University of Modena and Reggio Emilia
"Enzo Ferrari" Engineering Department

Master Degree Thesis
of Computer Science Engineering

Faceted Browsing: Analysis and implementation of a Big Data solution using Apache Solr.

Candidate:
Paolo Malavolta

Advisor:
Prof. Sonia Bergamaschi
Co-Advisor:
Prof. H.V. Jagadish
Dott. Ing. Francesco Guerra

Academic Year 2012 – 2013

Key words:

Big Data
Faceted Browsing
Bayesian Networks
Apache Solr
Keywords Search

ABSTRACT

Data is pervasive – and we generate more every day. We live in the era of big data, in which we produce and collect digital information on an unprecedented scale. This data is generated by terabyte through social media interactions, commerce and banking, business operations, healthcare activities, and scientific exploration. By developing tools and techniques to comb through and make sense of big data, decisions that previously were based on guesswork, intuition, or on painstakingly constructed models of reality can now be made based on the data itself.

We're talking about a lot of data, which is increasingly being gathered by the ubiquitous information-sensing mobile devices, remote sensing technologies, software logs, cameras, microphones, radio-frequency identification readers, and wireless sensor networks around us. In 2010, enterprises and users stored more than 13 exabytes of new data; this is over 50,000 times the data in the Library of Congress (LOC). As of 2012, 2.5 quintillion (2.5×10^{18}) bytes of new data were created every day.

Big data analysis now drives nearly every aspect of modern society, from manufacturing and retail, through mobile and financial services, through the life sciences and physical sciences. Big data is transforming how research is conducted, with giant repositories of specialized research data, such as biological or astronomical data, collected and shared on a massive scale. There is an entire discipline of bioinformatics that is devoted to the curation and analysis of such data.

Big data holds great promise for reducing healthcare cost and improving quality, for transforming the way education is delivered, and for use in assembling and acting on societal data. The ability to continue to use big data to make new connections and discoveries will help to drive the breakthroughs of tomorrow.

One of the most valuable means through which to make sense of big data, and thus make it more approachable to most people, is through data visualization. Data visualization is way finding, both literally, like the street signs that direct you to a highway, and figuratively, where colors, size, or position of abstract elements convey information. In either sense, the visual, when correctly aligned, can offer a shorter route to help guide decision making and become a tool to convey information critical in all data analysis. However, to be truly actionable, data visualizations should contain the right amount of interactivity. They have to be well designed, easy to use, understandable, meaningful, and approachable.

This thesis is focused on a new approach to visualize huge amount of data.

The project consists in designing and implementing a bayesian suggestion algorithm into an existing and widely used enterprise platform such as *Solr* that I have adapted and extended to work in a Big Data scenario.

The bayesian suggestion algorithm is a key ingredient in a big data scenario, because a query can generate so many results that the user can be confused. Thus, the generation of just the better results together with the result path the user has chosen can be very useful.

Moreover, I developed a software in Java, based on bayesian networks for generating queries from keyword queries over relational databases. So, using Java, I have provided a software that permits us to generate query without performing joins because they are the most requiring database operations.

The research area of keyword search over relational databases has become popular since it allows non-expert users to formulate queries without the need to learn a structured query language (SQL) and understand how the data is represented inside a database. The existing techniques for keyword search over structured sources are based on a-priori instance-analysis that scans the database instance and constructs an index, or a similar structure, which is used at run-time to easily identify and retrieve the information requested by the user. In this thesis, I experimented how an index based approach can exploit a Bayesian network for improving the accuracy of the results in case of user keywords composed of multiple terms.

OUTLINE OF THE THESIS

The Thesis is organized as follow:

- *Chapter 1* introduces the goals and achievements of the thesis.
- In *Chapter 2* I will describe what faceted browsing means, how it is born, how and why it is important in a Big Data context and what consequence it makes.
- *Chapter 3* provides a Big Data solution for *Apache Solr*.
- *Chapter 4* shows the study and development of an extension of *Solr* to make multi-faceted querying and faceted navigation (modifying its front-end) executable on *Solr*.
- *Chapter 5* is just a brief description about what can we do with *Bayesian Networks* and which is the most important algorithm used.
- In *Chapter 6* I will show a Bayesian solution used together with *Solr* for a better user search experience.
- *Chapter 7* introduces the topic of *keywords search* over relational databases showing some currents researches.
- *Chapter 8* provides a software that, by using Bayesian networks, permits to perform keywords search over relational databases.
- Finally, *Chapter 9* provides conclusions and some ideas for future works.

This thesis is part of a joint project between UniMoRe and UMich, in which two research thesis were carried out in parallel. The two thesis intended to join the big data visualization and Bayesian networks techniques.

The role played by me in the project, was the study, analysis and implementation of an improvement in the front end *Apache Solr* (faceted browsing), further adding recommendations based on Bayesian networks and providing it with a carrier suitable for use contexts in big data.

The second thesis within the project, carried out by Emanuele Charalambis, was instead based on the study of Bayesian networks in big data environments.

As a final point, the study of Bayesian Networks (BN) allowed me to perform an additional task of research, that is the development of a simple software, based on BNs to be used for performing keyword search over relational databases.

SOMMARIO

I dati sono pervasivi - e sono generati ogni giorno di più. Viviamo nell'era dei *Big Data*, in cui produciamo e raccogliamo informazioni digitali su larga scala senza precedenti. Questi dati sono nell'ordine dei terabyte e vengono generati attraverso le interazioni dei social media, delle banche, nelle operazioni commerciali, attività sanitarie, e nella ricerca scientifica. Con lo sviluppo di strumenti e tecniche per gestire, e quindi dare un senso a questa grande mole di dati, decisioni che in precedenza erano basate su congetture, intuizioni, o su modelli faticosamente costruiti della realtà, ora possono essere effettuate sulla base dei dati stessi.

Stiamo parlando di una mole di dati molto elevata, raccolti sempre più spesso dagli onnipresenti dispositivi mobile, dalle tecnologie di telerilevamento, dai software, telecamere, microfoni, lettori a radio frequenza e dalle reti di sensori wireless intorno a noi. Nel 2010, le imprese e gli utenti hanno memorizzato più di 13 exabyte di nuovi dati; cioè oltre 50.000 volte i dati contenuti nella *Library of Congres*s (LOC). A partire dal 2012, 2,5 trilioni ($2,5 \times 10^{18}$) di byte di nuovi dati vengono creati ogni giorno.

La *Big Data analysis* guida ora quasi ogni aspetto della società moderna, dalla produzione e vendita al dettaglio, attraverso i servizi di telefonia mobile e finanziari, attraverso le scienze della vita e le scienze fisiche. I Big Data stanno trasformando il modo di condurre le ricerche; con i repository di dati di ricerca specializzati i dati biologici o astronomici sono raccolti e condivisi su larga scala. C'è un'intera disciplina della bioinformatica che è dedicata al management e all'analisi di tali dati.

I Big Data rappresentano una grande promessa per la riduzione dei costi sanitari e per migliorarne la qualità, per trasformare il modo in cui l'istruzione viene insegnata e per l'uso nel modo di agire sui dati della società odierna. La possibilità di continuare a utilizzare i Big Data per nuove interazioni e scoperte aiuterà a guidare le innovazioni di domani.

Uno dei mezzi più importanti attraverso i quali dare un senso ai Big Data, e quindi renderli più accessibili alla maggior parte delle persone, è attraverso la loro visualizzazione. La *Big Data Visualization* è un modo per trovare, sia in senso letterale, così come i segnali stradali che ci dirigono ad una strada, sia in senso figurato, dove i colori, le dimensioni o la posizione di elementi astratti veicolano informazioni. In entrambi i sensi, quello visivo, quando correttamente configurato, deve essere in grado di offrire un percorso più breve per aiutare il processo decisionale e diventare uno strumento per trasmettere informazioni critiche in tutte le analisi dei dati. Tuttavia, affinchè questo obiettivo sia realmente perseguibile, la data visualization devono contenere la giusta quantità di interattività. Deve perciò essere ben progettata, facile da usare, comprensibile, significativa e accessibile.

Questa tesi si concentra su un nuovo approccio per visualizzare enormi quantità di dati.

Il progetto consiste nella realizzazione di un algoritmo di suggerimento bayesiano in una piattaforma di ricerca enterprise largamente utilizzata quale è *Solr* che ho previamente adattato per lavorare in contesti Big Data.

Questo è importante perché una query può generare tanti risultati e l'utente può perciò essere confuso. Così, la generazione dei soli risultati migliori può essere molto utile insieme con il percorso scelto dall'utente.

Inoltre, ho sviluppato un software per il keyword search su database relazionali. Così, usando Java, ho sviluppato un software che permette di generare query senza eseguire join che sono le operazioni di database più onerose.

L'area di ricerca del keyword search su database relazionali è diventata popolare in quanto permette agli utenti non esperti di formulare query senza la necessità di imparare un linguaggio di interrogazione strutturato e capire come i dati vengono rappresentati all'interno di un database. Le tecniche esistenti per il keyword search sulle fonti strutturate sono basati su una istanza di analisi a priori che analizza l'istanza di database e costruisce un indice, o una struttura simile, che viene utilizzata in fase di esecuzione, identificando così facilmente e recuperando le informazioni richieste dall'utente.

In questa tesi, ho sperimentato come un approccio basato su indice può sfruttare una rete bayesiana per migliorare l'accuratezza dei risultati in caso di keywords utente composte da più termini.

OUTLINE OF THE THESIS

La Tesi è organizzata come segue:

- Il *Chapter 1* è una breve descrizione sugli obiettivi svolti in questa tesi.
- Nel *Chapter 2* descriverò cosa significa faceted browsing, come è nato, come e perchè è importante in un contesto Big Data e a quali conseguenze porta.
- Il *Chapter 3* fornisce una soluzione Big Data per *Apache Solr*.
- Il *Chapter 4* mostra lo studio e lo sviluppo di una estensione per *Solr* che renda possibile il multi-faceted querying e il faceted browsing modificando il suo front-end.
- Il *Chapter 5* è solo una breve descrizione di cosa possiamo fare con le *Reti Bayesiane* e quali sono gli algoritmi più utilizzati.
- Nel *Chapter 6* mostrerò una soluzione bayesiana usata in *Solr* per una migliore ricerca da parte dell'utente.
- Il *Chapter 7* introduce l'argomento del *keywords search* su database relazionali, mostrando a che punto è la ricerca.
- Il *Chapter 8* fornisce un software che, usando le reti bayesiane, permette il keywords search su database relazionali.
- In ultimo, il *Chapter 9* trae le conclusioni di questa tesi e propone qualche idea per possibili sviluppi futuri.

