commonly



Field-Based Spatial Modeling. Figure 2. A map of vegetation cover class conceptualized as a nominal field, superimposed on the Santa Barbara area. Each area denotes a particular type of vegetation.

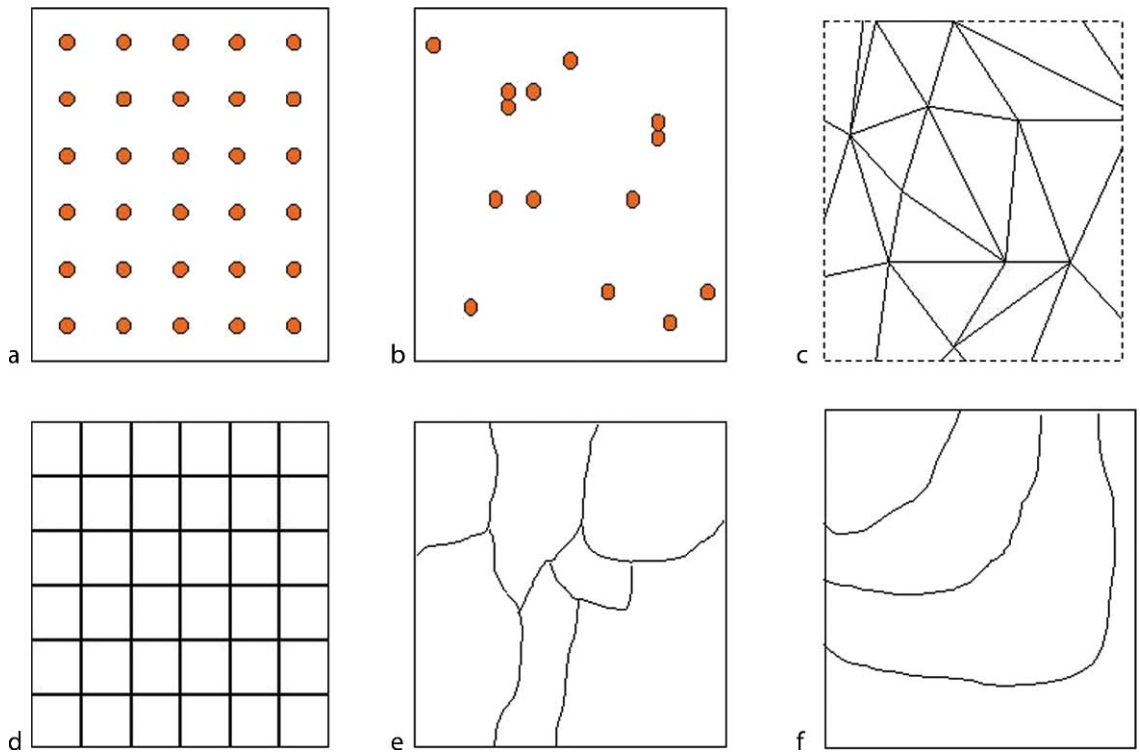
employed in spatial databases and geographic information systems. Two are based on a raster, in other words the discretization of space into a regular array with finite spacing, while the remaining four use vector methods, specifying the position of each element of the discretization as coordinates and representing volumes as collections of polygonal faces, areas as polygons, and lines as polylines.

Figure 3a shows one of the raster options, the representation of a spatially continuous phenomenon as a collection of sample values regularly spaced over a rectangular array. This is the method most commonly employed in the representation of terrain, in what are known as digital elevation models (DEMs). The spacing of sample heights implies a well-defined level of spatial resolution. Note, however, that since it is impossible to lay a rectangular grid over a curved surface, it is similarly impossible to construct a DEM of a significant part of the Earth's surface with a precisely constant spatial resolution.

Figure 3d shows the other raster option. Here the study area has been divided into rectangular cells, and a single measurement provided in each cell. This is often the mean value in the case of measurements on interval and ratio scales, but in other cases may be a modal value or some more complex function of the values of the field within the cell. Again spatial resolution is constant and well defined. This approach is most commonly encountered in the discretization of images, such as those obtained from satellites.

Figure 3b shows a vector option. In this case the field has been sampled at a set of irregularly spaced locations, and the value of f reported. Sample locations may be selected using some specified rule, or may be a historic artifact, as they are in the case of weather data, when the points correspond to weather-observation stations. Nothing is known about the values of f between sample locations, though a wide range of techniques exist for making intelligent guesses under the rubric of spatial interpolation.

Figure 3c shows another vector option involving irregularly spaced sample points, though in this case a set of triangles have been created by connecting them. The field variable is assumed to vary linearly between points, ensuring continuity of value across triangle edges. In essence this option, which is generally known as a triangulated irregular network (TIN) or triangular mesh, adds a specific method of spatial interpolation to Option 3b. It is commonly used as a method for representing terrain, where it is particularly efficient for terrains characterized by long uniform slopes and sharp ridges; and as an internal representation in contouring algorithms.



Field-Based Spatial Modeling. Figure 3. The six common methods of discretizing a field. See text for explanation.

Figure 3e shows the discretization of a field as a collection of space-exhausting, non-overlapping areas, that will themselves be discretized geometrically as polygons. The field variable is assumed uniform within each area. This is the most common discretization when the field variable is nominal or ordinal, as it is for maps of soil class and land cover class for example. It is also commonly used to represent data collected by statistical agencies and aggregated by reporting zones such as counties or census tracts. In such cases, the number of variables recorded for each reporting zone may be very large. For some of these variables, termed spatially intensive, the value reported will represent the mean within the zone of a variable such as average income, or a proportion such as percent black, or a density per unit area. In other cases the reported value will be a total, such as total population or total income, corresponding to the integration of a density over the reporting zone's area; this type of value is termed spatially extensive.

Finally, Fig. 3f shows the last example, in which the field is represented as a collection of digitized isolines. This method is commonly used to capture the contours shown on topographic maps. While it is effective for visual purposes, its value from an analytic perspective is far less than the DEM or TIN because of the very uneven sampling that is achieved.

Key Applications

Many examples of phenomena conceptualized as fields have already been cited; this section addresses the functions that are commonly applied to field representations, and the software that implements them. The functions described in this section are commonly found in GIS packages, and they and many others are reviewed by De Smith, Goodchild, and Longley [2].

A variety of functions are available to manipulate representations of topography. Most use the DEM option (3a) though very powerful algorithms have been described for similar operations on TINs. Visualization is a common requirement, and functions have been developed to compute surface gradient and hence simulated illumination; to compute and plot isolines; and to compute solar insolation as a key variable in understanding vegetation patterns on rugged topography. Another class of algorithms concern visibility, and can be used to compute the area visible by an observer positioned a given height above the terrain. Algorithms

have been described for computing the most exposed and most concealed points on a landscape, as well as most exposed and most concealed routes; and to position a minimal number of observers such that the entire landscape can be observed.

Another important collection of algorithms concerns drainage. Starting with a DEM, it is possible to compute drainage directions as a vector field, and to integrate these into catchments and stream channels. DEMs are used to predict and manage flooding, and to plan modifications to the landscape such as the construction of levees.

Reference was made earlier to methods of spatial interpolation, which address the task of predicting the value of a field at locations where it has not been measured. Most often these methods are applied to the representations illustrated in Fig. 3a and 3b, but a related technique known as areal interpolation has been devised for the task of predicting the values associated with areas that do not match (cut across the boundaries of) the reported areas of 3d and 3e.

Of particular interest are algorithms that produce representations of fields from collections of discrete objects. They include density estimation, which produces a field of feature density, most often of points; and calculation of the distance from any point in the plane to the nearest of a collection of discrete objects.

The most powerful collections of field-based manipulation functions clearly exist for raster data, because of the possibility that a collection of fields can be represented using a set of co-registered rasters. This is the principle of raster GIS, most clearly illustrated by packages with firm roots in raster-based discretizations, such as Idrisi and GRASS. Most have adopted a common language for expressing instructions known as map algebra [6]. A powerful alternative geared particularly to simulation is PCRaster (http://pcraster.geo.uu.nl), developed at the University of Utrecht and having its own manipulation language that is considerably more succinct and powerful than map algebra.

Future Directions

The comparative importance of continuous fields and discrete objects is a matter of continual debate. On the one hand, the majority of GIS applications occur in worlds where the objects of interest are well-defined and often man-made. On the other hand, many phenomena in the natural world are essentially continuous, and representing them as collections of discrete objects invites error and misuse. Cognitive scientists would argue that humans are hard-wired to see the world as a collection of discrete objects, while environmental scientists might argue that continuous fields are one of the most significant breakthroughs in the history of science, lying at the root of such developments as the calculus and hydrodynamics. Many of the forms of analysis commonly used in the environmental sciences are based on fields; while most of those commonly used in the social sciences are based on discrete objects.

GIS software today embraces both conceptualizations. But it does so in a somewhat unsatisfactory manner, in which representations of continuous fields must be reduced to collections of discrete objects, without any record of that process of reduction. Thus it is impossible for the user of a GIS database to enquire whether a collection of points in the database represents a set of points sampling a field, or a set of isolated point-like objects in an otherwise empty space. Unfortunately this means that inappropriate operations can easily be performed. Nothing prevents the GIS user from applying spatial interpolation to points conceptualized as discrete objects, or from computing a density field of sample points; the former is of course more disastrous than the latter. Similarly nothing prevents a GIS user from moving an isoline so that it crosses another, or from making two polygons in a Fig. 3e representation overlap.

Several interesting and potentially powerful research directions have been pursued in the hope of eventually improving this situation. One approach has been to argue that the user should be able to interact with the concept of a field directly, rather than with the elements of one of the six representations of Fig. 3 as at present. Another has been to search for a visual paradigm for handling fields that matches and has similar power to the visual representation of sets of discrete objects that is found in UML (Unified Modeling Language) and related methods. Now that GIS technology includes the capability to design databases in UML and to create and populate the necessary tables automatically, it would make good sense to develop parallel methods for handling fields. Finally, much recent effort has gone into finding ways of reconciling and bridging the field/object dichotomy [4].

Cross-references

► Digital Elevation Models
► Geographic Information System

► Raster Data Management and Multi-Dimensional Arrays
► Spatial Data Analysis
► Triangulated Irregular Network
► Unified Modeling Language

Recommended Reading

1. Couclelis H. People manipulate objects (but cultivate fields): beyond the raster-vector debate in GIS. In Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, A.U. Frank, I. Campari, U. Formentini (eds.). Springer, Berlin, 1992, pp. 65–77.
2. De Smith M.J., Goodchild M.F., and Longley P.A. Geospatial Analysis: A Comprehensive Guide to Principles. Techniques and Software Tools. Winchelsea Press, UK, 2007.
3. Goodchild M.F. Modeling error in objects and fields. In Accuracy of Spatial Databases, M.F. Goodchild, S. Gopal (eds.). Taylor and Francis, Bristol/London, 1989, pp. 107–114.
4. Goodchild M.F., Yuan M., and Cova T.J. Towards a general theory of geographic representation in GIS. Int. J. Geogr. Inf. Sci., 21(3):239–260, 2007.
5. Longley P.A., Goodchild M.F., Maguire D.J., and Rhind D.W. Geographic Information Systems and Science, Wiley, West Sussex, 2005.
6. Tomlin C.D. Geographic Information Systems and Cartographic Modeling. Prentice Hall, Englewoods, NJ, 1990.
7. Worboys M.F. and Duckham M. GIS: A Computing Perspective, CRC Press, Boca Raton, FL, 2004.
8. Zhang J.X. and Goodchild M.F. Uncertainty in Geographical Information. Taylor and Francis, London/New York, 2002.

File Compression

► Text Index Compression

File Format

► Image Representation

Filter/Refinement Query Processing

► Multi-Step Query Processing

Finding of Observation

► Clinical Observation

Finiteness

► Safety and Domain Independence

First-Order Logic: Semantics

Val Tannen
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Predicate calculus; Predicate logic
FOL

Definition

This entry should be read in conjunction with the companion entry *First-Order Logic: Syntax* where the terms vocabulary, variable, formula, etc. are defined.

To give semantics to first-order sentences, first-order *structures* (a.k.a. models or interpretations), and the "*holds true*" (a.k.a. satisfaction or validity) relationship is used between sentences and structures. Both are detailed below. This allows the definition of *logical consequence* (a.k.a. logical implication), $\Gamma \models \varphi$, whose meaning is that the sentence φ holds true in any structure in which the sentences in the set Γ hold true. Proof systems for first-order logic should be assessed against logical consequence whether *sound* and/or *complete*. The semantic of FOL is also used for *definability*, a concept that helps understanding the limitations of formalisms based on FOL.

As introduced by E. F. Codd into database technology, the *relational model* is based on finite first-order structures for a vocabulary that has only relational symbols.

Key Points

Fix a first-order vocabulary. A *structure* $\mathcal{A}$ for this vocabulary consists of a non-empty set A (sometimes called the "universe (of discourse)" of the structure) together with a mapping that assigns to every symbol s in the vocabulary a corresponding *meaning* $s^{\mathcal{A}}$ over A. For constants c, $c^{\mathcal{A}}$ is just an element of A, for n-ary relation symbols R, $R^{\mathcal{A}}$ is an n-ary relation over A (i.e., a subset of A^n), and for n-ary function symbols f one has a function $f^{\mathcal{A}} : A^n \to A$.

Next, the goal is to define when a sentence φ *holds true* in a structure $\mathcal{A}$, written $\mathcal{A} \models \varphi$. To do this, one should use *assignments*, which are partial functions from the set of all variables to the universe A of $\mathcal{A}$. It

$$\bar{\mu}(x) = \mu(x) \qquad \bar{\mu}(c) = c^{\mathcal{A}} \qquad \bar{\mu}(f(t_1,\ldots,t_n)) = f^{\mathcal{A}}(\bar{\mu}(t_1),\ldots,\bar{\mu}(t_n))$$

$$\mathcal{A},\mu \models t_1 = t_n \text{ iff } \bar{\mu}(t_1) = \bar{\mu}(t_2) \qquad \mathcal{A},\mu \models R(t_1,\ldots,t_n) \text{ iff } (\bar{\mu}(t_1),\ldots,\bar{\mu}(tn)) \in R^{\mathcal{A}}$$
$$\mathcal{A},\mu \models \textit{true} \qquad \mathcal{A},\mu \not\models \textit{false}$$
$$\mathcal{A},\mu \models \varphi_1 \vee \varphi_2 \text{ iff } \mathcal{A},\mu \models \varphi_1 \text{ or A}, \mu \models \varphi_2 \qquad \mathcal{A},\mu \models \varphi_1 \wedge \varphi_2 \text{ iff } \mathcal{A},\mu \models \varphi_1 \text{ and } \mathcal{A},\mu \models \varphi_2$$
$$\mathcal{A},\mu \models \neg\varphi \text{ iff } \mathcal{A},\mu \not\models \varphi \qquad \mathcal{A},\mu \models \varphi_1 \to \varphi_2 \text{ iff } \mathcal{A},\mu \models \varphi_2 \text{ whenever } \mathcal{A},\ \mu \models \varphi_1$$

$$\mathcal{A},\mu \models \exists x\,\varphi \text{ iff } \textit{there is } a \in A \text{ such that } \mathcal{A},\mu[x := a] \models \varphi$$
$$\mathcal{A},\mu \models \forall x\,\varphi \text{ iff } \textit{for each } a \in A,\ \mathcal{A},\mu[x := a] \models \varphi$$

suffices to consider assignments of finite domain (defined only on finitely many variables). By extending assignments $\mu : Vars \to A$ to $\bar{\mu} : Terms \to A$, define $\mathcal{A}, \mu \models \varphi$ where φ is a formula and such that μ is defined on (at least) all the free variables of φ:
where $\mu[x := a]$ is the same as μ except that $\mu[x := a](x) = a$. The definition above is uncomfortably "circular" if one thinks of FOL or other logical formalisms as part of a foundation for mathematics. But this definition of truth is part of a more useful point of view, due to Hilbert and Tarski, in which logic is a mathematical symbol "game" and one can use ordinary math to study it (this use of math is called "metamathematics").

If φ is a sentence it is possible to define when it holds in a structure $\mathcal{A}$: $\mathcal{A} \models \varphi$ means $\mathcal{A}, \emptyset \models \varphi$ where $\emptyset$ is the assignment defined nowhere (empty domain). A sentence is *valid* if it holds in all structures. At first validity seems computationally absurd (there isn't even a "set of all structures", because Russell's Paradox would be lurking in the fold). However, a very important result in metamathematics, Gödel's Completeness Theorem [2], shows that validity is equivalent to provability in one of the many equivalent proof systems for FOL. It follows that the set of all valid sentences is in fact recursively enumerable (r.e.), a reasonable foundation for attempting automated theorem-proving for FOL. However, Church and Turing have shown that validity is undecidable [2], an important inherent limitation that also affects applications to databases.

In fact, Gödel's Completeness Theorem shows more. If Γ is a set of sentences and φ a sentence, one defines *logical consequence* by

$$\Gamma \models \varphi \quad \textit{iff} \quad \textit{for all } \mathcal{A},\ \textit{if } \mathcal{A} \models \psi \textit{ for each } \psi \in \Gamma \textit{ then } \mathcal{A} \models \varphi$$

Fix a proof system with provability relation $\vdash$. The proof system is *sound* if $\Gamma \vdash \varphi$ implies $\Gamma \models \varphi$ and is *complete* if $\Gamma \models \varphi$ implies $\Gamma \vdash \varphi$. Gödel's Completeness Theorem shows that any of the equivalent proof systems for FOL is sound and complete.

A sentence is *finitely valid* if it holds in all finite structures. Finite validity is co-r.e. (just enumerate finite structures up to isomorphism). However, Trakhtenbrot's Theorem [3] shows that it is undecidable, hence it cannot be r.e. (in fact, it shows that it is co-r.e.-complete). Therefore, one cannot hope to find a sound and complete proof system for just the finitely valid sentences.

Finally, consider a structure $\mathcal{A}$ and an n-ary relation $D \subseteq A^n$. Say that D is *first-order definable in* $\mathcal{A}$ if there exists an FOL formula φ with exactly n distinct free variables $x_1,\ldots,x_n$ such that

$$\textit{for all } a_1,\ldots,a_n \in A, \quad (a_1,\ldots,a_n) \in D \textit{ iff } \mathcal{A}, [x_1 := a_1,\ldots,x_n := a_n] \models \varphi$$

Definability in a given structure is trivial if the structure is finite. For finite structures look at the definability of *queries*.

Cross-references

- ▶ First-Order Logic
- ▶ First-Order Logic: Syntax
- ▶ Relational Calculus

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases: The Logical Level. Addison Wesley, Reading, MA, 1994.
2. Enderton H.B. A Mathematical Introduction to Logic, 2nd edn. Academic, New York, 2000.
3. Libkin L. Elements of Finite Model Theory. Springer, 2004.

First-Order Logic: Syntax

Val Tannen
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Predicate calculus; Predicate logic; FOL

Definition

First-order logic (FOL) is a formalization of the most common kind of mathematical reasoning. It is characterized by the *quantification* of *variables* that range over a "universe of discourse" (a set of values). Less complex reasoning is captured by *propositional* (a.k.a. Boolean or sentential) logic. More complex reasoning is captured by *second-order* or even *higher-order* logic.

The syntactic aspects of FOL comprise a *vocabulary* (a.k.a. language or signature), *formulae* and, in particular, *sentences* (a.k.a. assertions), and a *proof system* (one of many equivalent ones!), *structures* (a.k.a. models or interpretations), and the *satisfaction* (a.k.a. truth or validity or "holds in") relationship between sentences and structures. All are detailed below.

FOL is the source of the *relational* paradigm that was introduced by E. F. Codd in 1970 and has been dominating database technology for 30+ years.

Key Points

A first-order *vocabulary* consists of a set of *constant* symbols, for each integer $n \geq 0$ a set of n-ary *relation* (a.k.a. predicate) symbols, for each integer $n \geq 1$ a set of n-ary *function* symbols. For some authors the constants are the 0-ary function symbols. The 0-ary relation symbols are called "propositional constants" (although, in propositional logic they are called "variables"!). In addition, first-order vocabularies are often assumed to contain the equality symbol—a distinguished binary relation symbol – as well as *true* and *false* as distinguished propositional constants. For applications in Computer Science, these sets of symbols are usually finite.

Next, a vocabulary and an infinite set of *variables* are fixed. First-order *terms* are defined by the grammar

$$t ::= c \mid x \mid f(t_1,...,t_n)$$

where c ranges over constants, x over variables, f over n-ary function symbols, and t, 1,...,t_n over terms, for each integer $n \geq 1$. First-order *formulae* (sometimes called *well-formed formulae* of *wff's* are defined by the grammar

$$\varphi ::= R(t_1,...,t_n) \mid t_1 = t_2 \mid \mathit{true} \mid \mathit{false} \mid (\varphi_1 \vee \varphi_2) \mid (\varphi_1 \wedge \varphi_2) \mid (\neg\varphi) \mid (\varphi_1 \rightarrow \varphi_2) \mid (\exists x\varphi) \mid (\forall x\varphi)$$

where R ranges over n-ary relation symbols, t_1, t_2,...,t_n over terms (for each integer $n \geq 0$), x over variables, and φ, φ_1, φ_2 over formulae. Formulae of the form $R(t_1,...,t_n)$, $t_1 = t_2$, *true* or *false* are called *atoms* (a.k.a atomic formulae). The logical connectives for disjunction, conjunction, negation, and implication are as in propositional logic. Characteristic of first-order logic is *existential* ($\exists$) and *universal* ($\forall$) quantification with variables x ranging over a "universe of discourse". The parentheses are typically omitted based on simple precedence and associativity rules.

Next, one defines when a variable x *is free* (a.k.a. "occurs free") in a formula φ (inductively on the syntax of φ): (i) if α is an atom, then x is free in it iff it is used to build one of the terms that make up α; (ii) x is free in $(\varphi_1 \vee \varphi_2)$ or in $(\varphi_1 \wedge \varphi_2)$ or in $(\varphi_1 \rightarrow \varphi_2)$ iff it is free in φ_1 or in φ_2; (iii) x is free in (φ) iff it is free in φ; (iv) x is free in $(\exists y\, \varphi)$ or $(\forall y, \varphi)$ iff it is free in$\varphi$and $x \neq y$. Every formula has a finite set of variables that occur syntactically in it. Some of these are free, as defined above. The others are called *bound* because they must appear in some quantified subformula: $(\exists y\, \psi)$ or $(\forall y\, \psi)$, in which case it is said that ψ is the *scope* of the (bound) quantified variable y. For clarity, it is good practice to use fresh variables for each quantification. However, this is not required by the definition, for example

$$\exists x\, E(a,x) \wedge (\exists y\, E(x,y) \wedge (\exists x\, E(y,x) \wedge (\exists y\, E(x,y) \wedge E(y,b))))$$

says that there exists a path of length five from vertex a to vertex b in a directed graph with edge relation E, and it does so with only two bound variables although there are four intermediate vertices. Bound variable "reusing" can be exploited, see *finite variable logics* [3].

A formula without free variables is called a *sentence*. Sentences are logical assertions that may or may not be *true* by themselves. However, to give them meaning, one needs to give meaning more generally to formulae.

Still part of the syntax of FOL are the *proof systems* which describe effective procedures for deriving sentences from other sentences. A common style of proof system, due to Hilbert, uses sentences that are asserted as *axioms*, such as *tautologies* from propositional logic (eg., De Morgan's Laws) or the *substitution* axioms $(\forall x\varphi \rightarrow \varphi[x := t]$ where $\varphi[x := t]$ is the result of substituting every free occurrence of x in φ with the term t) and *inference rules* such as *modus ponens* (from φ and $\varphi \rightarrow \psi$ infer ψ). In such a system, a proof is a list of sentences such that each of them is either an axiom or is inferred by some rule from previously listed sentences. The proof is for the last sentence in the list. The sentences proved in a proof system are called *theorems*. More

generally, a proof system defines a *provability* (a.k.a. inference or derivability) relationship between sentences φ and sets Γ of sentences: $\Gamma \vdash \varphi$ iff there exists a proof of φ that can use sentences in Γ as additional axioms (in particular, φ is a theorem iff $\emptyset \vdash \varphi$.

Quite a few styles of proof systems have been proposed, for FOL and for other logics, moreover the choice of axioms/rules often varies within each style. Of course, all the proof systems for FOL have been shown to be equivalent, that is, they define the same provability relationship, in particular the same theorems. More importantly, all proof systems are characterized by the following; (i) proofs are finite objects; (ii) it is decidable whether a proof is correctly formed; (iii) the proved sentence is totally computable from the proof. It follows that the theorems of FOL form a recursively enumerable set (although, by the result of Church and Turing, not a decidable one).

Cross-references

▶ First-Order Logic
▶ Semantics

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of databases: the logical level. Addison Wesley, Reading, MA, USA, 1994.
2. Enderton H.B. 1A Mathematical Introduction to Logic, Academic, London, 2000.
3. Libkin L. Elements of Finite Model Theory. Springer, Berlin, 2004.

First-Order Query

▶ Relational Calculus

Fisheye Views

▶ Distortion Techniques

Fixed Time Span

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

Synonyms

Constant span

Definition

A time span is *fixed* if it possesses the special property that its duration is independent of the assumed context.

Key Points

As an example of a fixed span, "one hour" always, assuming a setting without leap seconds, has a duration of 60 minutes. To see that not all spans are fixed, consider "one month," which is a prime example of a variable span in the Gregorian calendar. The duration of this span may be any of 28, 29, 30, and 31 days, depending on the context, i.e., the specific month.

Cross-references

▶ Calendar
▶ Temporal Database
▶ Time Interval
▶ Time Span
▶ Variable Time Span

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399, Springer, Berlin Heidelberg New York, 1998. pp. 406–413.
2. 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, Berlin Heidelberg New York, 1998, pp. 367–405.

Flajolet-Martin Algorithm

▶ FM Synopsis

Flajolet-Martin Sketch

▶ FM Synopsis

Flake

▶ Snippet

Flash

▶ Storage Devices

Flex Transactions

Aidong Zhang[1], Bharat Bhargava[2]
[1]State University of New York at Buffalo, Buffalo, NY, USA
[2]Purdue University, West Lafayette, IN, USA

Synonyms

Flexible transactions; S-transactions; ConTracts

Definition

In database systems, a *transaction* is a sequence of actions performed on data items in a database. In a distributed database environment, a *global transaction* is a set of subtransactions, where each *subtransaction* is a transaction accessing the data items at a single local site. The flex transaction model supports flexible execution control flow by specifying two types of dependencies among the subtransactions of a global transaction: (i) execution ordering dependencies between two subtransactions, and (ii) alternative dependencies between two subsets of subtransactions.

Key Points

Flexible transaction models, such as ConTracts, Flex Transactions, S-transactions, and others [1–3], increase the failure resilience of global transactions by allowing alternate subtransactions to be executed when a local database site fails or a subtransaction aborts. In a non-flexible transaction, a global subtransaction abort is followed either by a global transaction abort decision or by a retry of the global subtransaction. With the flexible transaction model, there is an additional option of switching to an alternate global transaction execution. The alternative global transaction execution is achieved by executing alternative or contingency transactions.

Flexibility allows a flexible transaction to adhere to a weaker form of atomicity, termed *semi-atomicity*, while still maintaining its correct execution in the distributed database environment. Semi-atomicity allows a flexible transaction to commit as long as a subset of its subtransactions that can represents the execution of the entire flexible transaction commit. Flexible transactions, S-tractions, and ConTracts are instances of Open-nested Transaction Model. The following example is illustrative:

A client at bank b_1 wishes to withdraw \$50 from her savings account a_1 and deposit it in her friend's checking account a_2 in bank b_2. If this is not possible, she will deposit the \$50 in her own checking account a_3 in bank b_3. With flexible transactions, this is represented by the following set of subtransactions:

t_1: Withdraw \$50 from savings account a_1 in bank b_1
t_2: Deposit \$50 in checking account a_2 in bank b_2
t_3: Deposit \$50 in checking account a_3 in bank b_3

In this global transaction, either $\{t_1,t_2\}$ or $\{t_1,t_3\}$ is acceptable, with t_3 being a contingency of t_2 but $\{t_1,t_2\}$ preferred. If t_2 fails, t_3 may replace t_2. The entire global transaction thus may not have to be aborted even if t_2 fails.

Cross-references

- ▶ Atomicity
- ▶ Concurrency Control Manager
- ▶ ConTract
- ▶ Distributed transaction management

Recommended Reading

1. Wächter H. and Reuter A. The ConTract model. In Database Transaction Models for Advanced Applications, A.K. Elmagarmid (ed.). Morgan Kaufmann, Los Altos, CA, 1992.
2. Zhang A., Nodine M., and Bhargava B. Global scheduling for flexible transactions in heterogeneous distributed database systems. IEEE Trans. Knowl. Data Eng., 13(3):439–450, 2001.
3. Zhang A., Nodine M., Bhargava B., and Bukhres O. Ensuring relaxed atomicity for flexible transactions in multidatabase systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 67–78.

Flexible Metric Computation

▶ Learning Distance Measures

Flexible Transactions

▶ Flex Transactions

Flowcharts

▶ Activity Diagrams

FM Sketch

▶ FM Synopsis

FM Synopsis

Phillip B. Gibbons
Intel Labs Pittsburgh, Pittsburgh, PA, USA

Synonyms

FM sketch; Flajolet-Martin sketch; Flajolet-Martin algorithm

Definition

Given a multi-set S of values from a domain $\mathcal{D}$, the *distinct-values estimation* problem is to estimate the number of distinct values in S, using only one pass over S and only small working space. The *FM Synopsis* algorithm, developed by Flajolet and Martin in the mid-1980s [7], provides provably accurate distinct-values estimation using only $O(\log(|\mathcal{D}|))$ space. The basic technique makes use of a hash function $h()$ that maps each value in $\mathcal{D}$ to one of $m \approx \log(|\mathcal{D}|)$ bit positions, according to a geometric distribution. Specifically, $h()$ maps half the values in $\mathcal{D}$ to position 0, one-quarter of the values in $\mathcal{D}$ to position 1, one-eighth of the values in to position 2, and so on. The steps of the FM Synopsis algorithm are:

1. Initialize a bit vector M of m bits to all 0s.
2. For each item in S do: Set $M[h(v)]$ to 1, where $v \in \mathcal{D}$ is the value of the item.
3. Estimate the number of distinct values as $2^Z/.77351$, where Z is the position of the least significant unset bit in M.

To reduce the variance in this estimator, Flajolet and Martin take the average over tens of applications of this procedure, with different random hash functions $h()$.

Historical Background

Distinct-values estimation was one of the first non-trivial data stream problems studied. A decade before data stream synopsis structures and their desirata were defined and popularized, the FM Synopsis met the desirata for a data stream synopsis structure: (i) it needs only one pass through the data, (ii) it yields highly accurate answers regardless of the data distribution, (iii) it takes only logarithmic space, and (iv) it requires only constant per-item processing time and only constant time to produce an estimate given the synopsis structure.

Prior to the popularization of the FM Synopsis algorithm, a common approach for estimating the number of distinct values in a multi-set S was to collect a random sample of S and then apply sophisticated estimators based on the distribution of the values in the sample [2]. However, all known sampling-based estimators provide unsatisfactory results on some data sets of interest, and moreover, estimating the number of distinct values within a small constant factor (with probability $> \frac{1}{2}$) requires (in the worst case) that *nearly all of S be sampled* [3]. Thus, streaming approaches, such as the FM Synopsis algorithm, are preferred over sampling-based approaches, whenever feasible.

Recent work has built upon the FM Synopsis approach, improving the accuracy guarantees on the estimation, proving lower bounds, and considering other settings such as sliding windows and distributed streams.

Foundations

Figure 1 depicts two data sets, S_1 and S_2, where the number of distinct values, also called the zeroth frequency moment F_0, is 15 for S_1 and 6 for S_2. These data sets can help illustrate the challenges of sampling-based estimators for F_0. Consider the following 33% sample of a data set of 24 items:

A, T, A, R, E, T, P, H

Given this 33% sample (with its 6 distinct values), does the entire data set have 6 distinct values, 18 distinct values (i.e., the 33% sample has 33% of the distinct values), or something in between? Note that this particular sample can be obtained by taking every third item of either S_1 (where $F_0 = 15$) or S_2 (where $F_0 = 6$) from Fig. 1. Thus, despite sampling a large (33%) percentage of the data, estimating F_0 remains challenging, because the sample can be viewed as fairly representative of either S_1 or S_2 – two data sets with very different F_0s.

Data set S_1: R, X, A, D, A, T, Y, R, A, A, T, R, T, U, E, A, C, T, F, L, P, B, V, H
Data set S_2: T, A, A, R, H, T, A, T, A, E, T, R, A, T, E, P, H, T, R, P, P, E, A, H

FM Synopsis. Figure 1. Two example data sets of $n = 24$ items from a domain $\mathcal{D} = \{A, B, \ldots, Z\}$.

Now consider the FM Synopsis algorithm outlined above. Using a hash function ensures that all items with the same value will select the same bit position; thus the final bit vector M is independent of any duplications among the item values. Flajolet and Martin's analysis assumes an idealized random hash function that maps each value $v \in \mathcal{D}$ to a bit position $b \in [0..m-1]$ such that $b = i \in [0..m-2]$ with probability $2^{-(i+1)}$ and $b = m-1$ with probability $2^{-(m-1)}$. Accordingly, $M[i]$ is expected to be set if there are at least 2^{i+1} distinct values. Because bit $Z-1$ is set but not bit Z, there are likely greater than 2^Z but fewer than 2^{Z+1} distinct values. Flajolet and Martin's analysis shows that $E[Z] \approx log_2(.77351 \cdot F_0)$, so that $2^Z/.77351$ is a good choice in that range.

To reduce the variance in the estimator, Flajolet and Martin take the average over tens of applications of this procedure (with different hash functions). Specifically, they take the average, $\bar{Z}$, of the Z's for different hash functions and then compute $\lfloor 2^{\bar{Z}}/.77351 \rfloor$. An example is given in Fig. 2.

The error guarantee, space bound, and time bound are summarized in Theorem 1. The space bound does not include the space for representing the hash functions, and the time bound assumes that computing $h(v)$ is a constant time operation.

Theorem 1 [7] *Consider a multi-set* S *of* n *items with values from a domain* $\mathcal{D}$. *The FM Synopsis algorithm with* k *(idealized) hash functions produces a distinct-values estimator with standard error* $O(1/\sqrt{k})$, *using* k · m *memory bits for the bit vectors, for any* $m > \log_2(\min(n, |\mathcal{D}|)) + 4$. *For each item, the algorithm performs* O(k) *operations on memory words of at most* $\max(m, \log_2(|\mathcal{D}|))$ *bits.*

Optimizations

The space needed for the k bit vectors, $M_1[\cdot]$, $M_2[\cdot]$,..., $M_k[\cdot]$, can be significantly reduced, using the following simple compression technique [4,7,10]. Note that at any point during the processing of the data set, each bit vector $M_j[\cdot]$ consists of three parts, where the first part is all 0s, the second part (the *fringe*) is a mix of 0s and 1s, and the third part is all 1s (as in Fig. 2). Moreover, the bit vectors are likely to share many bits in common, because each $M_j[\cdot]$ is constructed using the same algorithm on the same data. The technique is to interleave the bits, as follows:

$$M_1[m-1], M_2[m-1],..., M_k[m-1],$$
$$M_1[m-2],..., M_k[m-2],..., M_1[0],..., M_k[0],$$

and then run-length encode the resulting bit vector's prefix of all 0s and suffix of all 1s. Specifically, a simple encoding scheme is used that ignores the all 0s prefix and consists of (i) the interleaved bits between the all 0s prefix and the all 1s suffix and (ii) a count of the length of the all 1s suffix. An example is given in Fig. 3. Note that the interleaved fringe starts with the maximum bit set among the k fringes and ends with their minimum unset bit. Each fringe is likely to be tightly centered around the current $\log_2(F_0)$, e.g., for a given hash function and $c > 0$, the probability that no bit than $\log_2(F_0) + c$ or larger is set is $(1 - 2^{-(\log_2(F_0)+c)})^{F_0} \approx e^{-1/2^c}$. At any point in the processing of the data set, a similar

Item number	Item value v	Hash function 1 b	$M_1[\cdot]$	Hash function 2 b	$M_2[\cdot]$	Hash function 3 b	$M_3[\cdot]$
1.	R	1	000010	1	000010	0	000001
2.	X	0	000011	2	000110	0	000001
3.	A	0	000011	1	000110	0	000001
4.	D	0	000011	0	000111	0	000001
5.	A	0	000011	1	000111	0	000001
6.	T	1	000011	0	000111	1	000011
7.	Y	3	001011	0	000111	1	000011
8.	R	1	001011	1	000111	0	000011
		$Z = 2$		$Z = 3$		$Z = 2$	

The estimate for F_0 is $\left\lfloor \frac{2^{(2+3+2)/3}}{.77351} \right\rfloor = 6$.

FM Synopsis. Figure 2. Example run of the FM Synopsis algorithm on the first 8 items of S_1, using $k = 3$ hash functions. Each bit vector $M[\cdot]$ is $m = 6$ bits long, with $M[0]$ being the rightmost bit (the least significant bit). The estimate, 6, matches the number of distinct values in the first 8 items of S_1.

argument shows that the interleaved fringe is expected to be fewer than $O(\mathrm{k}\log\mathrm{k})$ bits. Also, the count of the all 1s suffix is at most $k \cdot m$. Thus, the encoding is expected to use fewer than $\log_2 m + O(k \log k)$ bits.

The number of operations per data item can be reduced from $O(k)$ to $O(1)$, using a variant of the FM Synopsis algorithm called *Probabilistic Counting with Stochastic Averaging (PCSA)* [7]. In the PCSA algorithm (see Fig. 4), k bit vectors are used (for k a power of 2) but only a single hash function g(). The (idealized) random hash function g() is assumed to map each value $v \in \mathcal{D}$ to a uniformly random value in $\mathcal{D}$. For each data item with value v, the $\log_2 k$ least significant bits of $g(v)$ are used to select a bit vector. Then the remaining $m - \log_2 k$ bits of $g(v)$ are used to select a position within that bit vector, according to a geometric distribution (as in the basic FM Synopsis algorithm). To compute an estimate, PCSA averages over the positions of the least significant 0-bits, computes 2 to the power of that average, and divides by the bias factor .77351. To compensate for the fact that each bit vector has seen only $1/k$th of the distinct items on average, the estimate is multiplied by k.

The error guarantee, space bound, and time bound are summarized in the following theorem. The space bound does not include the space for representing the hash function. The time bound assumes that computing g and b are constant time operations. Although some of the operations use only m-bit words, the word size is dominated by the $\log_2 n$ bits for the item value v. The analysis assumes that the bit vectors are compressed using the optimization in Fig. 3.