Questa tesi è parte di un progetto congiunto tra UniMoRe e UMich, in cui due tesi di ricerca sono state condotte in parallelo. Le due tesi hanno l'obiettivo di unire i Big Data e le reti bayesiane.

In questo progetto il ruolo svolto da questa tesi è quello dello studio, analisi e implementazione di un miglioramento del front-end di *Apache Solr* (faceted browsing), aggiungendo inoltre raccomandazioni basate su reti bayesiane e dotandolo di un supporto adatto per essere utilizzato in contesti Big Data.

La seconda tesi di ricerca, realizzata da Emanuele Charalambis, è invece basata sullo studio delle reti bayesiane in ambienti Big Data.

Come punto finale, lo studio delle reti bayesiane, mi ha al contempo permesso di svolgere un task supplementare di ricerca, sviluppando un semplice software che permette di usare le BNs per il keyword search su database relazionali.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my advisor Professor Sonia Bergamaschi and co-advisors Professor H.V. Jagadish and Professor Francesco Guerra for their patient guidance, valuable suggestions and useful critiques during the planning and development of this research work.

I would like to offer my special thanks to my family and all my real friends for their support, encouragement and friendship throughout my studies and my life.

Finally, I would like to extend my gratitude to Ing. Massimo Ragni and ASSI (*Associazione Specialisti Sistemi Informativi*) association.

The work presented in this thesis was partially supported by ASSI.

Thank you at all!

COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY of MICHIGAN

October 29, 2013

To whom it may concern:

I am pleased to extend an invitation to Mr. Paolo Malavolta, a Master student at the University of Modena e Reggio Emilia, Modena, Italy to visit the University of Michigan at Ann Arbor, Department of Computer Science and Engineering, during the period of November 1, 2013 to January 29, 2014 to conduct collaborative research in the field of Big Data management, faceted browsing and Bayesian networks.

The travelling expenses will be covered by a fund provided by Professor Sonia Bergamaschi of Department of Engineering "Enzo Ferrari", University of Modena e Reggio Emilia, Modena, Italy.

During his visit, Mr. Malavolta will be self-financed.

I will not provide financial support but I will provide research facilities in my lab to conduct collaborative research.

Sincerely,

H V Jagadish
Bernard A Galler Collegiate Professor of
Elec. Engg. and Computer Science.
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121

COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY of MICHIGAN

February 25, 2014

To whom it may concern:

This is to confirm that Mr. Paolo Malavolta spent three months from November 2013 through January 2014 working on research under my direction in the Software Systems Research Laboratory located in the Bob and Betty Beyster Building at the University of Michigan.

During this period, he worked on research problems in Big Data management, faceted browsing and Bayesian networks, he participated in research meetings of the DataBase group, and he attended various seminars that were organized in the DataBase group or in Computer Science and Engineering department.

Sincerely,

H V Jagadish
Bernard A Galler Collegiate Professor of
Elec. Engg. and Computer Science.
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Utilizing computers had always begged the question of interfacing. The methods by which human has been interacting with computers has travelled a long way. The journey still continues and new designs of technologies and systems appear more and more every day and the research in this area has been growing very fast in the last few decades.

The growth in *Human-Computer Interaction* (HCI) field has not only been in quality of interaction, it has also experienced different branching in its history. Instead of designing regular interfaces, the different research branches have had different focus on the concepts such as intelligent adaptive interfaces rather than command/action based ones, and finally active rather than passive interfaces

Sometimes called as *Man-Machine Interaction* or *Interfacing*, the concept of Human-Computer Interaction/Interfacing was automatically represented with the emerging of computer, or more generally machine, itself. The reason, in fact, is clear: most sophisticated machines are worthless unless men can use them properly. This basic argument simply presents the main terms that should be considered in the design of HCI: functionality and usability [1].

Why a system is actually designed can ultimately be defined by what the system can do i.e. how the functions of a system can help towards the achievement of the purpose of the system. *Functionality* of a system is defined by the set of actions or services that it provides to its users. However, the value of functionality is visible only when it becomes possible to be efficiently utilized by the user [2]. *Usability* of a system with certain functionality is the range and degree by which the system can be used efficiently and adequately to accomplish certain goals for certain users. The actual effectiveness of a system is achieved when there is a proper balance between the functionality and usability of a system [3].

Having these concepts in mind and considering that the terms computer, machine and system are often used interchangeably in this context, HCI is a design that should produce a fit between the user, the machine and the required services in order to achieve a certain performance both in quality and optimality of the services [4]. Determining what makes a certain HCI design good is mostly subjective and context dependant. The available technology could also affect how different types of HCI are designed for the same purpose. One example is using commands, menus, *graphical user interfaces* (GUI), or virtual reality to access functionalities of any given computer.

The advances made in last decade in HCI have almost made it impossible to realize which concept is fiction and which is and can be real. The trust in research and the constant twists in marketing cause the new technology to become available to everyone in no time.

At the point where the GUI was about to become popular, HCI researchers summarized the important opportunities it provided, under the name *Direct Manipulation*. In fact some of these things were already possible with text interfaces, and they remain relevant in more recent generations of hardware. It is also possible to use GUI libraries to create bad user interfaces that do not support these principles – just being graphical doesn't make it good!

The principles of Direct Manipulation as described are:

- An object that is of interest to the user should be continuously visible in the form of a graphical representation on the screen

- Operations on objects should involve physical actions (using a pointing device to manipulate the graphical representation) instead of commands with complex syntax
- The actions that the user makes should be rapid, should offer incremental changes over the previous situation, and should be reversible
- The effect of actions should immediately be visible, so that the user knows what has happened

A further inference problem is that, in addition to the user not knowing what is happening inside the system, the system doesn't "know" what is happening inside the user. Advanced systems can be designed to record and observe user interactions, and on the basis of that data, make inferences about what the user intends to do next, and present shortcuts, usability cues or other aids. These kinds of *"intelligent user interface"* are becoming more common, but they can also introduce severe usability problems.

Many intelligent user interfaces emerge from the *machine learning* community, and especially *Bayesian inference* techniques. Bayesian techniques are more appropriate to user interfaces than other techniques for a range of reasons:

- They don't rely on large training sets (as is the case with neural net approaches), so they can adapt more quickly to individual users
- Bayesian consideration of prior probabilities corresponds better to commonsense human reasoning under uncertainty.
- Bayes formula provides a consistent way to combine data from user interactions with historical data and heuristic rules.

An inference framework provides a valuable analytic perspective on many current trends in user interaction. For example, the behavior of Google, or of recommender systems such as Amazon or Facebook friend finder, use inference techniques to apply statistical data and guess what the user really wants. It remains the case that when the system makes inaccurate inferences, the results will be annoying, confusing, or even damaging.

This thesis intends to provide a good solution to visualize a huge amount of data, also called *Big Data*. To obtain this I have joined *faceted browsing* and *Bayesian network* together and I have implemented it into a very popular enterprise search engine focused on Big Data such as *Apache Solr*, previously adapted to work well with Big Data.

The knowledge about Bayesian network also takes me to propose a simple algorithm to do *keyword search* over relational database without using join functions.

This thesis is part of a joint project with *UniMoRe* and *UMich*, in which two research projects were carried out in parallel. The two projects intended to join the Big Data and Bayesian networks.

In these projects, the role played by me has seen me involved in the study, analysis and implementation of an improvement in the front-end *Apache Solr* (faceted browsing), further adding to recommendations based on Bayesian networks and providing it with a carrier suitable for use contexts in big data.

The second research project, carried out by Emanuele Charalambis, was instead based on the study of Bayesian networks in big data environments.

As a final point, the study of Bayesian networks at the same time I was also allowed to perform an additional task of research, developing a simple software that allows me to use the BNs to search on relational databases.

2 FACETED BROWSING & BIG DATA

To learn better what *faceted browsing*, or *faceted navigation*, means now I will just give a brief look about the problem. I will use a domain popular with faceted search researchers and practitioners: wine. The facets typically used to characterize wine include varietal (the type of grape, e.g., Merlot), vintage (the year in which a wine was produced), region, rating, price, vintner, and so on. How can I apply a faceted representation of the data to help someone (whom I will call the user) find a wine that meets his or her particular tastes?

2.1 PARAMETRIC SEARCH

A *parametric search* interface is essentially a Boolean search interface for a faceted content collection: it allows users to formulate queries by visually specifying a set of constraints on the facet values. A query is typically an AND of ORs: values selected within a single facet are combined using a logical OR, whereas constraints associated with different facets are combined using a logical AND. The system responds to a query with the set of objects in the collection that satisfy it.

Let me use a concrete example to see how parametric search works. Consider a user interested in red wines from France with a rating of at least 90 and a price at most $10. Parametric search allows him/her or her to specify this query as the following set of constraints: {Varietal: Red, Region: France, Rating: >90, Price: <$10}. A parametric search interface, such as the one depicted in Figure 2.1, presents each facet for independent specification.

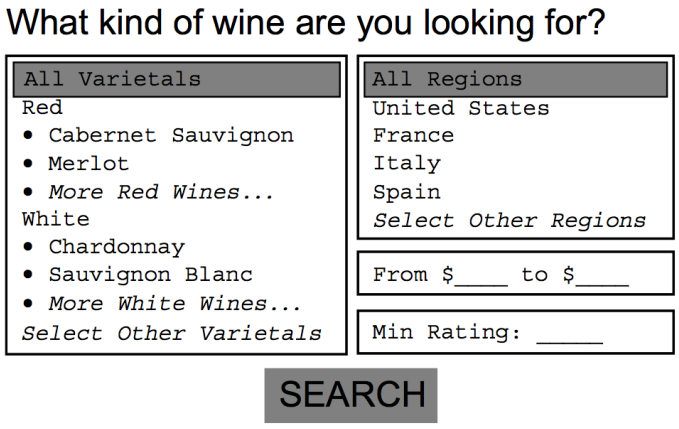


Figure 2.1 – A simple parametric search interface.

Note that the simple interface pictured does not enable OR-ing within a facet; I will discuss the interface challenge of allowing multiple selections from a facet in Section 2.3.3.

As should be clear from even this simple example, different kinds of facets lend themselves to different query metaphors.

For a facet that has nominal values (i.e., a list of enumerable categorical values), it makes sense for the user to see a list of options and select one or more of these individually. If the list is large (e.g., a

wineries facet), then further effort is necessary to avoid information overload, as I will discuss in detail in Section 2.5.3.

For a hierarchical facet, such as the varietal facet in my example, the user might select a nonleaf value, such as Red, or a leaf value, such as Merlot. A user might see both leaf and nonleaf options simultaneously or might work top-down through the hierarchy.

For numerical facets, such as price or rating, the user most likely want to select a range, possibly unbounded on one side. In the visualized interface, there is no guidance as to what are reasonable bounds; a more sophisticated interface might provide such guidance.

I will consider interface issues specific to faceted search in the Section 2.3. Nonetheless, I raise some of them now to emphasize how interface design is a critical concern in any application that querying against a faceted collection.

Importantly, parametric search is a form of set retrieval: it offers a subset of Boolean search functionality, albeit over facets rather than unstructured text. Like Boolean search over text, it suffers from the problem that users struggle to formulate their queries. They run into a "million or none" problem: underspecified queries return too many results, whereas over specified queries return no results. Parametric search offers expressivity, but it does not offer users guidance through the space of possible queries.

2.2 FACETED NAVIGATION

Faceted navigation fills in the piece that is missing in parametric search: guidance. Parametric search requires that the user express an information need as a query in one shot, making selections across all facets of interest. In contrast, faceted navigation allows the user to elaborate a query progressively, seeing the effect of each choice in one facet on the available choices in other facets.

To illustrate the difference, let me return to my previous wine example. A user might want to start by establishing a budget of $10. This selection constrains the other facets — for example, there are no French wines for less than $10. Further selections in the varietal and region facets constrain the options in the unspecified facets, for example, selecting Spain as a region eliminates Sauvignon Blanc as an option for the varietal facet. Eventually, the user chooses to see all results that match the specified constraints. The faceted navigation interface in Figure 2.2 illustrates this sequence of steps.

Faceted navigation delivers an experience of progressive query refinement or elaboration. From a user's perspective, faceted navigation eliminates the "dead ends" that can result from selecting unsatisfiable combinations of constraints among the facets. In fact, most combinations of facet values are unsatisfiable because the set of satisfiable combinations is typically a sparse subset of the set of all possible combinations. Thus, faceted navigation addresses the "million or none" problem of parametric search.

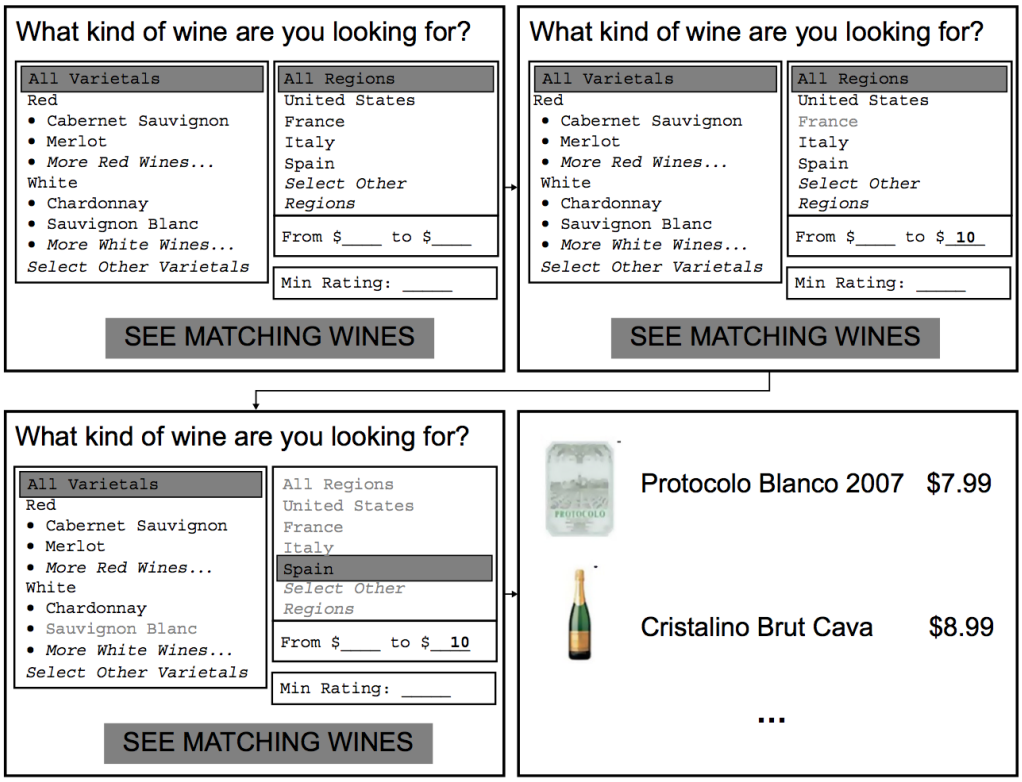


Figure 2.2 – Facet navigation interface.

Not all kinds of facets, however, lend themselves to faceted navigation interfaces. For example, in the interface depicted above, the user could still arrive at a dead end by selecting wines for less than $1. A more sophisticated user interface might avoid this possibility, for example, by dividing numerical values into discrete ranges. In general, faceted navigation only makes sense for facets whose values can be presented through tractable choice lists.

But how do I handle textual data? That question leads me to the next section—and the subject of this chapter: faceted search.

2.3 FACETED SEARCH

I finally arrive at *faceted search*, the topic that motivated this chapter. The goal of this section, however, is to provide a minimal definition of faceted search that will serve as the basis for the next chapters.

The parametric search and faceted navigation approaches described are unaware to unstructured text and instead assume documents to be collections of values in a faceted classification system.

In practice, most of the document collections that interest me are semi-structured: a typical document contains a combination of unstructured text and structured attributes. The structured attributes are some times called metadata. A document's metadata consists of structured attributes about the document that are, at least in a logical sense, stored with the document.

When this structured content conforms to a faceted classification system, I can combine text search, applied to the unstructured text content, with faceted navigation of the structured content. This approach is the essence of faceted search.

Let me go back to my wine example, but this time assuming a richer document model where each wine is also associated with descriptive text. The example in Figure 2.3 illustrates how faceted search combines a text-oriented search with faceted navigation.

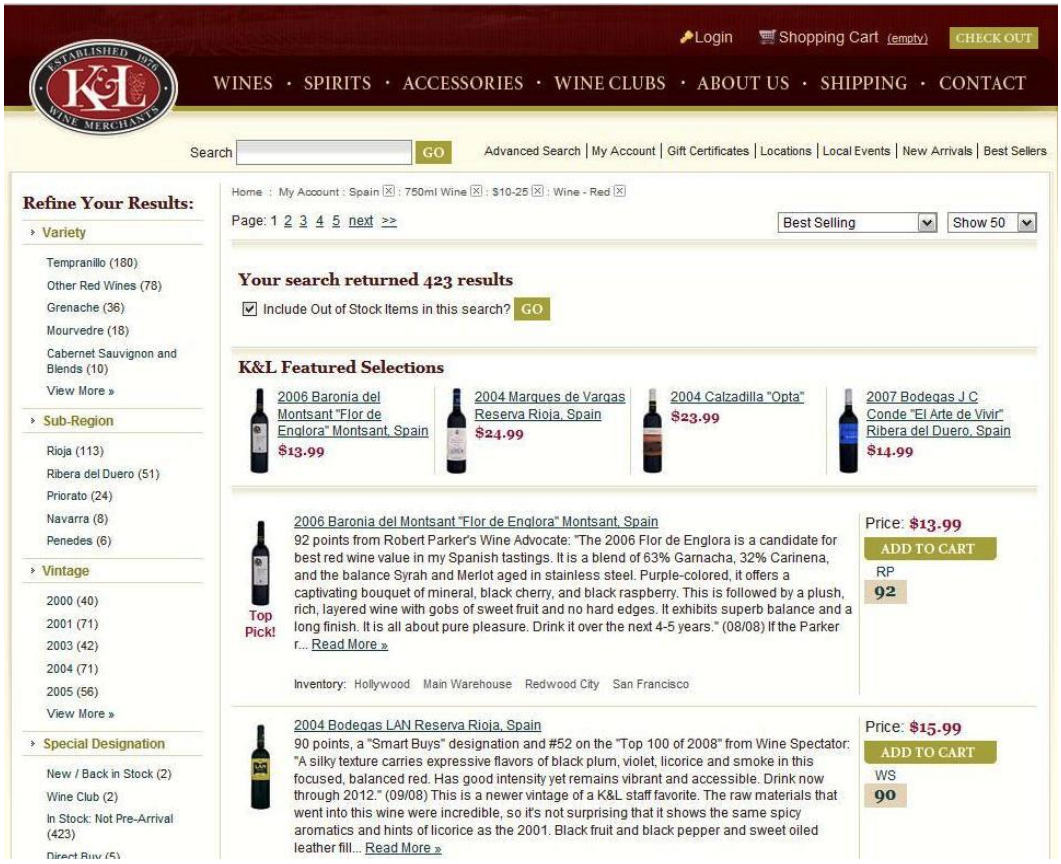


Figure 2.3 – Faceted search interface.

This time, the user does not start by using the facets but rather performs a free-text search for wines that contain the word "sophisticated" in their description. The user then uses the price facet to narrow these results to those wines less than $10. The system returns the wines that match these filters, sorted by rating (another facet). The user can further refine this query by selecting values from other facets, such as type, country, and so on.

So there you have it: faceted search.

2.3.1 ORGANIZING FACETS AND FACET VALUES

In this section, I consider the front-end problem: how do I organize the information that I do present?

I approach the challenge of information overload in two steps: reducing the number of facets and values presented (as discussed in Section 2.5.3) and then organizing them as effectively as possible. There are three general strategies for organizing facets:

- Use a static order that does not change as the user navigates.

- Dynamically rank the order of presentation of the facets on the basis of their estimated utility to the user.
- Organize similar or related facets into groups.