Theorem 2 *Consider a multi-set S of n items with values from a domain $\mathcal{D}$. The PCSA algorithm with* k *bit vectors and an (idealized) hash function produces a distinct-values estimator with standard error* $0.78/\sqrt{k}$, *using an expected* $\log_2 m + O(k \log k)$ *memory bits for the bit vectors, for any* $m > \log_2(\min(n, |\mathcal{D}|)/k) + 4$. *For each item, the algorithm performs* $O(1)$ *operations on memory words of at most* $\max(m, \log_2(|\mathcal{D}|))$ *bits.*

Extensions

There has been considerable follow-on work since the introduction of the FM Synopsis algorithm. Some highlights are provided below; further details can be found in [8].

Alon, Matias and Szegedy [1] showed how a variant of the FM Synopsis algorithm could obtain guaranteed accuracy without the assumption of idealized hash functions. Specifically, they consider standard *linear hash functions* of the form $g(v) = a \cdot v + b$, where a and b are chosen uniformly and independently at random from a suitable range and all arithmetic is computed modulo that range. Their variant estimates F_0 as 2^R, where R is the position of the most significant bit that is set to 1. Moreover, it directly keeps track of the maximum position set to 1, instead of maintaining a bit vector.

Durand and Flajolet [5] presented a variant that uses only $\approx k \log_2 \log_2(\min(n, |\mathcal{D}|)/k)$ bits for its synopses and provides a standard error of $1.05/\sqrt{k}$, assuming idealized hash functions.

The FM Synopsis approach can be readily extended to handle item *deletions* by replacing each bit in $M[\cdot]$ with a running counter that is incremented on

$M_j[\cdot]$'s:	00001111, 00001011, 00000111, 00010111, 00011111, 00001111
Interleaved:	000000000000000000000011011001110111111111111111
End-encode:	11011001110, 16

FM Synopsis. Figure 3. Example FM Synopsis compression, for $k = 6$ and $m = 8$.

1. for $j := 1,\ldots, k$ and $i := 0,\ldots,m-1$ do $M_j[i] := 0$
2. for each item in S do:
   $x := g(v) \bmod k$, where $v \in \mathcal{D}$ is the value of the item // Note: k is a power of 2
   $b :=$ the largest $i \geq 0$ such that the i rightmost bits in $\lfloor g(v)/\mathrm{k} \rfloor$ are all 0
   $M_x[\mathrm{b}] := 1$
3. return $\left\lfloor \frac{k}{.77351} 2^{\bar{Z}} \right\rfloor$, where $\bar{Z} := \frac{1}{k}\sum_{j=1}^{k} \min\{i : M_j[i] = 0\}$ // least significant unset bit

FM Synopsis. Figure 4. The PCSA algorithm [7].

insertions and decremented on deletions. This increases the synopsis space by a logarithmic factor. Similarly, the FM Synopsis approach can be extended to handle *sliding windows*, where the problem is to estimate the number of distinct values over a sliding window of the w most recent items, for some fixed w. The idea is to keep track of the sequence number of the most recent item that set each FM bit. Then, when estimating the number of distinct values within the current sliding window, only those FM bits whose associated sequence numbers are within the window are considered to be set. This too increases the synopsis space by a logarithmic factor. See [8] for an overview of algorithms that improve upon these basic schemes for deletions and sliding windows.

Finally, note that the FM Synopsis approach can be readily adapted to a *distributed* setting in which a set of observers each processes the portion of the multi-set S that it sees. The goal is to estimate the number of distinct values in S by (i) having each observer compute a small synopsis for the values it sees, and then (ii) combining the synopses to output a highly accurate estimate. The FM Synopsis algorithm is perfectly suited for this task. It can be applied to each data set portion independently, using the exact same hash function at all observers, to generate a bit vector $M[\cdot]$ for each portion. Because the same hash function is used by all observers, the bit-wise OR of their bit vectors yields the exact bit vector that would have been produced by running the FM Synopsis algorithm on any interleaving of the data portions. Thus, estimating F_0 from this bit-wise OR provides the same error guarantees as in the original algorithm. Moreover, the per-observer space bound and the per-item time bound also match the space and time bounds in Theorems 1 and 2.

Key Applications

Estimating the number of distinct values in a data set is a well-studied problem with many applications. The statistics literature refers to this as the problem of estimating the number of *species* or *classes* in a population. The problem has been extensively studied in the database literature, as a tool for summarizing the diversity of data values in a data set, both to help guide query optimization and to provide fast approximate answers to distinct-value queries (e.g., a select count(distinct) query in SQL). Distributed distinct-values estimators are useful for network resource monitoring, in order to estimate the number of distinct destination IP addresses, source-destination pairs, requested urls, etc. In network security monitoring, determining sources that send to many distinct destinations can help detect fast-spreading worms.

Distinct-values estimation can also be used as a general tool for duplicate-insensitive counting: Each item to be counted views its unique id as its "value," so that the number of distinct values equals the number of items to be counted. Duplicate-insensitive counting can be used to avoid double-counting items that are counted while in motion, to compute the number of distinct neighbors at a given hop-count in a network, and to compute the size of the transitive closure of a graph. In a sensor network, duplicate-insensitive counting together with multi-path in-network aggregation enables robust and energy-efficient answers to count queries [9]. Moreover, duplicate-insensitive counting is a building block for duplicate-insensitive computation of other aggregates, such as sum and average.

Experimental Results

The number of bit vectors, k, in the FM Synopsis algorithm is a tunable parameter trading off space for accuracy. The relative error as a function of k has been studied empirically in [4,7,9,10] and elsewhere, with $<$ 15% relative error reported for $k = 20$ and $<$10% relative error reported for $k = 64$, on a variety of data sets. These studies show a strong diminishing return for increases in k. Theorems 1 and 2 show that the standard error is $O(1/\sqrt{k})$. Thus, reducing the standard error from 10% to 1% requires increasing k by a factor of 100! In general, to obtain a standard error at most ε it is required that $k = \Theta(1/\varepsilon^2)$. Estan, Varghese and Fisk [6] present a number of techniques for further improving the constants in the space versus error trade-off, including using multi-resolution and adaptive bit vectors.

Cross-references

▶ Approximation and Data Reduction Techniques
▶ Stream Mining
▶ Synopsis Structure

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. J. Comput. Syst. Sci., 58:137–147, 1999.
2. Bunge J. and Fitzpatrick M. Estimating the number of species: a review. J. Am. Stat. Assoc., 88:364–373, 1993.

3. Charikar M., Chaudhuri S., Motwani R., and Narasayya V. Towards estimation error guarantees for distinct values. In Proc. 19th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2000, pp. 268–279.
4. Considine J., Li F., Kollios G., and Byers J. Approximate aggregation techniques for sensor databases. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 449–460.
5. Durand M. and Flajolet P. Loglog counting of large cardinalities. In Proc. 11th European Symposium on Algorithms. 2003, 604–617.
6. Estan C., Varghese G., and Fisk M. Bitmap algorithms for counting active flows on high speed links. In Proc. 3rd ACM SIGCOMM Conf. on Internet Measurement. October, 2003, pp. 153–166.
7. Flajolet P. and Martin G.N. Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci., 31:182–209, 1985.
8. Gibbons P.B. *I*Distinct-values estimation over data streams. In Data Stream Management: Processing High-Speed Data Streams, M. Garofalakis, J. Gehrke, R. Rastogi (eds.). Springer, Secaucus, NJ, USA, 2009.
9. Nath S., Gibbons P.B., Seshan S., and Anderson Z. Synopsis diffusion for robust aggregation in sensor networks. ACM Trans. on Sensor Networks, 4(2), 2008.
10. Palmer C.R., Gibbons P.B., and Faloutsos C. ANF: a fast and scalable tool for data mining in massive graphs. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 81–90.

F-Measure

Ethan Zhang[1,2], Yi Zhang[1]
[1]University of California-Santa Cruz, Santa Cruz, CA, USA
[2]Yahoo! Inc., Santa Clara, CA, USA

Synonyms

Harmonic mean of recall and precision

Definition

Assume an information retrieval (IR) system has recall R and precision P on a test document collection and an information need. The *F-measure* of the system is defined as the weighted harmonic mean of its precision and recall, that is, $F = \frac{1}{\alpha\frac{1}{P}+(1-\alpha)\frac{1}{R}}$, where the weight $\alpha \in [0,1]$. The balanced F-measure, commonly denoted as F_1 or just F, equally weighs precision and recall, which means $\alpha = 1/2$. The F_1 measure can be written as $F_1 = \frac{2PR}{P+R}$.

Key Points

The F-measure can be viewed as a compromise between recall and precision. It is high only when both recall and precision are high. It is equivalent to recall when $\alpha = 0$ and precision when $\alpha = 1$. The F-measure assumes values in the interval [0,1]. It is 0 when no relevant documents have been retrieved, and is 1 if all retrieved documents are relevant and all relevant documents have been retrieved.

Cross-references

- ▶ Precision
- ▶ Recall
- ▶ Standard Effectiveness Measures

Focused Retrieval

- ▶ XML Retrieval
- ▶ Structured Document Retrieval

Focused Web Crawling

Soumen Chakrabarti
Indian Institute of Technology of Bombay, Mumbai, India

Synonyms

Web resource discovery; Topic-directed Web crawling

Definition

The world-wide Web can be modeled as a very large graph with nodes representing pages and edges representing hyperlinks. Thanks to dynamically generated content, the Web graph is infinitely large. Page content and hyperlinks change continually. Any centralized Web search service must first fetch a large number of Web pages over the Internet using a Web crawler, and then subject the local copies to indexing and other analysis. At any time during its execution, a Web crawler has a set of pages that have been fetched, and a frontier of unexplored hyperlinks encountered on fetched pages. Given finite network resources, it is critical for the crawler to choose carefully the subset of frontier hyperlinks it should fetch next. Depending on the application and user group, it may be beneficial

to preferentially acquire pages that are highly linked, pages that pertain to specific topics, pages that are likely to mention specific structured information, pages that score highly with respect to queries submitted frequently to the search engine, pages that change frequently, and so on. Focused Web crawling is a generic term for employing hyperlink and text mining techniques to prioritize the crawl frontier to maximize the harvest of qualified or preferred pages, while minimizing communication and computation effort on other pages. The network resources thus saved may be used, for example, to monitor crawled pages more aggressively for changes. Focused Web crawling is commonly used to build vertical search services catering to one or few topical interests.

Historical Background

Despite giant leaps in communication, storage and computing power during the last decade, crawlers have always struggled to keep up with Web content generation and modification. The Web graph is actually infinite, owing to dynamically generated hyperlinks and pages. A crawler begins with URL references to a fraction of nodes in the Web graph. Many of these URLs are stale because the pages they referred to are inaccessible. The number of pages collected by the crawler makes little sense in this context, compared to which specific pages are collected.

Around 1997, researchers began to propose objectives that a crawler may seek to optimize, and several forms of crawl prioritization. Some objectives [8,14] were related to the Web graph structure. Others were determined by textual content of pages [4,12]. The term "focused crawling" is generally used for the latter class, although later generations of focused crawlers use all available features and clues [7,10,17,] to guide themselves, including workloads collected from query logs [15]. As of late 2007, work on prioritizing crawls and crawl refreshing continues apace, with increased stress on balancing resources between monitoring crawled pages for changes and discovering new sites and pages.

The success of focused Web crawling depends on benign patterns and regularities in the Web graph, which started to get seriously investigated as social network phenomena around 1999. The Web as a whole and many major communities have a strongly connected core of authoritative and popular pages [2,11]. Well-connected subgraphs of the Web are also likely to be topically coherent [6,9]. Without these helpful properties, focused crawlers cannot succeed.

Foundations

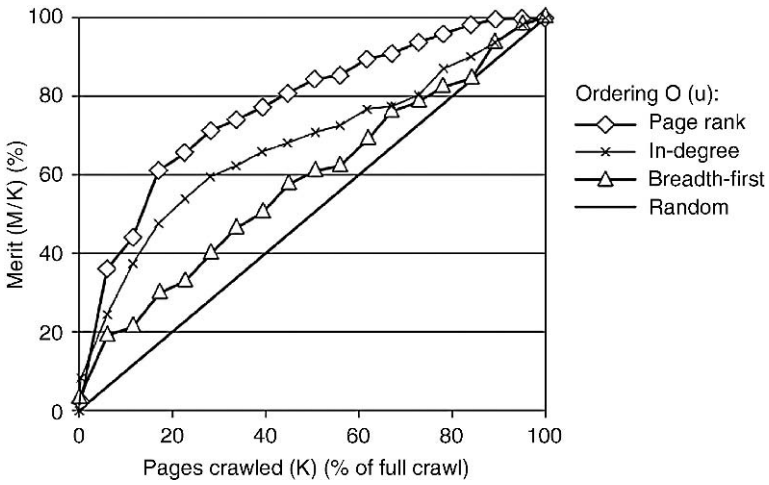
Setting Crawl Priorities Using Web Graph Properties

Initial work on focused Web crawling set reasonable fixed policies and evaluated their effect on various measures of crawl quality [8]. To characterize crawl quality, each page u is assigned an *importance* $I(u)$. For simplicity, one can initially assume that the crawler starts from just one URL u_0, and stops when K pages have been fetched. The perfect crawler collects K of the N pages reachable from u_0 by following hyperlinks with the largest possible importance. Let these "hot pages" have importance scores $I(u_1) \leq ... \leq I(u_K)$. An imperfect crawler will also fetch K pages, but only M of those will have importance at least $I(u_k)$. Then the figure of merit of the crawler will be M/K. As a baseline, if a "random" crawler could somehow collect K random samples from the N reachable pages without following links, the expected number of hot pages in the sample would be MK/N leading to a figure of merit equal to K/N.

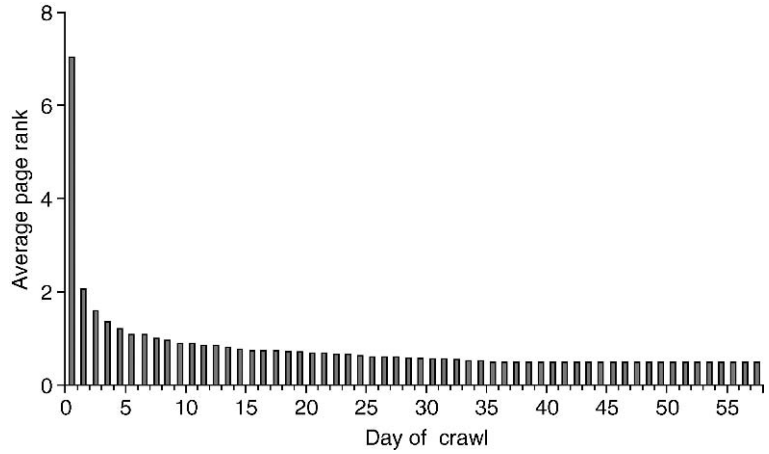
Suppose $I(u)$ is the indegree of u in the "whole" Web graph (suppose some arbitrary subset of the infinite graph is fixed a-priori as the "universe"). In this case, $I(u)$ can be known only after crawling the universe. Another global property is PageRank [16]. At any time, a real crawler only has access to the subgraph crawled thus far. It may use the indegree and PageRank computed on the crawled subgraph as approximations to the true indegree and PageRank.

Figure 1 [8] shows M/K as a percent plotted against K/N also plotted as a percent. The random crawler is understandably the worst, and all the other crawlers fare better. Somewhat surprisingly, the best crawler is the one that is guided not by indegree but PageRank computed with respect to the currently crawled subgraph. A later breadth-first crawl [14] starting from http://www.yahoo.com, covering 7 million hosts and 328 million pages, confirmed that pages with large true PageRank are acquired very quickly using the breadth-first policy (see Fig. 2).

Cho et al. [8] also considered page importance defined by the occurrence of query words on the page, and showed that favoring URLs v that were frequently referred from pages u with query words in or near the anchor text of the (u, v) link resulted in



Focused Web Crawling. Figure 1. Comparison of harvest rates of large indegree pages using various frontier prioritization policies [8].



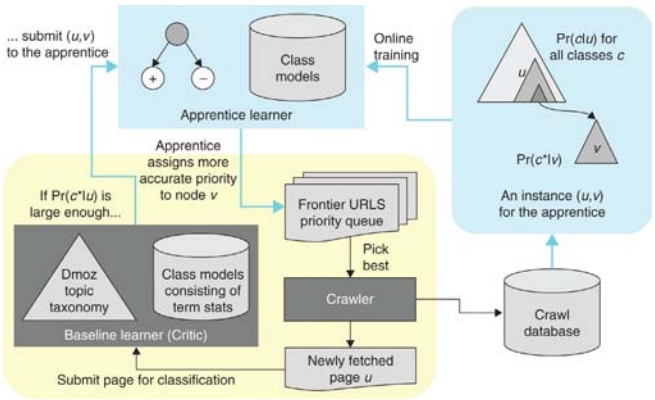
Focused Web Crawling. Figure 2. Page fetched early on in a breadth-first crawl from well-known seed URLs tend to have high PageRank [14].

improved acquisition rate for relevant pages. Similar results were reported by Hersovici et al. [12].

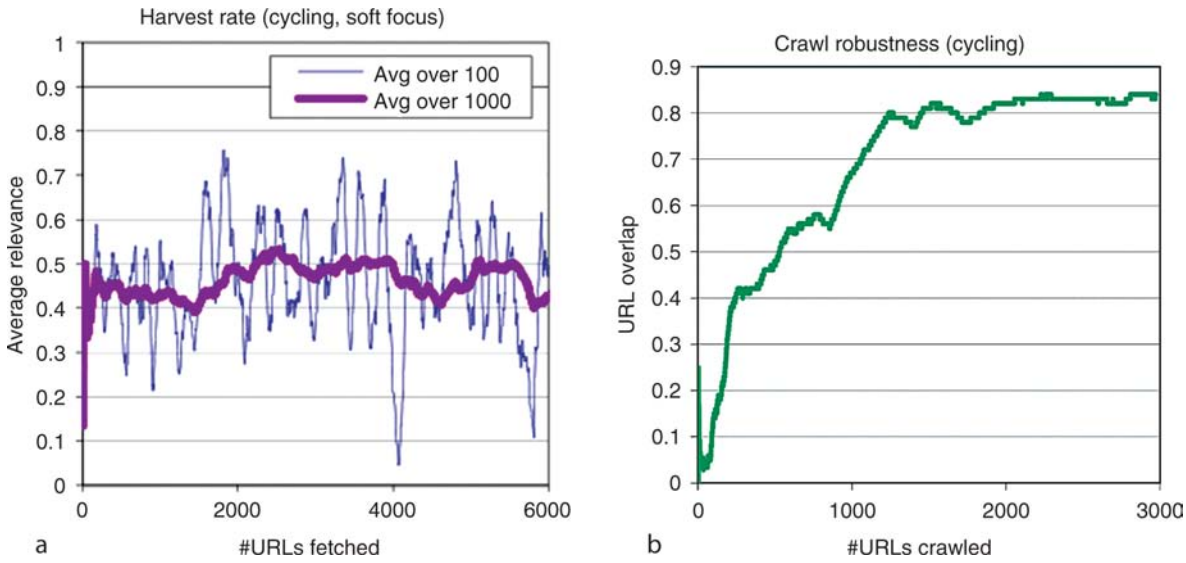
Basic Topic-Focused Crawler

During 1997–1998, document classification witnessed renewed interest and enhancements from the use of hyperlinks [5]. Rather than use occurrence of a few query words as an indicator of payoff during a crawl, Chakrabarti et al. [4] coupled a pre-trained text classifier with a crawler as shown in Fig. 3. The classifier may be trained on a topic taxonomy such as the one published by Yahoo! or the Open Directory Dmoz, and a few topics (nodes) c^* in the taxonomy (and all their descendants, by inheritance) flagged as positive or rewarding for the crawler.

This organization depends on the (verified [9,6]) premise that pages within a short link distance of each other are topically similar or related. In particular, if $\Pr(c^*|u)$ is large and hyperlink (u, v) exists, then the hypothesis is that $\Pr(c^*|v)$ is also likely to be large quite often. For reasonably broad topics about which there are sufficiently dense link communities on the Web, this hypothesis holds up well and leads to reliable rates of page collection. Some sample results are shown in Fig. 4.



Focused Web Crawling. Figure 3. Basic topic-focused crawler [3].

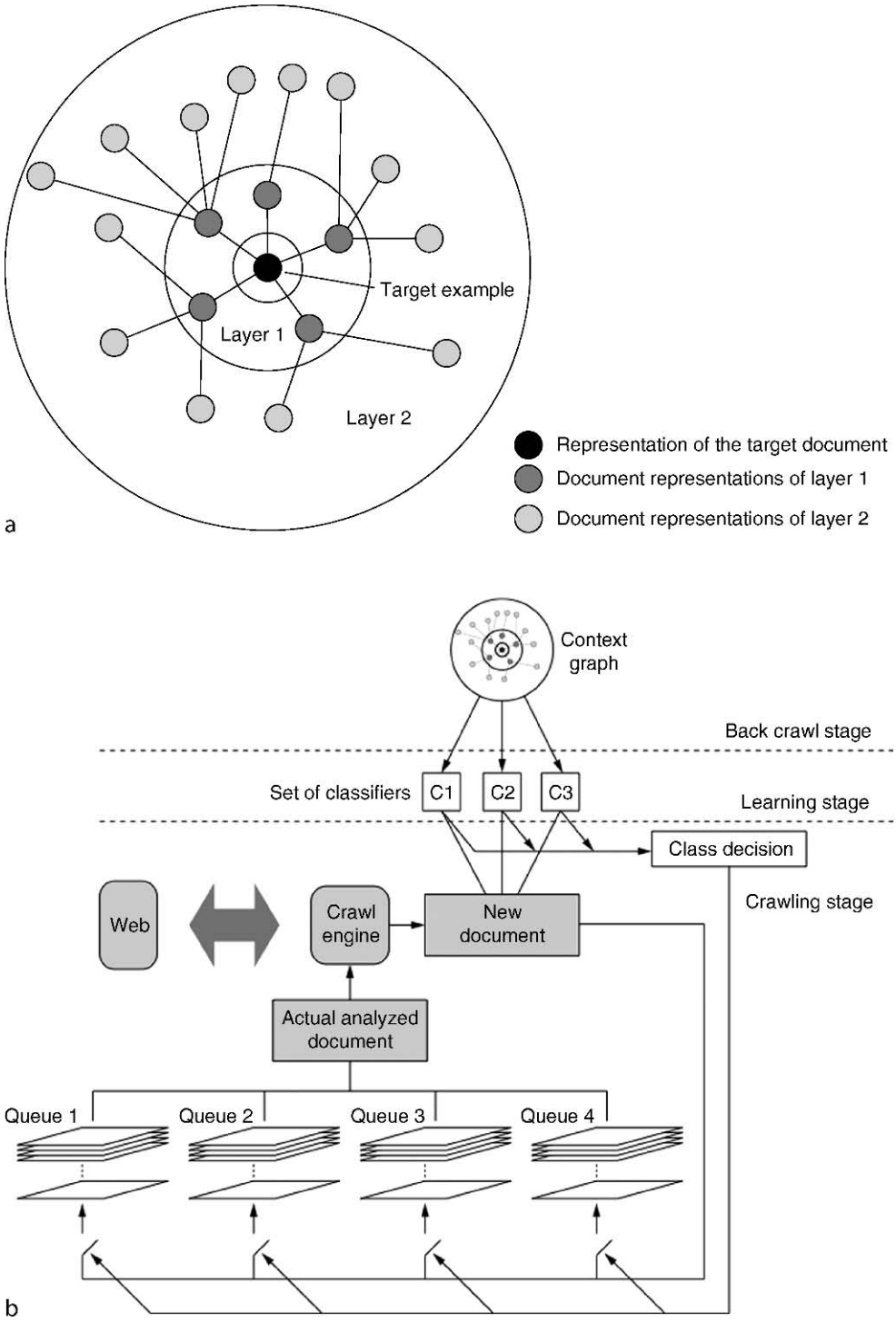


Focused Web Crawling. Figure 4. (a) The basic focused crawler manages to keep up a reasonable "harvest rate" of collecting relevant pages. (b) Started from different seed URLs, the focused crawler navigates to the dominant communities on the focus topic and visits largely overlapping sites and pages [4].

Focused Crawling Using Context Graphs

The next enhancement to focused crawling was the use of *context graphs* [10]. The first generation of topic-focused crawlers was myopic: the relevance of page u was used as a surrogate for the benefits of crawling outlink (u, v), and this prediction was limited to only one hop. In Reinforcement Learning [18], an area of Machine Learning, techniques have been evolved to anticipate more distant payoffs. In focused crawling with context graphs, during the training phase, the trainer presents to the system paths $u_1,...,u_g$ of various lengths (up to some modest limit, say, four hops) leading to goal nodes. The goal node u_g is a node on the topic/s of focus. The system includes a supervised learning module that then learns to predict, given a node u, the (minimum) distance to a goal node. When the trained crawler is deployed, after fetching page/node u, it estimates the shortest link distance from u to a goal node, and uses this as the priority for expanding outlinks of u; see Fig. 5.

Figure 6 shows the benefits from going beyond a greedy one-hop prediction paradigm. A similar technique



Focused Web Crawling. Figure 5. Focused crawling using context graph [10].

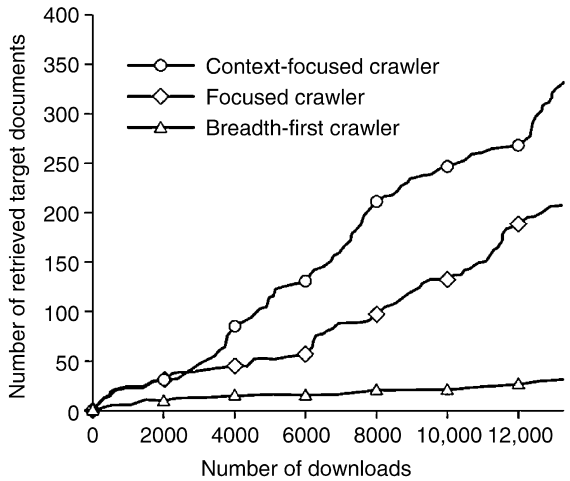
has been implemented by McCallum et al. [17]. More recently, Baberia et al. [1] have cast the problem of estimating the distance to a goal node as an ordinal regression, and given scalable and effective algorithms.

Training a Focused Crawler Online

The focused crawlers discussed thus far are all trained once, off-line, and then start harvesting the Web. The next generation of focused crawlers [7] "learn on

the job." Compare Fig. 7 with the basic focused crawler shown in Fig. 3. In the new edition, once a page u is approved by the baseline topic classifier for getting its outlinks visited, each outlink (u, v) is submitted to an *apprentice*. Initially, the apprentice is inexperienced, and might assign the same priority to each outlink (u, v) that the baseline learner does. As pages continue to be fetched, the baseline classifier classifies each of them, and the apprentice "watches." The baseline classifier acts as a *critic*, pointing out where the apprentice was right and wrong. The apprentice uses features not only from the whole of u, but also features specific to the outlink (u,v), exploiting the DOM tree of u.

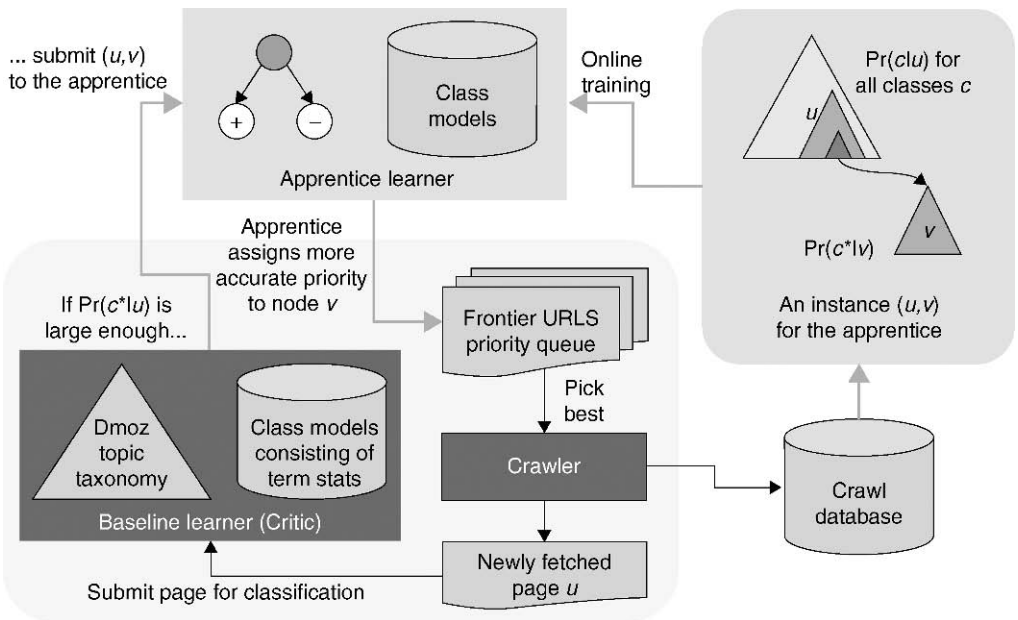
Gradually, the apprentice learns to predict, over and above the critic's relevance judgment of u, the expected profit from traversing link (u,v). Continually training the apprentice is much easier than training the baseline classifier. The apprentice can be a simple regression algorithm, whereas the baseline classifier has to handle much more data and many topics. As the apprentice is switched into the decision process, the number of fetched pages that must be discarded because they are off-topic, also called the "loss rate," drops dramatically. This is shown in Fig. 8a. Overall, as Fig. 8b shows, a focused crawler that learns online can significantly reduce the loss rate and thus boost the harvest rate.



Focused Web Crawling. Figure 6. Context graphs increase harvest of relevant pages [10].

Reinforcement Learning Using Markov Models

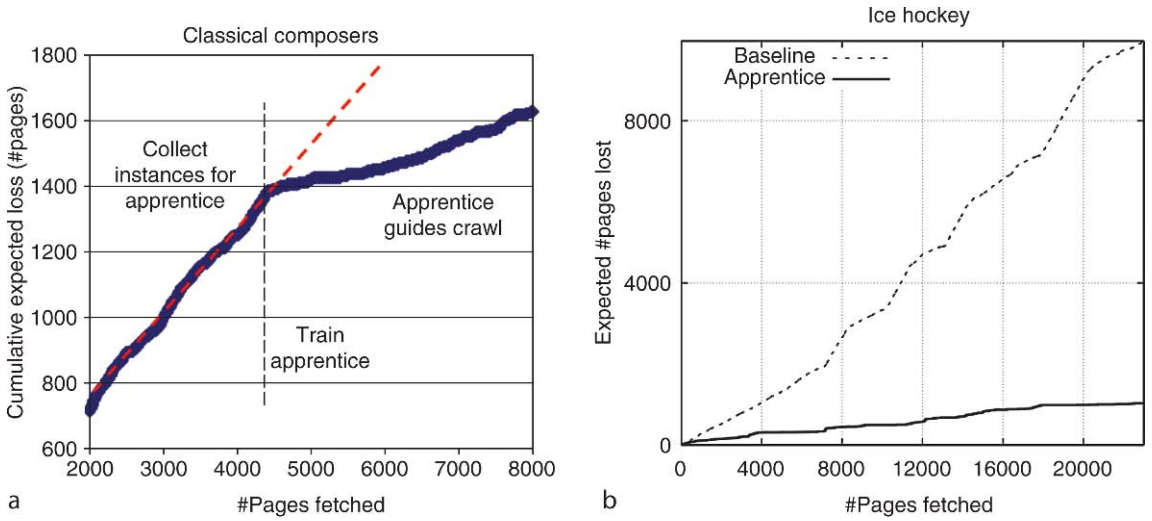
The next step in improving the ability of focused crawlers to spot distant rewards was taken by learning a more detailed representation of the path from the current node to a goal node, rather than just the estimated distance [19]. This is expected to be effective especially for crawling tasks where paths leading into goal nodes have some regularity. For example, paths leading from the root page of a Computer Science



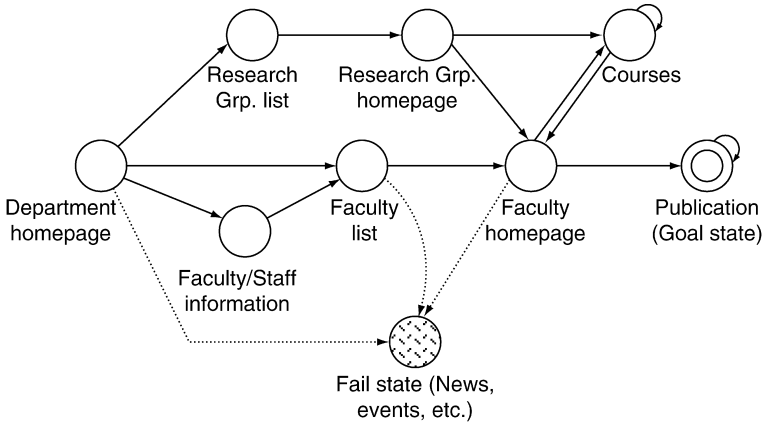
Focused Web Crawling. Figure 7. Topic-focused crawler that learns online during a crawl [7].

Department to research papers about computer vision, or paths leading from the root page of a company to a set of profiles of its top officers, are fairly regular. In fact, one can even give informal but crisp descriptions of the intermediate pages. In the first example, the intermediate pages will most likely be a roster of faculty members with research interests, followed by a faculty homepage, which either has links to PDF files, or links to a page listing publications, which then links to PDF files. In the second example, a surfer is most likely to navigate through an "about us" page, which may list the top office-bearers, each with a link to a profile page, or the profiles may be inlined. An example state transition diagram is given in Fig. 9.

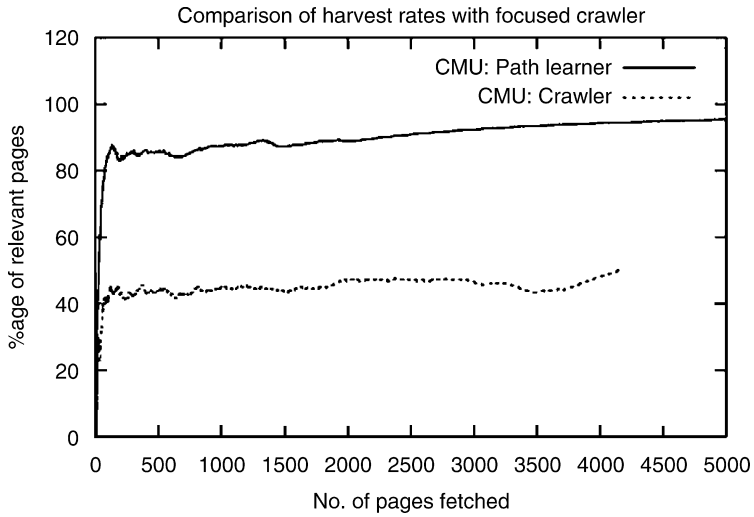
A hidden Markov model (HMM) is ideal for capturing this form of regularity. For label prediction in HMMs, Conditional Random Fields (CRFs) [13] are the best-known technique. Training the CRF involves slightly more work than in the case of context graphs: here, intermediate nodes must be assigned labels. Once this is done, the problem of learning a transition model is very similar to other sequence-labeling applications, like part-of-speech tagging (nodes are words, labels are parts of speech) and named-entity recognition (nodes are words, labels are named entity types like *person* or *location*). Over and above the general framework, smart state and feature design are a must for good accuracy. Also, during crawling, the crawl path up to



Focused Web Crawling. Figure 8. A focused crawler that "learns on the job" improves harvest rate significantly [7].



Focused Web Crawling. Figure 9. Example state transitions modeling paths from a department homepage to research papers [19].



Focused Web Crawling. Figure 10. Using a CRF to predict the expected reward continuing from the current path leads to significantly improved harvest [8].

the current node is assigned various label sequences with various probabilities, and these must contribute to a distance-discounted reward function as in reinforcement learning [18]. For crawling tasks with path regularities, this form of reward prediction is more accurate than the standard topic-focused crawler, leading to substantially improved page harvest rates; see Fig. 10.

Key Applications

Focused crawling, broadly interpreted as goal-directed crawl prioritization with resource constraints, is of interest to almost anyone that has to run a crawler of any substantial scale. All crawlers include a module for frontier prioritization, but only some degree of machine learning usually qualifies it to be explicitly called a "focused crawler." Focused crawling is especially beneficial in applications where content pertaining to a limited topic area must be collected from a much larger Web collection, possibly the whole Web, and subjected to topic-specific post-processing, such as information extraction specific to particular people, regions, languages, or products and services.

Future Directions

Focused crawlers are now available in the public domain (see http://ivia.ucr.edu/ and http://combine.it.lth.se/). Many companies in the vertical Web search space implement focused crawlers. While many of the technical issues around focused crawling are now reasonably solved, the operational details of starting, monitoring, and maintaining a focused crawler remain relatively unknown in the public domain.

URL to Code

A public domain focused crawler implementation is available from http://ivia.ucr.edu/ under the Limited Gnu Public License. Another public domain version is available from the ALVIS project, see http://combine.it.lth.se/. Also see the Wikipedia page on Focused Crawling at http://en.wikipedia.org/wiki/Focused_crawler.

Cross-references

- ▶ Classification
- ▶ Digital Libraries
- ▶ Document Databases
- ▶ Document Links and Hyperlinks
- ▶ Graph Database
- ▶ Incremental
- ▶ Indexing the Web
- ▶ Information Retrieval
- ▶ Personalized Web Search
- ▶ Relevance Feedback
- ▶ Social Networks
- ▶ Text Categorization
- ▶ Text Mining
- ▶ Web Characteristics and Evolution
- ▶ Web Crawler Architecture
- ▶ Web Crawler
- ▶ Web Harvesting

Recommended Reading

1. Babaria R., Saketha Nath J., Krishnan S., Sivaramakrishnan K.R., Bhattacharyya C., and Murty M.N. Focused crawling with scalable ordinal regression solvers. In Proc. 24th Int. Conf. on Machine Learning, 2007, pp. 57–64.
2. Broder A. et al. Graph structure in the Web: experiments and models. In Proc. 9th Int. World Wide Web Conference, 2000, pp. 309–320.
3. Chakrabarti S. Mining the Web: Discovering Knowledge from Hypertext Data, Morgan-Kauffman, 2002.
4. Chakrabarti S., van den Berg M., and Dom B. Focused crawling: a new approach to topic-specific Web resource discovery. Comput. Netw., 31:1623–1640, 1999.
5. Chakrabarti S., Dom B., and Indyk P. Enhanced hypertext categorization using hyperlinks. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 307–318.
6. Chakrabarti S., Joshi M.M., Punera K., and Pennock D.M. The structure of broad topics on the Web. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 251–262.
7. Chakrabarti S., Punera K., and Subramanyam M. Accelerated focused crawling through online relevance feedback. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 148–159.
8. Cho J., Garcia-Molina H., and Page L. Efficient crawling through URL ordering. In Proc. 7th Int. World Wide Web Conference, 1998, pp. 161–172.
9. Davison B.D. Topical locality in the Web. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 272–279.
10. Diligenti M., Coetzee F., Lawrence S., Giles C.L., and Gori M. Focused crawling using context graphs. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 527–534.
11. Dill S., Ravi Kumar S., McCurley K.S., Rajagopalan S., Sivakumar D., and Tomkins A. Self-similarity in the Web. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 69–78.
12. Herseovici M., Jacovi M., Maarek Y.S., Pelleg D., Shtalhaim M., and Ur S. The shark-search algorithm – an application: tailored Web site mapping. In Proc. 7th Int. World Wide Web Conference, 1998, pp. 317–326.
13. Lafferty J., McCallum A., and Pereira F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proc. 18th Int. Conf. on Machine Learning, 2001, pp. 282–289.
14. Najork M. and Weiner J. Breadth-first search crawling yields high-quality pages. In Proc. 10th Int. World Wide Web Conference, 2001, pp. 114–118.
15. Pandey S. and Olston C. User-centric Web crawling. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 401–411.
16. Page L., Brin S., Motwani R., and Winograd T. The PageRank citation ranking: bringing order to the Web. Manuscript, Stanford University, 1998.
17. Rennie J. and McCallum A. Using reinforcement learning to spider the web efficiently. In Proc. 16th Int. Conf. on Machine Learning, 1999, pp. 335–343.
18. Sutton R.S. and Barto A.G. Reinforcement Learning: An Introduction. MIT, March 1998.
19. Vinod Vydiswaran V.G. and Sarawagi S. Learning to extract information from large Websites using sequential models. In Proc. 11th Int. Conf. on Management of Data, 2005, pp. 3–14.