The choice of using a static facet order vs. ranking facets is an interesting trade-off.

On one hand, a static order has the advantage or reinforcing the user's mental model—because the user will always see the same facets in the same order. This strategy works best when the number of facets is small in that all facets are visible at all times.

On the other hand, a static ordering is less effective when the number of facets is too large to be displayed at once or when some facets only apply to particular query contexts. For example, on an e-commerce site that sells consumer electronics, some facets only apply to small subsets of the product catalog, such as wattage for audio speakers or megapixels for digital cameras. Those facets should only be displayed when the user is looking at narrow result sets. In general, the same kinds of utility measures used for filtering can also be used for ranking.

2.3.2 THE SEARCH BOX

Although my discussion of faceted search has focused on the use of facets for refinements, it is important to remember that faceted search is still a type of search—and that often the entry point into a faceted search system is the search box. Without the search box, I would only have a faceted navigation system—a useful interface for many applications but too limited to enjoy the broad success of faceted search.

Combining free-text search and faceted refinement is powerful: it allows users to create semi-structured queries and thus access structured and unstructured contents. However, the search box also raises significant design challenges for application developers.

The designer of a faceted search system must make a number of choices about how the search box behaves:

- Should a search query adhere to the current query filters?
- Should search look at all of the text in each document, or should it be restricted to specific fields?
- How should the search handle multiword queries by default? Should the words be combined as an OR (i.e., match any word), as an AND (i.e., match all words), or as a phrase (i.e., the words must all occur in a document and in that exact sequence), and should the user be given a choice in this matter?
- Should search queries be subject to query expansion, such as matching words that are variants of query terms?
- Should systems present multiple search boxes, a parameterizable search box, or an advanced search interface?

These are open-ended questions and only represent a subset of the questions about search behavior that face designers of faceted search applications. I will try to supply some answers—or, at least, guidance.

First, let me consider the question of whether the search query should respect the current query filters. In the most common use case for faceted search, a user initially enters a free-text search query and then follows it up by one or more refinements using the facets. For example, a user types in "digital cameras" and refines on a specific megapixel range. But how do I handle deviations from this common case, for example, when the user first narrows the document collection by selecting a facet value and then performs a free-text search? Does the text search adhere to the faceted refinement or start a new query from scratch?

The conventional and probably safest approach is for free-text search to clearing all other filters or to offer users the options to search within the current results, for example, by clicking on a check box indicating that the user is explicitly choosing to search only within the set of results that is currently being viewed.

Multiword search queries and query expansion also raise issues of precision and recall. However, because of their familiarity with web search engines, most users have become accustomed to search engines that interpret multiword search queries as an unordered conjunction (an AND, not an OR), for example, a search for faceted navigation returns documents containing both words but not necessarily in that order or even next to each other. While I should not be fatalistic about conventions, I must recognize that flouting them will incur some amount of user confusion.

2.3.3 MULTIPLE SELECTIONS FROM A FACET

The most common use case for faceted search or navigation is to select at most one value per facet, but there are at least two ways from which a user might select multiple values from the same facet:

- Disjunctive (OR) selection. Selecting a range (e.g., a price or date range) may be a kind of disjunctive selection, depending how the values are represented.
- Conjunctive (AND) selection.

The design challenge is to communicate to users whether selecting multiple values from a particular facet is disjunctive or conjunctive—particularly if the site offers both behaviors. Users are notoriously bad at inferring Boolean logic from subtle cues.

It is important to use an interface that not only is self-consistent but also adheres to familiar conventions. There are fewer interfaces that allow conjunctive selection from the same facet so, is necessary to implement a disjunction within the facet and a conjunction across the facets like figure 2.4 shows.

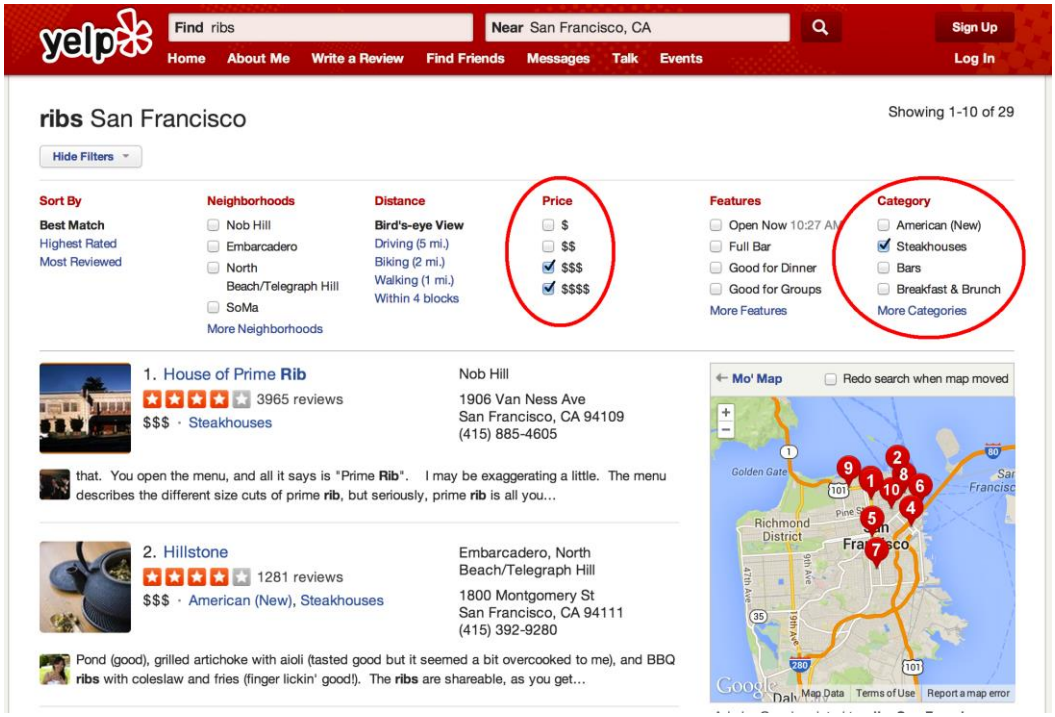


Figure 2.4 – Disjunctive and conjunctive selection on yelp.com

2.4 COMMERCIAL APPLICATIONS

This brief section makes no attempt to enumerate the many companies that have built such applications but rather highlights some of the most visible milestones in the commercialization of faceted search.

2.4.1 Car.com

This screen capture (Figure 2.5) from *cars.com* represents a typical example of a faceted navigation interface. Like most such interfaces, it is divided between a small query panel on the left and a results panel on the right. The left-hand side also includes a small summary digest describing the overall result set.

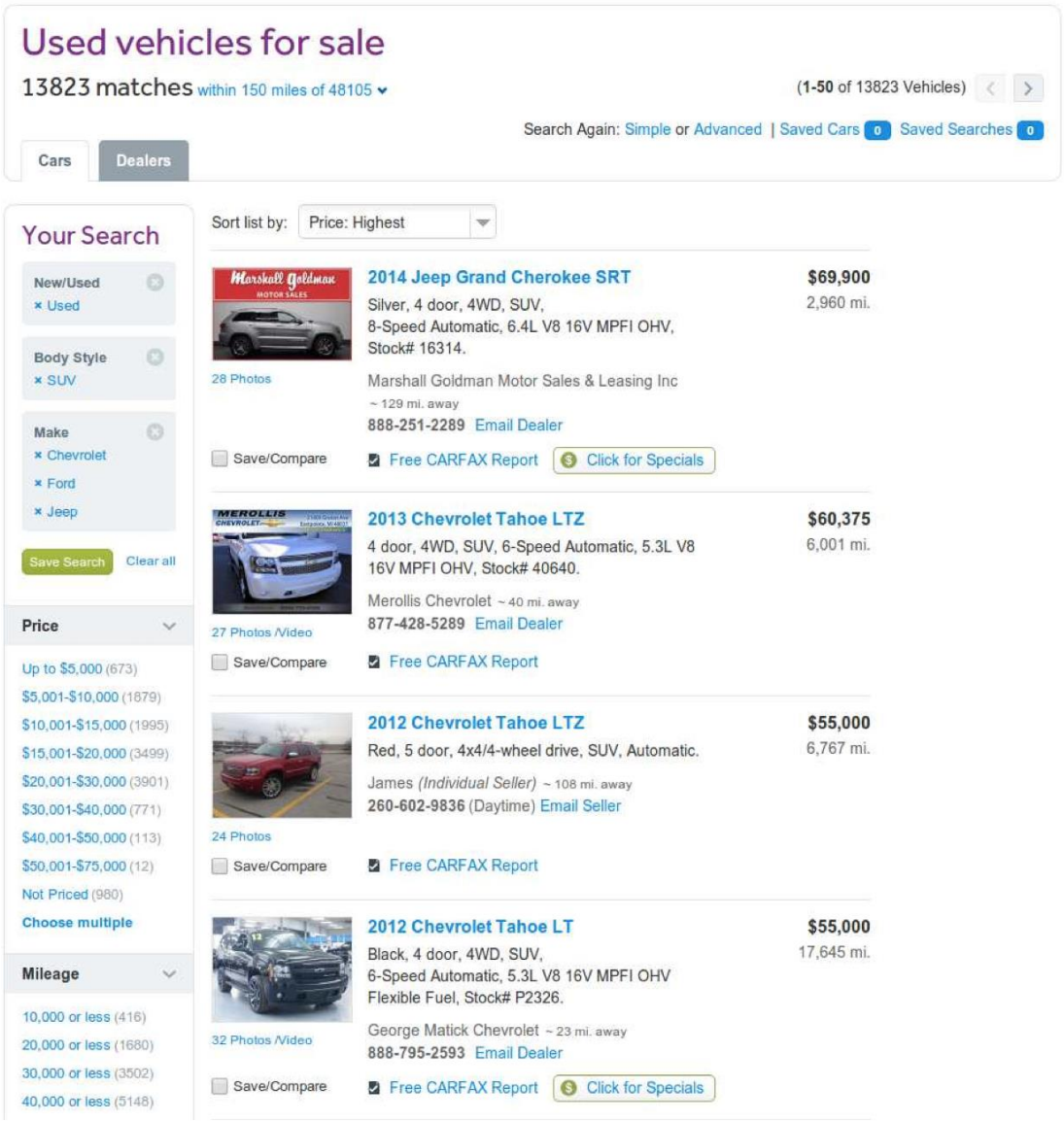


Figure 2.5 – Car.com faceted interface.

2.4.2 Amazon

This screen capture (Figure 2.6) from *amazon.com* shows many add-on features that are seen in typical faceted interface.

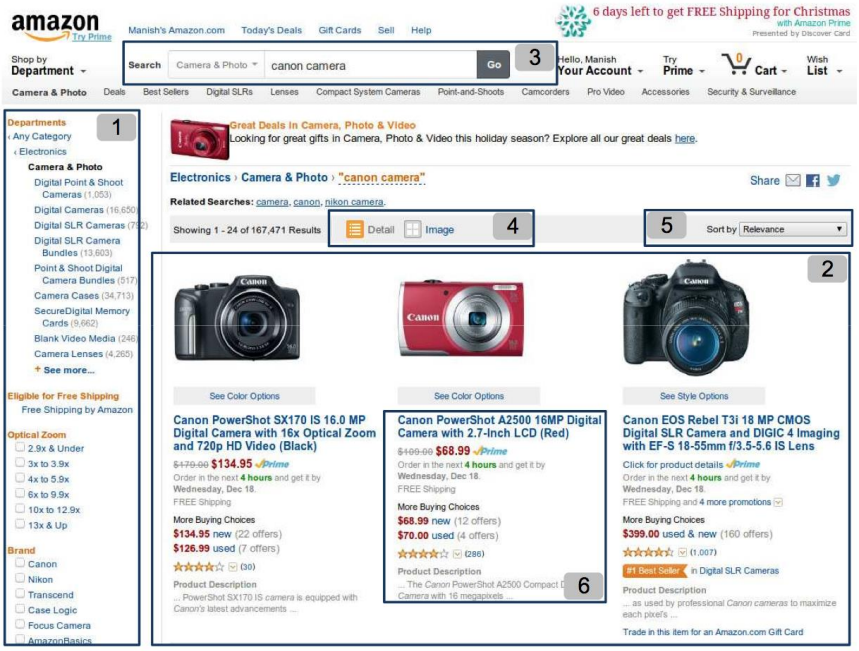


Figure 2.6 – Amazon faceted interface.

2.5 FACETED BROWSING OVER BIG DATA

2.5.1 SCALE

The value of a document collection is often correlated to its size; when it comes to information, more—or at least access to more—is better. Moreover, because faceted search enables exploration, a faceted search system potentially offers more utility for large document collections than a conventional search engine because it can make more of the information accessible in practice.

Unfortunately, the benefits of scale also incur costs, either in hardware requirements or in operational complexity. There are also efficiency concerns, but I will discuss those in the following section.

For faceted search systems, scale is itself a multifaceted concern. When I talk about scale, I primarily mean the following factors:

- Number of documents
- Number of facet values per document
- Searchable text per document

The precise storage requirements depend heavily on the particular data structures used to implement faceted search.

For small document collections, all of this information can be kept in main memory—an approach that, as I will see in the next section, has attractive efficiency implications. For larger collections, however, an in-memory approach may not be a realistic option.

As of the time of this writing, random-access memory (RAM) capacity for all but the most expensive servers is measured in gigabytes (1G = 10^9 bytes), whereas external storage capacity is measured in

terabytes ($1T = 10^{12}$ bytes). Moreover, the cost of RAM accelerates sharply at the high end. Hence, at least for the foreseeable future, the cost of external storage is far lower than the cost of RAM, which means that most storage-intensive applications are likely to require some—if not most—of the storage to be external.

It is also possible to distribute storage across multiple servers. Such a distribution may be a scaling strategy: using distributed in-memory storage to avoid the accelerating cost of server RAM. Distributed storage may also be an external requirement imposed on an application developer for other reasons. Although the storage for a faceted search should scale linearly, some care is required to avoid inefficient computation—particularly network latency.

2.5.2 EFFICIENCY

Faceted search is computationally demanding.

When a faceted search system processes a query, its first task is to determine the set of documents that satisfy the query constraints.

The subsequent task, however, is to show the facet values available for refining the set of results. In addition, faceted search applications often show the counts associated with these refinements. The task of computing refinements is significantly more demanding than that of computing the set of results.

Unfortunately, commercial providers of faceted search systems do not disclose the proprietary techniques they use to achieve computational efficiency.

It is also important to decide how to measure efficiency, particularly when serving concurrent users. If the main concern is throughput, that is, the average number of queries processed per second, then most approaches make it possible to improve efficiency linearly by using additional servers that maintain copies of the document collection (although distributing the collection can complicate replication). In contrast, there are no straightforward approaches to reduce latency, that is, the time experienced by a user for a single query. However, bandwidth problems can be "cured" with money.

As noted in the previous section, it is possible to distribute storage across multiple servers. Such an approach can reduce query latency through intra-query parallelization, as can distributing query processing among multiple processes or threads on a single server. All of these distribution approaches, however, introduce complexity.

Distributed storage across multiple servers introduces operational complexity, network latency that may outweigh the gains from distribution, and more complex query processing.

Distributed query processing on a single server is more attractive operationally, but introduces its own complexity.

The other significant factor likely to affect efficiency is the frequency of data updates. Many faceted search systems assume that indexing documents is a relatively slow, offline process compared to the time-sensitive task of servicing queries while a user waits. In the presence of data updates, however, I have to consider how users experience update latency—that is, a lag between when an update arrives and when users experience its effect in query results—and the degradation of query processing efficiency that results from computational resources spent on the indexing.

A full discussion of how to optimize the computational efficiency of a faceted search system is one of the scopes of this thesis. What I hope this section makes clear is that the computation cost of faceted search is significantly more expensive and complex than that of ordinary search engines.

2.5.3 INFORMATION OVERLOAD

The computational challenges discussed in the previous two sections are serious, but a challenge that is (at least) as serious is the scarcity of the most valuable resource of all—the user's attention. As Herbert Simon, the Nobel Laureate who pioneered the study of bounded rationality, says, "a wealth of information creates a poverty of attention and a need to allocate that attention efficiently" [5]. The wealth of information offered by faceted search systems threatens to overwhelm users and thus requires me to prioritize what information I show to users efficiently, thus optimizing the allocation of users' attention.

There are essentially two factors that can lead to information overload in a faceted search system: the number of facets and the number of facet values. Given the limitations of screen real estate and human attention, I cannot always show users all facets and all facet values. There are ways to prune both.

The number of facets reflects the number of ways a document can potentially be classified. In theory, there is no limit to the number of facets; there are infinitely many potential taxonomies to classify a document collection. In practice, of course, the number of facets is finite, but it may be quite large.

There is also the issue of dependence among facets. City, state, and country may exist as three distinct facets rather than a single hierarchical facet; language and nationality are highly correlated and yet clearly distinct facets; and yet other facets, such as medical subject within a library catalog, may only apply to a subset of a document collection. At best, designing a faceted classification scheme with independent facets requires extraordinary effort on the part of information architects; at worst, it is an impossible task because such a set of independent facets would not match cleanly to the way users conceive of the information space. Either way, I cannot require that facets be independent of one another.

Rather, I need to face the possibility of a large number of heterogeneous, interdependent facets. I cannot present all of these to users all the time—not so much because of the computational costs (although these could be significant) but because the information would overwhelm users. Instead I must attempt to determine the most valuable facets to present to users.

3 BIG DATA SOLUTION FOR APACHE SOLR

In this chapter I will describe an enterprise search platform such as *Apache Solr* and how to improve it for Big Data visualization. I have decided to use it because *Solr* is the most full text search platform used by big enterprises for their services, like for example *Macy's*, *the Guardian*, *Netflix* and many others, and is free and open-source, but it doesn't have an integrated Big Data solution and good enterprise solutions are very expensive.

3.1 BRIEF DESCRIPTION

Solr [6] is written in *Java* and runs as a standalone full-text search server within a servlet container such as *Jetty*. *Solr* uses the *Lucene* Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it easy to use from virtually any programming language. Solr's powerful external configuration allows it to be tailored to almost any type of application without Java coding, and it has an extensive plugin architecture when more advanced customization is required.

3.2 HOW TO ENABLE A BIG DATA SOLUTION

Solr running by default on a light servlet container like *Jetty* [7] but, such as is described above, to enable a Big Data solution for *Solr* we need to change it with a more powerful servlet container.

So, I need to integrate together these applications:

- *Solr*
- *Tomcat*
- *Apache Zookeeper*
- *Hadoop File System* (HDFS)

3.3 WHY USE TOMCAT INSTEAD JETTY ?

Jetty and *Tomcat* are open servlet containers, both of them support HTTP server, HTTP client and javax.servlet container. In this section, we will quick view the difference between *Jetty* and *Tomcat*, and give the generic idea about which is the better one.

Jetty is a uniformly excellent tool about some features. It has been started around 1998 and claims to be a "100% Java HTTP Server and Servlet Container". It is foremost a set of software components that offer HTTP and servlet services. Jetty can be installed as a standalone application server or be easily embedded in an application or framework as a HTTP component. It is a simple servlet engine, as do a feature rich servlet engine or part of a full *JEE* environment do.

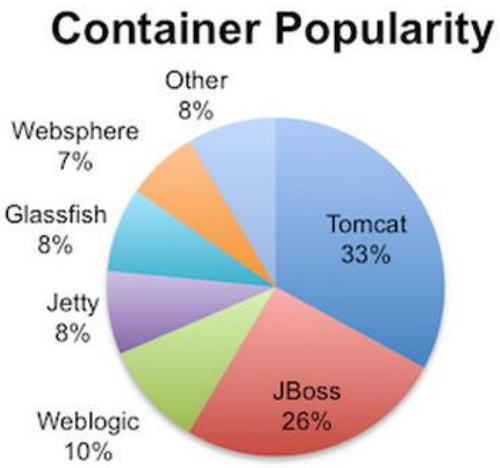


Figure 3.1 – Container popularity.

The following figure gives a generic idea about which *Java* Containers/App Servers are most used and so, I can see that *Tomcat* and *Jetty* are bigger winner open source containers. Tomcat is the absolute primary container over all others.

Jetty Features and Powered:

- Full-featured and standards-based.
- Embeddable and Asynchronous.
- Open source and commercially usable.
- Dual licensed under *Apache* and *Eclipse*.
- Flexible and extensible, Enterprise scalable.
- Strong Tools, Application, Devices and Cloud computing supported.
- Low maintenance cost.
- Small and Efficient.