Focus-Plus-Context

► Distortion Techniques

FOL

► First-Order Logic: Semantics
► First-Order Logic: Syntax

F

FOL Modeling of Integrity Constraints (Dependencies)

Alin Deutsch
University of California-San Diego, La Jolla, CA, USA

Synonyms

Relational integrity constraints; Dependencies

Definition

Integrity constraints (also known as *dependencies* in the relational model) are domain-specific declarations which indicate the intended meaning of the data stored in a database. They complement the description of the structure of the data (e.g., in the relational model the structure is given by listing the names of tables and the names and types of their attributes). Integrity constraints express properties that must be satisfied by all instances of a database schema that can arise in the intended application domain (e.g., "no two distinct employees may have the same ssn value", "departments have a single manager", etc.).

Historical Background

The reference textbook [1] provides a comprehensive, unifying overview of the many special classes of relational dependencies, their modeling in first-order logic (FOL), and the key problems in the study of dependencies. This entry is a condensed form of Chap. 10 in [1] (excluding the material on reasoning about dependencies). For more detail on the motivation and history of relational dependency theory, see the excellent survey papers [10,14,22].

Historically, *functional* dependencies were the first class to be introduced (by Codd [3]). *Multi-valued* dependencies were discovered independently in [5,9,24]. These were followed by a proliferation of dependency classes, typically expressed using ad hoc syntax. These include

join dependencies, and *inclusion* dependencies (a.k.a. referential integrity constraints) [4]. Fagin [9] first introduced *embedded multi-valued* dependencies (MVDs that hold in the projection of a relation), while [15] introduced the distinct class of *projected* JDs. Other classes include *subset* dependencies [20], *mutual* dependencies [17], *generalized mutual* dependencies [16], *transitive* dependencies [18], *extended transitive* dependencies [19], and *implied* dependencies [12].

The movement towards ever-refined classifications of dependencies was prompted mainly by research on automatic reasoning about dependencies, in particular their axiomatization. This movement was soon countered by the drive to unify the treatment of the various classes by finding a formalism that subsumes all of them. Nicolas [17] is credited with first observing that FDs, MVDs and others have a natural representation in first-order logic. In parallel, Beeri and Vardi [2] introduced *tuple-generating* and *equality-generating* dependencies expressed in a tableaux-based notation, shown to be equivalent in expressive power to Fagin's *typed embedded dependencies* [8], which were expressed in first-order logic. The class DED$^{\neq}$ was introduced in [7] as an extension of embedded dependencies with disjunction and non-equalities (see also [13] for a particular case of DED$^{\neq}$s). Deutsch et al. [6] further extends the DED$^{\neq}$ class to allow negated relational atoms. Research on using arbitrary first-order logic sentences to specify constraints includes [11,17,21].

In contrast, *algebraic dependencies* are an alternative unifying framework developed by Yannakakis and Papadimitriou [23]. Algebraic dependencies are statements of containment between queries written in relational algebra, and have the same expressive power as first-order logic.

Foundations

This section lists a few fundamental classes of relational dependencies, subsequently showing how they can be expressed in first-order logic. In the following, $R(U)$ denotes the schema of a relation with name R and set of attributes U.

Functional Dependencies

A *functional dependency (FD)* on relations of schema $R(U)$ is an expression of the form

$$R : X \rightarrow Y, \tag{1}$$

where $X \subseteq U$ and $Y \subseteq U$ are subsets of R's attributes. Instance r of schema $R(U)$ is said to *satisfy* FD *fd*, denoted $r \models fd$, if whenever tuples $t_1 \in r$ and $t_2 \in r$ agree on all attributes in X, they also agree on all attributes in Y:

$$r \models fd \Leftrightarrow \text{for every } t_1, t_2 \in r \text{ if } \pi_X(t_1) = \pi_X(t_2) \text{ then } \pi_Y(t_1) = \pi_Y(t_2).$$

Here, $\pi_X(t)$ denotes the projection of tuple t on the attributes in X.

For instance, consider a relation of schema

review(paper, reviewer, track)

listing a conference track a paper was submitted to, and a reviewer it was assigned to. The fact that every paper can be submitted to a single track is stated by the functional dependency

$$review : \ paper \rightarrow track.$$

Key Dependencies

In the particular case when $Y = U$, a functional dependency of form (1) is called a *key dependency*, and the set of attributes X is a called a *key for* R.

Join Dependencies

A *join dependency (JD)* on relations of schema $R(U)$ is an expression of the form

$$R : \bowtie [X_1, X_2, \ldots, X_n], \tag{2}$$

where for each $1 \leq i \leq n$, $X_i \subseteq U$, and $\bigcup_{1 \leq i \leq n} X_i = U$. Instance r of schema $R(U)$ *satisfies* JD *jd*, denoted $r \models jd$, if the n-way natural join of the projections of r on each of the attribute sets X_i yields r:

$$r \models jd \Leftrightarrow r = \Pi_{X_1}(r) \bowtie \Pi_{X_2}(r) \bowtie \ldots \bowtie \Pi_{X_n}(r).$$

Here, $\Pi_X(r)$ denotes the projection of relation r on the attributes in X.

In the example, assume that a paper may be submitted for consideration by various tracks (e.g., poster or full paper), and that reviewers are not tied to the tracks. It makes sense to expect that for any given paper p, any track information listed with a reviewer of p is also listed with all other reviewers of p, since track and reviewer information are not correlated. This is expressed by requiring that the join of the projection of *review* on *paper, track* and of the projection on *paper, reviewer* yields the *review* table:

$$review : \bowtie [\{paper, track\}, \{paper, reviewer\}].$$

Multi-Valued Dependencies

In the particular case when $n = 2$, a join dependency of form (2) is called a *multi-valued dependency (MVD)*. Because MVDs were historically introduced and studied before JDs, they have their own notation: an MVD $R : \bowtie [X_1, X_2]$ is denoted

$$R : \ X \twoheadrightarrow Y, \tag{3}$$

where $X = X_1 \cap X_2$ and $Y = X_1 \setminus X_2$.

In the running example, the join dependency turns out to be a multi-valued dependency, which can be expressed using the following MVD-specific syntax:

$$review : \ paper \twoheadrightarrow track.$$

Inclusion Dependencies

Functional and join dependencies and their special-case subclasses each pertain to single relations. The following class of dependencies can express connections between relations. An *inclusion dependency (IND)* on pairs of relations of schemas $R(U)$ and $S(V)$ (with R and S not necessarily distinct) is an expression of the form

$$R[X] \subseteq S[Y], \tag{4}$$

where $X \subseteq U$ and $Y \subseteq V$. Inclusion dependencies are also known as *referential constraints*. Relations r and s of schemas $R(U)$, respectively $S(V)$ satisfy inclusion dependency *id*, denoted $r, s \models id$, if the projection of r on X is included in the projection of s on Y:

$$r, s \models id \Leftrightarrow \ \Pi_X(r) \subseteq \Pi_Y(s).$$

When R and S refer to the same relation name, then $r = s$ in the above definition of satisfaction.

In the running example, assume that the database contains also a relation of schema

$$PC(member, \ affiliation)$$

listing the affiliation of every program committee member. Then one can require that papers be reviewed only by PC members (no external reviews allowed) using the following IND:

$$review[reviewer] \subseteq PC[member].$$

Foreign Key Dependencies

In the particular case when Y is a key for relations of schema S ($S : Y \rightarrow V$), INDs of form (4) are called *foreign key dependencies*. Intuitively, in this case the projection on X of every tuple t in r contains the key of a tuple from the "foreign" table s.

In the running example, assuming that every PC member is listed with only one primary affiliation, *member* is a key for *PC*, so the IND above is really a foreign key dependency.

Expressing Dependencies in First-Order Logic

Embedded Dependencies Despite their independent introduction and widely different syntax, it turns out that all classes of dependencies illustrated above (and many more, including the ones mentioned in the historical background section) can be expressed using a fragment of the language of first-order logic. This fragment is known as the class of *embedded dependencies*, which are formulas of the form

$$\begin{aligned} &\forall x_1 ... \forall x_n \ \varphi(x_1, ..., x_n) \rightarrow \\ &\exists z_1 ... \exists z_k \ \psi(y_1, ..., y_m), \end{aligned} \tag{5}$$

where $\{z_1,...,z_k\} = \{y_1,...,y_m\} \setminus \{x_1,...,x_n\}$, and φ is a possibly empty and ψ is a non-empty conjunction of relational and equality atoms. A *relational atom* has form $R(w_1,...,w_l)$, and an *equality atom* has form $w = w'$, where each of $w,w',w_1,...,w_l$ are variables or constants. The particular case when all atoms in ψ are equalities yields the class known as *equality-generating dependencies (EGD)*, while the case when only relational atoms occur in ψ defines the class of *tuple-generating dependencies (TGD)*.

The power of embedded dependencies is illustrated next by expressing the classes of dependencies described above.

Functional Dependencies

Assume without loss of generality that in (1), $|X| = k$, $|Y| = l$, and $|U \setminus (X \cup Y)| = m$, and that the ordering of attributes in U is $U = X, Y, Z$. Then any functional dependency of form (1) is expressible as the embedded dependency (actually an EGD):

$$\begin{aligned} &\forall x_1 ... \forall x_k \ \forall y_1 ... \forall y_l \ \forall y'_1 ... \forall y'_l \ \forall z_1 ... \forall z_m \\ &\forall z'_1 ... \forall z'_m \\ &R(x_1, ..., x_k, y_1, ..., y_l, z_1, ..., z_m) \wedge \\ &\quad R(x_1, ..., x_k, y'_1, ..., y'_l, z'_1, ..., z'_m) \rightarrow \\ &\quad y_1 = y'_1 \wedge ... \wedge y_l = y'_l \end{aligned}$$

In the particular case when X is a key, $m = 0$ and there are no z_i, z_i' variables in the above embedded dependency.

The functional dependency *review* : *paper* → *track* in the running example, can be expressed as the following embedded dependency:

$$\forall p \forall r \forall t \forall r' \forall t' \ review(p, r, t) \wedge review(p, r', t') \rightarrow t = t'.$$

Join Dependencies

Join dependencies of form (2), are expressed by observing that for every relation r of schema $R(U)$, the inclusion

$$r \subseteq \Pi_{X_1}(r) \bowtie \Pi_{X_2}(r) \bowtie \ldots \bowtie \Pi_{X_n}(r)$$

holds trivially. Therefore only the opposite inclusion needs to be expressed,

$$\Pi_{X_1}(r) \bowtie \Pi_{X_2}(r) \bowtie \ldots \bowtie \Pi_{X_n}(r) \subseteq r.$$

This end requires the following notation. Recalling that the set of attributes U in the schema of R is ordered, let $pos(A)$ denote the position of an attribute $A \in X$ in the ordered set U. For a set of attributes $X \subseteq U$, $pos(X)$ denotes the set $\{pos(A) | A \in X\}$. In the following, given a tuple of variables $\bar{u}$, $\bar{u}[k]$ denotes the kth variable in $\bar{u}$.

1. Let $\{\bar{u}_i\}_{1 \le i \le n}$ be a family of tuples of $|U|$ variables each, such that for every $1 \le i < j \le n$ and every $1 \le k \le |U|$, $\bar{u}_i[k] = \bar{u}_j[k]$ if and only if $k \in pos(X_i \cap X_j)$.
2. Let $\bar{w}$ be a tuple of $|U|$ variables, such that for every $1 \le k \le |U|$ and every $1 \le i \le n$, if $k \in pos(X_i)$ then $\bar{w}[k] = \bar{u}_i[k]$. It is easy to check that $\bar{w}$ is well-defined: indeed, since $\bigcup_i X_i = U$, for each k there is at least one i with $k \in pos(X_i)$. Moreover, by definition of the family $\{\bar{u}_i\}_i$, $\bar{u}_i[k] = \bar{u}_j[k]$ whenever $k \in pos(X_i)$ and $k \in pos(X_j)$.
3. Finally, let $V = \{v_1, \ldots v_m\} = \bigcup_{i=1}^{n} \bar{u}_i$ (with each tuple $\bar{u}_i$ viewed as a set of variables). Notice that variables occurring in several $\bar{u}_i$ tuples appear just once among V.

Then the join dependency of form (2) is given by the embedded dependency (a TGD, really):

$$\forall v_1 \ldots \forall v_m \ R(\bar{u}_1) \wedge \ldots \wedge R(\bar{u}_n) \rightarrow R(\bar{w}).$$

Join dependency *review* : ⋈[{*paper, track*}, {*paper, reviewer*}] is expressed as the following embedded dependency:

$$\forall p \forall r_1 \forall t_1 \forall r_2 \forall t_2 \ review(p, r_1, t_1) \wedge review(p, r_2, t_2) \rightarrow review(p, r_2, t_1).$$

Inclusion Dependencies

To express inclusion dependencies, assume without loss of generality that in (4) $R(U) = R(Z, X)$ and $S(V) = S(Y, W)$. Then the inclusion dependency (4) is captured by the following embedded dependency (TGD):

$$\forall z_1 \ldots \forall z_{|Z|} \ \forall x_1 \ldots \forall x_{|X|} \ R(z_1, \ldots, z_{|Z|}, x_1, \ldots, x_{|X|}) \rightarrow \exists w_1 \ldots \exists w_{|W|} \ S(x_1, \ldots, x_{|X|}, w_1, \ldots, w_{|W|}).$$

In the running example, the inclusion dependency *review*[*reviewer*] ⊆ *PC*[*member*] is expressible as the embedded dependency

$$\forall p \forall r \forall t \ review(p, r, t) \rightarrow \exists a \ PC(r, a).$$

Other Classes of Dependencies

Embedded dependencies turn out to be sufficiently expressive to capture virtually all other classes of dependencies studied in the literature.

Other Constraints

By employing more expressive sub-languages of first-order logic, one can capture additional, naturally occurring constraints. For instance, extending embedded dependencies with disjunction, one obtains the language of *disjunctive embedded dependencies (DED)* of form

$$\forall \bar{x} \ \varphi(\bar{x}) \rightarrow \bigvee_{i=1}^{l} \exists \bar{z}_i \ \psi_i(\bar{y}_i), \qquad (6)$$

where $\bar{x}$ is a tuple of variables, and so are $\bar{z}_i$, $\bar{y}_i$ for every $1 \le i \le l$. Analogously to (5), $\bar{z}_i = \bar{y}_i \setminus \bar{x}$. φ and each ψ_i are conjunctions of relational and equality atoms as in (5). If in addition one allows *non-equality atoms* of the form $w \neq w'$, one obtains the class of DEDs with non-equality, DED$^{\neq}$, which is in turn a fragment of first-order logic.

Cardinality Constraints

The language DED$^{\neq}$ can express cardinality constraints. In the running example, $\delta_1 \in$ DED$^{\neq}$ states that every paper has at least two reviews:

$$(\delta_1) \quad \forall p \forall r_1 \forall t \ review(p, r_1, t) \rightarrow \exists r_2 \ review(p, r_2, t) \wedge r_1 \neq r_2.$$

δ_2 below states that every paper receives at most two reviews:

$$(\delta_2) \quad \forall p \forall r_1 \forall r_2 \forall r_3 \forall t \ \ review(p, r_1, t) \wedge review(p, r_2, t) \wedge review(p, r_3, t) \wedge r_1 \neq r_2 \rightarrow r_3 = r_1 \vee r_3 = r_2.$$

Note that the conjunction of δ_1 and δ_2 requires each paper to receive precisely two reviews.

Domain Constraints

The language DED$^{\neq}$ can be employed to restrict the domain of an attribute. Such restrictions are commonly known as *domain constraints*. For instance, in the running example, this is how to specify that the conference has only three kinds of tracks: "research", "industrial" and "demo":

$$(\delta_3) \ \forall_p \forall_r \forall_t \ review(p, r, t) \rightarrow t = \text{"}research\text{"} \vee t = \text{"}industrial\text{"} \vee t = \text{"}demo\text{"}.$$

Representational Constraints

Many application are based on data models that are richer than the relational model (e.g., object-oriented, object-relational, XML, RDF models). However, they often leverage the mature relational technology by supporting the storage of their data in a relational database. The resulting relations satisfy certain constraints that stem from the original data model. This entry refers to them as *representational constraints*. In order to maintain the relational storage and to efficiently process queries over it, it is imperative to exploit these representational constraints. An obstacle to doing so is the fact that, depending on the original model they encode relationally, representational constraints tend to not fit neatly into any of the classes of relational integrity constraints devised for native relational data. Again, first-order logic comes to the rescue.

Representational constraints for the relational representation of XML are illustrated next. While there are many possible representations, they are all equivalent to the following simple one [7] which captures the fact that in the XML data model, elements are the tagged nodes of a tree. The tree is represented using the following relations (among others):

$$elem(node, tag)$$
$$child(source, target)$$
$$desc(source, target)$$

where the *elem* relation lists for every element e, the identifier of the tree node modeling e, and the tag of e; the *child* table is the edge relation of the XML tree, according to which *source* is the identifier of the parent node and *target* the identifier of the child node; *desc* is the descendant relation in the tree, whose *target* node is a descendant of the *source* node. Any instance storing an actual XML tree in these tables must satisfy, among others, the following constraints, all expressible in DED$^{\neq}$: every element has at most one tag (expressed in (7) below); every element has at most one parent (8); children of a node are also descendants of this node (9); the descendant relation is transitive (10); if two elements have a common descendant, then they either coincide or one is the descendant of the other (11).

$$\forall n \forall t_1 \forall t_2 \ \ elem(n, t_1) \wedge elem(n, t_2) \rightarrow t_1 = t_2, \quad (7)$$

$$\forall n \forall p_1 \forall p_2 \ \ child(p_1, n) \wedge child(p_2, n) \rightarrow p_1 = p_2, \quad (8)$$

$$\forall s \forall t \ \ child(s, t) \rightarrow desc(s, t), \quad (9)$$

$$\forall s \forall u \forall t \ \ desc(s, u) \wedge desc(u, t), \rightarrow desc(s, t), \quad (10)$$

$$\forall n_1 \forall n_2 \forall d \ \ desc(n_1, d) \wedge desc(n_2, d), \rightarrow n_1 = n_2 \vee desc(n_1, n_2) \vee desc(n_2, n_1). \quad (11)$$

Key Applications

The role of integrity constraints is to incorporate more semantics into the data model. This in turn enables an improved schema design, as well as the delegation to the database management system (DBMS) of the task of enforcing and exploiting this semantics.

Schema Design

Integrity constraints are useful for selecting the most appropriate schema for a particular database application domain. It turns out that the same information can be stored in tables in many ways, some more efficient than others with respect to avoiding redundant storage of data, improved update and query performance, and better readability. While in the early days of database application development, appropriate schema selection started out as an art, it quickly evolved into a science enabling automatic schema design tools (also known as "wizards"). Such tools are based on database theory research that proposes schema design methodology starting from a single, "universal relation", which is then decomposed into new relations that satisfy desirable normal forms that take advantage of the known integrity constraints [1]. For instance, in

the running example, both the FD *review* : *paper* → *track* and the MVD *review* : *paper* ↠ *track* suggest decomposing relation *review* into two tables, one associating papers with their track, and one associating papers with their reviewers, to avoid the redundant listing of track information with every reviewer.

Automatic Integrity Enforcement

For the purpose of integrity enforcement, the DBMS automatically checks every update operation for compliance with the declared integrity constraints, automatically rejecting non-compliant updates. The database administrator can therefore rest assured that the integrity of her data will be preserved despite any bugs in the applications accessing the database. This guarantee is all the more important when considering that several applications are usually running against the same database. These applications are not always under the control of the database administrator, and are often developed by third parties, which renders their code unavailable for verification. Even with full access to the application code, verification (like all software verification) is technically challenging and does not scale well with increasing number of applications or modifications to their code.

Query Optimization

The query optimizer can exploit its constraint-derived understanding of the data to automatically rewrite queries for more efficient execution. Delegating this task to the DBMS ensures that queries are efficiently executed even when they are issued by applications whose developers write the queries in sub-optimal forms due to insufficient insight into the integrity constraints. More importantly, automatic optimization inside the DBMS handles the queries generated by tools rather than human developers. These queries are usually quite far from optimal. This problem is especially prevalent when queries over a rich data model *M* are translated to relational queries over the relational representation of *M* (e.g., XQuery queries over relational storage of XML). The *chase* is a very powerful tool for reasoning about (and exploiting in optimization) dependencies specified in first-order logic (see [1] for references to the papers that independently discovered the chase, for various classes of dependencies).

A Unified View of Dependencies

Given the plethora of classes of dependencies formulated and studied independently in the literature, the task of specifying the meaning of an application domain via integrity constraints would be daunting if the developer had to fit them into these classes. In addition, tailoring the tasks of schema design, optimization and integrity enforcement to every class of dependencies (and every combination of such classes) is impractical. First-order logic provides a formalism for the simple specification of integrity constraints, with well-understood semantics and sufficient expressive power to capture all common classes of dependencies, and beyond. The insight that virtually all dependency classes are expressible in the same formalism set the foundation for their uniform treatment in research and applications.

Cross-references

- ▶ Chase
- ▶ Equality-Generating Dependencies
- ▶ Tuple-Generating Dependencies

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of databases. Addison-Wesley, Reading, MA, 1995.
2. Beeri C. and Vardi M.Y. The implication problem for data dependencies. In Proc. Int. Conf. on Algorithms, Languages and Programming, 1981, pp. 73–85.
3. Codd E.F. Relational completeness of database sublanguages. In Courant Computer Science Symposium 6: Data Base Systems. R. Rustin (ed.). Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 65–98.
4. Date C.J. Referential integrity. In Proc. 7th Int. Conf. on Very Data Bases, 1981, pp. 2–12.
5. Delobel C. Normalization and hierarchical dependencies in the relational data model. ACM Trans. Database Syst., 3(3):201–222, 1978.
6. Deutsch A., Ludäscher B., and Nash A. Rewriting queries using views with access patterns under integrity constraints. Theor. Comput. Sci., 371(3):200–226, 2007.
7. Deutsch A. and Tannen V. XML queries and constraints, containment and reformulation. Theor. Comput. Sci., 336 (1):57–87, 2005.
8. Fagin R. Horn clauses and database dependencies. J ACM, 29(4):952–985, 1982.
9. Fagin R. Multivalued dependencies and a new normal form for relational databases. ACM Trans. Database Syst., 2(3):262–278, 1977.
10. Fagin R. and Vardi M.Y. The theory of data dependencies: A survey. In Mathematics of Information Processing: Proceedings of Symposia in Applied Mathematics, vol. 34, M. Anshel and W. Gewirtz (eds.). American Mathematical Society, Providence, RI, 1986, pp. 19–27.
11. Gallaire H. and Minker J. Logic and databases. Plenum Press, New York, 1978.
12. Ginsburg S. and Zaiddan S.M. Properties of functional dependency families. JACM, 29(4):678–698, 1982.

13. Grahne G. and Mendelzon A.O. Tableau techniques for querying information sources through global schemas. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 332–347.
14. Kannellakis P.C. Elements of relational database theory. In Handbook of Theoretical Computer Science. J. Van Leeuwen (ed.). Elsevier, Amsterdam, 1991, pp. 1074–1156.
15. Maier D., Ullman J.D., and Vardi M.Y. On the foundations of the universal relation model. ACM Trans. Database Syst., 9(2):283–308, 1984.
16. Mendelzon A.O. and Maier D. Generalized mutual dependencies and the decomposition of database relations. In Proc. 5th Int. Conf. on Very Data Bases, 1979, pp. 75–82.
17. Nicolas J.-M. First order logic formalization for functional, multivalued, and mutual dependencies. Acta Inf., 18(3):227–253, 1982.
18. Paredaens J. Transitive dependencies in a database scheme. Technical Report R387, MBLE, Brussels, 1979.
19. Parker D.S. and Parsaye-Ghomi K. Inference involving embedded multivalued dependencies and transitive dependencies. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1980, pp. 52–57.
20. Sagiv Y. and Walecka S.F. Subset dependencies and a completeness result for a subclass of embedded multivalued dependencies. J ACM, 29(1):103–117, 1982.
21. Vardi M.Y. On decomposition of relational databases. In Proc. 23rd Annual Symp. on Foundations of Computer Science, pp. 176–185, 1982.
22. Vardi M.Y. Trends in theoretical computer science. In Fundamentals of dependency theory. E. Borger Computer Science Press, Rockville, MD, 1987, pp. 171–224.
23. Yannakakis M. and Papadimitriou C. Algebraic dependencies. J Comput. Syst. Sci., 25(2):3–41, 1982.
24. Zaniolo C. Analysis and Design of Relational Schemata for Database Systems. PhD Thesis, University of California, Los Angeles, 1976. Technical Report UCLA-Eng-7669, Department of Computer Science.

Forever

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

Synonyms

Infinity; Positive infinity

Definition

The distinguished value *forever* is a special valid-time instant, namely the one following the largest granule in the valid-time domain. Forever is specific to valid time and has no transaction-time semantics.

Key Points

The value *forever* is often used as the valid end time for currently-valid facts. However, this practice is semanticallys incorrect, as such an end time implies that the facts are true during all future times. This usage occurs because database management systems do not offer means of storing the ever-changing current time, *now*, as an attribute value. To fix the incorrect semantics, the applications that manipulate and query the facts may interpret the semantics specially. However, the better solution is to extend the database management system to support the use of the variable *now* as an attribute value.

Concerning the synonyms, "infinity" and "positive infinity" both appear to be more straightforward, but have conflicting mathematical meanings. In addition, the time domain used for valid time may be finite, making the use of "infinity" inappropriate. Furthermore, the term positive infinity is longer and would imply the use of "negative infinity" for its opposite. Forever is intuitive and does not have conflicting meanings.

Cross-references

- ▶ Now in Temporal Databases
- ▶ Temporal Database
- ▶ Time Instant
- ▶ Until Changed
- ▶ 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, Berlin Heidelberg New York, 1998, pp. 367–405.
2. Torp K., Jensen C.S., and Snodgrass R.T. Effective timestamping in databases. VLDB J., 8(3–4):267–288, February 2000.

Form

Jessie Kennedy, Alan Cannon
Napier University, Edinburgh, UK

Synonyms

Forms-based interfaces

Definition

Paper-based forms are structured documents with empty slots that can be filled in by a user. In database systems, a form is an interface containing interaction

elements that allow a user to input, update or query data in a database. Forms may also contain static displays of data taken from the database. Documents containing only structured query results and no interaction elements are generally referred to as reports rather than forms. Database interfaces comprising of a series of on-screen forms are referred to as forms-based interfaces.

Commonly a single form corresponds with one table in relational databases and one form instance corresponds with one record. More complex forms may contain or update data from multiple tables. Forms are also used to interact with other databases such as object-oriented or XML databases.

Forms can be designed and created using various visual programming tools such as: Forms specification languages; GUI Builders; Automatic GUI Generation Systems; or by using third and fourth generation programming languages directly. Such tools may be included in the DBMS itself, in a related suite of proprietary programs or be independent applications.

Historical Background

With the advent of more general-purpose commercial database systems in the mid-1960s, the provision of forms-based interfaces for business applications became an issue for developers. Early applications were however still being custom built using various programming languages which was a very time intensive development effort. By the late 1970s/early 1980s a number of tools were becoming available to alleviate the load on the developer, for example, the Forms Application Development System (FADS) developed at U.C. Berkeley between 1979 and 1982. Later the ideas from FADS would develop into the widely used and more sophisticated Application-By-Forms (ABF) tool developed by Ingres Corporation between 1983 and 1986 [11]. Equally, the Oracle database system also introduced a tool for developing forms called Fast Forms (later called SQL*Forms) in 1984 [8]. Other examples include Control Center Forms module in dBASE IV, Forms EXPRESS in R:BASE, Paradox 3.0, Formanager, NFQL [3]. These forms were designed to be used with the alphanumerical terminals then in common usage and accordingly, the forms were character-based with some menus. These forms also tended to be relatively simple, conforming to the underlying database structure with one form usually representing one table and one record per screen.

As Graphical User Interfaces (GUIs) became available and popular for business applications in the late 1980s/1990s, forms took advantage of the new medium. Standard GUI techniques such as multiple windows, checkboxes and point and click selection were employed (e.g., Figure 1). Both DBMS and stand alone tools continued to provide support, with updated visual programming tools for GUI based forms (e.g., OracleForms, Ingres's W4GL, Microsoft Access's Form Tool/Wizard). The more powerful DBMS independent development tools had facilities for supporting skilled designers to build forms-based interfaces. While this sped development and empowered designers, it did not support inexperienced designers in avoiding bad designs and still required substantial developer resources. The trend of including forms generation as part of a DBMS or a related application suite that aimed to empower the end-user in customizing their own relatively simple forms has thus continued to date. DBMS provided tools, while increasing in sophistication, are however still restricted to either relatively simple applications or require an expert application developer. Attempts to reduce the reliance on developers by making greater use of automatic GUI creators (including the DBMS tools) have not been generally successful in generating complex forms-based interfaces.

In the last decade, the emphasis has shifted from a traditional client-server delivery method for forms, to a web-based one. Web forms are now prevalent across the internet (e.g., Figure 2). These web forms use HTML with various types of scripting or more recent alternatives such as XForms, XQForms [9].

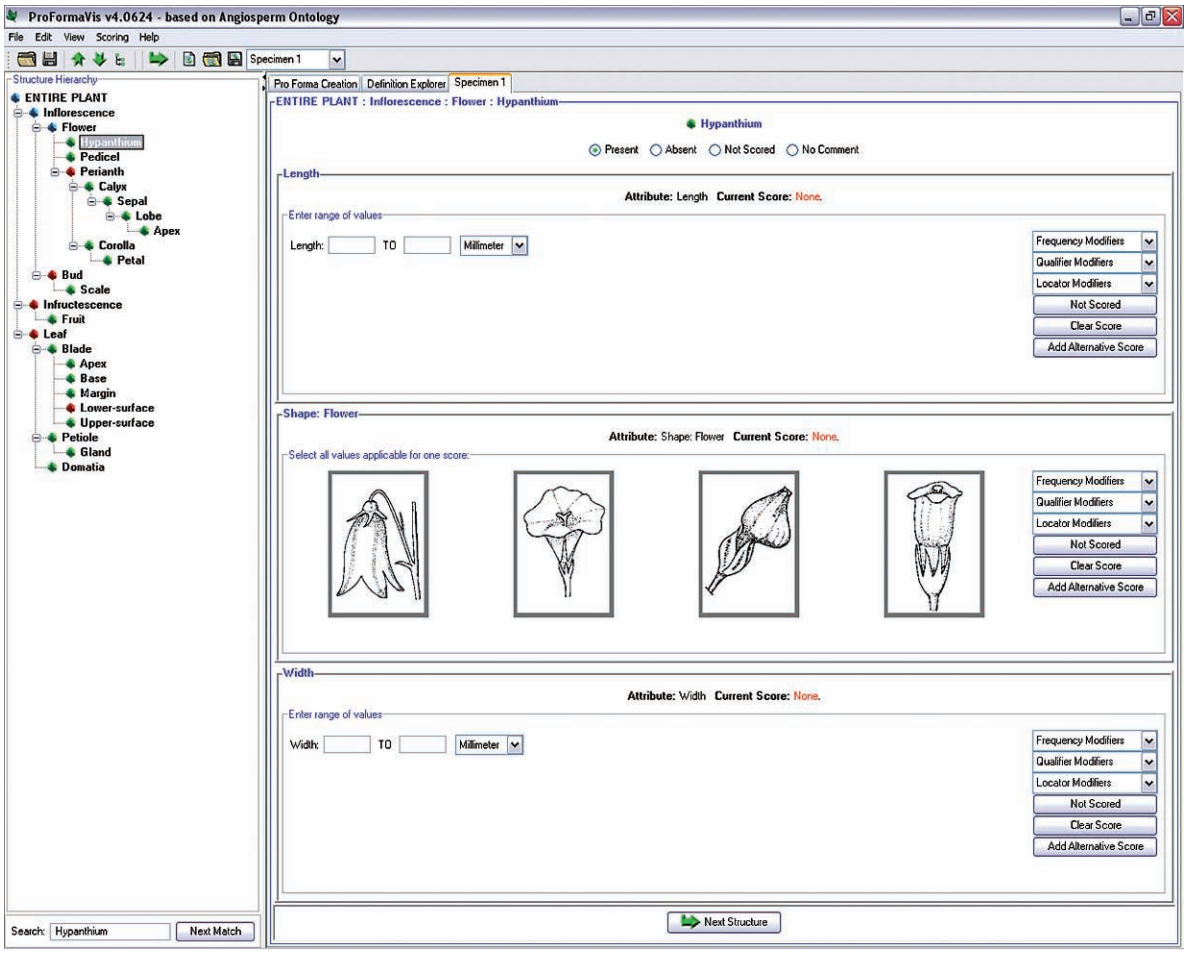
Foundations

Form Composition

In database systems, a form is an interface containing interaction elements that allow a user to input, update or query data in a database. Each form instance normally equates with a single database record, although exceptions exist and one database record may be spread over multiple forms.

Forms can be character-based, GUI-based (Fig. 1) or web-based (Fig. 2), but all basically comprise of four main elements that are tied together within logical containers such as a windows or frames:

1. Data entry interaction objects
2. Static labels

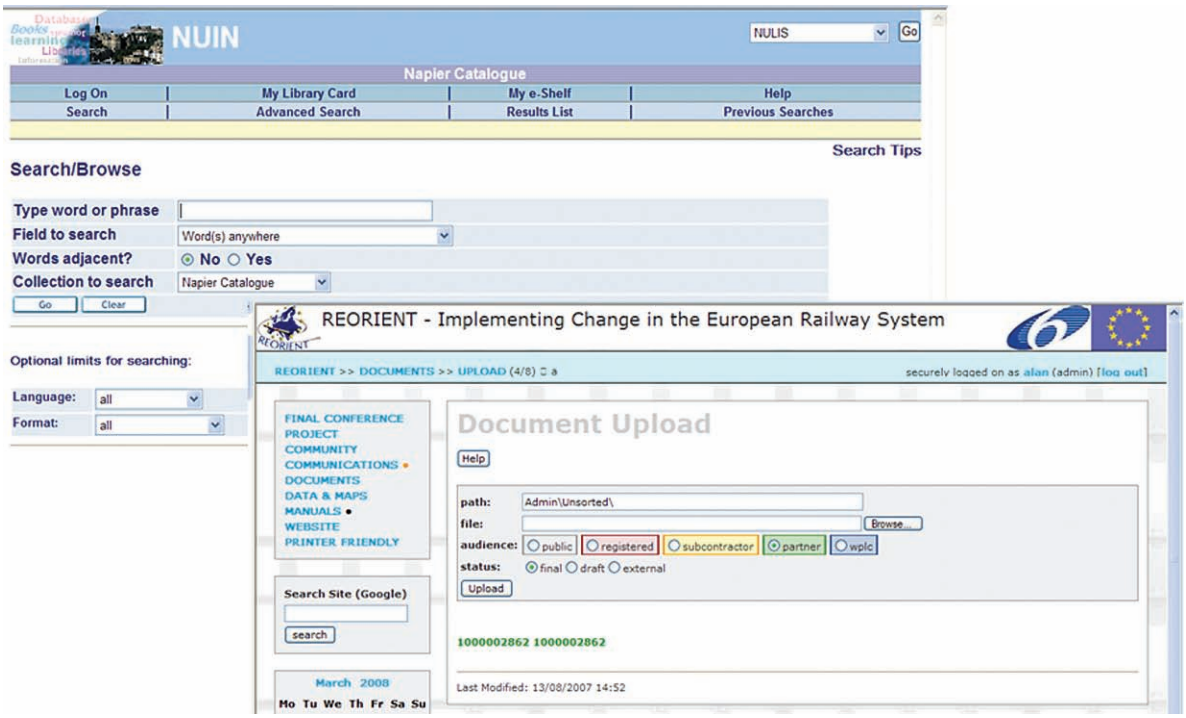


Form. Figure 1. Example of a GUI form for data entry [8].

3. Displayed data
4. Control elements

The data entry interaction objects are the slots into which data can be entered or altered by users. These elements are each bound to an associated attribute of a database table (or equivalent in non-relational databases). These comprise of such elements as text boxes, checkboxes, calendar objects, or any other of the interaction widgets that can be used to enter data. They may be initially blank or contain data, either taken from the existing record in the database (i.e., the current values of the bound attribute), from another section of the database (e.g., a range of possible values from a related table), from default attribute values specified in the database or from default values programmed into the form. The entered data can be used to fill a new record, update an existing one or query the database according to the underlying application logic. Each of the data entry interaction objects is usually associated with a static label specifying to the form user what data should be inserted. These are not usually bound to an attribute of the database. Other static data displayed in a form include existing data from the database of relevance to the form (e.g., other identifying data from the record being edited) or the results of user queries. This data may be a direct representation or it may be data manipulated by calculations programmed into the form. The last main elements of a form are control elements such as menus or buttons. These provide the form user with navigational and other controls over form operations. The most obvious examples of these are submit controls, that commit changes to the database.

Underlying business logic is programmed into a form to provide relevant database interaction, navigation controls, error checking and data validation. SQL or other query language code is normally embedded into



Form. Figure 2. Example of a two web-based forms for querying Napier University Library Catalog and for uploading a document to an on-line document repository.

form fields to support data updates or queries. A form can also contain hidden fields that are also sent to a database along with any other update or query (e.g., user id). This coding may be implemented directly by a developer using a third or fourth generation programming language or semi-automatically by development tools based on user interaction or totally automatically by forms generation tools directly from database schema.