Tomcat Features and Powered:

- Famous open source under *Apache*.
- Easier to embed Tomcat in your applications, e.g. in *JBoss*.
- Implements the Servlet 3.0, JSP 2.2 and JSP-EL 2.2 support.
- Strong and widely commercially usable and use.
- Easy integrated with other application such as *Spring*.
- Flexible and extensible, Enterprise scalable.
- Faster JSP parsing.
- Stable.

From a small benchmark test we can also see that *Jetty* is a good servlet container but for my project Tomcat is better because is wide supported and faster.

```
1 jetty 8080 Requests per second:    573.72 [#/sec] (mean)
2 tomcat 8888  Requests per second:    1228.50 [#/sec] (mean)
```

Figure 3.2 – Jetty vs. Tomcat requests.

Let us see, *Tomcat* handles 1228 requests per second but *Jetty* only goes 573 requests, so at least from this benchmark it reveals *Tomcat* does better.

Furthermore, after a run test, we can see those results like shown in figure 3.3 and 3.4.

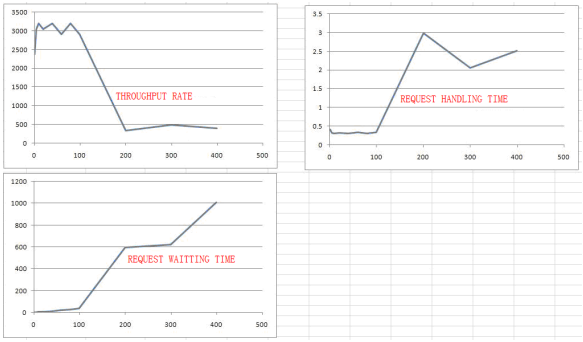


Figure 3.3 – Tomcat.

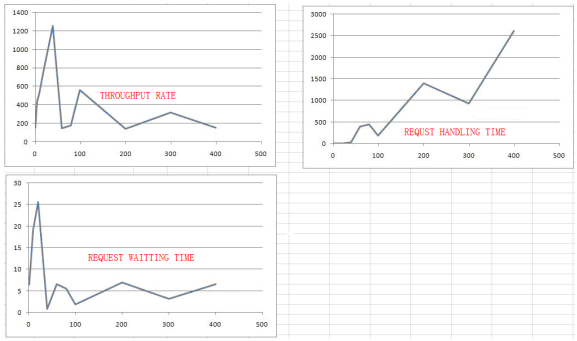


Figure 3.4 – Jetty.

These test show that in order to manage or visualize huge amount of data a better servlet container like Tomcat is needed.

3.4 BASIC CONCEPTS

On a single node, *Solr* has a core that is essentially a single index. If I want multiple indexes, I need to create multiple cores. With *SolrCloud*, a single index can span multiple *Solr* instances. This means that a single index can be made up of multiple cores on different machines.

The cores that make up one logical index are called a collection. A collection is essentially a single index that can span many cores, both for index scaling as well as redundancy. If, for instance, I wanted to move my two-core *Solr* setup to *SolrCloud*, I would have 2 collections, each made up of multiple individual cores.

In *SolrCloud* I can have multiple collections. Collections can be divided into slices. Each slice can exist in multiple copies; these copies of the same slice are called shards. One of the shards within a slice is the leader, designated by a leader-election process. Each shard is a physical index, so one shard corresponds to one core.

It is important to understand the distinction between a core and a collection. In classic single node *Solr*, a core is basically equivalent to a collection in that it presents one logical index. In *SolrCloud*, the cores on multiple nodes form a collection. This is still just one logical index, but multiple cores host different shards of the full collection. So a core encapsulates a single physical index on an instance. A collection is a combination of all of the cores that together provide a logical index that is distributed across many nodes.

Thus, the best way to manage Big Data with *Solr* is troughed *sharding* and *replication*.

In detail, sharding is a type of data partitioning that separates very large data into smaller, faster, more easily managed parts called data shards. The word shard means a small part of a whole. On The Server Side, I can explain sharding how to breaking up my data into many, much smaller data that share nothing and can be spread across multiple servers.

Technically, sharding is a synonym for horizontal partitioning. In practice, the term is often used to refer to any database partitioning that is meant to make a very large database more manageable.

The governing concept behind sharding is based on the idea that as the size of a database and the number of transactions per unit of time made on the database increase linearly, the response time for querying the database increases exponentially. Additionally, the costs of creating and maintaining a very large database in one place can increase exponentially because the database will require high-end computers. In contrast, data shards can be distributed across a number of much less expensive commodity servers.

On the other side, replication is the process of creating and managing duplicate versions of a database. Replication not only copies a database but also synchronizes a set of replicas so that changes made to one replica are reflected in all the others. In order to provide high availability, I can create replicas, or copies of each shard that run in parallel with the main core for that shard. The architecture consists of the original shards, which are called the leaders, and their replicas, which contain the same data but let the leader handle all of the administrative tasks such as making sure data goes to all of the places it should go. This way, if one copy of the shard goes down, the data is still available and the cluster can continue to function.

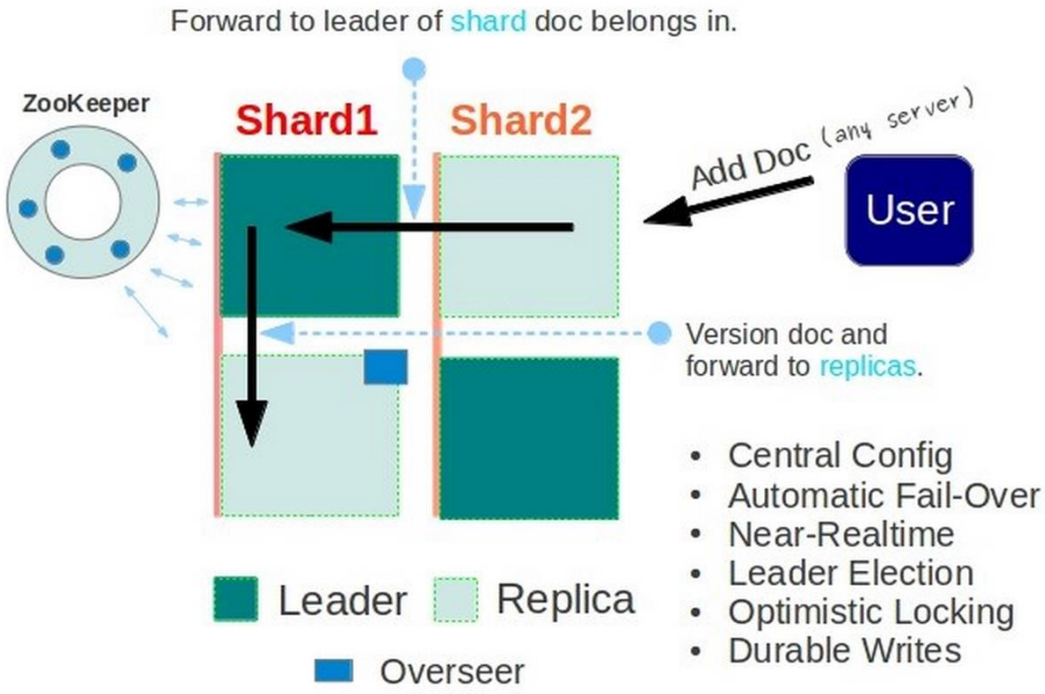


Figure 3.5 – Solr sharding and replication schema.

In the end, I could also provide a distributed solution that makes a quick access to data over a large number of nodes.

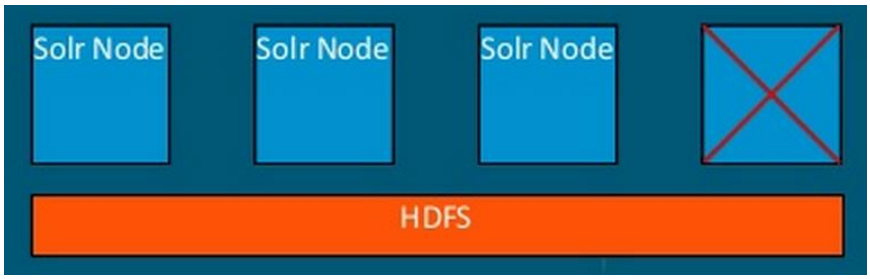


Figure 3.6 – HDFS ans Solr schema.

3.5 FIRST STEP: CHANGE SERVLET CONTAINER

In this section I will explain how to change the embedded *Jetty* to *Tomcat*. After have done this step I will have a working *Solr*, running on *Tomcat*.

The first step is the download of a binary distribution of Tomcat from its website [8]. Next I have to setup the tomcat application, first issue the following command to copy `solr.war` into the application root.

```
sudo cp /Users/paolo/Desktop/solr4.6.0/example/webapps/solr.war
     /Users/paolo/Desktop/solr4.6.0/example/solr/solr.war
```

One of the things about running *Solr* in a container different from than the embedded *Jetty* is that I need to setup the logging. This step is crucial, without doing this *Solr* will never start and I will pull all of my hair out trying to figure out why. Based on the instructions from the *Apache Solr Wiki*, I copy all the jar files,

```
sudo cp solr4.6.0/example/lib/ext
/Users/paolo/Desktop/apache-tomcat-7.0.50/lib
```

Now copy the logging configuration file.

```
sudo cp solr/example/resources/log4j.properties  /usr/share/tomcat7/lib
```

At this point I should set my log path by changing `solr.log` into `log4j.properties`. So, I have set it to `/Users/paolo/Desktop/solr4.6.0/logs`. Next I add *Solr* to the Catalina configuration folder

```
cd /Users/paolo/Desktop/apache-tomcat-7.0.50/conf/Catalina/localhost
                        sudo nano solr.xml
```

The configuration file should look like this:

```
<?xml version='1.0' encoding='utf-8'?>
<Context
      path="/solr/home"
      docBase="/Users/paolo/Desktop/solr4.6.0/example/solr/solr.war"
      allowlinking="true"
      crosscontext="true"
      debug="0"
      antiResourceLocking="false"
      privileged="true">
<Environment
```

```
      name="solr/home"
      type="java.lang.String"
      value="/Users/paolo/Desktop/solr4.6.0/example/solr"
      override="true"/>
</Context>
```