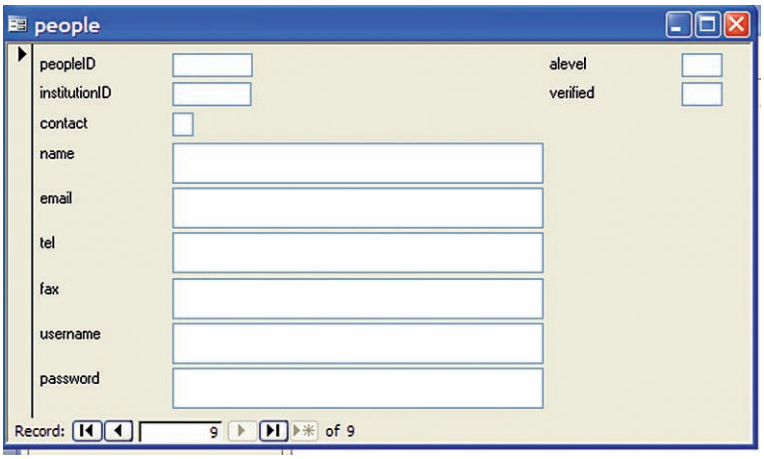
Generating Forms

Forms-based interfaces include sets of forms and the application logic that ties them together. Traditionally their generation has been a task for an expert application developer. To decrease that reliance, there have been attempts to allow end-users to develop their own forms-based interfaces. The main methods of generation are similar to any other user interface.

Traditional programming with languages that support database calls remain a common method for generation by application developers, particularly with fourth Generation languages and their associated Interface Development Environments. They permit powerful customized applications to be built.

Other visual development tools are also used both by application developers and more causal users. These can be stand-alone tools or linked to proprietary DBMS. The latter option is increasingly common, providing specific support for forms-based database interfaces. These all generally work on similar principles where the forms are designed visually with tools that allow fields, labels and other elements to be placed on the screen, using a WISYWIG system. The underlying application logic is then either selected with menus (often point and select) or by adding code in a database query language, programming or scripting language. Web based development for a forms-based interface is similar in nature (e.g., Macromedia's ColdFusion, Adobe's Dreamweaver).

Some DBMS visual development tools (e.g., MS Access, OracleForms, Paradox) have support for automatically generating forms from the database schema (Fig. 3), though the resulting interfaces are certainly at the simpler end of the forms and applications, being tied to a simple view of the database schema (such as in Figure 3), with only basic validation by data type for example being possible (for example by dragging the table attributes to XHTML form controls [10]).



Form. Figure 3. Example of a simple form created using Microsoft Access Form Wizard.

Other non-DBMS tools support automatic generation of the forms from various abstractions, where a presentation model controls the selection and layout of user interfaces, based on the modeled tasks and/or domain. These are known as model based approaches. The various approaches use different models, but usually focus on a task or domain/data model. Some approaches support completely automatic interface generation (e.g., MECANO, JANUS) and others provide support and guidance to designers to finalize an interface (e.g., MASTERMIND). Most of the approaches claim to connect to relational databases and one, Teallach, claims to connect to any object-orientated database [1]. These approaches have not, however, been widely adopted despite a strong research base [12], partly since they are generic systems and it is difficult to get good abstract domain and presentation models for specific applications. A related ontology based approach exists where an end-user edits a light-weight ontology to define the data model for a particular application. The system can then produce a forms-based data entry interface for that data model, based on a domain specific presentation model determined by the application developer [2].

Key Applications

Business Database Applications

The primary use for forms-based interfaces remains business applications, although as database technology continues to become more accessible, an increasing number of other casual users are utilizing databases and attendant forms-based interfaces for hobby and other leisure uses.

End-User Programming and Domain Specific Application Development

Empowering domain users to take over some of the application developer workload in creating forms for their own applications promises gains in development efficiency by reducing the need for a developer to learn about a domain and potentially in the applicability of the resulting forms. Improving the usability and utility of the various Integrated Development Environments for end-users is one way that database researchers can achieve this.

Ubiquitous Computing

A variety of new interface challenges must be met on different types of displays such as PDAs, mobile phones, and wall screens. It seems likely that the forms metaphor will continue to be used on these new devices and means of effectively tailoring forms to the requirements of the various displays and interaction methods must be found.

Data Quality

The phrase 'Garbage-In, Garbage-Out' remains as true today as at the dawn of computing. With forms being used constantly for data entry and updates, the control of entered data remains an important consideration. The design and data validation constraints placed on forms will continue to affect the quality of the data entered by users.

Meta-Data Capture

Forms-based interfaces are the norm for many systems such as data repositories and content management systems, which require to capture meta-data. The design and applicability of the forms used in these applications are vital to encourage users to enter meta-data, a task they are generally reluctant to undertake.

Knowledge Bases

Forms-based interfaces are also used by various ontological systems to populate knowledge bases (e.g., Protégé Frames). Existing ontology based approaches for data entry are, however, generally still limited to using automatically generated forms-based data entry interfaces unless manual editing is used. A form tends to be generated for each class instance with the choice of interaction widgets derived from the slot data types. Links between forms are based on class subsumption relationships.

Cross-references

- ▶ Database Management System
- ▶ Data Visualization
- ▶ DBMS Interface
- ▶ Direct Manipulation
- ▶ GUIs for Web Data Extraction
- ▶ Interface
- ▶ Multimodal Interfaces
- ▶ Ontology-Based Data Models
- ▶ Query language
- ▶ Table
- ▶ Views
- ▶ Visual Interfaces
- ▶ Web Services
- ▶ XML

Recommended Reading

1. Barclay P., Griffiths T., McKirdy J., Kennedy J., Cooper R., Paton N., and Gray P. Teallach – a flexible user-interface development environment for object database applications. J. Vis. Lang. Comput., 14(1):47–77, 2003.
2. Cannon A., Kennedy J., Paterson T., and Watson M. Ontology-driven automated generation of data entry interfaces. In Key Technologies for Data Management: 21st British National Conf. on Databases, 2004, pp. 150–164.
3. Embley D.W. NFQL: the natural forms query language. ACM Trans. Database Syst., 14(2):168–211, 1989.
4. Fry J.P and Sibley E.H. Evolution of data-base management systems. ACM Comput. Surv., 8(1):7–42, 1976.
5. Jagadish H.V., Chapman A., Elkiss A., Jayapandian M., Li Y., Nandi A., and Yu C. Making database systems usable. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 13–24.
6. Molina P.A. Review to model-based user interface development technology. In Proc. 1st Int. Workshop on Making Model-Based User Interface Design Practical: Usable and Open Methods and Tools. CEUR Workshop Proceedings 103 CEUR-WS.org, 2004.
7. Myers B., Hudson S.E., Pausch R. Past, present, and future of user interface software tools. ACM Trans. Comput. Hum. Interact., 7:(1)3–28, 2000.
8. Oracle Corporation. Oracle Forms. 2008. Available at: http://www.oracle.com/technology/products/forms/index.html
9. Petropoulos M., Vassalos V., and Papakonstantinou Y. XML query forms (XQForms): declarative specification of XML query interfaces. In Proc. 10th Int. World Wide Web Conference, 2001, pp. 642–651.
10. Raggett D., Le Hors A., and Jacobs I. XHTML HTML 4.01 Specification W3C Recommendation, 1999. Available at: URL: http://www.w3.org/TR/html4/
11. Rowe L.A. A retrospective on database application development frameworks. ACM SIGMOD Rec., 21(1):5–10, 1992.
12. Trætteberg H., Molina P.J., and Nunes N.J. Making model-based UI design practical: usable and open methods and tools. In Proc. 9th Int. Conf. on Intelligent User interface, 2004, pp. 376–377.

Forms-based Interfaces

- ▶ Form

Fourth Normal Form

Marcelo Arenas
Pontifical Catholic University of Chile, Santiago, Chile

Synonyms

4NF

Definition

Let $R(A_1,...,A_n)$ be a relation schema and Σ a set of unctional and multivalued dependencies over $R(A_1,...,A_n)$. Then (R, Σ) is said to be in Fourth Normal Form (4NF) if for every nontrivial multivalued dependency $X \twoheadrightarrow Y$ implied by Σ, it holds that X is a superkey for R.

Key Points

In order to avoid update anomalies in database schemas containing functional and multivalued

dependencies, 4NF was introduced by Fagin in [2]. As for the case of BCNF, this normal form is defined in terms of the notion of superkey as shown above. For example, given a relation schema $R(A, B, C)$ and a set of functional dependencies $\Sigma = \{A \rightarrow B\}$, it does not hold that $(R(A, B, C), \Sigma)$ is in 4NF since $A \twoheadrightarrow B$ is a nontrivial multivalued dependency implied by Σ and A is not a superkey for R. Similarly, relation schema $S(A, B, C)$ and set of multivalued dependencies $\Sigma = \{A \twoheadrightarrow C\}$ is not in 4NF since A is not a superkey for S. On the other hand, relation schema $T(A,B,C)$ and set of functional and multivalued dependencies $\Sigma = \{AB \rightarrow C, AC \rightarrow B, A \twoheadrightarrow B\}$ is in 4NF since AB, AC and A are all superkeys for T.

It should be noticed that relation schema $S(A, B, C)$ above is in BCNF if $\Sigma = \{A \twoheadrightarrow C\}$ since no nontrivial functional dependency can be inferred from Σ. In fact, 4NF is strictly stronger than BCNF; every schema in 4NF is in BCNF, but there exist schemas (as the one shown above) that are in BCNF but not in 4NF.

For every normal form two problems have to be addressed: How to decide whether a schema is in that normal form, and how to transform a schema into an equivalent one in that normal form. As for the case of BCNF, it can be tested efficiently whether a relation schema is in 4NF. A relation schema (R, Σ) is in 4NF if and only if for every nontrivial functional dependency $X \rightarrow Y \in \Sigma$, it holds that X is a superkey, and for every nontrivial multivalued dependency $X \twoheadrightarrow Y \in \Sigma$, it holds that X is a superkey. Thus, it is possible to check efficiently whether (R, Σ) is in 4NF by using a polynomial time algorithm for functional and multivalued dependency implication [1]. On the negative side, given a relation schema S, it is not always possible to find a database schema S' such that S' is in 4NF and S' is a lossless and dependency preserving decomposition of S.

Cross-references

- ▶ Boyce-Codd Normal Form
- ▶ Normal Forms and Normalization
- ▶ Second Normal Form (2NF)
- ▶ Third Normal Form

Recommended Reading

1. Beeri C. On the membership problem for functional and multivalued dependencies in relational databases. ACM Trans. Database Syst., 5(3):241–259, 1980.
2. Fagin R. Multivalued dependencies and a new normal form for relational databases. ACM Trans. Database Syst., 2(3):262 –278, 1977.

FQL

Peter M.D. Gray
University of Aberdeen, Aberdeen, UK

Definition

At about the same time that Shipman was developing DAPLEX, Buneman and Frankel proposed the highly influential FQL functional query language [1], based on Backus's FP functional programming paradigm. A major motivation for this work was that it is in principle possible to combine arbitrary *computable functions* with *stored database functions* into a functional query language which is not limited to a predefined set of operators.

FQL was intended for implementation over existing DBMSs and an abstract implementation based on the lazy evaluation of stream data was described. In later publications [2,3], FQL was extended with features from the functional programming language ML. In this version, function definition is simpler than in the variable-free FP-like syntax and new higher-order functions can be defined in addition to the built-in ones. Also a set of built-in metalevel functions is provided, as in EFDM. Thus, object-level querying and meta level querying can be carried out in the same functional syntax.

Key Points

FP includes a built-in set of *functionals*, like higher-order functions, including function composition, $\circ$, and sequence construction. In addition to these, FQL provides the functionals !, * and ^.

Given an abstract entity type A, !A is a 0-argument function which returns a *stream* of all entities of type A. For example, given the abstract entity type COURSE, !COURSE returns a stream of all the entities of type COURSE.

Given any type A, *A is the type consisting of streams of entities of type A. For example, given the abstract entity types STUDENT and COURSE, the function `ATTENDS : STUDENT -> *COURSE` takes a STUDENT and returns a stream of the COURSEs attended by the STUDENT.

Given a function f: A → B, *f is the function of type *A → *B which takes as an argument a stream of entities of type A, a_1, a_2,... say, and returns the stream $f(a_1)$, $f(a_2)$,.... For example, given the function `CNAME : COURSE -> STRING` the query `COURSE o *CNAME` returns the name of every COURSE.

Given a function f: A → B or f: A → *B, the function ^f: B → *A is the inverse (or converse) of f. In a later version of FQL [2], the functionals ! and ^ are dropped.

Cross-references

► Functional Query Language

Recommended Reading

1. Buneman P. and Frankel R.E. FQL – A Functional Query Language. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 52–58.
2. Nikhil R. An incremental, strongly-typed database language. Technical Report PhD Thesis, University of Pennsylvania, 1984.
3. Nikhil R. Practical polymorphism. In Proceedings of Functional Programming and Computer Architecture'85, LNCS 201. 1985, pp. 319–333.

Fractal

Ke Deng
University of Queensland, Brisbane, QLD, Australia

Synonyms

Koch snowflake; Space-filling curve

Definition

A fractal is "a rough or fragmented geometric shape that can be subdivided in parts, each of which is (at least approximately) a reduced-size copy of the whole" [2]. This term is introduced by French mathematician Benoit Mandelbrot (born 1924) in 1975 and was derived from the Latin fractus meaning "broken" or "fractured". A simple fractal example is Koch snowflake. The method of creating this shape is to recursively replace each line segment with 4 line segments in a finer scale as shown in Fig. 1.

Key Points

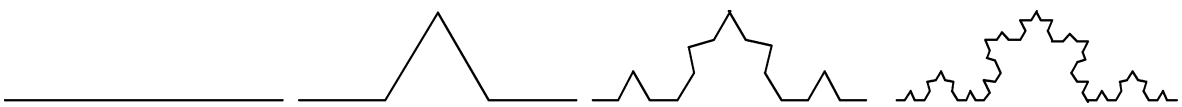
Fractals are self-similar structures (at least approximately or stochastically) which are appropriate for representing the geometry in nature. Some examples include clouds, snow flakes, coastlines, crystals, cauliflower or broccoli, and mountain ranges. In fractal geometry, the fractal dimension, D, is a statistical quantity that gives an indication of how completely a fractal appears to fill space, as one zooms down to finer and finer scales. Among several choices, Hausdorff dimension (also known as the Hausdorff-Besicovitch dimension) is a popular definition of fractal dimension and it is:

$$D = \lim_{\varepsilon \to 0} \frac{\log N(\varepsilon)}{\log(1/\varepsilon)}.$$

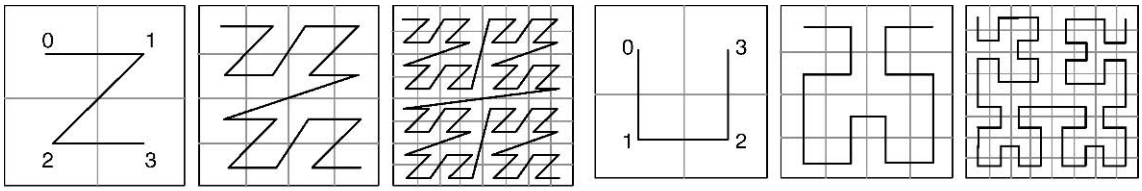
Where $N(\varepsilon)$ is the number of balls of radius at most ε to cover the fractal. The Hausdorff dimension of Koch snowflake is:

$$D = \frac{\log 4^n}{\log 3^n} = \frac{\log 4}{\log 3} = 1.2618...$$

Another case of fractals is the space-filling curves which is first described by Giuseppe Peano (1858–1932) and has Hausdroff dimension $D = 2$. Space-filling curve is useful in computer science in two general situations, managing multi-dimensional points and storing k-dimensional array on the disk [1]. This attributes to two features. (i) Space-filling curve is a continuous curve capable of mapping from d-dimensional space to 1-dimensional space. Using this mapping, a point in the cube can be described by its spatial coordinates, or by the length along the curve, measured from one its ends. Any one dimensional data structure can be used such as binary search trees, B-trees or Hash tables. (ii) Space-filling curve has good locality preserving behaviour such that points



Fractal. Figure 1. Recursively replace each line segment with 4 line segments in a finer scale in Koch snowflake.



Fractal. Figure 2. Two space filling curves.

are close together in the 1-dimensional space are mapped from points that are close together in the d-dimensional space. The reverse property is not true, i.e., not all adjacent cells in the d-dimensional space are adjacent or even close on the curve. A group of contiguous cells in d-dimensional space will typically be mapped to a collection of segments on the space-filling curve. There segments are called clusters. Z-order curve and Hilbert curve are space-filling curves as shown in Fig. 2a,b.

Cross-references

► Space-Filling Curve

Recommended Reading

1. Faloutsos C., and Roseman S., Fractals for secondary key retrieval. In Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1989, pp. 247–252.
2. Mandelbrot B. B., Fractal Geometry of Nature, W. H. Freeman, San Francisco, CA,1977.

Frequency Moments

David Woodruff
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

L_p norms; L_p distances

Definition

Consider a stream (i.e., an ordered list) $\mathcal{S} = a_1, a_2, \ldots, a_n$ of elements $a_i \in [m] \stackrel{\text{def}}{=} \{1,2,\ldots,m\}$. For $i \in [m]$, its frequency f_i is the number of times it occurs in $\mathcal{S}$. The k th frequency moment F_k of $\mathcal{S}$, for real $k > 0$, is defined to be $F_k(\mathcal{S}) = \sum_{i \in [m]} f_i^k$. Interpreting 0^0 as 0, one can also define F_0 this way, so that it equals the number of distinct elements in $\mathcal{S}$. Observe that $F_1 = n$ is the length of $\mathcal{S}$. In the database community, F_2 is known as the repeat rate or Gini's index of homogeneity. It is also natural to define $F_\infty = max_{1 \le i \le m} f_i$.

It is usually assumed that n is very large and that algorithms which compute the frequency moments do not have enough storage to keep the entire stream in memory. It is also common to assume that they are only given a constant (usually one) number of passes over the data. It is further assumed that the stream is presented in an arbitrary, possibly worst-case order. This necessitates the use of extremely efficient randomized approximation algorithms. An algorithm A (ϵ, δ)-approximates the k th frequency moment F_k if for any input stream $\mathcal{S}$, $\Pr[|A(\mathcal{S}) - F_k(\mathcal{S})| \le \epsilon F_k(\mathcal{S})] \ge 1 - \delta$, where the probability is over the coin tosses of A. Here, $A(\mathcal{S})$ means that A is presented items in $\mathcal{S}$ one-by-one. Efficiency is measured in terms of the amount of memory and update time of the algorithm.

Key Points

The frequency moments were introduced by Alon, Matias, and Szegedy [1] in their seminal paper in 1996, and are important statistics for massive databases. Indeed, efficient algorithms for estimating F_0 can be used by query optimizers for finding the number of unique values of an attribute without having to perform an expensive sort on the entire column. Efficient algorithms for F_2 are useful for determining the output size of self-joins and for error estimation in the context of query result sizes and access plan costs. Moreover, F_k for $k \ge 2$ can indicate the amount of skew of a data stream, and this can determine which algorithms to use for data partitioning. These values can also be used to detect denial-of-service attacks. In general, F_k for large k can be used to approximate the

most frequent value, potentially more efficiently than computing this value directly.

There is a large body of work on bounding the memory required of F_k-approximation algorithms. In their original work, Alon, Matias, and Szegedy surprisingly showed that F_2 can be (ϵ, δ)-approximated using only $O(\frac{\lg 1/\delta}{\epsilon^2}(\lg n + \lg m))$ bits of memory. It is now known that F_k for $k \leq 2$ can be (ϵ, δ)-approximated in 1-pass using $O(\frac{\lg 1/\delta}{\epsilon^2})$ bits of space, up to a *polylog nm* factor, and there is an almost matching $\Omega(1/\epsilon^2)$ lower bound for 1-pass algorithms. For $k > 2$, a sequence of work showed that F_k can be approximated in $O(m^{1-2/k}log1/\delta)$ space, up to a *poly* $(lognm,1/\epsilon)$ factor, and there is an almost matching $\Omega(m^{1-2/k})$ bound. The memory required for approximating F_∞ is $\Theta(m)$. Note that for $k > 2$ the memory required depends polynomially on m, whereas for $k \leq 2$ the dependence is logarithmic. The known algorithms use a clever combination of hashing (with limited independence), sketching, and bucketing ideas. The corresponding lower bounds come from reductions from problems in communication complexity, and draw from tools in extremal combinatorics and information theory.

There are several natural questions about the computational complexity of frequency moments which remain unanswered. For instance, for constant δ, it is unknown if F_k for $0 \leq k \leq 2$ can be approximated more efficiently if more than one pass (but still a constant number) is allowed. Also, it is unknown how efficiently F_k can be approximated if the stream elements arrive in a random order.

Cross-references

- ▶ Data Mining
- ▶ Data Stream
- ▶ Stream Mining
- ▶ Stream Processing
- ▶ Stream Sampling
- ▶ Streaming Applications

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments, J. Comput. Syst. Sci., 58:137–147, 1999.
2. Bar-Yossef Z., Jayram T., Kumar R., and Sivakumar D. An information statistics approach to data stream and communication complexity. In Proc. 43rd Annual Symp. on Foundations of Computer Science, 2002, pp. 209–218.
3. Indyk P. and Woodruff D. Optimal approximations of the frequency moments of data streams. In Proc. 37th Annual ACM Symp. on Theory of Computing, 2005, pp. 202–208.

Frequent Concepts

▶ Closed Itemset Mining and Nonredundant Association Rule Mining

Frequent Elements

▶ Frequent Items on Streams

Frequent Graph Patterns

Jun Huan
University of Kansas, Lawrence, KS, USA

Synonyms

Graph database mining

Definition

There are three key concepts in mining graph databases: (i) labeled graph, (ii) subgraph isomorphism, and (iii) graph support value. Based on the concepts, the problem of frequent subgraph mining could be defined in the following discussion.

Definition 1. *A **labeled graph** G is a quadruple $G = (V, E, \Sigma, \lambda)$ where V is a set of vertices or nodes and $E \subseteq V \times V$ is a set of undirected edges. Σ is a set of (disjoint) vertex and edge labels, and $\lambda: V \cup E \rightarrow \Sigma$ is a function that assigns labels to vertices and edges. Typically a total ordering is defined on the labels in Σ.*

With the previous definition, a *graph database* is a set of labeled graphs.

Definition 2. *A graph $G' = (V', E', \Sigma', \lambda')$ is **subgraph isomorphic** to $G = (V, E, \Sigma, \lambda)$, denoted by $G' \subseteq G$, if there exists a 1-1 mapping $f: V' \rightarrow V$ such that*

- $\forall v \in V', \lambda'(v) = \lambda(f(v))$
- $\forall (u,v) \in E', (f(u),f(v)) \in E$, *and*

- $\forall(u,v) \in E', \lambda'(u,v) = \lambda(f(u), f(v))$.

The function f is a *subgraph isomorphism* from graph G' to graph G. For simplicity, G' *occurs* in G if $G' \subseteq G$. Given a subgraph isomorphism f, the image of the domain V' ($f(V')$) is an embedding of G' in G.

Example 1. *Figure 1 shows a graph database of three labeled graphs. The mapping (isomorphism)* $q_1 \rightarrow p_3$, $q_2 \rightarrow p_1$, and $q_3 \rightarrow p_2$ *demonstrates that graph Q is isomorphic to P and so Q occurs in P. Set* $\{p_1, p_2, p_3\}$ *is an embedding of Q in P. Similarly, graph S occurs in graph P but not Q.*

Definition 3. *Given a graph database* $\mathcal{G}$, *the* support *of a graph C, denoted by sup*(C), *is the fraction of graphs in* $\mathcal{G}$ *in which C occurs.*

Problem Statement: Given a graph database $\mathcal{G}$ and a support threshold $0 < \sigma \leq 1$, the *frequent subgraph mining* problem is to identify all graphs whose support value is at least σ.

There are several possible extensions to the problem of frequent subgraph mining. For example, the node may contain multiple label rather than one [2] and the identified patterns may contain node with or without labels [6]. Special cases for frequent subgraph mining include frequent tree mining, frequent path mining, and frequent cycle mining from graphs. The current introduction emphasizes the fundamental problem frequent subgraph mining and covers little about the mentioned extensions.

Historical Background

Graph database mining is an active research field. Recent graph mining algorithms can be roughly divided into three categories. The first category uses a level-wise search strategy, including AGM [6] and FSG [7]. The second category takes a depth-first search strategy, including gSpan [11] and FFSM [4]. The third category works by first mining frequent trees and then construct cyclic graphs based on tree pattern. This category includes SPIN [5] and GASTON [8]. See [1] for a recent survey.

The following description of frequent subgraph mining is mainly based on the algorithms FFSM and SPIN. Both algorithms are built on top of a data structure termed CAM tree (graph Canonical Adjacency Matrix tree), which is a compact representation of a space of graphs. Using CAM tree, FFSM (Fast Frequent Subgraph Mining, [4] supports an incremental subgraph isomorphism check and hence is scalable for large graphs. SPIN (SPanning tree based mINing, [5]) reduces the total number of mined patterns by identifying only the *maximal* ones, i.e., the set of frequent subgraphs such that none of their supergraphs is frequent.

Foundations

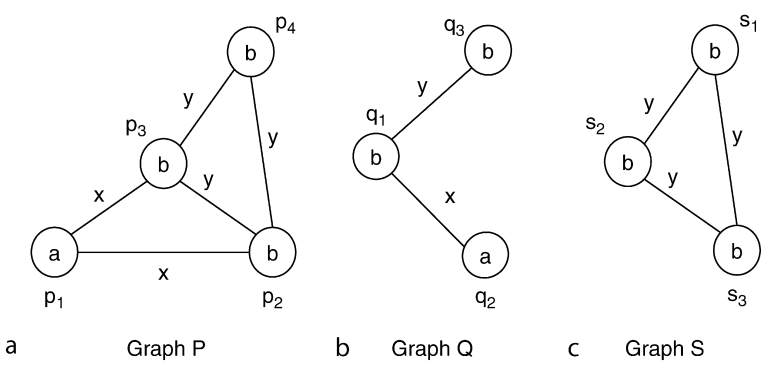
Three topics are important for designing efficient subgraph mining algorithms:

1. Graph Canonical Form.
2. Candidate Graph Proposing.
3. Candidate Support Value Computation.

In the following, the three topics are discussed in details.

Canonical Adjacency Matrix

For labeled graph, a natural way to present a graph is using an modified adjacency matrix. In this representation, every diagonal entry of the modified adjacency matrix is filled with the label of the corresponding node and every off-diagonal entry is filled with the label of the corresponding edge, or zero if there is no



Frequent Graph Patterns. Figure 1. A database of three labeled graphs.

edge. This is slightly different from the widely used adjacency matrix representation for unlabeled graphs.

One of the critical problems in graph mining is the graph isomorphic problem: given two graphs P and Q, determine whether P is isomorphic to Q. This problem may be solved following a common theme such that a graph is first transformed to a unique presentation (referred to as the *canonical form* of the graph) and secondly, two graphs are compared using the transformed presentations. A graph canonical form is specially designed such that if two graphs are isomorphic to each other, their canonical forms are the same and vice versa.

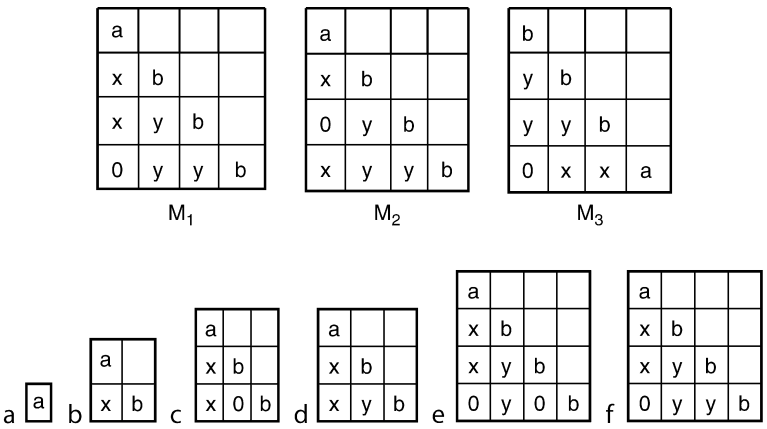
Following convention, in the following a capital letter is used to denote an adjacency matrix and the corresponding lower case letter (augmented with subscripts) is used to denote an individual entry of the adjacency matrix. For instance, $m_{i,j}$ denotes the entry on the ith row and jth column of an $n \times n$ adjacency matrix M, $0 < j \leq i \leq n$. Note that the adjacency matrix is not unique for a given graph: a permutation of the set of vertices in a graph may generate a different adjacency matrix. There is a total of $n!$ different permutations for a graph vertex set with size n, there may be up to $n!$ adjacency matrices for the graph. Figure 2 (top) shows three adjacency matrices for the labeled graph P in Fig. 1 (Only the lower triangular part of an adjacency matrix is drawn since the upper half is a mirror of the lower half.). In order to enable a unique representation for each graph, the code of adjacency matrices is defined below. Graph code helps in defining a total order among all adjacency matrices.

Definition 4. *Given an $n \times n$ adjacency matrix M of a graph G with n vertices, the* **code** *of M, denoted by code (M), is the sequence formed by concatenating lower triangular entries of M (including diagonal entries) in the order: $m_{1,1}m_{2,1}m_{2,2}...m_{n,1}m_{n,2}...m_{n,n-1}m_{n,n}$ $(0 < j \leq i \leq n)$.*

For an adjacency matrix M, each diagonal entry of M is a *node entry* and each off-diagonal none-zero entry in the lower triangular part of M is an *edge entry*. Edge entries are ordered according to their relative positions in the code of the matrix. For example the *first* edge entry of M is $m_{2,1}$ and the *last* edge entry is the one appears rightmost in *code*(M).

Example 2. *Figure 2 shows codes and edge entries for the labeled graph P showing in Fig. 1. For example for adjacency matrix M_1, the edge entry set is $\{m_{2,1}, m_{3,1}, m_{3,2}, m_{4,2}, m_{4,3}\}$ where $m_{2,1}$, $m_{4,3}$, and $m_{4,2}$ are the first, last, second-to-last edge entries of M, respectively. After applying the total ordering, the relationships are: code (M_1) = "axbxyb0yyb" $\geq$ code(M_2) = "axb0ybxyyb" $\geq$ code(M_3) = "bybyyb0xxa." Submatrices of CAM are shown at the bottom. Matrix 3.a is the proper maximal submatrix of matrix 3.b, which itself is the proper maximal submatrix of 3.c and so on so forth.*

Standard lexicographic order on sequences is usually used to define a total order of two arbitrary codes p and q. Given a graph G, its *canonical form*, denoted by $\varphi(G)$, is the maximal code among all its possible codes. The adjacency matrix M that produces the canonical form is the G's *canonical adjacency matrix* (CAM). For example, the adjacency matrix M_1 shown in Fig. 2 is the CAM of the graph P in Fig. 1, and *code*(M_1) is the canonical form of the graph.



Frequent Graph Patterns. Figure 2. Top: three adjacency matrices for the graph P in Fig. 1. Bottom: examples of maximal proper submatrices.

CAM Tree

Efficient frequent subgraph mining algorithm relies on efficient data structure. Below a widely data structure called CAM tree is discussed. CAM tree utilizes a key property of the previous canonical form, i.e., a "prefix" of the canonical form is also maximal, which is stated in the following theorem.

Definition 5. *Given an $n \times n$ matrix N and a $m \times m$ matrix M, let $m_{l,k}$ be the last edge entry of M, a matrix N is the* ***maximal proper submatrix*** *of M if N is obtained by removing $m_{l,k}$ from M*

Several examples of the maximal proper submatrices are given at the bottom of Fig. 2.

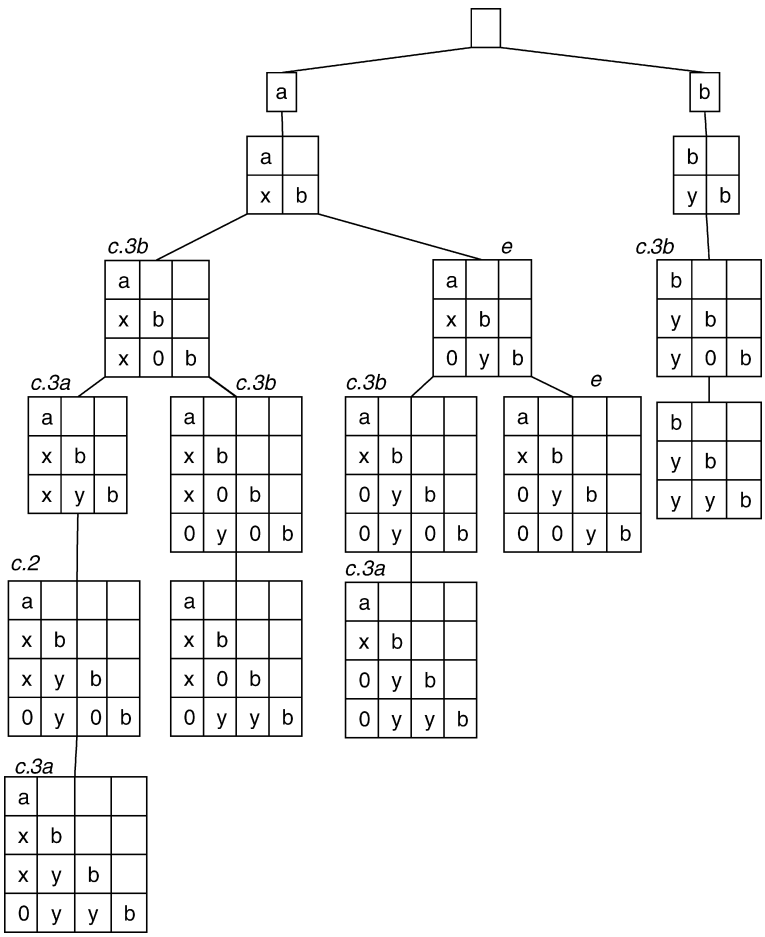
Corollary 1 *Given a CAM M of a connected graph G and a submatrix N of M, N represents a connected subgraph of G.*

Proof 1 *Since N must represent a subgraph of G, it is sufficient to show that the subgraph N represents is connected. To prove this, it is sufficient to show that in N there is no such row i (with the exception of the first row) that i doesn't contains any edge entry. The rest of the proof is trivial and could be found in* [6].

Corollary 2 *Given a connected graph G with CAM M, a submatrix N of M, and a graph H which N represents, N is the CAM of H.*

Proof 2 *For simplicity, M is assumed to have only one edge entry in the last row. It is trivial to show that code (N) is a prefix of code(M). This suggests that code$(N) \geq$ code$(CAM(H))$, where $CAM(H)$ stands for the canonical adjacency matrix of graph H. Therefore N must be the CAM of the graph H.*

For a CAM with at least two edge entries in the last row, a similar proof could be used.



Frequent Graph Patterns. Figure 3. The CAM Tree of the graph P in Fig. 1. Every matrix obtained by a join operation is specified by a label starting with c. and then the type of the join operation e.g., c. 3a stands for join case3a. A CAM obtained by an extension operation is labeled with e.

Usually an empty matrix is considered as the maximal proper submatrix of any matrix with size 1. With this assumption, all the CAMs of connected subgraphs of a graph G may be organized as a rooted tree as follows:

- The root of the tree is an empty matrix;
- Each node in the tree is a distinct connected subgraph of G, represented by its CAM;
- For a given none-root node (with CAM M), its parent is the graph represented by M's maximal proper submatrix;

The tree obtained in this fashion is denoted as the *CAM tree* of the graph G. Figure 3 shows the CAM tree of the graph P from Fig. 1.

Frequent Subgraph Mining of a Graph Database

With the CAM tree data structure, the space of subgraphs could be organized in a single CAM tree. If such a tree is built in advance (regardless of the required space and computational capacity), any traversal of the tree (depth-first, level-wise, random) reveals the set of distinct subgraphs of the graph database. For each of such subgraph, its support value may be determined by a linear scan of the graph database and finally frequent ones can be reported. This method clearly suffers from the huge number of available subgraphs in a graph database and hence has poor scalability to large databases.

In the following pseudo code, an algorithm which takes advantage of the following simple fact is presented: if a subgraph G is infrequent (support of G is less than a user posted threshold), none of its supergraphs are frequent. This suggests that an algorithm can stop building a branch of the tree as early as its finds out that the current node has insufficient support.

In the pseudo code below, symbol $CAM(G)$ denotes the CAM of the graph G. *is CAM* is a function computes whether the matrix X is the CAM of the graph it represents or not.

FFSM
input: a graph database GD and a support threshold $f\,(0 < f \leq 1)$
output: set S of all G's connected subgraphs.

```
1: C ← {the CAMs of the frequent edges}
2: F ← {the CAMs of the frequent nodes and edges}
3: FFSM-Explore (C,F);
```

FFSM-Explore
input: C, a suboptimal CAM list and F, a set of frequent connected subgraphs' CAMs
output: set F contains CAMs of all frequent subgraphs searched so far.

```
1: for X ∈ C do
2:   if (isCAM(X)) then
3:     F ← F ∪ {X}
4:     C′ ← ∅
5:     for Y ∈ C do
6:         C′← C′∪ FFSM-Join(X,Y)
7:     end for
8:     C′← C′ ∪ FFSM-Extension (X)
9:     remove CAM(s) from C′ that is either
       infrequent or not suboptimial
10:    FFSM-Explore(C′,F)
11:  end if
12: end for
```

Key Applications

Frequent subgraph mining has many applications. Below two applications in the emergent area of bioinformatics and cheminformatics are reviewed. Other applications of frequent subgraph mining could be found in recent reviews.

Pattern Discovery from Chemical Structures

Several investigators have applied the idea of frequent subgraph mining in cheminformatics. In these applications, a graph is used to model a chemical structure where a node represents an atom and an edge represents a chemical bond in the chemical structure. Frequent subgraph mining is then utilized to discover task-relevant and problem-specific features (*descriptors* as usually called in cheminformatics literature) in the chemical structures. Finally the features are utilized to build a predictive model to map chemical structures to their target properties [9].

Pattern Discovery from Protein Structures

Huan et al. have applied data mining algorithms to analyze 3D structures of biomoelcules including proteins and chemicals. Their approach adopted geometric graph representations of protein 3D structure using a geometric technique called Delaunay Tessellation and its recent extension to almost-Delaunay. These techniques produce sparser but information-preserving graph representations than conventional distance-based methods. With

the graph database mining techniques, they have identified structure patterns that occur frequently in a family of proteins but rarely in other families, or *protein family-specific fingerprints*. It has been demonstrated that patterns obtained from comparing multiple structures have clear linkages to known functional features such as the catalytic sites in enzymes [3].

Cross-references

- ▶ Data Mining
- ▶ Graph Database
- ▶ Semi-Structured Data

Recommended Reading

1. Han J., Cheng H., Xin D., and Yan X. Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov., 14, 2007.
2. Huan J., Prins J., Wang W., Carter C., and Dokholyan N.V. Coordinated evolution of protein sequences and structures with structure entropy. Tech. Rep. Computer Science Department, 2006.
3. Huan J., Wang W., Bandyopadhyay D., Snoeyink J., Prins J., and Tropsha A. Mining protein family specific residue packing patterns from protein structure graphs. In Proc. 8th Annual Int. Conf. on Research in Computational Molecular Biology, 2004, pp. 308–315.
4. Huan J., Wang W., and Prins J. Efficient mining of frequent subgraph in the presence of isomorphism. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 549–552.
5. Huan J., Wang W., Prins J., and Yang J. SPIN: mining maximal frequent subgraphs from graph databases. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 581–586.
6. Inokuchi A., Washio T., and Motoda H. An apriori-based algorithm for mining frequent substructures from graph data. In Principles of Data Mining and Knowledge Discovery, 4th European Conf., 2000, pp. 13–23.
7. Kuramochi M. and Karypis G. Frequent subgraph discovery. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 313–320.
8. Nijssen S. and Kok J. A quickstart in frequent structure mining can make a difference. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 647–652.
9. Smalter A., Huan J., and Lushington G. Structure-based pattern mining for chemical compound classification. In Proc. 6th Asia Pacific Bioinformatics Conf, 2008, pp. 39–48.
10. Vanetik N. and Gudes E. Mining frequent labeled and partially labeled graph patterns. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 91–102.
11. Yan X. and Han J. gSpan: graph-based substructure pattern mining. In Proc. 2002 IEEE Int. Conf. on Data Mining, 2002, pp. 721–724.

Frequent Items on Streams

Ahmed Metwally
Google Inc., Mountain View, CA, USA

Synonyms

Frequent elements; Heavy hitters; Hot items

Definition

Frequent items are the items that mostly represent the stream, since these are the items that occur more than a given user threshold. Formally, given a stream, S, of size N from an alphabet, A, a *frequent item*, $E_i \in A$, is an item whose *frequency*, or number of occurrences, F_i exceeds a specific user support ϕN, where $0 \leq \phi \leq 1$. There cannot be more than $\lfloor \frac{1}{\phi} \rfloor - 1$ such items. Finding the frequent items exactly in one pass requires $O(min(A,N))$ in-memory space [6]. Frequent items can be defined on the entire stream or on a sliding window of fixed or variable size (see Stream Models). Similarly, frequent items can be defined on append-only streams as well as streams with item deletions (see Stream Mining).

Historical Background

Even before the stream processing model was proposed, the early work in [3], and [9] searches for a majority item that occur more than $\frac{N}{2}$ times. This work was extended in [15] to search for $\lfloor \frac{1}{\phi} \rfloor - 1$ items, each occurring more than ϕN times.

Foundations

Due to the high cost of the exact solution of the frequent items problem, researchers proposed approximate problem definitions. This section discusses the problem approximations and their proposed algorithms.

Problem Approximations

The approximate problem definition in [3,9,15] is to report $\lfloor \frac{1}{\phi} \rfloor - 1$ items regardless of the number of items that really satisfy the frequency constraint. However, all the really frequent items should be among the reported set of items. This introduces *false positives*, which are reported items whose frequencies do not really exceed ϕN.

The *hot items* problem [6], is to report the frequent items with high probability. This raises the issue of

having both false positives and *false negatives*, where a false negative is a frequent item that is not reported.

Finding the ε-*deficient* frequent items was proposed by Manku and Motwani [13] to report all the items with frequencies exceeding $\lceil \phi N \rceil$. In addition, no reported item can have a frequency less than $\lceil (\phi - \varepsilon)N \rceil$, where $\varepsilon \in [0, \phi]$ is a user-defined error. The estimated frequencies of the reported items also have to be reported.

Proposed Algorithms

Several algorithms [6,7,8,10,11,13,14] were proposed to solve the approximate frequent items problems. In [14], these algorithms are classified into *Counter-based* and *Sketch-based* algorithms.

Counter-Based Techniques

Counter-based algorithms monitor the frequency of a subset of A. They keep a counter for each monitored item. The counter of a monitored item, E_i, is updated whenever E_i is observed in the stream. The algorithms mainly differ in how they handle non-monitored items.

The *Sticky Sampling* algorithm [13] solves the ε-deficient frequent items probabilistically. It slices S into rounds of exponentially increasing length. The probability an item is sampled, i.e., starts to be monitored, at any round, r, is $\frac{1}{r}$. At rounds' boundaries, for every monitored item, a coin is tossed until a success occurs. The counter is decremented for every unsuccessful toss, and is deleted if it reaches 0. Since decrementing counters follow a geometric distribution, the probability of sampling an item is constant throughout S. *Sticky Sampling* has a space bound of $O(\frac{2}{\epsilon}\ln(\phi^{-1}\delta^{-1}))$, where δ is the failure probability.

The *Lossy Counting* algorithm [13] solves the ε-deficient frequent items with success probability of 1, though with a space bound greater than that of *Sticky Sampling*. S is broken up into rounds of equal length, $\frac{1}{\epsilon}$. Throughout every round r, non-monitored items are added to the monitored set. When a new item is added in round r, it is given the benefit of doubt. Its initial count is set to $r - 1$, and the maximum possible overestimation, $r - 1$, is recorded for the new item. Thus, *Lossy Counting* guarantees that monitored items can have their frequencies overestimated by no more than εN, and never underestimated. At the end of each ound, r, every item, E_i, whose estimated frequency s less than r is evicted in order to reduce the space footprint. *Lossy Counting* [13] has a space bound of $O(\frac{1}{\epsilon}\ln(\epsilon N))$.

The *Frequent* algorithm [7] solves the problem of [15], which asks for a maximum of $\lfloor \frac{1}{\phi} \rfloor - 1$ items, each of which has frequency exceeding ϕN. *Frequent*, a rediscovery of the algorithm proposed by [15], outputs a list of exactly $\frac{1}{\phi} - 1$ items with no guarantee on which items, if any, have frequency more than ϕN. This algorithm extends the early work done in [3,9] for finding a majority item using only one counter. The algorithm in [3,9] monitors the first item in the stream. For each observation, the counter is incremented if the observed item is the monitored one, and is decremented otherwise. If the counter reaches 0, it is assigned the next observed item, and the algorithm is then repeated. When the algorithm terminates, the monitored item is the candidate majority item. *Frequent* [7] keeps $\frac{1}{\phi} - 1$ counters to monitor items. If a monitored item is observed, its counter is incremented, else all counters are decremented in an $O(1)$ operation, using a lightweight data structure. In case any counter reaches 0, it is assigned the next observed item. The discussion assumes hashing takes constant time. If the data structure is not used, decrementing all counters has an $O(1)$ amortized cost. The algorithm uses only $\frac{1}{\phi} - 1$ counters, but reports back all the monitored items, which could yield a high ratio of false positives. The same algorithm and data structure were proposed independently in [11].

The *Space-Saving* algorithm [14] solves the ε-deficient frequent items with success probability of 1 with a space bound of $\frac{1}{\epsilon}$, and gives the same guarantees on overestimation error of *Lossy Counting*. Moreover, the guaranteed error rate, ε, provably decreases with the increase in data skew. If the observed item, x, is monitored, its counter is incremented. If x is not monitored, i.e., no counter is assigned to x, then the item that currently has the least estimated hits, *min*, is evicted and the counter is assigned to x. The intuition behind replacing the item with the least hits is to sacrifice the least amount of information about the stream history. Since the actual hits of x can be any number between 1 and *min* + 1, *Space-Saving* assumes x has been observed *min* + 1 times, since it gives items the benefit of doubt not to underestimate frequencies. Like *Lossy Counting*, for each monitored item the algorithm keeps track of its maximum possible over-estimation resulting from the initialization of its counter when starting monitoring it. To guarantee constant processing time per observation, the item with the least estimated hits has to be identified in

$O(1)$ time. To do that, *Space-Saving* uses a data structure similar to that in [7] in order to keep the counters always sorted by their estimated hits. However, not using the data structure still guarantees an amortized constant time per stream observation.

In general, counter-based techniques have fast per-item processing since counters can be stored in hash tables.

Sketch-Based Techniques

Sketch-based techniques do not monitor a subset of items, but rather provide, with less stringent guarantees, frequency estimation for all items by using arrays of counters. Usually, each counter monitors the frequencies of a subset of the item domains. Similarly, each item is hashed into the space of counters using a family of hash functions, and the hashed-to counters are updated for every hit of this item. These "representative" counters are then queried for the item frequency.

The *GroupTest* algorithm [6] solves the hot items problem, with a probability of failure of δ. The *FindMajority* algorithm was first devised to detect the majority item, by keeping a system of a global counter and $\lceil log(|A|) \rceil$ counters. Assuming that items' IDs are in the range $1...|A|$. A hit to item E is handled by updating the global counter, and all counters whose index corresponds to a 1 in the binary representation of E. At any time, counters whose value are more than half the global counter correspond to the 1s in the binary representation of the candidate majority item, if it exists. *GroupTest* is a simple generalization of *FindMajority*, which keeps only $O(\frac{\frac{1}{\phi}-1}{\delta}\log(\frac{1}{\phi}-1))$ of such systems, and uses a family of universal hash functions to map each item to $O(\frac{\log(\frac{1}{\phi}-1)}{\delta})$ *FindMajority* systems that monitor the occurrences of the item. When queried, the algorithm discards systems with more than one, or with no hot items. Also proposed is an elegant scheme for suppressing false positives by checking that all the systems a hot item belongs to report this item as being hot. *GroupTest* is generally accurate and can handle streams with both item insertions and deletions. However, it offers no information about items' frequencies or order.

The *Multistage Filters* approach, proposed by [8], which was also independently proposed by [10], is similar to *GroupTest*. The *Multistage Filters* algorithm hashes every item to a number of counters that are updated every time the item is observed in the stream. The item is considered to be frequent if the smallest of its representative counters satisfies the user required support. The algorithm by [8] judges an item to be frequent or not while updating its counters. If a counter is estimated to be frequent, it is added to a specialized set of counters for monitoring frequent items, the *flow memory*. To decrease the false positives, [8] proposed some techniques to reduce the over-estimation errors in counters. First, once an item is added to the flow memory, its counters are not monitored anymore by the *Multistage Filters*. Second, the algorithm increments only the counter(s) of the minimum value. Without the second error-reduction technique, the algorithm can be used for streams of both item insertions and deletions.

The *hCount* algorithm [10], does not employ the error reduction techniques employed by [8]. However, it keeps a number of imaginary items, which have no hits. At the end of the stream, all the items in the alphabet are checked for being frequent, and the over-estimation error for each of the items is estimated to be the average number of hits for the imaginary items. Both the *hCount* algorithm [10], and the *Multistage Filters* [8] require a number of counters bounded by $\frac{e}{\epsilon} * \ln\left(\frac{-|A|}{\ln \delta}\right)$.

In general, sketch-based techniques are not affected by the ordering of items in the stream, and are usually capable of handling streams with item insertions and deletions. On the other hand, they are probabilistic, and an item observation or a query about its frequency entails calculations across several counters.

Generalization for Sliding Windows

In [1], generalizations of the algorithm in [15], and *Sticky Sampling* were proposed for sliding windows of fixed and variable sizes (see Stream Models). Limiting the discussion to the deterministic [15] algorithm and the fixed size sliding window, the generalization monitors the stream at several levels of exponentially increasing granularities. That is, the finest granularity level, *level*-0, keeps copies of the algorithm that tracks the frequent items in sub-windows of size $f(W, \varepsilon)$, where W is the fixed size of the sliding window. These sub-windows are non-overlapping and contiguous such that their union is equivalent to the whole sliding window, modulo the oldest sub-window that contains expired items. The second finest granularity, *level*-1, keeps copies of the algorithm that tracks the frequent items in sub-windows of size $2 \times f(W, \varepsilon)$, and so forth. A boundary of a sub-window at any level has to cooccur with a boundary at the next finer level. Due to keeping several levels, and keeping several

algorithm copies at each level, the space requirement is $O(\frac{1}{\epsilon}(\log(\epsilon N))^2)$.

The case for variable size sliding window was handled essentially by forming new levels of coarser granularity from existing levels when the window size multiplies by a factor of 2. The requirement here is to be able to merge two copies of the algorithm, each has observed n items, to form a copy of the algorithm that is equivalent to observing the entire $2 \times n$ items. If the window shrinks to be a smaller size than the granularity of the coarsest level, the coarsest level is dropped.

At query time, the results of the algorithm copies at all levels are combined together to discover the frequent items in the entire sliding window. This requires the ability to combine the output of several algorithm copies while maintaining the error guarantees of the algorithm. The combining process does not include algorithm copies monitoring sub-windows containing expired items, does not include algorithm copies monitoring items that are already combined, and favors copies of the algorithm at the coarser levels.

The generalization of [15] to sliding windows at [12] does not keep several copies of the algorithm. Instead, the counters are incremented in a way that expires occurrences of the item that are older than the sliding window boundary. The space requirement is still $O(\frac{1}{\epsilon})$, thought the time for processing each element is also $O(\frac{1}{\epsilon})$. The case for variable size sliding window was handled essentially by adding new levels when the window size multiplies by a factor of 2. The requirement here is to be able to from a copy of a sliding window algorithm with error $2 \times \varepsilon n$ from another copy with error εn, where n is the number of items observed by the original copy. Like [1], if the window shrinks to be a smaller size than the granularity of the coarsest level, the coarsest level is dropped. The work done by Lee and Ting [12] was recently improved further in [16] to require $O(1)$ for processing each stream element using a doubly circularly-linked list to link the occurrences of the items sorted by their expiration time. Hence, the algorithm can expire occurrences of the items more efficiently.

Key Applications

Current motivation of the stream frequent items problem include network traffic analysis which is important for caching, routing, accounting, detecting network-level attacks, and maintenance [7,8]. In [14], the problem was applied for the application of increasing the revenue of Internet advertising networks. Motivated by these networks applications, [2] implemented *Space-Saving* and *Lossy Counting* on network processing units (NPUs), a special networking architecture with associative memories. In addition, [13] considered using frequent items queries to alleviate finding frequent itemsets (see Association rules), as well as iceberg queries and iceberg cubes. In specific, [5] generalized the problem to the case of multi-dimensional items in the presence of domain hierarchies (see Hierarchical heavy hitter mining on streams).

Experimental Results

Recently, Cormode and Hadjieleftheriou [4] conducted experiments comparing the most promising algorithms on both real and synthetic data, and noticed the competitive benefits of *Space-Saving* when processing append-only streams. In the case of streams with deletions, different sketch-based algorithms had different advantages.

In [13], the two proposed algorithms, *Lossy Counting* and *Sticky Sampling*, were compared on synthetic Zipfian data. *Lossy Counting* perform bettered even though the theoretical space bound of *Sticky Sampling* is better. However, the experiments were run with relatively small alphabets.

In [6], the algorithms *GroupTest*, *Lossy Counting*, and *Frequent* were compared on both real telephone connection and synthetic data. The data had both item insertions and deletions. Both *Lossy Counting* and *Frequent* were made to handle item deletions by decrementing the counters of the deleted items if such counters exist. Under these conditions, on synthetic data *GroupTest* performed best, followed by *Frequent*, and then *Lossy Counting* had the least precision and recall (the highest false positives and negatives). However, *GroupTest* used approximately 3× the space used by *Lossy Counting*, and 6× the space used by *Frequent*. On real data, *Lossy Counting* performed better than *Frequent*, but *GroupTest* was more accurate than both.

In [8], the proposed techniques were compared against commercial sampling-based algorithms for traffic measurement on real Internet backbone traffic traces. The results favored the *Multistage Filters* technique.

In [14], the algorithms *Space-Saving*, *GroupTest*, and *Frequent* were compared on real Internet advertising traffic as well as synthetic data, both with large alphabets. Throughout the experiments, *Frequent* attained a precision of less than 0.2. *Space-Saving* was

the only algorithm that neither reported false positives nor reported false negatives, even though it consumed roughly the same space of *Frequent*. On real data, *Space-Saving* consumed approximately one fifth of the space used by *GroupTest*, and on synthetic data, *Space-Saving* consumed at most one eighth of the space used by *GroupTest*.

Nagender Bandi et al. [2] compared how the hardware and software implementations of *Space-Saving* and *Lossy Counting* perform. The results of *Lossy Counting* conformed with those in [13]. *Lossy Counting* can use space much smaller than its bounds under small alphabets. *Lossy Counting* used less space than *Space-Saving* when the data was skewed and the skew was not estimated. However, *Space-Saving* used less space when the data skew was estimated since it monitored less items. The hardware implementation of *Space-Saving* ran significantly faster than *Lossy Counting*, while the software implementation ran only slightly faster.

Cross-references

- ▶ Hierarchical Heavy Hitter Mining on Streams
- ▶ Histograms on Streams
- ▶ Quantiles on Streams
- ▶ Stream Mining
- ▶ Stream Models
- ▶ Stream Sampling

Recommended Reading

1. Arasu A. and Manku G. Approximate counts and quantiles over sliding windows. In Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2004, pp. 286–296.
2. Bandi N., Metwally A., Agrawal D., and Abbadi A.E. Fast data stream algorithms using associative memories. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 247–256.
3. Boyer R. and Moore J. A Fast Majority Vote Algorithm. Tech. Rep. 1981-32, Institute for Computing Science, University of Texas, Austin, 1981.
4. Cormode G. and Hadjieleftherion M. Finding Frequent Items in Data Streams. Proc. VLDB, 1(2):1530–1541, 2008.
5. 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.
6. Cormode G. and Muthukrishnan S. What's Hot and What's Not: Tracking Most Frequent Items Dynamically. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2003, pp. 296–306, an extended version appeared in ACM Trans. on Comput. Syst., 30(1):249–278, 2005.
7. Demaine E., López-Ortiz A., and Munro J. Frequency estimation of internet packet streams with limited space. In Proc. 10th ESA European Symposium on Algorithms, 2002, pp. 348–360.
8. Estan C. and Varghese G. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. ACM Trans. Comput. Syst., 21(3):270–313, 2003.
9. Fischer M. and Salzberg S. Finding a Majority Among N Votes: Solution to Problem 81-5. J. Algorithms, 3:376–379, 1982.
10. Jin C., Qian W., Sha C., Yu J., and Zhou A. Dynamically maintaining frequent items over a data stream. In Proc. Int. Conf. on Information and Knowledge Management, 2003, pp. 287–294.
11. Karp R., Shenker S., and Papadimitriou C. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. ACM Trans. Database Syst., 28(1):51–55, 2003.
12. Lee L. and Ting H. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 290–297.
13. Manku G. and Motwani R. Approximate frequency counts over data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 346–357.
14. Metwally A., Agrawal D., and El Abbadi A. Efficient computation of frequent and top-k elements in data streams. In Proc. 10th Int. Conf. on Database Theory, 2005, pp. 398–412, an extended version appeared in ACM Trans Database Syst., 31(3):1095–1133, 2006.
15. Misra J. and Gries D. Finding repeated elements. Sci. Comput. Program., 2:143–152, 1982.
16. Zhang L. and Guan Y. Frequency estimation over sliding windows. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 1385–1387.

Frequent Itemset Mining with Constraints

Carson Kai-Sang Leung
University of Manitoba, Winnipeg, MB, Canada

Synonyms

Constrained frequent itemset mining; Frequent pattern mining with constraints; Frequent set mining with constraints

Definition

Let $\texttt{Item} = \{ item_1, item_2, \ldots, item_m \}$ be a set of domain items, where each item represents an object in a specific domain. Each object is associated with some attributes or auxiliary information about the object. A transaction $t_i = \langle tID, I_i \rangle$ is a tuple, where *tID* is a unique identifier and $I_i \subseteq \texttt{Item}$ is a set of items. A set of items is also known as an *itemset*. A transaction database *TDB* is a collection of transactions. An itemset S is *contained* in a transaction $t_i = \langle tID, I_i \rangle$ if $S \subseteq I_i$.

The *support* (or *frequency*) of an itemset S in a database *TDB* is the number (or percentage) of transactions in *TDB* containing S. An itemset is *frequent* if its support exceeds or equals a user-specified support threshold *minsup*. A user-specified constraint C is a predicate on the powerset of Item (i.e., C: $2^{\text{Item}} \mapsto$ {true, false}). An itemset S *satisfies* a constraint C if $C(S)$ evaluates to true. Hence, given (i) a transaction database *TDB*, (ii) a user-specified support threshold *minsup* and (iii) user-specified constraints C, the problem of *frequent itemset mining with constraints* is to find from *TDB* a set of frequent itemsets that satisfy C.

Historical Background

The research problem of frequent itemset mining was first introduced by Agrawal et al. [1] for market basket analysis in the context of association rule mining. Specifically, frequent itemset mining, which aims to find frequent itemsets from a transaction database, is an important first step of association rule mining. Once the frequent itemsets are found, they are used in the second step to generate association rules. The rules reveal the buying patterns in consumer behavior. For instance, they tell how the presence of some itemsets is associated with the presence of some other itemsets in the shopping baskets of consumers. This information is useful in making decisions in applications such as customer targeting, shelving, and sales promotion.

Besides the mining of association rules, frequent itemset mining also plays an essential role in many important data mining tasks – such as the mining of maximal itemsets, closed itemsets, correlation, causality, and sequential patterns. This explains why frequent itemset mining has been the subject of numerous studies since its introduction. Most of these studies focused either on improving efficiency of frequent itemset mining (e.g., the Apriori frequent itemset mining framework [2] and its tree-based counterpart [7]) or on extending the initial notion of frequent itemsets to other patterns such as maximal itemsets, closed itemsets, and correlated itemsets. Regardless whether they focused on these performance or functionality issues, these studies basically considered the data mining exercise in isolation. They relied on a computational model in which the computer does almost everything and the user is *not* engaged in the mining process. In other words, this model does not explore how data mining can interact with the human user. Consequently, the model provides little or no support for user focus.

However, in many applications, it is not uncommon for the user to have certain broad phenomena in mind, on which to focus mining. Without user focus, the mining process is treated as an impenetrable black-box – only allowing the user to set the support threshold *minsup* at the beginning and to get all the frequent itemsets at the end. The user does not have the opportunity to specify his interest via the use of constraints. As a result, the user often needs to wait for a long period of time for numerous frequent itemsets, out of which only a tiny fraction may be interesting to the user. This motivates the introduction of the research problem of *frequent itemset mining with constraints*, which gives the user opportunities to express his focus in mining by means of a rich set of constraints that captures application semantics.

Foundations

When compared with its traditional unconstrained counterpart, *frequent itemset mining with constraints* provides user focus in the sense that the user has opportunities to express his interest – via the use of constraints. By using the constraints, mining can be focused and computation can be saved. Over the past decade, several studies on frequent itemset mining with constraints have been proposed. For example, Ng, Lakshmanan and their colleagues [8,9,11] proposed a framework and algorithms for mining frequent itemsets with constraints. Within their framework, the user can use a rich set of constraints – which captures the semantics of itemsets – to guide the mining process for finding only those frequent itemsets that satisfy the constraints. These constraints include SQL-style aggregate constraints as well as domain constraints. The *SQL-style aggregate constraints* are of the following form:

$$agg(S.attribute)\ \theta\ constant,$$

where *agg* is an SQL-style aggregate function (e.g., *min*, *max*,*sum*,*avg*) and θ is a Boolean comparison operator (e.g., $=,\neq,<,\leq,\geq,>$). For example, by specifying the aggregate constraint $min(S.Price) \geq \$20$, the user expresses his interest in finding every itemset S such that the minimum price of all items in S is at least \$20. Other examples of aggregate constraints include $avg(S.Temperature) < -18^{\circ}C$ (which expresses that the average temperature of items in S is below $-18^{\circ}C$) and $max(S.Qty) > 15$ (which expresses that the maximum quantity of items in S is more than 15). *Domain*

constraints, which are non-aggregate constraints, can be of the following forms:

1. *S.attribute* θ *constant*, where θ is a Boolean comparison operator (e.g., $=,\neq,<,\leq,\geq,>$);
2. *constant* $\in$*S.attribute*;
3. *constant* $\notin$ *S.attribute*; or
4. *S.attribute* φ *set of constants*, where φ is a set comparison operator (e.g., $\subseteq,\not\supseteq,\subset,\not\subset,=,\neq,\supseteq,\not\supseteq,\supset,\not\supset$).

Examples of domain constraints include *S.ManufactureYear* $\neq$ 2008 (which expresses that each item in *S* is not manufactured in 2008), 2kg $\in$*S.Weight* (which expresses that a 2kg-item must be in *S*) and *S.Type* $\supseteq$ {*snack*, *soda*} (which expresses that *S* must contain some snacks and soda). These aggregate constraints and domain constraints can be categorized into several overlapping classes – such as *anti-monotone constraints*, *succinct constraints*, *monotone constraints*, and *convertible constraints* – based on properties of constraints. Figure 1 shows the characterization of some commonly used constraints.

To find frequent itemsets that satisfy the aforementioned constraints, Ng, Lakshmanan and their colleagues proposed algorithms – called *CAP* (*C*onstrained *Ap*riori) [11] and *DCF* (*D*ynamic *C*onstrained *F*requent-set computation) [8] – that exploit properties of these constraints. By doing so, both CAP and DCF ensure that the computational effort in mining frequent itemsets satisfying the constraints is proportional to the selectivity of the constraints. For instance, both CAP and DCF algorithms exploit a nice property of *anti-monotone constraints*: For an anti-monotone constraint C_{am} (e.g., *min*(*S.Price*) $\geq$ \$20, *S.ManufactureYear* $\neq$ 2008), if an itemset *S* does not satisfy C_{am} then supersets of *S* are guaranteed not to satisfy C_{am}. Hence, whenever *S* does not satisfy C_{am}, both CAP and DCF do not need to generate any superset of *S* as a candidate and thus do not need to count its support (or frequency). Consequently, by pushing C_{am} into the mining process, the mining algorithms save computation.

Note that many algorithms for mining frequent itemsets with or without constraints (e.g., Apriori [2], CAP [11] and DCF [8]) have also been exploiting a special anti-monotone constraint – namely, the *frequency constraint support*(*S*) $\geq$*minsup*, which states that the support (or frequency) of a frequent itemset *S* should equal or exceed the user-specified support threshold *minsup*. If *S* is infrequent, then the mining algorithms do not consider supersets of *S* because they are guaranteed to be infrequent. This property is commonly known as the Apriori property.

In addition to anti-monotone constraints, both CAP and DCF algorithms also exploit *succinct constraints* (e.g., *min*(*S.Price*) $\geq$ \$20, 2kg $\in$ *S.Weight*). For frequent itemset mining without constraint, it is well known that mining algorithms based on the Frequent-Pattern tree (FP-tree) [7] outperform their

	Anti-monotone	Succinct	Monotone	Convertible anti-monotone	Convertible monotone
$S.attr\ \theta\ const, \theta \in \{=,\leq,\geq\}$	√	√		√	
$const \in S.attr$		√	√		√
$const \notin S.attr$	√	√		√	
$S.attr \subseteq set\ of\ constants$	√	√		√	
$S.attr \supseteq set\ of\ constants$		√	√		√
$min(S.attr) \leq const$		√	√		√
$min(S.attr) \geq const$	√	√		√	
$max(S.attr) \leq const$	√	√		√	
$max(S.attr) \geq const$		√	√		√
$sum(S.attr) \leq const, \forall item_j \in S (item_j.attr \geq 0)$	√			√	
$sum(S.attr) \geq const, \forall item_j \in S (item_j.attr \geq 0)$			√		√
$sum(S.attr) \leq non\text{-}negative\ const$				√	
$sum(S.attr) \geq non\text{-}negative\ const$					√
$sum(S.attr) \leq non\text{-}positive\ const$					√
$sum(S.attr) \geq non\text{-}positive\ const$				√	
$avg(S.attr)\ \theta\ const, \theta \in \{\leq,\geq\}$				√	√
$support(S) \geq minsup$	√			√	

Frequent Itemset Mining with Constraints. Figure 1. Characterization of commonly used domain, aggregate and frequency constraints.

Apriori-based counterparts. As both CAP and DCF are Apriori-based, Leung et al. [10] proposed an FP-tree based algorithm – called *FPS* (*FP*-tree based mining of Succinct constraints) – for mining frequent itemsets that satisfy succinct constraints. By pushing the succinct constraint C_{suc} into the mining process, all three algorithms (CAP, DCF and FPS) directly generate precisely all and only those itemsets that satisfy C_{suc} by using a precise "formula" called a *member generating function* (*MGF*). As a result, there is no need to generate and then exclude itemsets not satisfying C_{suc}. For example, itemsets satisfying the succinct constraint $min(S.Price) \geq \$20$ can be generated by first selecting items with price at least \$20 from the domain `Item` and then combining these selected items:

$$\{X | X \subseteq \sigma_{Price \geq \$20}(\text{Item}),\ X \neq \emptyset\}.$$

As another example, itemsets satisfying the succinct constraint 2kg $\in S.Weight$ can be generated by combining at least one 2kg-item (i.e., at least one mandatory item) with some optional items of any weight:

$$\{Y \cup Z | Y \subseteq \sigma_{Weight=2\text{kg}}(\text{Item}),\ Y \neq \emptyset,\ Z \subseteq \sigma_{Weight \neq 2\text{kg}}(\text{Item})\}.$$

As the third example, itemsets satisfying the succinct constraint $S.Type \supseteq \{snack, soda\}$ can be generated by combining some snacks and some soda (which are mandatory items) with some optional items of other types:

$$\{X \cup Y \cup Z | X \subseteq \sigma_{Type=snack}(\text{Item}),\ X \neq \emptyset,\ Y \subseteq \sigma_{Type=soda}(\text{Item}),\ Y \neq \emptyset,\ Z \subseteq \sigma_{Type \neq snack \wedge Type \neq soda}(\text{Item})\}.$$

Among the succinct constraints in these three examples, the first one is also anti-monotone but the last two are not. In general, itemsets satisfying a succinct and anti-monotone constraint can be generated using only mandatory items (e.g., those with price $\geq$ \$20), whereas itemsets satisfying succinct but not anti-monotone constraints require both mandatory items and optional items.

Besides anti-monotone constraints and succinct constraints, there have been studies that handle other classes of constraints. For instance, Grahne et al. [6] exploited *monotone constraints* when finding correlated frequent itemsets. Since supersets of any itemset S satisfying a monotone constraint C_m (e.g., 2kg $\in$ *S. Weight*) are guaranteed to satisfy C_m, Grahne et al. pushed C_m into the mining process so that they do not need to perform further constraint checking on any superset of S once S satisfies C_m. Hence, computation is saved. Bucila et al. [4] proposed a dual mining algorithm – called *DualMiner* – that exploits both anti-monotone constraints and monotone constraints simultaneously to find itemsets satisfying the constraints.

Knowing that some constraints such as $avg(S.Temperature) < -18^{\circ}\text{C}$ are not anti-monotone or monotone in general, Pei and his colleagues [12,13] converted these "tough" constraints into anti-monotone constraints or monotone constraints by sorting items in each transaction in some order. They proposed the $\mathcal{FIC}$ algorithms (mining *F*requent *I*temsets with *C*onvertible constraints) to handle these *convertible constraints*. Specifically, the $\mathcal{FIC}^{\mathcal{A}}$ algorithm deals with *convertible anti-monotone constraints*, and the $\mathcal{FIC}^{\mathcal{M}}$ algorithm deals with *convertible monotone constraints*. For example, by arranging items in non-descending order of temperature, $\mathcal{FIC}^{\mathcal{A}}$ does not need to consider an itemset S if its prefix does not satisfy a convertible anti-monotone constraint C_{cam}. This is because S is guaranteed not to satisfy C_{cam} whenever the prefix of S does not. Similarly, by arranging items in non-ascending order of temperature, $\mathcal{FIC}^{\mathcal{M}}$ does not need to perform further constraint checking on an itemset S' if its prefix satisfies a convertible monotone constraint C_{cm}. This is because S' is guaranteed to satisfy C_{cm} whenever the prefix of S' does. Along this research direction, Bonchi and Lucchese [3] exploited "tough" constraints involving the aggregate functions variance and standard deviation.

Besides the aforementioned domain and aggregate constraints, the user can also express his interest by specifying other constraints. For example, Srikant et al. [14] considered *item constraints* that allow the user to impose a Boolean expression over the presence or absence of items in the itemset. Gade et al. [5] mined closed frequent itemsets that satisfy *block constraints*, which determine the significance of an itemset S by considering the dense blocks formed by items within S and by transactions associating with S. Yun and Leggett [15] proposed an algorithm for mining *weighted* frequent itemsets with *length decreasing support constraints*.

Key Applications

Frequent itemset mining – with or without constraints – plays an essential role in the mining of various patterns and relationships, which include maximal itemsets, closed itemsets, association rules, correlation, causality, sequential patterns, episodes, partial periodicity, emerging patterns, as well as frequent structures and trends. Moreover, frequent itemset mining is also useful in many data mining tasks such as associative classification, outlier detection, iceberg-cube computation, and stream mining. The knowledge discovered from frequent itemset mining can reveal important information in many real-life applications. Examples of these applications include market basket analysis (e.g., modeling of customer purchase behaviors, customer targeting, shelving, sales promotion), bioinformatics (e.g., order-preserving clustering of microarray data), Web mining (e.g., mining of Web contents, Web structures, or Web usages), mining for software reliability (e.g., software bug mining), as well as network management and intrusion detection (e.g., finding frequent routing paths, detecting signatures for intrusions).

Data Sets

Data sets commonly used for experimental evaluation for frequent itemset mining with constraints are similar to those used for frequent itemset mining *without* constraints. These data sets include the following:

1. IBM synthetic data generated by a data generator program developed at the IBM Almaden Center [2] (www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html)
2. Data from the UCI Machine Learning Repository (archive.ics.uci.edu/ml/)
3. Data from the Frequent Itemset Mining Dataset Repository (fimi.cs.helsinki.fi/data/)
4. Real-world data sets from the KDD Cup 2000, where the data sets contain purchase data from a real online retailer (www.sigkdd.org/kddcup/index.php?section=2000&method=task).

Cross-references

- ▶ Approximation of Frequent Itemsets
- ▶ Association Rule Mining on Streams
- ▶ Classification by Association Rule Analysis
- ▶ Closed Itemset Mining and Non-redundant Association Rule Mining
- ▶ Data Mining
- ▶ Frequent Itemsets and Association Rules
- ▶ Sequential Patterns
- ▶ Stream Mining

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 207–216.
2. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
3. Bonchi F. and Lucchese C. Pushing tougher constraints in frequent pattern mining. In Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conf., 2005, pp. 114–124.
4. Bucila C., Gehrke J., Kifer D., and White W. DualMiner: a dual-pruning algorithm for itemsets with constraints. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 42–51.
5. Gade K., Wang J., and Karypis G. Efficient closed pattern mining in the presence of tough block constraints. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 138–147.
6. Grahne G., Lakshmanan L.V.S., and Wang X. Efficient mining of constrained correlated sets. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 512–521.
7. Han J., Pei J., and Yin Y. Mining frequent patterns without candidate generation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 1–12.
8. Lakshmanan L.V.S., Leung C.K.-S., and Ng R.T. Efficient dynamic mining of constrained frequent sets. ACM Trans. Database Syst., 28(4):337–389, 2003.
9. Lakshmanan L.V.S., Ng R., Han J., and Pang A. Optimization of constrained frequent set queries with 2-variable constraints. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 157–168.
10. Leung C.K.-S., Lakshmanan L.V.S., and Ng R.T. Exploiting succinct constraints using FP-trees. ACM SIGKDD Explor., 4(1):40–49, 2002.
11. Ng R.T., Lakshmanan L.V.S., Han J., and Pang A. Exploratory mining and pruning optimizations of constrained associations rules. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 13–24.
12. Pei J. and Han J. Can we push more constraints into frequent pattern mining? In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 350–354.
13. Pei J., Han J., and Lakshmanan L.V.S. Mining frequent item sets with convertible constraints. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 433–442.
14. Srikant R., Vu Q., and Agrawal R. Mining association rules with item constraints. In Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining, 1997, pp. 67–73.
15. Yun U. and Leggett J. WLPMiner: weighted frequent pattern mining with length-decreasing support constraints. In Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conf., 2005, pp. 555–567.

Frequent Itemsets and Association Rules

Hong Cheng, Jiawei Han
University of Illinois at Urbana-Champaign, Urbana, IL, USA

Synonyms

Frequent Patterns; Large Itemsets

Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items, and $DB = \{T_1, T_2, \ldots, T_m\}$ be a transaction database, where T_i ($i \in [1 \ldots m]$) is a transaction containing a set of items in I. The *support* (or occurrence frequency) of an itemset A, where A is a set of items from I, is the number of transactions containing A in DB. An itemset A is *frequent* if A's support is no less than a user-specified *minimum support threshold* θ. An itemset A which contains k items is called a k-itemset.

Historical Background

Frequent itemset mining was first proposed by Agrawal et al. [2] for market basket analysis in the context of association rule mining. It analyzes customer buying habits by finding associations between the different items that customers place in their "shopping baskets." For instance, if customers are buying milk, how likely are they going to also buy cereal (and what kind of cereal) on the same trip to the supermarket? Such information can lead to increased sales and maximize the profit by helping retailers do selective marketing and arrange their shelf space.

Since there are usually a large number of distinct single items in a typical transaction database, and their combinations may form a very huge number of itemsets, it is challenging to develop scalable methods for mining frequent itemsets in a large transaction database. The first frequent itemset mining algorithm was Apriori, proposed by Agrawal and Srikant [3]. An interesting *downward closure* property, called Apriori property, was observed: A *k-itemset is frequent only if all of its sub-itemsets are frequent*. This implies that frequent itemsets can be mined by first scanning the database to find the frequent 1-itemsets, then using the frequent 1-itemsets to generate candidate frequent 2-itemsets, and check against the database to obtain the frequent 2-itemsets. This process iterates until no more frequent k-itemsets can be generated for some k. This is the essence of the Apriori algorithm.

Scientific Fundamentals

In many cases, the Apriori algorithm significantly reduces the size of candidate sets using the Apriori principle. However, it can suffer from two nontrivial costs: (i) generating a huge number of candidate sets, and (ii) repeatedly scanning the database and checking the candidates by pattern matching.

Han et al. [10] devised an FP-growth method that mines the complete set of frequent itemsets without candidate generation. FP-growth works in a *divide-and-conquer* way. The first scan of the database derives a list of frequent items in which items are ordered in frequency-descending order. According to the frequency-descending list, the database is compressed into a frequent-pattern tree, or *FP-tree*, which retains the itemset association information.

An example database from [10] is shown in Table 1 and the corresponding FP-tree is shown in Fig. 1. In this way, the problem of mining frequent patterns in databases is transformed to that of mining the FP-tree.

The FP-tree is mined by starting from each frequent length-1 pattern (as an initial suffix pattern), constructing its *conditional pattern base* (a "subdatabase," which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern), then constructing its conditional FP-tree, and performing mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree. Continue with the example in [10], Fig. 2 shows the recursive mining process on the conditional FP-tree of item m. It derives three frequent patterns ($am : 3$), ($cm : 3$) and ($fm : 3$). Recursive calls of the FP-growth algorithm construct the conditional FP-trees of am, cm and fm (the conditional FP-tree of fm is empty in this example) respectively. The recursive mining process on these conditional FP-trees is shown in Fig. 2.

The FP-growth algorithm transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. Performance studies demonstrate that the method substantially reduces search time.

Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in *horizontal data format* (*i.e.*, {*TID: itemset*}), where TID is a

transaction-id and *itemset* is the set of items bought in the transaction TID. Alternatively, mining can also be performed with data presented in *vertical data format* (i.e., {*item: TID_set*}).

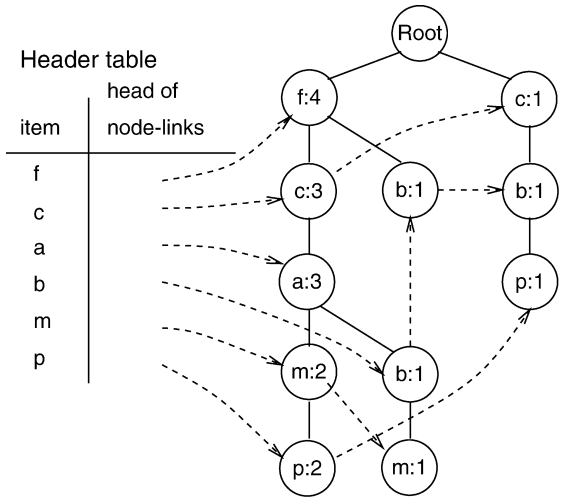
Zaki [20] proposed Eclat (Equivalence CLASS Transformation) algorithm by exploring the vertical data format. The first scan of the database builds the TID_set of each single item. Starting with a single item ($k = 1$), the frequent $(k + 1)$-itemsets grown from a previous k-itemset can be generated according to the Apriori property, with a depth-first computation order similar to FP-growth [10]. The computation is done by intersection of the TID_sets of the frequent k-itemsets to compute the TID_sets of the corresponding $(k + 1)$-itemsets. This process repeats, until no frequent itemsets or no candidate itemsets can be found.

Besides taking advantage of the Apriori property in the generation of candidate $(k + 1)$-itemset from frequent k-itemsets, another merit of this method is that there is no need to scan the database to find the support of $(k + 1)$-itemsets (for $k \geq 1$). This is because the TID_set of each k-itemset carries the complete information required for counting such support.

There are many alternatives and extensions on frequent itemset mining, e.g., hashing technique [14], partitioning technique [16], sampling approach [18], dynamic itemset counting [4], and so on. A FIMI (Frequent Itemset Mining Implementation) workshop dedicated to the implementation methods of frequent itemset mining was reported by Goethals and Zaki [9].

Frequent Itemsets and Association Rules. Table 1. Example transaction database D, $\theta = 3$

TID	Items bought	(Ordered) frequent items
100	*f,a,c,d,g,i,m,p*	*f,c,a,m,p*
200	*a,b,c,f,l,m,o*	*f,c,a,b,m*
300	*b,f,h,j,o*	*f,b*
400	*b,c,k,s,p*	*c,b,p*
500	*a,f,c,e,l,p,m,n*	*f,c,a,m,p*

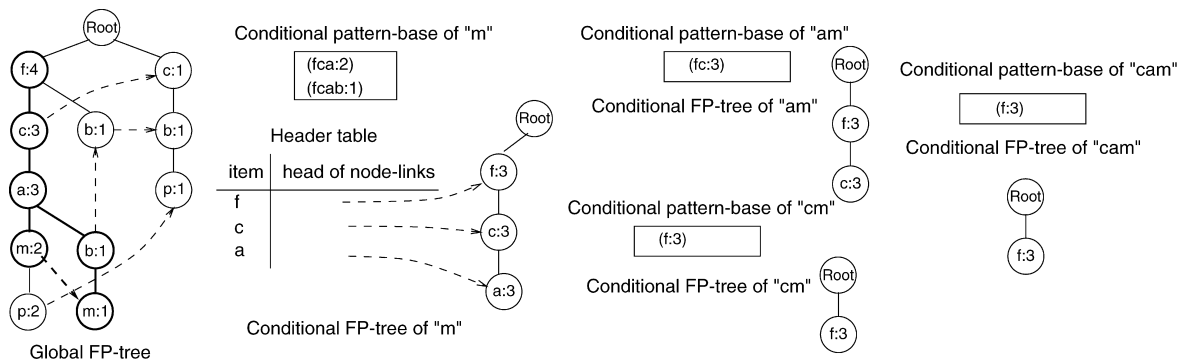


Frequent Itemsets and Association Rules. Figure 1. The FP-tree for database in Table 1.

Key Applications

Association and Correlation Analysis

Frequent itemset mining naturally leads to the discovery of associations and correlations among items in large transaction data sets. The concept of association rule was introduced together with that of frequent itemset [2]. An association rule r takes the form of $\alpha \rightarrow \beta$, where α and β are itemsets, and $\alpha \cap \beta = \phi$. support and confidence are two measures of rule



Frequent Itemsets and Association Rules. Figure 2. Mining the conditional FP-tree for item *m*.

interestingness, where $support(r) = support(\alpha \cup \beta)$ and $confidence(r) = \frac{support(\alpha \cup \beta)}{support(\alpha)}$ for a rule r.

Sometimes, an association rule may not be interesting, especially when mining at a low support threshold or mining for long patterns. To mine interesting rules, a correlation measure has been used to augment the support-confidence framework of association rules. This leads to the correlation rules of the form $\alpha \Rightarrow \beta[support,confidence,correlation]$. There are various correlation measures including lift, χ^2, *cosine* and *all_confidence*.

Frequent Pattern-based Classification and Clustering

Frequent itemsets have been demonstrated to be useful for classification, where association rules are generated and analyzed for use in classification [6,11,13]. The general idea is that strong associations between frequent patterns and class labels can be discovered. Then the association rules are used for prediction. In many studies, associative classification has been found to be more accurate than some traditional classification methods, such as C4.5.

Cluster analysis in high-dimensional space is a challenging problem. Since it is easy to compute frequent patterns in subsets of high dimensions, it provides a promising direction for high-dimensional subspace clustering. Two algorithms CLIQUE [1] and ENCLUS [5] were proposed, both of which used the Apriori property to mine interesting subspaces.

Biological Data Analysis

An important experimental application of frequent itemsets is the exploration of gene expression data, where the joint discovery of both the set of conditions that significantly affect gene regulation and the set of co-regulated genes is of great interest. Wang et al. proposed pCluster [15], a pattern similarity-based clustering method for microarray data analysis, and demonstrated its effectiveness and efficiency for finding subspace clusters in high-dimensional space.

Cong et al. [7] proposed to discover top-k covering rule groups for each row of gene expression profiles. It uses a row enumeration technique and introduces several pruning strategies to make the rule mining process very efficient. A classifier is constructed from the top-k covering rule groups.

Web Mining and Software Bug Mining

Frequent itemset mining can also be applied to other application domains like Web and software debugging. Association rules discovered for pages that are often visited together can reveal user groups [8] and cluster web pages. Web access patterns via association rule mining in web logs were proposed in [15,17].

Frequent pattern mining has started playing an important role in software bug detection and analysis. PR-Miner [12] uses frequent itemset mining to extract application-specific programming rules from source code. A violation of these rules might indicate a potential software bug.

Future Directions

First, the set of frequent itemsets derived by most of the current mining methods is too huge for effective usage. The bottleneck of frequent itemset mining is not on whether users can derive the complete set of frequent patterns under certain constraints efficiently but on whether they can derive a compact but high quality set of patterns that are most useful in applications. There are proposals on reduction of such a huge set, including closed patterns, maximal patterns, approximate patterns, condensed pattern bases, representative patterns, etc. However, it is still not clear what kind of patterns will give satisfactory pattern sets in both compactness and representative quality for a particular application. Much research is still needed to substantially reduce the size of derived pattern sets and enhance the quality of retained patterns.

Second, although there are efficient methods for mining precise and complete set of frequent itemsets, approximate frequent patterns could be the best choice to handle noise or variations in many applications such as bioinformatics. How to define the approximate constraint and design efficient mining algorithms is an open question. Much research is still needed to make such mining effective and efficient.

Third, to make frequent itemset mining an essential task in data mining, much research is needed to further develop pattern-based mining methods. For example, classification is an essential task in data mining. How to construct a better classification model using frequent patterns than most other classification methods? What kind of frequent patterns are more effective and discriminative than other patterns? How to mine such patterns directly from data? These questions need to be answered before frequent patterns can play an essential role in several major data mining tasks, such as classification.

Experimental Results

In general, for every proposed method, there is an accompanying experimental evaluation in the corresponding reference. In addition, [9] in FIMI workshop provided a detailed and comprehensive experimental evaluation of many mining methods on a large set of benchmark data.

Data Sets

An IBM Quest synthetic data generator is available at http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html.

A large collection of real datasets can be found at http://fimi.cs.helsinki.fi/data/.

URL to Code

FIMI workshop website (http://fimi.cs.helsinki.fi/src/) contains the code for many different frequent itemset mining methods.

Cross-references

- ▶ Approximation of Frequent Itemsets
- ▶ Association Rule Mining on Streams
- ▶ Closed Itemset Mining and Non-redundant Association Rule Mining
- ▶ Frequent Graph Patterns
- ▶ Mase-pattern Mining
- ▶ Sequential Patterns

Recommended Reading

1. Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 94–105.
2. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
3. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
4. Brin S., Motwani R., Ullman J.D., and Tsur S. Dynamic itemset counting and implication rules for market basket analysis. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 255–264.
5. Cheng C.-H., Fu A.W., and Zhang Y. Entropy-based subspace clustering for mining numerical data. In Proc. 5th ACM SIGKOD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 84–93.
6. Cheng H., Yan X., Han J., and Hsu C. Discriminative frequent pattern analysis for effective classification. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 716–725.
7. Cong G., Tan K.-L., Tung A.K.H., and Xu X. Mining top-k covering rule groups for gene expression data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 670–681.
8. Eirinaki M. and Vazirgiannis M. Web mining for web personalization ACM Trans. Int. Tech., 3:1–27, 2003.
9. Goethals B. and Zaki M. An introduction to workshop on frequent itemset mining implementations. In Proc. ICDM'03 International Workshop on Frequent Itemset Mining Implementations, 2003, pp. 1–13.
10. Han J., Pei J., and Yin Y. Mining frequent patterns without candidate generation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 1–12.
11. Li W., Han J., and Pei J. CMAR: Accurate and efficient classification based on multiple class-association rules. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 369–376.
12. Li Z. and Zhou Y. PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code. In Proc. ACM SIGSOFT Symp. on Foundations Software Eng., 2005, pp. 306–315.
13. Liu B., Hsu W., and Ma Y. Integrating classification and association rule mining. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 80–86.
14. Park J.S., Chen M.S., and Yu P.S. An effective hash-based algorithm for mining association rules. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 175–186.
15. Pei J., Han J., Mortazavi B.-A., and Zhu H. Mining access patterns efficiently from web logs. In Advances in Knowledge Discovery and Data Mining, 4th Pacific-Asia Conf., 2000, pp. 396–407.
16. Savasere A., Omiecinski E., and Navathe S. An efficient algorithm for mining association rules in large databases. In Proc. 21st Int. Conf. on Very Large Data Bases, 1995, pp. 432–443.
17. Srivastava J., Cooley R., Deshpande M., and Tan P. Web usage mining: discovery and applications of usage patterns from web data. SIGKDD Explor., 1:12–23, 2000.
18. Toivonen H. Sampling large databases for association rules. In Proc. 22nd Int. Conf. on Very Large Data Bases, 1996, pp. 134–145.
19. Wang H., Wang W., Yang J., and Yu P.S. Clustering by pattern similarity in large data sets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 418–427.
20. Zaki M.J. Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng., 12:372–390, 2000.

Frequent Partial Orders

Antti Ukkonen
Helsinki University of Technology, Helsinki, Finland

Definition

Given a set D of n *partial orders* on S, and a threshold $\sigma \leq n$, a partial order P is a *frequent partial order* (FPO)

if it is compatible with more than σ partial in D. Typically D contains total orders either on S or arbitrary subsets of S.

Historical Background

A natural extension of association rule mining is to make use of temporal information. This was first done in [1], where the authors present algorithms for mining *frequently occurring sequences* of sets of items in a database of transactions. Each of such sequences can be seen as a partial order on the complete set of items. For more recent work on the same topic please see [13,812]. The slightly different problem of mining *frequent episodes* from a sequence of events is presented in [7]. In this case an episode is a partial order over the set of all possible events. The problem differs from the one of [1] by considering a stream of events (for example notifications and alerts generated by devices in a telecommunications network) instead of a stationary set of transactions. The same problem is also discussed e.g., in [4,5].

Common for the references above is that they do not explicitly call the mined patterns partial orders. One of the first papers to do so is [6], where the authors discuss the problem of finding a number of partial orders that are a good compact description for a set of input sequences. The approach taken differs from the traditional setting of frequent pattern mining, however. A similar problem is addressed recently in [3], albeit in a more theoretical setting.

The problem of finding frequent partial orders as defined in this article was first addressed in [9,10]. The authors present an efficient algorithm for finding frequent closed partial orders (The term closed appears in the same meaning as in the context of *frequent closed itemsets*.) in a database of strings with the restriction that each symbol of the alphabet may occur only once or not at all in a string.

Foundations

The basic idea of frequent pattern mining can be formalized as follows: given a database D, a pattern class $\mathcal{P}$ and a threshold σ, find all instances of $\mathcal{P}$ that are supported by more than σ rows of D. The precise definition of a support depends on the contents of D and the pattern class $\mathcal{P}$. In case of frequent partial orders the pattern class is the set of all partial orders on some fixed set S, and the database can contain either total or partial orders on S. All orders in D that are compatible with the partial order P form the *support* of P, denoted $s(P)$. Given D and a threshold σ, the problem is to find all partial orders P so that $|s(P)| \geq \sigma$.

A pattern is *closed* if it can not be augmented without decreasing the size of its support. According to this definition the partial order P is closed in the database D if $|s(P \cup (u, v))| < |s(P)|$ for all $u, v \in S$. It can be argued that finding only the frequent closed patterns is of interest, as the non-closed ones contain less information but can be considered only equally "reliable," as they have the same support as a closed pattern. In the remainder of this section two methods for finding *frequent closed partial orders* are discussed. The first one is based on using existing algorithms for mining *frequent closed itemsets*, while the second one is a dedicated method for finding frequent partial orders.

Finding Frequent Closed Partial Orders Using Frequent Itemset Mining Algorithms

Since its original development in the early 1990s, association rule mining and especially the discovery of frequent itemsets has been a widely studied topic. As a result, there currently exist a myriad of efficient algorithms for mining frequent closed itemsets. Turns out that any of these can be used to find frequent closed partial orders if the input is in a suitable format.

Let D contain total orders on S. Usually each total order $T \in D$ is given as a list of symbols. For example, let $T = \langle a, b, c, d, e \rangle$, meaning that a comes first in T, b comes second, and so on. This is the *list representation* of T, which is a compact and intuitive way of representing total orders, but can not as such be used with frequent itemset mining algorithms, because they only consider the occurrence of a symbol and not their position relative to the other symbols in the list.

Any order relation can also be expressed as a set of ordered pairs (u, v). The pair (u, v) belongs to T when u appears before v in the list representation of T. Returning to the example, the *set representation* of T is

$$T = \{(a,b),(a,c),(a,d),(a,e),(b,c),(b,d),(b,e),(c,d),(c,e),(d,e)\}.$$

This differs from the list representation by having pairs of symbols instead of single symbols as "items." Thus, if two total orders have the same pair (u, v), they must agree on the order between u and v.

Given a database D of total orders in the list representation, each member of D is converted to the set

representation. The resulting database is denoted $\hat{D}$. Each row of $\hat{D}$ is a set of (u, v) pairs that form a total order $T \in D$. This representation of the input has the consequence that every closed itemset (set of (u, v) pairs) that is frequent in $\hat{D}$ given the threshold σ can in fact be interpreted as a frequent closed partial order. An itemset I is closed if no more items can be added to I without decreasing the size of its support.

Note that it is important to find the frequent closed itemsets, as a frequent itemset might not correspond to a partial order. The closedness guarantees that the resulting sets of (u, v) pairs form a transitive relation, which is required of a partial order.

Using the set representation for finding frequent partial orders works, but it has some drawbacks. First of all, the storage requirements for $\hat{D}$ are much larger than for D, as each total order with a list representation of l symbols must be replaced with a set containing $\frac{1}{2}l(l-1)$ elements. In addition, $\hat{D}$ might not be sparse, meaning the number of items in each row (total order) is not necessarily small when compared to the total number of items. This in turn may lead to poor performance of the frequent itemset mining algorithms.

The most important problem with the above approach is, however, that in general a complete partial order P is not required. Very often it is sufficient to find its *transitive reduction* $tr(p)$. All of the ordering information contained in P is retained in $tr(p)$, which is better suited for analysis purposes. For example, when visualizing partial orders as directed acyclic graphs, it is in general better to use $tr(p)$. Obviously the transitive reduction can be computed afterwards given the frequent itemsets, but this is another computationally intensive step. Also, frequent partial orders can be found more efficiently when the transitive reduction is mined directly.

Finding Transitive Reductions of Frequent Closed Partial Orders Directly

Details of the approach are given in [10], this is only a summarization of the key ideas. This algorithm (called Frecpo in [10]) operates directly on the list representations in D and returns the set of the transitive reductions of all closed partial orders that are frequent in D given the threshold σ.

The basic framework is based on enumerating all representations of transitive reductions of partial orders in a depth-first fashion using a recursive algorithm. All closed partial orders that can not be frequent are pruned as soon as possible. The pruning is based on the observation that the frequency of the partial order $P \cup (u, v)$ is upper bounded by the frequency of P. In general this is called *antimonotonicity* of a pattern.

When entering a recursive call the algorithm has already constructed a frequent partial order P in the previous steps. This is called the *current pattern*, which is initially Ø. The algorithm first computes a list L of pairs that are added to P one after the other to create a new frequent partial order. For a pair (u, v) to be included in this list it must (i) belong to a transitive reduction and (ii) the frequency of the pattern $P \cup (u, v)$ must be above the threshold σ. When computing this list the algorithm only considers total orders that belong to the support of the current pattern P, denote this with D_P Initially $D_P = D$.

To construct the list L, the algorithm computes a table called the *detection matrix*. In this matrix for each pair (u, v) is stored the number of times u precedes v in D_P, and the set of items that appear between u and v in every total order where u precedes v. These items are called *anchors*. The frequency information can be used to prune infrequent pairs. If the frequency of the pair (u, v) is below σ in D_P, the partial order $P \cup (u, v)$ can not be frequent either. However, if the frequency of (u, v) is above σ in D_P, then $P \cup (u, v)$ must be a frequent partial order as well.

The set of anchors is used to identify forbidden pairs. A pair is forbidden if it's set of anchors is not empty. This is because a pair (u, v) can not belong to the transitive reduction if an item (the anchor) occurs between u and v in every total order where u and v occur. All pairs that are not infrequent or forbidden are added to the list L.

For details of the algorithm the reader is referred to [10], where it is also shown experimentally that Frecpo outperforms the algorithm based on frequent itemset mining by a considerable margin.

Key Applications

Frequent partial orders can be of interest in any application where the data can be viewed as a set of orders (or rankings) of some finite set.

For example, in certain voting systems the voters do not only place a single vote on one candidate, but are expected to rank the alternatives (or a subset thereof) according to their preferences. This voting

mechanism is employed for instance in the general election of Ireland. Finding FPOs of the candidates from this data can give more insight to the behavior of voters than traditional opinion polls. Preference data in general is a natural application for FPOs. Such data can be based on questionnaires, but also other sources.

A related application is clickstream analysis. Based on server log files it is possible to reconstruct the sequence in which a user visited different pages of a website. Finding FPOs from these sequences can give information about user preferences (if the pages correspond to different products, for example) or potential usability problems associated with navigation on the website in question.

Another promising application is in bioinformatics in the context of gene expression analysis. A gene expression data set usually contains expression levels of several genes in a number of conditions or tissues. Instead of looking at the actual expression value, which can be very noisy, the conditions or tissues can be ranked in decreasing order of expression and use the resulting rankings for further analysis. In this case the FPOs can be seen as a generalization of the order preserving submatrices, originally proposed in [2].

Cross-references

- ▶ Frequent Itemsets and Association Rules
- ▶ Sequential Patterns

Recommended Reading

1. Agrawal R. and Srikant R. Mining sequential patterns. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 3–14.
2. Ben-Dor A., Chor B., Karp R., and Yakhini Z. Discovering Local Structure in Gene-Expression Data: The Order Preserving Submatrix Problem. In Proc. 6th Annual Int. Conf. on Computational Biology, 2002, pp. 49–57.
3. Fernandez P.L., Heath L.S., Ramakrishnan N., and Vergara J.P. Reconstructing Partial Orders from Linear Extensions. In Proc. 4th SIGKDD Workshop on Temporal Data Mining: Network Reconstruction from Dynamic Data, 2006.
4. Gwadera R., Atallah M.J., and Szpankowski W. Reliable Detection of Episodes in Event Sequences. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 67–74.
5. Laxman S., Sastry P.S., and Unnikrishnan K.P. A fast algorithm for finding frequent episodes in event streams. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 410–419.
6. Mannila H. and Meek C. Global Partial Orders from Sequential Data. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 161–168.
7. Mannila H., Toivonen H., and Verkamo I. Discovering frequent episodes in sequences. In Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, 1995, pp. 210–215.
8. Pei J., Han J., Mortazavi-Asl B., Pinto H., Chen Q., Dayal U., and Hsu M.-C. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 215–224.
9. Pei J., Liu J., Wang H., Wang K., Yu P.S., and Wang J. Efficiently Mining Frequent Closed Partial Orders. In Proc. 2005 IEEE Int. Conf. on Data Mining, 2005, pp. 753–756.
10. Pei J., Wang H., Liu J., Wang K., Wang J., and Yu P.S. Discovering frequent closed partial orders from strings. IEEE Trans. Knowl. Data Eng., 18(11):1467–1481, 2006.
11. Wang J. and Han J. BIDE: Efficient Mining of Frequent Closed Sequences. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 79–90.
12. Yan X., Han J., and Afshar R. CloSpan: Mining Closed Sequential Patterns in Large Datasets. In Proc. SIAM International Conference on Data Mining, 2003, pp. 166–177.
13. Zaki M. SPADE: an efficient algorithm for mining frequent sequences. Mach. Learn. J., 42(1/2):31–60, 2000.

Frequent Pattern Mining with Constraints

▶ Frequent Itemset Mining with Constraints

Frequent Patterns

▶ Frequent Itemsets and Association Rules

Frequent Set Mining with Constraints

▶ Frequent Itemset Mining with Constraints

Frequent Subsequences

▶ Sequential Patterns

Freshness Control

▶ Replica Freshness

Full Text Inverted Index

► Inverted Files

Fully-Automatic Web Data Extraction

Cai-Nicolas Ziegler
Siemens AG, Munich, Germany

Synonyms

Web content extraction; Automatic wrapper induction; Web information extraction

Definition

Web documents contain abundant hypertext markup information, both for indicating structure as well as for giving page rendering hints, next to informative textual content. Fully-automatic Web data extraction is geared towards extracting all relevant textual information from HTML documents, without requiring human intervention throughout the process. Commonly, two types of automatic Web extraction paradigms are distinguished in this vein. First, the extraction of one single block of informative content, e.g., in case of news pages, which is also referred to as page cleaning [4]. Second, the extraction of recurring patterns across multiple blocks, typically the case for the extraction of search engine results. In the latter case, the extraction system will commonly also assign *labels* to the single atoms of each identified recurring block, such as the search result record's title, snippet, and URL.

Historical Background

Systems for extracting information from Web pages date back to the late mid-nineties. First approaches, coined wrapper induction systems [6], extracted structured information from HTML documents in a semi-automated fashion, see, e.g., [1] and [9]. A wrapper hereby refers to a set of learned rules that extract structured information records from HTML pages of like style, i.e., dynamically generated pages for which the same template has been used. One may think of two product pages from Amazon.com as an example. The rules have been learned inductively, by means of humans labeling a sufficient number of pages.

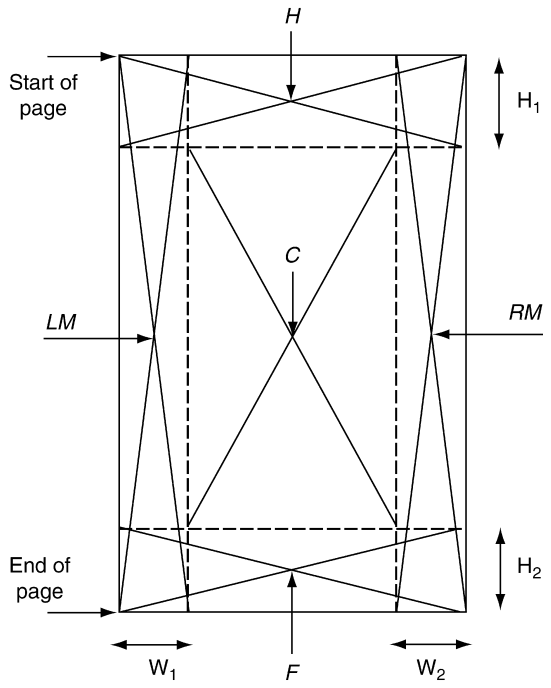
While these wrapper induction systems have been continuously improved to require less human intervention and less time for designing wrappers, they still do not scale to efforts that require the extraction of massive amounts of information from diverse types of Web document sources: Even with an effort of only five minutes per site template, coping with several thousands of those requires too many human resources for most project budgets.

Two areas have been identified where extraction works in a fully-automated fashion, the first referring to the extraction of results from any type of search engine, such as Web search engines like Google and Yahoo, product finders as they are found on Amazon and friends, and so forth. These systems are able to automatically generate wrappers as they exploit the fact that search result pages commonly feature large numbers of recurring patterns. Moreover, these automated search engine result wrappers also assign labels to the components of each entry found, such as the title of the search result snippet, the URL, and body text.

A second breed of automated page wrappers has evolved at the end of the nineties. Theses wrappers aim at extracting the purely informative content from Web pages, particularly *news* Web pages. As opposed to the automated search engine result wrappers, no labels are assigned to the different blocks of information. Hence, the generated output is commonly one single plain text document which contains the informative textual content only. Extractors of informative textual content have also been referred to as "page cleaners" [1].

Foundations

For the extraction of informative content from Web pages, also referred to as Web page cleaning [1], a broad range of diverse techniques has been investigated. Some systems use simple heuristics (see, e.g., [1] and [5]) and obtain accuracy scores that are reasonably good. For example, such heuristics comprise of rules that take into account the number of characters and text tokens which occur in one cohesive textual block [1]. Other approaches compute a set of features for each text block and implement a decision function which tells whether to discard or keep the block at hand. Such approaches are depicted in [1,6,7]. The respective decision functions are learned by means of non-linear optimization methods or common classifier training [7].



Fully-Automatic Web Data Extraction. Figure 1. The approach of Kovacevic et al. [5] segments an HTML page's visual representation into five regions.

Fully-Automatic Web Data Extraction. Figure 2. Sample search result entry (a) and its abstract shape (b), as used in ViNTS [13].

Next to approaches that operate either on the text level or logical structure of a Web page, e.g., its DOM tree, the exploitation of visual cues has been likewise researched: The location-based segmentation approach described in [8] constructs an M-tree to represent the HTML document's physical representation as seen in the browser window. To this end, the browsers screen coordinates of block segments are taken into account. The screen is divided into five areas: header, left and right menu, footer, and the center of the page (see Fig. 1). Heuristics are used to assign HTML blocks to the defined areas, the informative content is assumed to reside in the center of the page.

For extracting *recurring* information blocks from Web pages, in particular Web search engine results, state-of-the-art wrapper generators also exploit visual cues by analyzing an HTML page's graphical representation as rendered by a Web browser engine [9,10]. ViNTs [10] automatically generates search result record extraction rules using visual context features and tag structure information. To this end, ViNTS first analyzes the graphical representation without considering the tag structure to identify content regularities by means of so-called content lines (see Fig. 2 for a content line typical of search results on Amazon). Next, structural regularities among HTML blocks are combined with these visual features to generate wrappers. To weight the relevance of different extraction rules, visual and non-visual features are considered.

The ViPER system [9] builds on similar techniques as ViNTS, but extends its capabilities by not only allowing to identify recurring blocks that are aligned vertically, but also those aligned in a horizontal fashion. Horizontally aligned search results are becoming increasingly popular among online retailers, such as Overstock (http://www.overstock.com). Next to the exploitation of visual cues, ViPER used multiple sequence alignment techniques, known from bio-informatics, to identify structure and patterns in HTML tag sequences.

Key Applications

The extraction and automatic labeling of search results sees its application in various domains, primarily product-related data integration. For instance, price robots access large numbers of online retail shops and need to interpret search results for product searches so as to digest and incorporate the found information into their own databases. Moreover, meta search engines also have an increasing demand for the automated extraction of search results, as their operation is based on the merging of search results from several hundreds of search engines, for which the manual design of extraction wrappers is enormously time-consuming. Moreover, manually crafted wrappers may break easily, when the structural template of presented results changes. Automatically generated search engine wrappers are less prone to these deficiencies.

The extraction of informative content from arbitrary HTML documents serves many purposes. Clearly, when used as processing step for Web search engines, it can help to improve the precision of search results dramatically, as only an HTML document's informative content is considered for inclusion in the search

index. Pages where the search terms only occur in non-informative content, such as an advertisement or as part of a link list, are not considered as hits anymore.

Next to general-purpose search engines, the extraction of informative page content is likewise essential for special-purpose applications based on retrieval, such as reputation monitoring platforms [3,11]: These systems record the number of mentions of monitored keywords, such as company, brand, or product names, in order to allow for timeline-based trend analysis. Citation count numbers hence become more reliable, for the same reasons as those stated for Web search engines.

Automated Web content extraction for page cleaning is also at the heart of the CLEANEVAL competition (See *http://cleaneval.sigwac.org.uk/* for details.), which has become part of the "Web as Corpus" initiative (WAC) as of May 2007. The objective of WAC is to collect massive textual information from the Web in order to use it for natural language processing (NLP) and linguistic research, forming representative background corpora and language models.

Data Sets

For the CLEANEVAL competition, a dataset containing both unlabeled documents (for testing purposes) and labeled documents (for classifier training) can be downloaded from the indicated URL. The relatively new dataset is expected to serve as publicly accepted benchmark in the future.

For automatic extraction of search engine results, several smaller datasets are available, among those the Omini dataset (Available from Sourceforge via *http://sourceforge.net/projects/omini/*.) and MDR collection [8]. None of them may count as standard dataset, though.

Cross-references

- ▶ GUIs for Web Data Extraction
- ▶ Information Extraction
- ▶ Information Filtering
- ▶ Wrapper Induction

Recommended Reading

1. Crescenzi V., Mecca G., and Merialdo P. RoadRunner: towards automatic data extraction from large web sites. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 109–118.
2. Debnath S., Mitra P., and Giles C.L. Automatic extraction of informative blocks from webpages. In Proc. ACM Symp. on Applied Computing, 2005, pp. 1722–1726.
3. Glance N., Hurst M., Nigam K., Siegler M., Stockton R., and Tomokiyo T. Deriving marketing intelligence from online discussion. In Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2005, pp. 419–428.
4. Hofmann K. and Weerkamp W. Web corpus cleaning using content and structure. In Building and Exploring Web Corpora, C. Fairon, H. Naerts, A. Kilgarrif, and G. de Schryver (eds.). vol. 4, UCL, 2007, pp. 145–154.
5. Kovacevic M., Dilligenti M., Gori M., and Milutinovic V. Recognition of common areas in a web page using a visualization approach. In Proc. 10th Int. Conf. on Artificial Intelligence: Methodology, Systems, and Applications, 2002, pp. 203–212.
6. Kushmerick N., Weld D., and Doorenbos R. Wrapper induction for information extraction. In Proc. 15th Int. Joint Conf. on AI, 1997, pp. 119–128.
7. Lin S.H. and Ho J.M. Discovering informative content blocks from web documents. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 588–593.
8. Liu B., Grossman R., and Zhai Y. Mining data records in web pages. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 601–606.
9. Muslea I., Minton S., and Knoblock C. Hierarchical wrapper induction for semistructured information sources. Auton. Agent. Multi Agent Syst., 4(1–2):93–114, 2001.
10. Simon K. and Lausen G. ViPER: augmenting automatic information extraction with visual perceptions. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 381–388.
11. Ziegler C.N. and Skubacz M. Towards automated reputation and brand monitoring on the web. In Proc. IEEE/WIC/ACM Int. Conf. on Web Intelligence, 2006, pp. 1066–1070.
12. Ziegler C.N. and Skubacz M. Content extraction from news pages using particle swarm optimization on an linguistic and structural features. In Proc. IEEE/WIC/ACM Int. Conf. on Web Intelligence, 2007, pp. 242–249.
13. Zhao H., Meng W., Wu Z., Raghavan V., and Yu C. Fully automatic wrapper generation for search engines. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 66–75.

Fully Temporal Relation

▶ Bitemporal Relation

Functional Data Model

PETER M. D. GRAY
University of Aberdeen, Aberdeen, UK

Definition

Functional Data Models are a form of *Semantic Data Model* which appeared early in database history. They use the mathematical formalism of *function application*

to represent and follow associations between data items. Functions are usually applied to variables whose values may be object identifiers or record instances. Thus if P represents an entity instance of type Person, then *forename(P)* would return a string (e.g., "Peter"). (Note however that different systems may write this in a LISP style as (forename P) or in JAVA style as P.forename.) The function *town(P)* could be used to represent an *association* by returning the identifier for Peter's home town. This allows *function composition* so that *name(town(P)) = P.town.name = "Aberdeen"*.

Using functions gives several obvious advantages. Firstly the syntax is universally understood, and provides a means of expressing schemas, queries and constraints independently of any supplier-dependent language. This is very handy for integrating data held in heterogeneous databases – one of the earliest applications [11]. Secondly, functional expressions follow the rule of *Referential transparency* – expressions of equal value can be substituted for variables without altering the sense or value of the expression. This avoids the problems of side-effects in nested *procedure calls* found in programming languages. In consequence, optimizing functional expressions is much easier, as is parallel evaluation. In particular, the whole mathematical theory of *Comprehensions* worked out for functional programming can be applied to *Functional Query Languages*.

Historical Background

Functions provided an underlying formalism for data models from as early as Abrial's *access functions* for representing binary relationships between entities [1] and Florentin's *property functions* for representing the attributes of entities [4].

Kerschberg and Pacheco's *Functional Model of Data* [8] integrated these two uses for functions, modeling the universe of discourse by means of *entity sets* and *total functions*. Entity sets are represented by labeled nodes and functions by labeled arcs in a directed graph.

The main motivation for the Functional Model of Data was ease of conceptual modeling, and [8] showed how functional data models can be automatically transformed into relational and CODASYL data models for implementation within a DBMS.

The Functional Model of Data was further developed by Sibley and Kerschberg in [14]. In this development, the universe of discourse is again represented by labeled nodes and arcs. The arcs are again total functions, but the nodes are either entity sets or *value sets* (as opposed to a single set of character strings, C). A major motivation for this work was to derive a general, unifying conceptual data model which could then be specialized into either relational or CODASYL data models.

IS-EMPLOYEE	HAS-SALARY		
emp-id	name	emp-id	salary
Paul	"Paul"	Paul	19,000
Mary	"Mary"	Mary	22,000
John	"John"	John	19,000
...	...	...	...

At about the same time, Hammer and McLeod proposed the Semantic Data Model [7], as a higher-level model better suited to conceptual modelling of an application domain. A key innovation of the Semantic Data Model was the recognition of the importance of entity attributes based on *derived information*. Shipman used functions to implement a Semantic Data Model representing both data and derived information in the very influential DAPLEX language [13]. It was used to integrate a heterogeneous database network MULTIBASE [11].

Following this came fore-runners of *Object-Oriented Databases*. The GENESIS query language [2] supported a functional data model and was intended as a front end for DBMSs supporting either a relational or nested relational data model. The PROBE database system [3] supported the representation and manipulation of arbitrarily complex objects by means of a functional data model.

Foundations

The earliest functional data models were based on binary relational data, where every relational table had only two columns. The name of the table, considered as a verb relating the two columns, became the function name. One column was chosen to be the argument and the other to be the result. Where possible, this is done to make the function single-valued, as usual.

Consider two such tables representing facts such as

```
Paul is-employee ''Paul''
Mary is-employee ''Mary''
```

```
...
Paul has-salary 19000
Mary has-salary 22000
```

where each row refers to an individual employee and the right-hand column entries for that employee give their name and current salary respectively. Note that Paul and Mary are surrogates for object-identifiers, in order to make the example more readable.

A functional view could specify this information as the functions

```
is-employee :: emp-id -> name
has-salary :: emp-id -> salary
```

Where the relationship is one-to-many, as for example with

```
Paul has-child Jane
Paul has-child Sue
Paul has-child Norman
Mary has-child James
```

Then the function must return a *set of* emp-id. Thus *children*(*Paul*) = {*Jane*,*Sue*,*Norman*} and *children*(*Mary*) = {*James*}. Some values will be empty sets e.g., *children*(*John*) ={}. Historically, only garbage-collected languages (such as LISP and FP and Prolog) could deal with such variable-length lists, and they did not suit analysts used to the conventions of fixed-length tables in Codd's normal form! It was only gradually that people realized that it was perfectly straightforward to hold data in relational tables and to use the Functional Data Model as a high-level *View* on the same data. With the coming of Java, one did not even have to use a special purpose list processing language in order to handle or print such values. Likewise, although some implementations stored facts in a specially designed triple store [9], it was not essential to do so. Interestingly, such storage techniques are becoming favored again for storing RDF triples of web data (see below).

Note that the functional approach is specifying information in smaller units, that is with a finer semantic granularity than with a single n-ary table. Thus further information can readily be added, such as the function:

```
line-manager :: emp-id -> emp-id
```

A powerful argument for the functional approach, in addition to its greater flexibility, is that it facilitates the incremental development of systems where the schema evolves as more data is collected. Note also the convenience with which one can ask who is the line manager of a particular employee's line manager, by contrast with the SQL approach:

```
line-manager (line-manager(Paul))
```

instead of

```
SELECT L2.line-manager FROM line-manager L1 L2
WHERE L1.emp-id = Paul AND L2.emp-id = L1.line-manager
```

In the DAPLEX language, new abstract entities are created by means of the directive A NEW and values of base functions are assigned by means of the directive LET. For example, to create a new Student of name "Fred Jones" who attends the Biology and Biochemistry courses:

```
FOR A NEW Student
  BEGIN
    LET Name(Student) = "Fred Jones"
LET Attends(Student) =
     {THE CourseOfName("Biology"),
THE CourseOfName("Biochemistry")}
END
```

Note here the use of CourseOf Name(Biology) instead of just "Biology" as in a relational database. This is because variables in DAPLEX can denote actual *object identifiers* (as in Java, for example) whereas the analogous values in relational database columns will be *foreign key* values in order to identify the object. This is a big difference from the relational model and is common to most functional data models.

Semantic Web Vision and RDFS

The *semantic web* vision is to enable rich machine processing of web information sources. RDF stands for Resource Description FrameWork Model, which was first accepted by W3C as a Data Model described in *RDF Schema* in February 1999. (http://www.w3.org/TR/REC-rdf-syntax) A data interchange format is defined using XML syntax with tags starting < *rdf :* to encode *subject–predicate–object triples*. The *predicate* is just a function name, for example `has-salary`.

Thus, RDF is not unlike the Entity-Relational data model in its use of Entity identifiers as *subject*, and Property or Relationship names as *predicate* in RDF triples. However, it also includes features of object data models in its use of object identifiers and subclasses. This makes it very similar to the Functional Data Model.

The comparison below is illuminating and shows how schemas can be mapped between the two models.

Mapping a Functional Model to RDFS

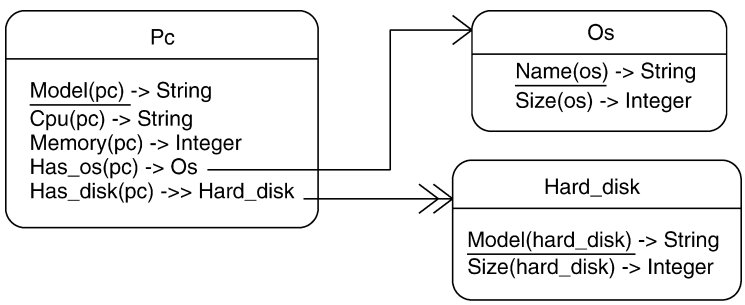
The RDFS Data Model abstracts over relational storage, flat files and object-oriented storage, following the principle of data independence. Thus, it shares with the Functional Data Model the advantage of not tieing one to any particular storage system. This is a great advantage to the programmer. The mapping to a particular knowledge source or data source can then take place separately through a wrapper. This makes it very much easier to integrate data from different sources, as is often required over the Web.

Consider the following example [5], where the functional data model schema of Fig. 1 is used to describe the `pc` and `os` classes and their relationship `has_os` in an application domain where components are put together to configure a workable PC.

The Functional Data Model is, of course, an extended ER model and it can be automatically mapped into an RDFS specification. A mapping program reads metadata from the database and *generates* the corresponding RDFS, as in Fig. 2, making this knowledge web-accessible. Related work by Risch [12] also shows how RDFS resources can be integrated and accessed by a functional query language. The basic rules used when mapping the schema declarations to RDFS are as follows:

- A class *c* defined as an `entity` (declared as *c* `->> entity`) maps to an RDF resource of type `rdfs:Class` (where `rdfs` is the namespace prefix for the RDFS descriptions).
- A class *c* declared to be a subtype of another class *s* (declared as *c* `->>`*s*) maps to an RDF resource of type `rdfs:Class`, with an `rdfs:subClassOf` property the value of which is the class named *s*.
- A function *f* declared on entities of class *c*, with result type *r* (declared as *f*(*c*) `->`*r*) maps to an RDF resource of type `rdf:Property` with an `rdfs:domain` of *c* and an `rdfs:range` of *r*.

Mapping a functional schema into RDFS has the advantage of making the domain model available to



Functional Data Model. Figure 1. This functional schema shows three entity classes. The single arrow means that each `pc` may have only one `os` installed. A double arrow means that a `pc` can have multiple `hard-disk`.

```
<rdfs : Class rdf : ID= "pc">
    <rdfs : subClassOf rdf : resource = "#Resource"/>
</rdfs : Class>

<rdfs : Class rdf : ID = "os">
    <rdfs : subClassOf rdf : resource = "#Resource"/>
</rdfs : Class>

<rdf : Description rdf : ID = "has_os">
    <rdf : type rdf : Property/>
    <rdfs : domain rdf : resource = "#pc"/>
    <rdfs : range rdf : resource = "#os"/>
</rdf : Description>
```

Functional Data Model. Figure 2. RDFS (RDF Schema) representation of schema of Fig. 1.

RDFS-ready software. Some semantic information is lost, because the cardinality of each attribute is not expressed in RDFS. Also information on the *key* of each entity class is omitted. However, this information could be represented by an extra metadata class declared in RDFS.

Constraints

Integrity Constraints are a very important part of data modelling. Sometimes this is just considered as part of type checking or range checks on values, but in a *Semantic Data Model* one can express constraints that represent semantics applying to data in a particular domain, that may not be obvious from a casual inspection of tables, or even from applying Data Mining techniques. For example, in DAPLEX one can enforce that all Students attend at least four courses:

```
DEFINE CONSTRAINT Number_of_courses
(Student) =>
COUNT(Attends(Student)) > 3
```

Comprehensions in *P/FDM* are also used to describe the semantics of *Integrity Constraints*, representing invariants that must be held true under updates. For this purpose the nested loop syntax used in queries is adapted, similarly to constraints in EFDM [10]:

```
constrain each t in seniortutor
each s in advisees(t) to have grade(s) > 60;
```

This constrains each person in the class `seniortutor` to have only advisees with grades over 60. Both universal(each) and existential(some) quantifiers are allowed, in any combination. Such integrity constraints are particularly important in Functional Data Models, since they provide a way to *extend the data model* with rich semantics defined with the full power of a Comprehension (like range-restricted FOL). By contrast, SQL and relational languages are often restricted to just providing range checks on data items.

Updating Functional Data

In DAPLEX, the value of a multi-valued function for a particular argument can be modified by using the built-in operators INCLUDE and EXCLUDE. For example, to modify one of the courses attended by Fred Jones from BioChemistry to Physiology:

```
FOR THE Student SUCH THAT Name
(Student) = "Fred Jones"
    BEGIN
      EXCLUDE Attends(Student) = THE
CourseOfName("BioChemistry")
      INCLUDE Attends(Student) = THE
CourseOfName("Physiology")
    END
```

It is immediately clear that the *updating* of functions defining a functional database contradicts the basic assumption of referential transparency. In a functional database, the essence of a function is conveyed by its name and its type, with which is associated its real-world semantics, and not by its current mapping, which will change as a result of database updates. The function `has-salary` specifying a person's salary as used earlier is a case in point.

The partial loss of referential transparency due to updates does not, however, alter the other considerable advantages of functional programs: that they represent a high-level, non-procedural but executable specification of what is required, and enable programs to be created in a top-down fashion with local detail encapsulated within the specification of the functions to which these details relate. Moreover a functional program that does not involve updates but accesses a database in read-only mode will be referentially transparent. Even with updates there is what might be termed local referential transparency between such updates; thus some advantages can nonetheless be gained [15].

Key Applications

The integration of data held in heterogeneous databases continues to be a very challenging problem. At one time people thought that all data could be forced into relational storage through the dominance of SQL. However, the emergence of semi-structured data, much of it held on the Web, has changed all that.

Future Directions

Functional data models have the right abstractions for dealing with heterogeneous data, but they need to be packaged for more convenient use with Web data. Interestingly, functional languages such as Python and Jython have a secure following among scientific users and others wanting to advance beyond Visual Basic. Once again, if they could be combined with a functional data modelling package that provides a view over data stored in different kinds of databases then the idea could very likely catch on.

URL to Code

http://www.csd.abdn.ac.uk/~pgray/FDMdownload.html http://user.it.uuse/~udbl/amos/download.html http://www.dcs.bbk.ac.uk/~ap/pfl/html

Cross-references

- ▶ Comprehensions (IN Functional Query Languages)
- ▶ Functional Query Languages
- ▶ Introductory article of [6], from where much of this is taken

Recommended Reading

1. Abrial J.R. Data Semantics. In Data Base Management. Klimbie. J.W. and K.L. Koffeman (eds.). North Holland, 1974.
2. Batory D.S., Leung T.Y., and Wise T.E. Implementation concepts for an extensible data model and data language. ACM Trans. Database Syst., 13(3):231–262, 1988.
3. Dayal U. et al. Simplifying complex objects: the PROBE approach to modelling and querying them. In Proc. Workshop on the Theory and Applications of Nested Relations and Complex Objects, 1987, pp. 17–37.
4. Florentin J.J. Consistency auditing of databases. Computer J., 17 (1):52–28, 1974.
5. Gray P.M.D., Embury S.M., Hui K.Y., and Kemp G.J.L. The evolving role of constraints in the functional data model. J. Intell. Inf. Syst., 12:113–137, 1999.
6. Gray P.M.D., Kerschberg L., King P.J.H., and Poulovassilis A., (eds.). The Functional Approach to Data Management. Springer, Berlin Heidelberg New York, 2004.
7. Hammer M.M. and McLeod D.J. The Semantic Data Model: a modelling mechanism for database applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1978, pp. 26–35.
8. Kerschberg L. and Pacheco J.E.S. A Functional Data Base Model. Technical Report 2/76, Departmento de Informatica, Pontificia Universidade de Sao Vincente, Rio de Janeiro, 1976.
9. King P.J.H., Derakhshan M., Poulovassilis A., and Small C. TriStarp – an investigation into the Implementation and Exploitation of Binary Relational Storage Structures. In Proc. British National Conf. on Databases, 1990, pp. 64–84.
10. Kulkarni K.G. and Atkinson M.P. EFDM: Extended Functional Data Model. Computer J., 29(1):38–46, 1986.
11. Landers T. and Rosenberg R.L. An overview of Multibase. Distributed Databases, H.-J. Schneider (ed.). North Holland, 1982, pp. 153–184.
12. Risch T. Functional queries to wrapped educational semantic Web meta-data. In The Functional Approach to Data Management, Chap. 19. P.M.D. Gray, L. Kerschberg, P.J.H. King, and A. Poulovassilis, (eds.). Springer, 2004.
13. Shipman D.W. The functional data model and the data language DAPLEX. ACM Trans. Database Syst., 6(1):140–173, 1981.
14. Sibley E.H. and Kerschberg L. Data architecture and data model considerations. In Proc. AFIPS National Computer Con., 1977, pp. 85–96.
15. Sutton D.R. and Small C. Extending functional database languages to update completeness. In Proc. British National Conf. on Databases, 1995, pp. 47–63.

Functional Dependencies for Semi-Structured Data

Gillian Dobbie[1], Tok Wang Ling[2]
[1]University of Auckland, New Zealand
[2]National University of Singapore, Sigapore, Singapore

Synonyms

Path functional dependencies; Extended functional dependencies

Definition

Functional dependencies are used in relational database design to show that the value of a set of attributes depends on the value of another set of attributes. Theory has been developed to manipulate a set of functional dependencies to describe equivalences of sets of functional dependencies. Semi-structured data differs from relational data in two important ways: semi-structured data is hierarchical and the structure of the data is less consistent. Traditional functional dependencies do not capture these differences so new functional dependencies with associated theory has been defined for semi-structured data.

Key Points

Functional dependencies for semi-structured data have been defined in the three recommended readings. While the syntax of functional dependencies defined over semi-structured data varies, the semantics is similar. In this article the syntax of Arenas and Libkin [1] is used but the notation differs a little since there is no distinction between attributes and subelements. There is a notion of an XML tree and a path in an XML tree. Consider the XML document in Fig. 1a, and the XML tree for that document in Fig. 1b. The internal nodes of an XML tree are labelled with tag-name:node-ID, e.g., the node label *student:v5* means this node represents an element *student* with node ID *v5*.

A path through this XML tree is something like *department.course.code*. A tree tuple is a subtree containing at most one occurence of each path. Note that there are two occurences of paths such as *department.course.student* in Fig. 1a, so there are two tree tuples, one including the paths for the student with student number *123456* and the other including the paths for the student with student number *234567*. The tree

tuples will be called t_1 and t_2 respectively. Below it is shown how the tree tuples t_1 and t_2 assign values to the paths in the XML tree in Fig. 1b:

The symbol, S, is a reserved symbol, and for a path $p.S$, if $t(p) = v$ then $t(p.S)$ is the value of the element or attribute at v. A functional dependency is an expression of the form $X \rightarrow Y$ where both X and Y are sets of paths. An XML tree satisfies a functional dependency $X \rightarrow Y$ if for any two tree tuples t_1 and t_2 in the tree $t_1(X) = t_2(X)$ then $t_1(Y) = t_2(Y)$. Note that $t_1(X) = t_2(X)$ means that $t_1(p) = t_2(p)$ for all $p \in X$. Using functional dependencies, key constraints, dependencies among attributes, and dependencies among object classes can be expressed.

The key constraint, that no two course elements will have the same code, is expressed as:

department.course.code.S → *department.course*

If there were another course in the document shown in Fig. 1a with code "*CS101*", then this functional dependency would no longer hold.

A functional dependency is used to show that there is only one record of a student taking a course. The functional dependency represents that two student subelements of the same course cannot have the same student number. This constraint is represented as:

{*department.course*, *department.course.student.stuNo.S*} → *department.course.student*

In contrast, the following functional dependency expresses that no two student elements will have the same student number:

{*department.course.student.stuNo.S*} → *department.course.student*

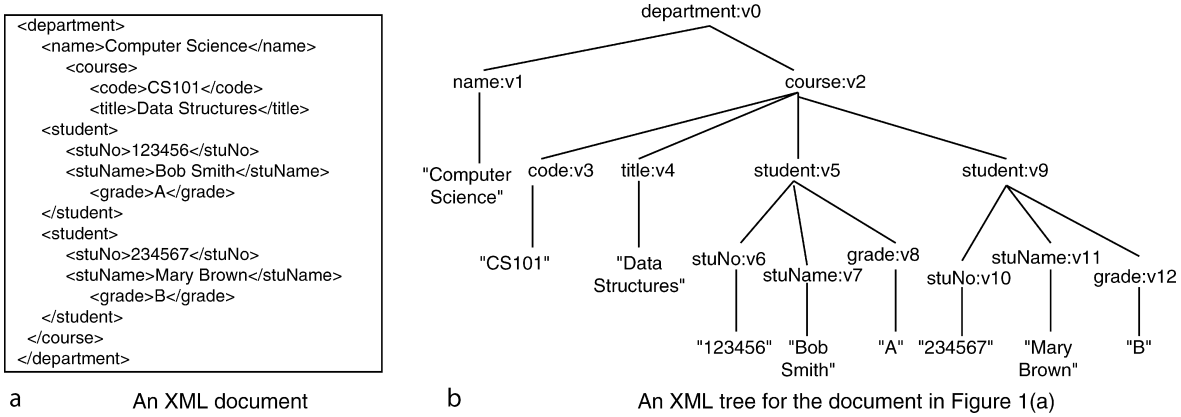
The constraint, that two student elements with the same value for *stuNo* must have the same value for *stuName*, is written:

department.course.student.stuNo.S → *department.course.student.stuName.S*

Note that this definition of functional dependencies combines node and value equality. This allows functional dependencies to be used to express not

t_1 (*department*) = υ0
t_1 (*department.name*) = υ1
t_1 (*department.name.S*) = "*Computer Science*"
t_1 (*department.course*) = υ2
t_1 (*department.course.code*) = υ3
t_1 (*department.course.code.S*) = "*CS*101"
t_1 (*department.course.title*) = υ4
t_1 (*department.course.title.S*) = "*Data Structures*"
t_1 (*department.course.student*) = υ5
t_1 (*department.course.student.stuNo*) = υ6
t_1 (*department.course.student.stuNo.S*) = "123456"
t_1 (*department.course.student.stuName*) = υ7
t_1 (*department.course.student.stuName.S*) = "*Bob Smith*"
t_1 (*department.course.student.grade*) = υ8
t_1 (*department.course.student.grade.S*) = "*A*"

t_2 (*department*) = υ0
t_2 (*department.name*) = υ1
t_2 (*department.name.S*) = "*Computer Science*"
t_2 (*department.course*) = υ2
t_2 (*department.course.code*) = υ3
t_2 (*department.course.code.S*) = "*CS*101"
t_2 (*department.course.title*) = υ4
t_2 (*department.course.title.S*) = "*Data Structures*"
t_2 (*department.course.student*) = υ9
t_2 (*department.course.student.stuNo*) = υ10
t_2 (*department.course.student.stuNo.S*) = "234567"
t_2 (*department.course.student.stuName*) = υ11
t_2 (*department.course.student.stuName.S*) = "*Mary Brown*"
t_2 (*department.course.student.grade*) = υ12
t_2 (*department.course.student.grade.S*) = "*B*"



Functional Dependencies for Semi-Structured Data. Figure 1. XML document and its corresponding XML tree.

only what the keys are but also to differentiate between absolute and relative constraints. That is, all students in a document have a unique student number can be expressed using an absolute constraint, and all students in a course have a unique student number can be expressed using a relative constraint. Note that the relative constraint is relative to another element or object class, in this case *course*, while the absolute constraint applies to the whole document. However, the richness of expressibility also means that functional dependencies in the semi-structured setting are more complicated than functional dependencies in the relational setting.

Cross-references

► Semi-structured Database Design

Recommended Reading

1. Arenas M. and Libkin L. A normal form for XML documents. ACM Trans. Database Syst., 29(1):195–232, 2004.
2. Lee M.L., Ling T.W., and Low W.L. Designing functional dependencies for XML. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 124–141.
3. Wu X., Ling T.W., Lee S.Y., Lee M.L., and Dobbie G. NF-SS: a normal form for semistructured schema. In Proceedings of ER Workshops LNCS, Vol. 2465/2002. 2001, pp. 292–305.

Functional Dependency

Solmaz Kolahi
University of British Columbia, Vancouver, BC, Canada

Synonyms

FD

Definition

Given a relation schema $R[U]$, a *functional dependency (FD)* is an expression of the form $X \rightarrow Y$, where $X, Y \subseteq U$. An instance I of $R[U]$ satisfies $X \rightarrow Y$, denoted by $I \models X \rightarrow Y$, if for every two tuples t_1, t_2 in I, $t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$. That is, whenever two tuples contain the same values for attributes in X, they must have the same values for attributes in Y. A functional dependency $X \rightarrow Y$ is called *trivial* if $Y \subseteq X$.

A *key dependency* is a functional dependency of the form $X \rightarrow U$. Then X is called a *superkey* for relation R.

Movies			
Title	*Director*	*Actor*	*Year*
The Godfather	Francis F. Coppola	Marlon Brando	1972
The Godfather	Francis F. Coppola	Al Pacino	1972
The Godfather	Francis F. Coppola	James Caan	1972
The Shining	Stanley Kubrick	Jack Nicholson	1980
The Shining	Stanley Kubrick	Shelley Duvall	1980

If there is no proper subset Y of X such that Y is a superkey, then X is called a *key*.

Key Points

Functional dependencies form an important class of integrity constraints that play a critical role in maintaining the integrity of data, query optimization and indexing, and especially schema design. The focus of the *normalization* technique in schema design is on avoiding redundancies caused by functional and other types of dependencies in relational databases. An example of such redundancies can be seen in the *year* column of the following table, where the functional dependency *title* → *year* holds.

In database design theory, there are normal forms that put restrictive conditions on data dependencies to control this kind of redundancy. Well-known normal forms that deal with FDs are second normal form (2NF), third normal form (3NF), and Boyce-Codd Normal Form (BCNF).

The implication problem for FDs can be solved in linear time. That is, given a set Σ of FDs, it is possible to check whether an FD $X \rightarrow Y$ is logically implied by Σ, denoted by $\Sigma \models X \rightarrow Y$, in the time that is linear in the size of Σ and $X \rightarrow Y$. A key concept for solving the implication problem is the *closure* of a set of attributes X, which is the set of attributes $A \in U$, denoted by X^+, such that $\Sigma \models X \rightarrow A$. There are efficient algorithms for finding the closure of a set of attributes [1].

The implication problem of functional dependencies can also be axiomatized. That is, there is a finite sound and complete set of inference rules, called *Armstrong axioms*, that can be used to solve the implication problem:

Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$.
Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$.
Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Cross-references

- ▶ Boyce-Codd Normal Form (BCNF)
- ▶ Key
- ▶ Normal Forms and Normalization
- ▶ Second Normal Form (2NF)
- ▶ Third Normal Form (3NF)

Recommended Reading

1. Abiteboul S., Hull R., Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, USA, 1995.

Functional Query Language

Peter M. D. Gray
University of Aberdeen, Aberdeen, UK

Definition

Functional Query Languages came from two lines of development:

1. From new functional programming languages such as FP that showed the value of *referential transparency*; this ensures that complex nested functional expressions can be evaluated inside-out (bottom-up) or top-down or even split up and done in parallel, with the same result. For optimization purposes this is vastly better than state-altering algorithms used in early CODASYL systems, or even code used today with embedded SQL (as in ODBC).

2. From requirements to provide a single query language and a single integrated schema over multiple autonomous, heterogeneous, distributed databases. This happened in the MULTIBASE project and resulted in the DAPLEX language [10]. It was the first functional query language to compute over instances of a *Functional Data Model*, for the purpose of abstracting away details of different storage schemas in a distributed DB.

Historical Background

No full implementation of DAPLEX was undertaken. However, later systems Adaplex, EFDM and *P/FDM* implemented large parts of the language. These showed how to integrate base functions (from the data model), derived functions, views, and integrity constraints into a single functional framework. The main drawback of DAPLEX was that for general computation it required either calling out to foreign functions, or embedding in a host programming language.

ADAPLEX was an embedding of a subset of DAPLEX into the programming language ADA. The data types of DAPLEX are reconciled with those of ADA by associating base functions with abstract entity types when these are declared. Also, derived functions are defined procedurally in ADA rather than in DAPLEX (this is an example of calling out to foreign functions).

EFDM [8] extended DAPLEX with procedural computation, recursive functions over abstract types and scalar types, built-in metalevel functions for interrogating the meta data, and a construct which can store EFDM statements in the database (like stored SQL procedures). The underlying database used persistent object storage techniques, similar to commercial object-oriented databases. The facilities for semantic integrity constraints considerably extend those of DAPLEX.

Following *FQL*, The functional database languages FDL [9] and PFL concentrated on a more general, computationally complete, functional language based on the lambda calculus.

This work was followed by the development of several functional languages, such as FAD [1] and FUGUE, for complex object and *object-oriented databases*. These capitalized on the fact that DAPLEX already had a built-in notion of *object identifier* for *entity* instances in its data model, but without committing to any particular implementation. This also inspired the Iris object-oriented database system, which used a LISP-like functional syntax. Iris in turned influenced the development of the original *AMOSQL* language used in active mediation of distributed data sources [5].

The OSQL query language [2] was intended for use with object-oriented databases. It unified functional and relational modeling by extending SQL to include entities and functions. It included primitives for creation of entities, for assignment of base function values, and for updating multi-valued base functions. Derived functions were defined by means of a SELECT clause and queries had an SQL-like syntax.

Foundations

One common thread running through almost all Functional Query Languages is the use of the functional programming concept of a *Comprehension* (q.v.) which deserves to be much more widely known. Buneman et al. [3] have generalized it for database use to include set and bag comprehensions. Fegaras [6] has related it to the algebraic structure of monoids which underlies *OQL*. This section makes general use of the

notion of a Comprehension, because it describes precisely almost all the computations done in functional query languages, despite their very different surface syntax!

The comprehension crosses the borders between the lambda calculus and the predicate calculus since functions, just like predicates, can be used either as generators or as filters in collecting up sets of values. However, what matters is the mathematical theory worked out in connection with functional programming, which can be used, for example, to develop and prove correct new analysis and optimization techniques for queries. Generators may just be based on finite sets (or subranges) of integers, and filters can also do calculations. For example, suppose one wants the set of all right-angled triangles with whole-number sides less than 50. The comprehension for this neatly expresses the mathematical requirements:

```
[(x,y,z) | x <- [1..50]; y <- [1..50];
z <- isqrt(x*x + y*y); z*z = x*x + y*y;
z <50]
```

Here for each value of `x` between 1 and 50, one must explore all `y` values between 1 and 50, generate a value for the longest side `z` using a function to calculate the integer part of the square root, and then test that `z` is less than 50. The results are returned in the form of triples, such as `(3,4,5)`. This is a typical use of functions in numerical calculation. It also introduces the subject of *nested loops*. Notice that where one generator appears to the right of another, then its values have to be considered in combination with all values from the previous one, which is just like using nested loops in an ordinary programming language.

Now consider a query involving nested generators, which corresponds to the use of joins in SQL, but which appears in the OQL object database query language [4] as:

```
select x.name
from x in students, y in x.takes, z in y.
taught_by
where z.rank = "full prof"
```

This gives the names of students who take courses taught by full professors. The equivalent comprehension is

```
[name(x) | x <- students; y <- takes(x);
z <- taught_by(y);rank(z) = "full
prof"]
```

In a programming language supporting assignment and `for` loops, one would write the computation using nested loops, which show more clearly the role of the generators:

```
result := [];
for x in students()
  for y in takes(x)
    for z in taught_by(y)
      if rank(z) = "full prof"
      then result := result ++ [name(x)];
```

Note that the variables are now treated as holding object identifiers. More significantly, each variable is only introduced once with an arrow, and this must come before it is used in a predicate or as a parameter of another generator. Apart from this, generators and predicates can be reordered without altering the value of the comprehension, since it behaves like a conjunction of booleans.

This reordering is made easier by using *converse functions* of the form `f_inv` (equivalent to `Conv f` used in Functional Programming); just like an *inverted index*, this may enable one to move selections nearer to generators, so as to gain efficiency. For example, the following comprehension produces the same result more quickly by applying the filter on full professor much earlier:

```
[name(x) | z <- lecturers; rank(z) =
"full prof";
  y <- taught_by_inv(z); x <- takes_inv
  (y)]
```

This technique is widely used. Note that it is not necessary for the end users themselves to use `f_inv`; this need only happen within the query optimizer module of the DBMS.

In order to show that list comprehensions can be more complex, consider another query which returns the codes of all courses that have some section taught by a senior lecturer:

```
[code(c) | c <- course;
some([t | s <- sections(c); t <-
lecturer(s);
  p <- position(t); p="SL"]) ]
```

Here, the inner comprehension depends on the variable `c` representing a course, which is bound by the outer comprehension. The function `some` is a predicate that tests whether its parameter (the inner comprehension) returns a non-empty list. It corresponds

to the EXISTS predicate in SQL. In fact, the whole query can be translated into SQL automatically.

The following subsections show how comprehensions are used in a number of different query languages. Despite variations in syntax and in emphasis, all these systems make use of the same mathematical theory for their transformation and optimization.

Comprehensions in P/FDM and Constraints

P/FDM, like the examples in *AMOSQL* [5], uses list comprehensions internally but provides an equivalent query language syntax which suits regular programmers. In P/FDM the above comprehension is rendered as

```
for each c in course such that
  some t in lecturer(sections(c)) has
position(t)="SL"
    print(code(c));
```

This syntax is very close to Shipman's original DAPLEX language.

Comprehensions in P/FDM are also used to describe the semantics of integrity constraints, representing invariants that must be held true under updates. For this purpose the nested loop syntax used in queries is adapted, similarly to constraints in EFDM:

```
constrain each t in seniortutor
  each s in advisees(t)
    to have grade(s) > 60;
```

This constrains each person in the class `seniortutor` to have only advisees with grades over 60. It requires that the following list comprehension always computes an empty list:

```
[ t | t <- seniortutor; s <- advisees
(t); not (grade(s) > 60)]
```

The constraint is equivalent to the following formula in predicate logic, which shows the correspondence between the generators and the nested quantifiers and also the similarity in use of conjunctions. This demonstrates a fascinating connection between comprehensions and the predicate calculus:

$$(\forall t)\ seniortutor(t) \Rightarrow ((\forall s,g)\ advisee(t,s) \wedge grade(s,g) \Rightarrow g > 60)$$

Comprehensions in Kleisli – Records, Lists and Mixed Types

In Kleisli [11], comprehensions are written with a comma as separator for conjunctions in place of a semicolon. Also, the first time that a variable iterates over a set of values (in some generator), its name needs to be prefixed with a backslash. Subsequent uses of the unprefixed variable, either in filters or in generators, make use of these bindings. Lastly, record field names are prefixed by a `#`. Thus, the initial example would read:

```
{p.#surname | \p <- person, \f <- p.
#forename, f = "Jim"}
```

If one wishes to create a *list* comprehension or *bag* comprehension, one simply encloses the above expression in different brackets `[ ... | ...]` or `{| ... | ...|}` respectively.

Thus the end user basically works with comprehensions, while the full power of functional programming is reserved for the implementers, working in Standard ML. This is an elegant use of functional programming but note that it is not essential to use a functional language for optimising comprehensions; for example, P/FDM uses Prolog while AMOS uses a version of LISP. Any good list processing language will do but functional languages are, of course, better at type checking.

Comprehensions in FDL and PFL

The functional database languages FDL [9] and PFL incorporate list comprehensions as part of a more general, computationally complete, functional language based on the lambda calculus. In such languages, comprehensions can be formalised as successive applications of a higher order function `flatmap`. Thus, for example, the following comprehension from earlier in this section:

```
[surname(p) | p <- person; f <- forename
(p); f = "Jim"]
```

would translate into the following expression:

```
flatmap (lambda p.
  flatmap (lambda f.if f = "Jim"
      then [surname(p)]
      else [])
    forename(p))
  person
```

`flatmap` is also an example of a *parametric polymorphism*, in that its type can be inferred to be `(a->[b])->[a]->[b]`, where `a` and `b` are *type variables* each of which can be replaced by *any* type. For

example, the first occurrence of `flatmap` above has type `(Person->[String])->[Person]->[String]` while the second occurrence has type `(String->[String])->[String]->[String]`.

PFL has a similar type system to FDL, but uses a class of functions called *selectors* which allow storage, querying and update of sets of values of the same type.

Key Applications

The fundamental advantage of list comprehensions as a database abstraction is that they preserve *Data Independence* in a very clean and simple way. They do it by working entirely in terms of sets and functional relationships, regardless of how they are stored. This will seem strange to programmers who are used to carefully choosing between arrays of records or parallel arrays or linked lists or B-trees etc.

Database people know that large collections of data may exist in different forms on different computers, and may need to change form by restructuring on a single computer. Thus it is necessary to do the translation from a list comprehension expressed against a conceptual schema into a specific storage schema at compile time (often close to run time). Likewise it may be necessary to send part of a query to a remote server, since data is increasingly distributed in different forms. Thus the list comprehension is a good choice for passing a complex computational request between computers, free from assumptions about data storage or whether functions are computed or based on stored values.

Future Directions

About a decade ago, Stonebraker was predicting that SQL would become *Galactic Dataspeak*. With the widespread adoption of JDBC and PHP to access MySQL or much bigger commercial SQL databases, this would appear to be so. However, on the Internet there is increasing pressure to provide functionality as *Web Services* which can work on various encapsulated forms of structured and semi-structured data. Functional Query languages may well find a niche as convenient scripting languages for implementing such Web services. They allow a much richer variety of computation and can easily evolve to cope with new storage schemas, because they adhere to the Principle of Data Independence. As such, they should prove much easier to maintain in the fast-changing world of the Web. The functional language *Python* and its JAVA version *Jython* are increasingly popular, and have an explicit syntax for *Comprehensions*.

URL to Code

http://www.csd.abdn.ac.uk/~pgray/FDMDownload.html
http://user.it.uu.se/~udbl/amos/download.html
http://www.dcs.bbk.ac.uk/~ap/pfl.html

Cross-references

- ▶ AMOSQL
- ▶ FQL
- ▶ Functional Data Model
- ▶ OQL
- ▶ P/FDM

For a general overview on the Functional Approach to modeling, integrating,querying and analyzing data see [7] from where much of this section is taken.

Recommended Reading

1. Bancilhon F., Briggs T., Khoshafian S., and Valduriez P. FAD, a powerful and simple database language. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 97–105.
2. Beech D. A foundation of evolution from relational to object databases. In Advances in Database Technology, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp. 251–270.
3. Buneman P., Libkin L., Suciu D., Tannen V., and Wong L. Comprehension syntax. ACM SIGMOD Rec., 23(1):87–96,1994.
4. Cattell R.G.G. (ed). The Object Data Standard: ODMG 3.0. Morgan Kaufmann, Los Altos, CA, 2000.
5. Fahl G., Risch T., and Sköld M. AMOS – an architecture for active mediators. In Proc. Workshop on Next Generation Information Technologies and Systems, 1993, pp. 47–53.
6. Fegaras L. and Maier D. Towards an effective calculus for Object Query Languages. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 47–58.
7. Gray P.M.D., Kerschberg L., King P.J.H., and Poulovassilis A. The Functional Approach to Data Management. Springer, Berlin, 2004.
8. Kulkarni K.G. and Atkinson M.P. EFDM : Extended Functional Data Model. Comput. J., 29(1):38–46, 1986.
9. Poulovassilis A. and King P.J.H. Extending the functional data model to computational completeness. In Advances in Database Technology, Proc. 2nd Int. Conf. on Extending Database Technology, 1990, pp. 75–91.
10. Shipman D.W. The functional data model and the data language DAPLEX. ACM Trans. Database Syst., 6(1):140 –173, 1981.
11. Wong L. Kleisli, a functional query system. J. Funct. Program., 10:19–56, 2000.

Fuzzy Information Retrieval

- ▶ Fuzzy Models

Fuzzy MCDM

► Fuzzy Set Approach

Fuzzy Models

Gabriella Pasi
University of Milano-Bicocca, Milan, Italy

Synonyms

Fuzzy information retrieval

Definition

The application of Fuzzy Set Theory to Information Retrieval is aimed to the definition of retrieval techniques capable of modeling, at least to some extent, the subjectivity, vagueness and imprecision that is intrinsic to the process of locating information relevant to some user's needs. In particular, Fuzzy Set Theory has been applied in the context of IR to the following main aims:

- To define generalizations of the Boolean retrieval model
- To deal with the imprecision and subjectivity that characterize the document indexing process
- To manage the user's vagueness in query formulation
- To soften the associative mechanisms, such as thesauri and documents' clustering algorithms, which are often employed to extend the functionalities of the basic IR scheme
- To define flexible aggregation strategies in meta search engines and to define flexible approaches to distributed IR
- To represent and inquiry semi-structured information (XML documents)

Historical Background

In 1979 one of the first proposals of application of Fuzzy Set Theory to IR was presented by Tadeusz Radecki [7]. His seminal paper, titled "Fuzzy set theoretical approach to document retrieval" constituted a first generalization of the Boolean retrieval model, followed by several subsequent proposals.

Fuzzy Set Theory was defined as a generalization of classical Set Theory, on which the Boolean information retrieval model is based [8]. A fuzzy set is a class of elements with unsharp boundaries suitable to represent vague concepts. Formally, a fuzzy subset A of a universe of discourse U is defined through a membership function μ_A: U → [0,1]; the value 1 indicates full membership of an element of U to A, the value 0 no membership, and a value between 0 and 1 partial membership. A finite fuzzy subset A of a set U is denoted by A = $\{\mu_A(u)/u\}$ with u∈U, in which the notation $\mu_A(u)/u$ indicates that with each element u of the universe U a membership degree $\mu_A(u)$ (in [0,1]) is associated, which denotes the membership of u to the subset.

The main aim of fuzzy generalizations of the Boolean IR model was to overcome its limitations, and to model relevance as a gradual property of documents with respect to a user's query, with the consequence of making an Information Retrieval System able to produce a document ranking [1,4,6]. In a fuzzy IR model a document is formally represented as a fuzzy subset of index terms (the membership value associated with a term represents its index term weight), and the Boolean query language can be extended by allowing the association of weights with query terms. In a generalized Boolean query, each term weight is interpreted as a specification of term importance, and it is formally defined as a flexible constraint (that is a constraint satisfied up to a partial extent) on the document representation. For example, a query term weight may be interpreted as a threshold, i.e., as a constraint satisfied by the documents having the indexing weight higher than the specified query term weight [1,4]. The exact matching of the Boolean model is in this way relaxed to a partial matching in fuzzy IR models, where the matching degree of a document to a query (the so called Retrieval Status Value of a document) is computed on the basis of an evaluation of the satisfaction of the flexible constraints specified in the query (by weighted terms).

Since Radecki's seminal paper, several new applications of Fuzzy Set Theory to IR have been proposed; some of them are described in the following section.

Foundations

In this section a more extensive explanation of Fuzzy IR models is presented, related in particular to the definition of flexible query languages (the key point of the above mentioned fuzzy generalizations of the Boolean model). Some other key applications of Fuzzy Set Theory to IR are also presented in distinct subsections.

Definition of Generalizations of the Boolean Query Language

By means of Fuzzy Set Theory two main generalizations of the Boolean query language have been proposed: the introduction of query term weights, and a generalization of the aggregation operators (the connectives AND and OR in the Boolean query language).

The first fuzzy models proposed in the literature introduced query term weights, by allowing the association of a weight with each query term. In this way the basic constraint specified in a query is a weighted term, identified by a pair <term, weight>. Query term weights express the "importance" of query terms as descriptors of users' needs, and are interpreted as flexible constraints on the index terms weights in each document representation. By such an extension, the structure of a Boolean query is maintained, by allowing weighted query terms to be aggregated by the AND, OR connectives and negated by the NOT operator. In the first fuzzy models query term weights were defined as numeric values in the range [0,1]. A numeric query term weight identifies a flexible constraint on the weighted document representations; such a constraint depends on the semantics of the query weight. Distinct semantics have been proposed for query weights, corresponding to distinct fuzzy generalizations of the Boolean model (distinct fuzzy IR models) [1,4,6]. The three main semantics for query term weights are: the relative importance semantics (query weights express the relative importance of pairs of terms in a query), the threshold semantics (a query weight expresses a threshold on index term weights), and the ideal index term weight semantics (a query weight expresses the "perfect" index term weight).

The choice of one out of the three proposed query weight semantics implies a distinct definition of the partial matching function which evaluates a weighted query term. A flexible constraint imposed by a weighted query term <t, *weight*> is formally defined as a fuzzy subset of the set [0,1] of the index term weights. For a given document d, the membership value $\mu_{weight}(F(d,t))$ is interpreted as the degree of satisfaction of the flexible constraint imposed by the *weight* associated with query term t by the index term weight of t in document d (the value F(d,t)). This means that the partial evaluation function is the membership function of the fuzzy subset identified by the query term weight (i.e., the function μ_{weight}). The result of the evaluation of a weighted query term is a fuzzy set: $\{\mu_{weight}(F(d,t))/d)\}$, in which the membership degree of a document is interpreted as the degree of relevance of the document to the query (the Retrieval Status Value), and is used to rank the documents.

In structured Boolean queries, the evaluation of the AND and OR connectives is applied to the fuzzy subsets returned by the weighted query terms evaluations.

In the context of Fuzzy Set Theory the connectives AND and OR are defined as aggregation operators belonging to the class of T-norms and T-conorms respectively. Usually, the AND is defined as the *min* (minimum) aggregation operator, and the OR as the *max* (maximum) aggregation operator. A Boolean expression on weighted query terms is usually evaluated by a bottom-up evaluation procedure (except in the case of relative importance semantics): first, each atomic selection condition (flexible constraint. i.e., each weighted term) in the query is evaluated for a given document, and then the aggregation operators are applied to the obtained values starting from the inmost operator in the query to the outermost operator.

As the association of a numeric value forces the user to quantify the qualitative concept of importance of query terms, some late models proposed in the literature have formalized linguistic extensions of the Boolean query language, based on the concept of linguistic variable [1,4]. The values of a linguistic variable are linguistic terms formally defined as fuzzy subsets of a given reference domain; for example in the case one wants to define *age* as a linguistic variable, some possible linguistic values can be *young*, *old*, formally defined as fuzzy subsets of the set [0,130] of the possible numeric values for the age of a human being. By using linguistic query weights, query terms can then be labeled by the words "*important*," "*very important*" or "*fairly important*." Similarly to the evaluation of numeric query term weights, a pair <t, *important*> expresses a flexible constraint evaluated by the function $\mu_{important}$ on the index term weights (the F(d,t) values). The evaluation of the relevance of a given document d to a query consisting of the pair <t, *important*> is then computed by applying the function $\mu_{important}$ to the value F(d,t). Also in this case the membership function of the fuzzy set associated with a linguistic weight depends on the weight semantics.

A second kind of generalization of the Boolean query language has concerned the definition of aggregation operators (AND and OR in the Boolean query language) [1,4,6]. When employing the Boolean retrieval model, if the AND is used for aggregating *M*

keywords in a user query, a document indexed by all keywords but one is not retrieved, thus causing the possible rejection of useful items. The opposite behavior characterizes the aggregation by OR. To express more flexible aggregations, the use of linguistic quantifiers (formally defined within Fuzzy Set Theory) has been proposed. Linguistic quantifiers, such as *at least 2* and *most*, specify more flexible document selection strategies. Linguistic quantifiers have been formally defined as averaging aggregation operators, the behavior of which lies between the behavior of the AND and the OR connectives, which correspond to the *all* and the *at least one* linguistic quantifiers.

Another recent research direction is aimed at the definition of flexible query languages to inquire XML documents. Fuzzy set theory is being applied to define extensions of XML query languages so as to make possible the expression of flexible selection conditions on both the documents' structure and contents.

Flexible Indexing of Semi-Structured Documents

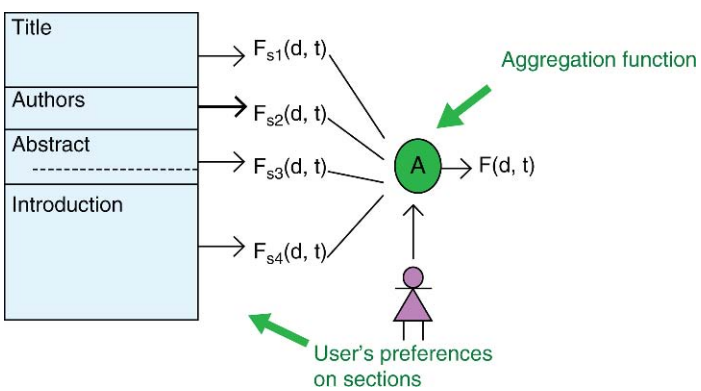
The diffusion of semi-structured documents has encouraged the definition of indexing models which take into account the information conveyed by the "structure" of the documents.

The usual tf*idf indexing schema adopted by IRSs does not take into account the distinct informative role that a term occurrence may have in distinct document sections. For example, considering the structure of a scientific paper, usually organized in sections such as *title, authors, abstract, introduction, references*, an occurrence of a term in the *title* has a distinct informative role than its occurrence in the *references* section. Moreover the usual indexing functions produce the same document representation to all users; this enhances the system's efficiency but implies a loss of effectiveness. In fact, when examining a structured document, users have their personal views of the document's information content. Users would naturally privilege the search in some subparts of the documents' structure, depending on their preferences. This observation has supported the idea of flexible and personalized indexing, first proposed in 1995 [2]. Such an indexing model is constituted by a static component and by an adaptive query-evaluation component; the static component provides an a priori computation of an index term weight for each logical section of the document.

The adaptive component may be activated by a user interaction during query formulation and provides an aggregation strategy of the n index term weights (where n is the number of sections) into an overall index term weight. The user is allowed to express preferences on the document sections, outlining those that the system should more heavily take into account in computing the overall index term weight. This user preference on the document structure is exploited to enhance the computation of index term weights: the importance of index terms is strictly related to the importance to the user of the logical sections in which they appear. The user may also decide which kind of aggregation to apply for producing the overall significance degree (see Fig. 1). This can be done by the specification of a linguistic quantifier such as *at least k* (with k an integer number) and *most*.

By adopting such an indexing model a same query may produce different document rankings if formulated by distinct users expressing distinct preferences on the documents sections.

More recently, an increasing number of approaches have proposed IR models which are based on concepts rather than keywords, thus modeling document representations at a higher level of granularity, trying to



Fuzzy Models. Figure 1. Sketch of the flexible indexing model.

describe the topical content and structure of documents. These efforts gave raise to the so called concept-based Information Retrieval, which aims at retrieving relevant documents on the basis of their meaning rather than their keywords. In this context some fuzzy set approaches to concept-based Information Retrieval have been proposed.

Fuzzy Associative Mechanisms

Associative retrieval mechanisms are defined to enhance the retrieval capability of traditional IRSs. They work by retrieving additional documents not directly indexed by the terms in a given query but indexed by terms associated to those specified in the query. The most common type of associative retrieval mechanism is based on the use of a thesaurus to associate entry terms with related terms. In traditional associative retrieval the associations are crisp.

The fuzzy associative retrieval mechanisms (first proposed in 1976) are based on the concept of fuzzy association [1,4,5]. A fuzzy association between two sets $X = \{x_1,...,x_m\}$ and $Y = \{y_1,...,y_n\}$ is formally defined as a fuzzy relation $f: X \times Y \rightarrow [0,1]$: the value $f(x, y)$ represents the degree of strength of the association existing between the values $x \in X$ and $y \in Y$.

In Information Retrieval, different kinds of fuzzy associations can be modeled depending on the semantics of the sets X and Y. Fuzzy associative mechanisms are represented by fuzzy thesauri, fuzzy pseudo-thesauri, and fuzzy clustering techniques.

Some authors have proposed the definition of fuzzy thesauri, where the links between terms are weighted to indicate strength of association. Moreover, this notion includes generalizations such as fuzzy pseudo-thesauri, and fuzzy associations based on a citation index [5].

Fuzzy associative mechanisms based on thesauri or clustering techniques have been defined in order to cope with the incompleteness characterizing either the representation of documents or the users' queries. Fuzzy thesauri and pseudo-thesauri can be used to expand the set of index terms of documents with new terms by taking into account their varying significance in representing the topics dealt with in the documents; the degree of significance of the associated terms depends on the strength of the associations with the documents' descriptors. An alternative use of fuzzy thesauri and pseudo-thesauri is to expand each of the search terms in the query with associated terms, by taking into account their distinct importance in representing the concepts of interest; the varying importance is dependent on the associations' strength with the search terms.

Fuzzy Approaches to Distributed Information Retrieval

In distributed information retrieval, there are two main models. In the first model, the information is considered as belonging to a unique, huge, centralized database which is distributed but centrally indexed for retrieval purposes. A second model is based on the distribution of the information on distinct repositories, independently indexed, and thus constituting distinct information sources. In this second case, the repositories reside on distinct servers, each of which can be provided with its own search engine (IRS).

The multi-source information retrieval paradigm is more complex than the centralized model. This paradigm presents additional problems, such as the selection of an appropriate information source for a given information need. This task of distributing retrieval is affected by uncertainty, since a decision must be taken based on an incomplete description of the information source. Furthermore, a common problem in both models is the list fusion task.

Some fuzzy methods have been defined to address the above mentioned problems of source selection and ranked list fusion [3]. In particular, a meta-search model has been recently proposed where the fusion of overlapping ordered lists into an overall ordered list is regarded as a group decision making activity in which the search engines play the role of the experts, the documents are the alternatives that are evaluated based on a set of criteria expressed in a user query, and the decision function is a soft aggregation operator which allows to take into account the lists priority, allowing to model a specific user retrieval attitude [3].

Key Applications

The main applications of Fuzzy Models in IR are aimed at defining new Information Retrieval Systems, to enhance query languages, to define new indexing algorithms, and to define associative mechanisms.

Cross-references

▶ Digital Libraries
▶ Text Indexing Techniques
▶ Text Retrieval

Recommended Reading

1. Bordogna G. and Pasi G. Modelling vagueness in information retrieval. In Lectures in Information Retrieval, M. Agosti, F. Crestani, G. Pasi (eds.). Springer, Berlin, 2001.
2. Bordogna G. and Pasi G. Personalized indexing and retrieval of heterogeneous structured documents. Inf. Retrieval, 8 (2):301–318, 2005.
3. Bordogna G., Pasi G., and Yager R.R. Soft approaches to distributed information retrieval. Int. J. Approx. Reasoning, 34 (2–3):105–120, 2003.
4. Kraft D.H., Bordogna G., and Pasi G. Fuzzy set techniques in information retrieval. In Fuzzy Sets in Approximate Reasoning and Information Systems, Series: The Handbooks of Fuzzy Sets Series. J.C. Bezdek, D. Dubois, H. Prade (eds.). Kluwer Academic, Norwell, MA, 1999, pp. 469–510.
5. Miyamoto S. Fuzzy Sets in Information Retrieval and Cluster Analysis. Kluwer Academic, Dordrecht, 1990.
6. Pasi G. Fuzzy Sets in Information Retrieval: State of the Art and Research trends. In Fuzzy Sets and Their Extensions: Representation, Aggregation and Models. Intelligent Systems from Decision Making to Data Mining, Web Intelligence and Computer Vision, Series: Studies in Fuzziness and Soft Computing. H. Bustince, F. Herrera, and J. Montero (eds.). Springer, Berlin, 2008, pp. 517–535.
7. Radecki T. Fuzzy set theoretical approach to document retrieval. Inf. Process. Manag., 15(5):247–260, 1979.
8. Zadeh L. Fuzzy Sets. Inf. Control, 8:338–353, 1965.

Fuzzy Multicriteria Decision Making

► Fuzzy Set Approach

Fuzzy Relation

Vilém Novák
University of Ostrava, Ostrava, Czech Republic

Definition

An n-ary fuzzy relation R is a fuzzy set (see FUZZY SET) in the universe $U_1 \times ... \times U_n$, i.e., $R \subsetq U_1 \times ... \times U_n$. A special case is a binary fuzzy relation in the universe $U \times V$, i.e., $R \subsetq U \times V$.

Key Points

Mathematically, an n-ary fuzzy relation is a function R: $U_1 \times ... \times U_n \rightarrow L$ where L is a residuated lattice.

Let $R \subsetq U \times V$ and $S \subsetq V \times W$ be two binary fuzzy relations. The *composition* of R and S is a fuzzy relation $R \circ S$ determined by the membership function

$$(R \circ S)(x,z) = \bigvee_{y \in V} (R(x,y) \wedge S(y,z)), \quad x \in U, z \in W. \tag{1}$$

The operation $\wedge$ in ((1)) can be replaced by arbitrary t-norm.

Cross-references

► Fuzzy Set
► Residuated Lattice
► Triangular Norms
► t-Norm

Recommended Reading

1. Klement E.P., Mesiar R., and Pap E. Triangular Norms. Kluwer, Dordrecht, 2000.
2. Klir G.J. and Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice-Hall, New York, 1995.
3. Novák V. Fuzzy Sets and Their Applications. Adam Hilger, Bristol, 1989.
4. Novák V., Perfilieva I., and Močkoř J. Mathematical Principles of Fuzzy Logic. Kluwer, Boston/Dordrecht, 1999.

Fuzzy Set

Vilém Novák
University of Ostrava, Ostrava, Czech Republic

Definition

A fuzzy set is a function $A : U \rightarrow L$ where U is an ordinary set of elements called *universe* and L is a scale which is usually supposed to have the structure of a residuated lattice (see RESIDUATED LATTICE). The function A is at the same time called also *membership function*, i.e., fuzzy set is identified with its membership function.

If $x \in U$ is an element then $A(x) \in L$ is a *membership degree* of x in the fuzzy set A. It can also be interpreted as a *degree of truth* of the fact that the element x belongs to the fuzzy set A. If U is a universe and A a fuzzy set then it is convenient to write $A \subsetq U$.

Key Points

Fuzzy sets can be explicitly written in the form

$$\{a/u \mid a \in L, u \in U\}$$

where $a \in L$ is a membership degree of an element $u \in U$.

Given a residuated lattice (see RESIDUATED LATTICE) as the structure of truth values, the basic operations with fuzzy sets can be defined as follows:

$$\begin{aligned} &\text{(union)} && (A \cup B)(x) = A(x) \vee B(x),\\ &\text{(intersection)} && (A \cap B)(x) = A(x) \wedge B(x),\\ &\text{(complement)} && \overline{A}(x) = \neg A(x) \end{aligned}$$

for all $x \in U$ where $\neg$ is a negation in residuated lattice. If the latter is a standard Łukasiewicz MV-algebra then

$$\overline{A}(x) = 1 - A(x).$$

Many other possible operations with fuzzy sets can be defined on the basis of operations in residuated lattice. In case that $L = [0,1]$, the union can be more generally defined using a t-conorm and the intersection using a t-norm.

Note that $\langle \{0,1\}, \vee, \wedge, \wedge, \rightarrow, 0, 1\rangle$ is also a residuated lattice because in this case $\otimes = \wedge$ is the ordinary conjunction and $\rightarrow$ is the classical boolean (material) implication. Therefore, the above operations naturally generalize classical set operations.

The *support* of a fuzzy set $A \subsetneq U$ is a classical set $\text{Supp}(A) = \{x \mid A(x) > 0\}$. The *kernel* of a fuzzy set is a set $\text{Ker}(A) = \{x \mid A(x) = 1\}$. A fuzzy set A is *normal* if $\text{Ker}(A) \neq \emptyset$.

A fuzzy set $A \subseteq \mathbb{R}$ is called a *fuzzy number* if it has one element kernel $\text{Ker}(A) = \{x_0\}$, bounded support, and $A(x) \geq A(y) \wedge A(z)$ holds for all $y \leq x \leq z$. Such a fuzzy set can thus be taken as interpretation of the expression "approximately x_0."

The above notions can be illustrated on a simple example. Let the universe be a finite set $U = \{a, b, c, d, e\}$ and let L be the standard Łukasiewicz MV-algebra $\langle [0, 1], \max, \min, \otimes, \rightarrow, 0, 1\rangle$. (see RESIDUATED LATTICE) Let

$$\begin{aligned} A &= \{{}^{0.1}/_{a}, {}^{1}/_{b}, {}^{0.8}/_{c}\},\\ B &= \{{}^{0.7}/_{b}, {}^{0.3}/_{c}, {}^{0.9}/_{d}, {}^{0.2}/_{e}\} \end{aligned}$$

(The omitted elements have membership degree equal to 0.). Then

$$\begin{aligned} A \cap B &= \{{}^{0.7}/_{b}, {}^{0.3}/_{c}\},\\ A \cap B &= \{{}^{0.1}/_{a}, {}^{1}/_{b}, {}^{0.8}/_{c}, {}^{0.9}/_{d}, {}^{0.2}/_{e}\},\\ \overline{A}(x) &= \{{}^{0.9}/_{a}, {}^{0.2}/_{c}, {}^{1}/_{d}, {}^{1}/_{e}\},\\ \text{Supp}(A) &= \{a, b, c\},\\ \text{Ker}(A) &= \{b\}. \end{aligned}$$

Example of other operation can be bold intersection

$$A \otimes B = \{{}^{0.7}/_{b}, {}^{0.1}/_{c}\}$$

using a Łukasiewicz product (this is also a t-norm) where, e.g., $(A \otimes B)(c) = \max\{0, 0.8 + 0.3 - 1\} = 0.1$.

Cross-references

- ▶ Fuzzy Relation
- ▶ Residuation
- ▶ Residuated Lattice
- ▶ Triangular Norms

Recommended Reading

1. Klir G.J. and Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice-Hall, New York, 1995.
2. Novák V. Fuzzy Sets and Their Applications. Adam Hilger, Bristol, 1989.
3. Novák V., Perfilieva I., and Močkoř J. Mathematical Principles of Fuzzy Logic. Kluwer, Boston/Dordrecht, 1999.

Fuzzy Set Approach

Vilém Novák
University of Ostrava, Ostrava, Czech Republic

Synonyms

Fuzzy MCDM; Fuzzy multicriteria decision making

Definition

The classical multicriteria decision making theory is based on the assumption that all the criteria can be characterized precisely so that it is possible to decide unambiguously, whether each alternative fulfils the given criterion or not. However, this is rarely the case in practice and so, the fuzzy set approach has been proposed which makes it possible to assume that the criteria can be evaluated imprecisely, for example "high quality, low reliability, very low weight," etc. Unlike classical approach which first dissolves imprecision and then constructs a decision model, the fuzzy set approach dissolves imprecision only at the very end, if necessary.

The basic concepts of fuzzy decision making are the following:

1. Decision based on the imprecisely defined set of alternatives, i.e., a fuzzy set of alternatives. This is called *decision in a fuzzy environment.*

2. Decision based on aggregation of imprecisely defined (fuzzy) preferences among alternatives.
3. Decision based on evaluation of alternatives using linguistic description of the decision situation.

The linguistic description in Case 3 consists of a set of fuzzy/linguistic IF-THEN rules, i.e., special expressions of the form

$$\text{IF } X \text{ is } \mathcal{A} \text{ THEN } Y \text{ is } \mathcal{B}$$

where X, Y are criteria and $\mathcal{A}, \mathcal{B}$ are imprecise characterizations (for example, evaluative expressions of natural language) of how well the given alternative meets the criteria. The linguistic expressions are often interpreted using fuzzy sets in various ways. The most advanced possibility takes fuzzy IF-THEN rules as conditional sentences of natural language.

A problem related to decision making is *classification*. The main task is to assign the given alternative to one of several classes. In practice, this may be quite difficult since the classes themselves can also be imprecise. For example, shoe numbers represent imprecise classes of foot sizes. The most advanced fuzzy classification methods are based on application of fuzzy IF-THEN rules.

Historical Background

The first paper on fuzzy multicriteria decision making has been written by Bellman and Zadeh in 1970 [1]. Their idea was very simple but at the same time quite powerful and it gave inspiration to many authors of the subsequent papers on fuzzy decision making. The paper started research leading to various kinds of approaches that improved the proposed method on one side and brought a lot of other new ideas and methods on the other side. After period of spontaneous development that lasted until mid of 1980s, the first systematic works appeared, namely the books [4,8].

A significant step forwards is the book by J. Fodor and M. Roubens [6] which is the first sound introduction to valued preference modeling using systematic use of fuzzy set theory and functional equations. Since then, many other papers and books have been published, e.g., [3,13]. Notable is the book by T. J. Saaty [11] which describes the, so called, *Analytic Hierarchy Process* that now belongs to one of the most widely used decision methods in practice.

The number of works on fuzzy multicriteria decision making now counts to thousands. There is also a working group EUROFUSE which regularly organizes various kinds of meetings or co-organizes conferences.

Foundations

Decision making in a fuzzy environment is realized as follows: Let a finite set of alternatives $A = \{a_1,...,a_n\}$ be given. The goal is to choose the best alternative according to m criteria that, however, may be delineated only imprecisely, for example *reasonable price, nice view, great reliability, high safety*, etc.

The criteria are specified by means of defining a fuzzy set of alternatives that fulfil the given criterion in various degrees. Hence, each criterion G_j, $j = 1,...,m$ is identified with a fuzzy set $G_j \subsetneq A$ of alternatives fulfilling it. The membership degree $G_j(a_i)$ expresses the degree, in which the given alternative $a_i \in A$ fulfils the criterion G_j.

At the same time also constraints must be specified with respect to each criterion, for example *affordable price, view to the forest, acceptable reliability, acceptable safety*, etc. Similarly as the criteria, the constraints are also specified as fuzzy sets of alternatives. This means that each constraint C_j is defined as a fuzzy set $C_j \subsetneq A$ of alternatives fulfilling C_j in various degrees. Consequently, for each j, $j = 1,...,m$, two fuzzy sets of alternatives are specified: a fuzzy set G_j representing criterion and a fuzzy set C_j representing constraint.

For example, the criterion G_j can be *reasonable price* while the constraint C_j can be *affordable price*. The latter means that not each reasonable price can be for the decision-maker at the same time also affordable. If a_i is a certain house that a decision-maker wants to buy then the membership degree $G_j(a_i)$ represents the degree in which the price of a_i is reasonable (e.g., $G_j(a_i) = 0.8$ means that it is in 80% true that the price of a_i is reasonable). Quite analogously, the membership degree $C_j(a_i)$ is a degree in which the price of a_i is affordable for the decision-maker. For example, the set A of houses (alternatives) may contain also a castle a_k at a very reasonable price – several millions of dollars – but this price can be completely non-affordable for the decision-maker because he/she has, say, less than one million at disposal. In this case, set $C_j(a_k) = 0$.

The final decision is a fuzzy set of alternatives obtained by composition

$$D = (G_1 \alpha C_1) \beta ... \beta (G_m \alpha C_m) \qquad (1)$$

where α, β are suitable fuzzy set operations. The best alternative is an alternative $a_k \in A$ with the highest membership degree $D(a_k)$.

Setting $\alpha = \beta = \cap$ (intersection) leads to *pessimistic decision*. With respect to the above example, $G_j(a_j) \wedge C_j(a_j)$ is a minimum of the degrees in which the house a_j has a reasonable and affordable price, i.e., the worse of both. Hence, for the castle a_k it is immediately obtained that $G_j(a_k) \wedge C_j(a_k) = 0$ and so $D(a_k) = 0$ which means that the castle is surely not chosen as the best alternative.

Setting $\beta = \cup$ (union) leads to *optimistic decision*. The best, however, seems compensatory decision which combines both intersection as well as union in various degrees. More details can be found in [19,14].

It is also possible to omit constraints from (1) and to consider the criteria $G_1,...,G_m$ only. Then, the problem is raised how all the membership degrees $G_j(a_i)$ should be aggregated to obtain the global satisfaction degree $D(a_i)$ of the given alternative a_i. This requires the use of a specific *aggregation operator* $\mathcal{H}^m$ so that

$$D(a_i) = \mathcal{H}^m(G_1(a_i),...,G_m(a_i)).$$

The symbol $\mathcal{H}^m$ denotes actual number of arguments of $\mathcal{H}$ (their number can vary).

Basic properties of aggregation operators are the following:

$$\begin{aligned}
&\mathcal{H}^1(a) = a,\\
&\mathcal{H}^m(0,...,0) = 0 \text{ and } \mathcal{H}^m(1,...,1) = 1,\\
&\text{if}(a_1,...,a_m) \le (b_1,...,b_m) \text{ then}\\
&\quad \mathcal{H}^m(a_1,...,a_m) \le \mathcal{H}^m(b_1,...,b_m).
\end{aligned}$$

Many other conditions can be imposed on aggregation operators, for example continuity, symmetry, associativity, idempotency, compensation, etc. The possible operators are classified as *conjunctive operators* (e.g., t-norms), *disjunctive operators* (e.g., t-conorms), *mean operators* (e.g., arithmetic mean), or more specific ones, such as *compensative operators* or *order weighted averaging operators* (OWA operators). Many details can be found in [2,7,13].

The criteria, however, are usually not equally important. Their relative importance is expressed using weights that can be numbers assigned to each criterion or/and constraint. The most popular method for weights assignment is *Analytical Hierarchy Process* (AHP) that itself can be used as a specific fuzzy decision method. It involves structuring multiple choice criteria into a hierarchy, assessing the relative importance of these criteria, comparing alternatives for each criterion, and determining an overall ranking of the alternatives. Thus, it can be divided into four phases:

1. *Decomposing*, i.e., the problem is structured into humanly-manageable sub-problems.
2. *Weighing*, i.e., a relative weight is assigned to each criterion, based on its importance within the node to which it belongs. A global priority is computed that quantifies the relative importance of a criterion within the overall decision model.
3. *Evaluating*, i.e., alternatives are scored and compared each one to others.
4. *Selecting*, i.e., an alternative fitting best the requirements is selected.

For the details about AHP, see [11].

Imprecisely defined preference is mathematically modeled using a binary *fuzzy preference relation* $R \subsetneq A \times A$. This is determined by triple of binary fuzzy relations $\langle P, I, J\rangle$ where $P \subsetneq A \times A$ is a strict fuzzy preference, $I \subsetneq A \times A$ is a fuzzy indifference, and $J \subsetneq A \times A$ is a fuzzy incomparability. This means, that $P(a_i, a_j)$ is a degree, in which the alternative a_i is strictly preferred to a_j. Similarly, $I(a_i, a_j)$ is the degree in which a_i is indifferent to a_j, and $J(a_i, a_j)$ is the degree in which a_i is incomparable with a_j. In other words, the given relation need not hold in full but only partially. This corresponds well with the real situations in which it needs not be fully convincing that, for example, "high building" is strictly nicer than "low" one, etc.

The definition of P, I, J and their further properties depend on the choice of the structure of truth values which is a specific residuated lattice. The solution is not unique and depends on other conditions that can be imposed on these relations. One possible solution is to assume that truth values form the standard Łukasiewicz MV-algebra and to put

$$P(a_i, a_j) = R(a_i, a_j) \wedge \neg R(a_j, a_i), \tag{2}$$

$$I(a_i, a_j) = R(a_i, a_j) \otimes R(a_j, a_i), \tag{3}$$

$$J(a_i, a_j) = \neg R(a_i, a_j) \otimes \neg R(a_j, a_i) \tag{4}$$

where $\otimes$ is the Łukasiewicz conjunction $a \otimes b = 0 \vee (a + b - 1)$. The detailed analysis and justification can be found in [6,13].

A very powerful general technique suitable also for applications in decision making as well as in classification are fuzzy/linguistic IF-THEN rules. The given decision situation is described using linguistic description and then, each alternative is judged using it on the basis of the measured characteristics for each criterion. The linguistic form of fuzzy/linguistic IF-THEN rules makes it possible to distinguish sufficiently subtly and, at the same time, aptly, various degrees of fulfilment of the respective criteria, their various importance and, moreover, it may also overcome possible discrepancies. Hence, the problem of assignment of weights to the criteria disappears. This makes fuzzy/linguistic IF-THEN rules very attractive for decision-making because the problem of weights assignment belongs to the most controversial problem in its applications. Another advantage is the possibility to include also information that can be quantified with great difficulties. For example "aesthetic quality," "overall impression," etc. are criteria which people can evaluate using expressions of natural language such as "very high, quite low, great, medium," etc. Such expressions (they are called *evaluative linguistic expressions*), however, can be used in fuzzy/linguistic IF-THEN rules without problems.

The procedure for decision support using fuzzy/linguistic IF-THEN rules is analogous to the AHP procedure mentioned above with the exception that the weighing phase is omitted. The first phase consists in decomposition of the decision problem into subproblems, each of which being characterized by a specific linguistic description. Hence, the decision situation is described using a hierarchical system of linguistic descriptions which, at the same time, renders evaluation:

$$\mathcal{R}_{1k}: \text{IF } G_{k1} \text{ is } \mathcal{A}_{11} \text{ AND } \ldots \text{ AND } G_{kn(k)} \text{ is } \mathcal{A}_{1n(k)} \text{ THEN } H_k \text{ is } \mathcal{B}_1$$

$$\ldots\ldots\ldots\ldots$$

$$\mathcal{R}_{1p(k)}: \text{IF } G_{k1} \text{ is } \mathcal{A}_{p(k)1} \text{ AND } \ldots \text{ AND } G_{kn(k)} \text{ is } \mathcal{A}_{p(k)n(k)} \text{ THEN } H_k \text{ is } \mathcal{B}_{p(k)}$$

$k = 1,\ldots,r$ and

$$\mathcal{R}_1: \text{IF } H_1 \text{ is } \mathcal{A}_{11} \text{ AND } \ldots \text{ AND } H_r \text{ is } \mathcal{A}_{1r} \text{ THEN } H \text{ is } \mathcal{B}_1$$

$$\ldots\ldots\ldots\ldots$$

$$\mathcal{R}_s: \text{IF } H_1 \text{ is } \mathcal{A}_{s1} \text{ AND } \ldots \text{ AND } H_r \text{ is } \mathcal{A}_{sr} \text{ THEN } H \text{ is } \mathcal{B}_s$$

where $H_1,\ldots,H_r$ are local evaluations, H is a global evaluation and $\mathcal{A}$'s and $\mathcal{B}$'s are evaluative linguistic expressions. The final decision is made on the basis of the values of the global evaluation H (usually, the higher, the better). A typical example of the above rules is, for example, "IF price is small AND maintenance is more or less medium THEN economical conditions are good," or "IF house is stylish AND place is very nice THEN aesthetic quality is significantly high," etc. On the basis of initial information about each alternative for all aspects, the inference proceeds. The most convenient inference method at this step is *perception-based logical deduction*. Note that other methods can be used as well but one must be careful about defining the shapes of fuzzy sets and so, the advantage of using natural language is limited. The best alternative is selected on the basis of the highest (or, possibly, the lowest) value of the global evaluation H.

A specific task belonging also to the realm of decision making is *classification*. In general this is a procedure that assigns an element from a finite set $\Omega = \{\omega_1,\ldots,\omega_c\}$ to a vector $\mathbf{x} \in S$, $S \subset \mathbb{R}$ of numbers. The set S is a *feature space* and Ω is a set of classes that can be, e.g., pieces of a figure, psychological categories, shoe numbers, diseases, etc. The need for fuzzy classification is raised in the moment when there is not a sufficient or precise information available or the character of classes is imprecise so that membership in them can be unclear and only some degree can be provided, etc. Such situations are very usual in the real life.

A specific case are fuzzy IF-THEN classifiers that are linguistic descriptions with clearly distinguished consequences. The latter can be either crisp or fuzzy numbers. Further elaboration is the same as above.

Key Applications

There are several thousands of real applications of fuzzy multicriteria decision making. One of the earliest of them was evaluation of the credit-worthiness of credit applicants developed in Germany (see [14]).

Other essential application is Yamaichi Fuzzy Fund which handles 65 industries and a majority of the stocks listed on Nikkei Dow. It is based on the application of fuzzy IF-THEN rules which are determined monthly by a group of experts and modified by senior business analysts when necessary. The system was tested for 2 years, and its performance in terms of the return and growth exceeds the Nikkei Avarage by

over 20%. For comparison, the system recommended "sell" 18 days before the Black Monday in 1987. The system went to commercial operations in 1988.

Some other ones are, evaluation of weapon systems, technology transfer strategy selection in biotechnology, aggregation of market research data and thousands others. Evaluation of weapon systems has been provided using the Analytical Hierarchy Process (AHP) based on fuzzy scales.

Typical features of the weapon system evaluation were the following: the objectives of the evaluations are generally multiple and generally in conflict, and the descriptions of the weapon systems are usually linguistic and vague. The details are in [5]. It should be stressed that such features are typical for most practical decision problems and this is the main reason for using fuzzy set theory. In case of linguistically provided information, especially fuzzy IF-THEN rules taken as conditional statements of natural language are convenient tool since they make possible to include also non-quantifiable information. Such application was described in [10].

Cross-references

- ▶ Classification
- ▶ Classification by Association Rule Analysis
- ▶ Clustering
- ▶ Decision Tree Classification
- ▶ Decision Trees
- ▶ Fuzzy Models
- ▶ Hierarchial Clustering
- ▶ Rule-Based Classification

Recommended Reading

1. Bellman R. and Zadeh L.A. Decision making in a fuzzy environment. Manage. Sci., 17:140–164, 1970.
2. Calvo T., Mayor G., and Mesiar R. (eds.). Aggregation Operators: New Trends and Applications, Physica-Verlag, Heidelberg 2002.
3. Carlsson C. and Fuller R. Fuzzy Reasoning in Decision Making and Optimization. Springer, Berlin, 2002.
4. Chen S.J. and Hwang C.L. Fuzzy multiple attribute decision-making, methods and applications. Lecture Notes in Economics and Mathematical Systems, Springer, Heildelberg, 1993.
5. Cheng C.H. and Mon D.-L. Evaluating weapon system by analitical hierarchy process based on fuzzy scales. Fuzzy Sets Syst., 63:1–10, 1994.
6. Fodor J. and Roubens M. Fuzzy Preference Modelling and Multicriteria Decision Support. Kluwer Academic, Dordrecht, 1994.
7. Grabisch M., Nguyen H., and Walker E. Fundamentals of Uncertainty Calculi, with Applications to Fuzzy Inference. Kluwer Academic, Dordrecht, 1995.
8. Kacprzyk J. and Yager R.R. Management Decision Support Systems Using Fuzzy Sets and Possibility Theory. Springer, Berlin, 1985.
9. Novák V. Fuzzy Sets and Their Applications. Adam Hilger, Bristol, 1989.
10. Novák V. Soft computing methods in managerial decision making. In Proc. 7th Czech-Japanese Seminar on Data Analysis and Decision Making under Uncertainty, 2004, pp. 63–68.
11. Saaty T.J. Fundamentals of Decision Making and Priority Theory With the Analytic Hierarchy Process. RWS Publications, 2000.
12. Sakawa M. Fuzzy Sets and Interactive Multiobjective Optimization, Applied Information Technology. Plenum, New York, 1993.
13. Slowinski R. (ed.). Fuzzy Sets in Decision Analysis, Operations Research and Statistics. Handbook of Fuzzy Sets Series. Kluwer Academic, Dordrecht, 1998.
14. Zimmermann H.-J. Fuzzy Set Theory and Its Applications. Dordrecht, Boston, 1985.
15. Zopounidis C., Pardalos P.M., and Baourakis G. Fuzzy Sets in Management, Economics and Marketing. World Scientific, 2001.

Fuzzy Time

- ▶ Temporal Indeterminacy

Fuzzy/Linguistic IF-THEN Rules and Linguistic Descriptions

Vilém Novák
University of Ostrava, Ostrava, Czech, Republic

Definition

Fuzzy/linguistic IF-THEN rules are structured expressions of natural language having the form

$$\mathcal{R}: \text{IF } X \text{ is } \mathcal{A} \text{ THEN } Y \text{ is } \mathcal{B} \tag{1}$$

where X, Y are variables and $\mathcal{A}, \mathcal{B}$ are expressions such as *small, very small, medium, roughly medium, more or less big, big*, etc. The latter are called *evaluative linguistic expressions*. Modeling their meaning in fuzzy set theory makes it possible to model the meaning of the whole rule. The part before THEN is called *antecedent*, the part after it is called *consequent*.

A *linguistic description* is a finite set of fuzzy/linguistic IF-THEN rules

$$\begin{aligned} &\mathcal{R}_1: \text{ IF } X \text{ is } \mathcal{A}_1 \text{ THEN } Y \text{ is } \mathcal{B}_1 \\ &\qquad\qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &\mathcal{R}_m: \text{ IF } X \text{ is } \mathcal{A}_m \text{ THEN } Y \text{ is } \mathcal{B}_m. \end{aligned} \tag{2}$$

Linguistic description can be taken as a special structured text in natural language which describes some situation.

Key Points

There are two possible ways how fuzzy/linguistic IF-THEN rules can be interpreted in fuzzy set theory:

(a) IF-THEN rule is assigned a fuzzy relation,

(b) IF-THEN rule is assigned a function from the set of contexts to the set of fuzzy relations.

Case (b) is more complicated but more realistic as a model of the meaning of linguistic expressions since (1) can be taken in this case as a conditional expression of natural language.

In case (a), the whole linguistic description (2) is assigned a fuzzy relation constructed using one of two possible formulas: let each of the expressions of the form "X is $\mathcal{A}_i$" be assigned a fuzzy set $A_i \subset U$ and "Y *is* $\mathcal{B}_i$" $i = 1,...m$ a fuzzy set $B_i \subset V$. Two possible fuzzy relations can be considered:

$$R^A(x,y) = \bigvee_{i=1}^{m} (A_i(x) \otimes B_i(y)), \tag{3}$$

$$R^I(x,y) = \bigwedge_{i=1}^{m} (A_i(x) \rightarrow B_i(y)), \tag{4}$$

where $\otimes$ is a t-norm and $\rightarrow$ is a residuation. Then (3) is called *disjunctive normal form* in which each IF-THEN rule is interpreted as *conjunction*; (4) is called *conjunctive normal form* in which each IF-THEN rule is interpreted as *implication*. Both forms (3) and (4) are two possible interpretations of the linguistic description (2).

The second interpretation of (2) is a set of functions $I(\mathcal{R}_i)$ assigned to the rules $\mathcal{R}_i$ in (2), $i = 1,...,m$. Each function has the form

$$I(\mathcal{R}_i) : C_X \times C_Y \rightarrow L^{U \times V} \tag{5}$$

where C_X, C_Y are sets of *contexts* for the variable X and Y, respectively. Each context $w_X \in C_X$ and $w_Y \in C_Y$ is a certain interval of elements in U and V, respectively. The couple of contexts $\langle w_X, w_Y \rangle$ is assigned via (5) a fuzzy relation of the form $A_i(x) \rightarrow B_i(y)$, $x \in U$, $y \in V$. This approach leads to a mathematical model of the meaning of the text (2). Human understanding to such expressions and deriving conclusions on the basis of them can be mimicked. The details can be found in [2].

Cross-references

- ▶ Approximate Reasoning
- ▶ Fuzzy Relation
- ▶ Fuzzy Set

Recommended Reading

1. Klir G.J. and Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice-Hall, New York, 1995.
2. Novák V. and Lehmke S. Logical structure of fuzzy IF-THEN rules. Fuzzy Sets Syst., 157:2003–2029, 2006.
3. Novák V. and Perfilieva I. On the semantics of perception-based fuzzy logic deduction. Int. J. Intell. Syst., 19:1007–1031, 2004.
4. Novák V., Perfilieva I., and Močkoř J. Mathematical Principles of Fuzzy Logic. Kluwer, Boston/Dordrecht, 1999.