Let's test the install, I want to use the *Tomcat* application manager but I need to configure users first. Add the following to `/etc/tomcat7/tomcat-users.xml`.

```
<tomcat-users>
<role rolename="manager-gui"/>
<user username="tomcat" password="secret" roles="manager-gui"/>
</tomcat-users>
```

Solr Browser allows me to setup the *Velocity* properties, for example, I can setup browse `requestHandler` using `velocity.properties` marked in red like is shown below:

```
<requestHandler name="/browse" class="solr.SearchHandler">
        <arr name="first-components">
           <str>redirect</str>
        </arr>
        <lst name="defaults">
         <str name="echoParams">explicit</str>
         <!-- VelocityResponseWriter settings -->
         <str name="wt">velocity</str>
         <str name="v.template">browse</str>
         <str name="v.layout">layout</str>
         <str name="v.properties">velocity.properties</str>
            ...
</requestHandler>
```

To change the log settings, for example, log file name and `Log4J`, I can add the following to `velocity.properties` file under `/example/solr/collection1/conf/velocity`.

```
runtime.log = /Users/paolo/Desktop/solr4.6.0/logs/velocity.log
runtime.log.logsystem.class=org.apache.velocity.runtime.log.SimpleLog4JLogS
ystem
```

Navigating to http://example.com:8080/manager/html, I should see the *Solr* application is running and all is good. If not, I can check the *Tomcat* logs at `/apache-tomcat-7.0.50/logs` to see any error messages.

3.6 SHARDING AND REPLICATION

Solr is designed to provide a highly available, fault tolerant environment that can index data for searching. It's a system in which data is organized into multiple pieces, or shards, that can be housed on multiple machines, with replicas providing redundancy for both scalability and fault tolerance, and a *ZooKeeper* server that helps to manage the overall structure so that both indexing and search requests can be routed properly.

If I would like to try this setup but I don't have an access to multiple servers there are two options:

- Use *Amazon EC2* micro instances (it's free).
- Use virtualization – I recommend *VMWare player*. It's free and fast.

3.6.1 SHARDING

So, I already have an istance of *Solr* running on *Tomcat*. To enable *Solr* for Big Data I also need to download *Apache ZooKeeper* [9].

Before I do any configuration lets check the host name.

```
$ hostname
ubuntu
```

Now look for it in `/etc/hosts`

```
$ sudo vim /etc/hosts
```

If I find something like this:

```
127.0.1.1       ubuntu
```

I need to change the IP to our LAN IP address. It will make at least one of my *Solr* nodes to register as "127.0.1.1". Localhost address doesn't make any sense from cloud's point of view. That will populate multiple issues with replication and leader election. It's later hard to guess that all of those problems come from this silly source.

After have unpacked *ZooKeeper*, the easier job is to setup it. I will do it only once on the first server. *ZooKeeper* scales well and I can run it on all *Solr* nodes if required. There is no need for this at the moment so I can take single server approach.

Create a directory for *ZooKeeper* data and set configuration to point to that place.

```
$ sudo mkdir -p /var/lib/zookeeper
$ cd zookeeper-3.4.5/
$ cp conf/zoo_sample.cfg conf/zoo.cfg
$ vim conf/zoo.cfg
```

Find `dataDir` and paste appropriate path.

```
dataDir=/var/lib/zookeeper
```

Start ZooKeeper.

```
$ bin/zkServer.sh start
JMX enabled by default
Using config: /home/lukasz/zookeeper-3.4.5/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

If I like I can use *ZooKeeper* client to connect with the server.

```
$ bin/zkCli.sh -server 127.0.0.1:2181
[zk: 127.0.0.1:2181(CONNECTED) 0] ls /
[zookeeper]
```

Type "quit" to exit.

Now lets insert *Solr* configuration into *ZooKeeper*. Go to *Solr* directory and have a look into `solr-webapp`. It should be empty.

```
$ cd solr-4.6.0/example/
$ ls solr-webapp/
```

Please notice I am using the example directory. In real live I obviously want to rename it to something better. The same with collections. I am going to use the default collection1 for this project.

If my `solr-webapp` doesn't have `solr.war` inside run *Solr* for few seconds to make it extract the file.

```
# java -jar start.jar
2014-03-20 09:38:58.132:INFO:oejs.Server:jetty-8.1.8.v20121106
2014-03-20 09:38:58.150:INFO:oejdp.ScanningAppProvider:Deployment monitor
/root/solr-4.6.0/example/contexts at interval 0
2014-03-20 09:38:58.153:INFO:oejd.DeploymentManager:Deployable added:
/root/solr-4.6.0/example/contexts/solr-jetty-context.xml
2014-03-20 09:38:58.209:INFO:oejw.WebInfConfiguration:Extract
jar:file:/root/solr-4.6.0/example/webapps/solr.war!/ to /root/solr-
4.6.0/example/solr-webapp/webapp
```

After this line I can press `ctrl+c` to stop the server.

```
$ ls webapps/solr.war
webapps/solr.war
```

Now I can start uploading configuration to the *ZooKeeper*.

```
$ cloud-scripts/zkcli.sh -cmd upconfig -zkhost 127.0.0.1:2181 -d
solr/collection1/conf/ -n default1
$ cloud-scripts/zkcli.sh -cmd linkconfig -zkhost 127.0.0.1:2181 -collection
collection1 -confname default1 -solrhome solr
$ cloud-scripts/zkcli.sh -cmd bootstrap -zkhost 127.0.0.1:2181 -solrhome
solr
```

Now if I login to *ZooKeeper* and run "`ls /`" command I should see the uploaded data.

```
$ bin/zkCli.sh -server 127.0.0.1:2181

[zk: localhost:2181(CONNECTED) 0] ls /
[configs, zookeeper, clusterstate.json, aliases.json, live_nodes, overseer,
collections, overseer_elect]
```

```
[zk: 127.0.0.1:2181(CONNECTED) 1] ls /configs
[default1]

[zk: 127.0.0.1:2181(CONNECTED) 3] ls /configs/default1
[admin-extra.menu-top.html, currency.xml, protwords.txt, mapping-
FoldToASCII.txt, solrconfig.xml, lang, stopwords.txt, spellings.txt,
mapping-ISOLatin1Accent.txt, admin-extra.html, xslt, scripts.conf,
synonyms.txt, update-script.js, velocity, elevate.xml, admin-extra.menu-
bottom.html, schema.xml]

[zk: localhost:2181(CONNECTED) 4]  get /configs/default1/schema.xml

// content of our schema.xml

[zk: 127.0.0.1:2181(CONNECTED) 5] quit
Quitting…
```

This step is obviously not required but it's good to know what happens inside each service and how to get there.

If we are impatient then you can go to "solr-4.6.0/example/" and run the service.

```
$ java -DzkHost=localhost:2181 -jar start.jar
```

I am almost there. The last thing is to "tell" *Solr* to use *ZooKeeper*. I already know how to do it from command line. When I run *Solr* from an external container I have to edit `solr.xml`.

```
$ nano solr-4.2.1/example/solr/solr.xml
```

Find top tag called `solr` and add `zkHost` attribute.

```
<solr persistent="true" zkHost="127.0.0.1:2181">
```

While we are editing `solr.xml` go to cores tag and set `hostPort` attribute to 8080.

```
<cores adminPath="/admin/cores"
      defaultCoreName="collection1"
      host="${host:}"
      hostPort="8080"
      hostContext="${hostContext:}"
      zkClientTimeout="${zkClientTimeout:15000}">
```

Restart *Tomcat*, open web browser and go to http://SERVER01_IP:8080/manager/html. I will be asked for username and password that I set in the previous section.

Find "/solr" in Applications list and click on "start" in commands column. If it fails with a message "*FAIL – Application at context path /solr could not be started*" it's most likely permissions issue.

Once the service is running I can access it under http://SERVER01_IP:8080/solr/#/.

That was first server. To have a cloud I need at least one more.

```
cd <SOLR_DIST_HOME>
cp -r example node2
```

The steps are exactly the same with a difference I can skip everything related to *ZooKeeper*. Make sure to set correct IP address for zkHost in `solr.xml`.

Run second server and go to http://SERVER01_IP:8080/solr/#/~cloud. I should see two servers replicating Collection1.

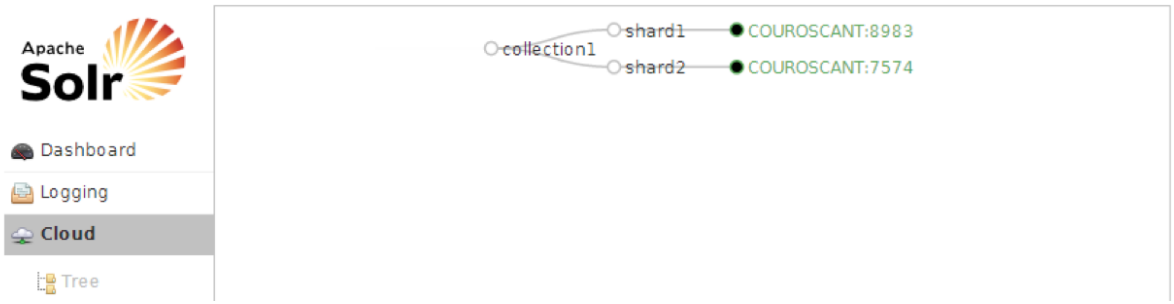


Figure 3.7 – Solr sharding.

Just to remind. If one of my servers has local IP like 127.0.1.1 there is a problem with my /etc/hosts file. If I made any mistake I can always start again. Stop *TomCat* servers, login to *ZooKeeper* and remove "*clusterstate.json*".

3.6.2 REPLICATION

Now that I have shards, I just have to make replication.

Start by creating two more fresh copies of the example directory:

```
cd <SOLR_DIST_HOME>
cp -r example node3
cp -r example node4
```

Just as when I created the first two shards, I can name these copied directories whatever I want.

So, in order to create these two replicas I need simple to redo the previous two sections. Notice that the parameters are exactly the same as they were for starting the second node; I am simply pointing a new instance at the original *ZooKeeper*. But if I look at the *Solr* admin page, I will see that it was added not as a shard, but as a replica for the first. This is because the cluster already knew that there were only two shards and they were already accounted for, so new nodes are added as replicas.

If I was to add additional instances, the cluster would continue this round-robin, adding replicas as necessary. Replicas are attached to leaders in the order in which they are started, unless they are assigned to a specific shard with an additional parameter of shardId (as a system property, as in -DshardId=1, the value of which is the ID number of the shard the new node should be attached to). Upon restarts, the node will still be attached to the same leader even if the shardId is not defined again (it will always be attached to that machine).

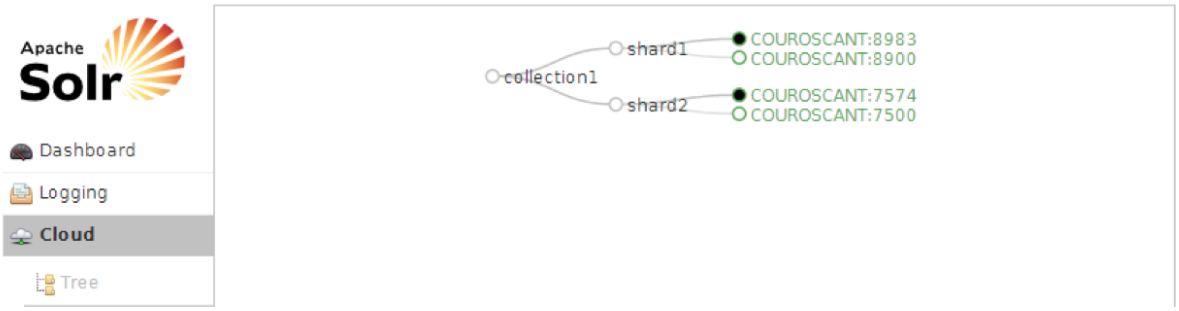


Figure 3.8 – Sharding and replication on Solr.

3.7 LAST STEP: RUNNING SOLR ON HDFS

Solr has support for writing and reading its index and transaction log files to the HDFS [10] distributed file system. This does not use *Hadoop Map-Reduce* to process *Solr* data, rather it only uses the HDFS file system for index and transaction log file storage.

3.7.1 BASIC CONFIGURATION

To use HDFS rather than a local file system, I must be using *Hadoop 2.0.x* and configure `solrconfig.xml` properly.

It's best to leave the data and update log directories as the defaults *Solr* comes with and simply specify the `solr.hdfs.home`. All dynamically created collections will create the appropriate directories automatically under the `solr.hdfs.home` root directory.

- Set solr.hdfs.home in the form hdfs://host:port/path
- I should specify a lock factory type of "hdfs" or none.

With the default configuration files, I can start *Solr* on HDFS with the following command:

```
java -Dsolr.directoryFactory=HdfsDirectoryFactory
     -Dsolr.lock.type=hdfs
     -Dsolr.hdfs.home=hdfs://host:port/path
```

3.7.2 THE BLOCK CACHE

For performance, the *HdfsDirectoryFactory* uses a Directory that will cache HDFS blocks. This caching mechanism is meant to replace the standard file system cache that *Solr* utilizes so much. By default, this cache is allocated off heap. This cache will often need to be quite large and you may need to raise the off heap memory limit for the specific JVM I am running *Solr* in. For the *Oracle/OpenJDK* JVMs, the follow is an example command line parameter that I can use to raise the limit when starting Solr:

```
-XX:MaxDirectMemorySize=20g
```

3.7.3 SETTINGS

The *HdfsDirectoryFactory* has a number of settings.

Solr HDFS Settings

Param	Example Value	Default	Description
solr.hdfs.home	hdfs://host:port/path/solr	N/A	A root location in HDFS for Solr to write collection data to. Rather than specifying an HDFS location for the data directory or update log directory, use this to specify one root location and have everything automatically created within this HDFS location.

Block Cache Settings

Param	Default	Description
solr.hdfs.blockcache.enabled	true	Enable the blockcache
solr.hdfs.blockcache.read.enabled	true	Enable the read cache
solr.hdfs.blockcache.write.enabled	true	Enable the write cache
solr.hdfs.blockcache.direct.memory.allocation	true	Enable direct memory allocation. If this is false, heap is used
solr.hdfs.blockcache.slab.count	1	Number of memory slabs to allocate. Each slab is 128 MB in size.

NRTCachingDirectory Settings

Param	Default	Description
solr.hdfs.nrtcachingdirectory.enable	true	Enable the use of NRTCachingDirectory
solr.hdfs.nrtcachingdirectory.maxmergesizemb	16	NRTCachingDirectory max segment size for merges
solr.hdfs.nrtcachingdirectory.maxcachedmb	192	NRTCachingDirectory max cache size

HDFS Client Configuration Settings

solr.hdfs.confdir pass the location of HDFS client configuration files - needed for HDFS HA for example.

Param	Default	Description
solr.hdfs.confdir	N/A	Pass the location of HDFS client configuration files - needed for HDFS HA for example.

Example

```
<directoryFactory name="DirectoryFactory"
class="solr.HdfsDirectoryFactory">
  <str name="solr.hdfs.home">hdfs://host:port/solr</str>
  <bool name="solr.hdfs.blockcache.enabled">true</bool>
  <int name="solr.hdfs.blockcache.slab.count">1</int>
  <bool name="solr.hdfs.blockcache.direct.memory.allocation">true</bool>
  <int name="solr.hdfs.blockcache.blocksperbank">16384</int>
  <bool name="solr.hdfs.blockcache.read.enabled">true</bool>
  <bool name="solr.hdfs.blockcache.write.enabled">true</bool>
  <bool name="solr.hdfs.nrtcachingdirectory.enable">true</bool>
  <int name="solr.hdfs.nrtcachingdirectory.maxmergesizemb">16</int>
  <int name="solr.hdfs.nrtcachingdirectory.maxcachedmb">192</int>
</directoryFactory>
```

Limitations

I must use an “append-only” *Lucene* index codec because HDFS is an append only file system. The currently default codec used by *Solr* is “append-only” and supported with HDFS.

4 IMPROVING SOLR WITH FACETING BROWSING

Solr comes by default with an own browsing web page that, like shown below, it hasn't a faceted navigation but a simple and unuseful parametric navigation such as described in *Chapter 2*.

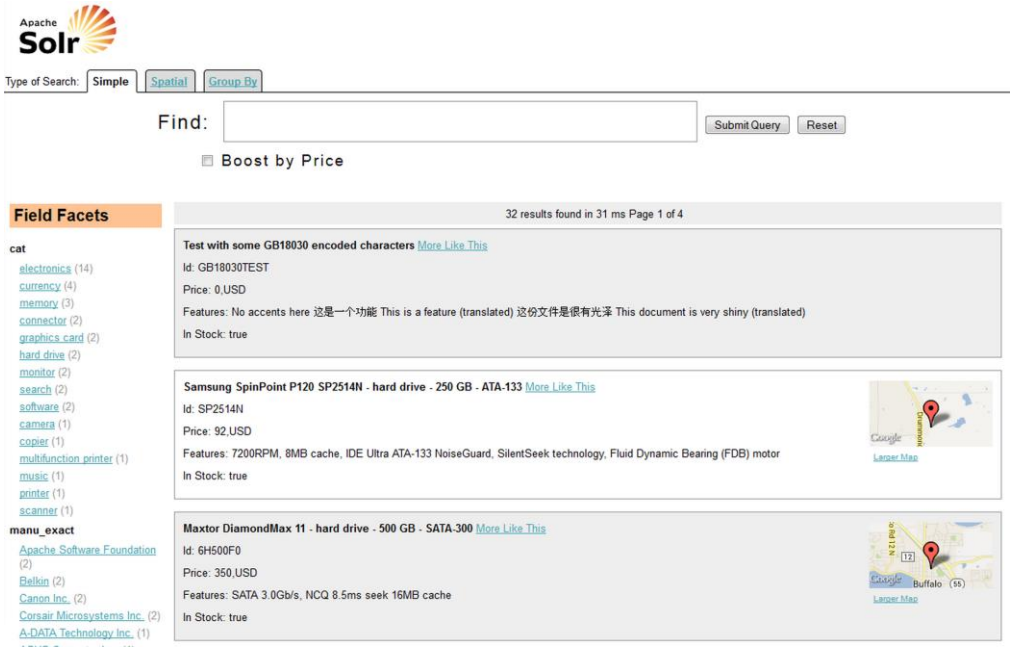


Figure 4.1 – Solr parametric navigation.

If one would like to have a faceted navigation allowing a better user search experience, there are some useful features to be emplyed:

- to use *Solr* like a search platform that returns a well format string (*Xml*, *jSon*, etc.) carrying on the result of a query;
- to use the provided *SolrJ* java client that offers a java interface to add, update, and query the *Solr* index.

On the other hand, this thesis proposes a different solution because I would like to obtain an integrated and ready to use visual application. In order to implement this, I need to learn how to generate a query to be executed by *Solr* that enables faceted browsing and, after this task, modifies the *Solr* front-end written in *Velocity* language. The query will be done and sent to *Solr* server using *Javascript*.

4.1 FACETED QUERY OVER SOLR

To understand how to create a well formed faceted query, first it is necessary to learn a more about the *Solr* query parser and the velocity response writer.

4.1.1 EDISMAX: EXTENDED QUERY PARSER

The *Extended DisMax Query Parser* (eDisMax) is a robust parser designed to process advanced user input directly. It is built on the original *DisMaxQParserPlugin* but adds many features. It searches for the query words across multiple fields with different boosts, based on the significance of each field. This parser supports full *Lucene QueryParser* syntax including boolean operators 'AND', 'OR', 'NOT', '+' and '-', fielded search, term boosting, fuzzy, grouping with parents, phrase search, phrase slop, numeric ranges, wildcard search and more. If there is a syntax error in the input, such as non-existing field name or unbalanced double-quotes, the input is gracefully searched as literal strings.

Extended DisMax is already configured in the example configuration, with the name *edismax*. Thus, to select the parser, you have to use defType=edismax in the query, or use the local-param syntax `{!edismax}`. *Solr* may provide virtual alias fields for users to query. This is useful either to provide a localized or easier name than what happens to be in the schema, or to provide an alias for a group of fields to support more advanced use cases such as "what" and "where" queries, even if there are no physical "what" or "where" fields.

The syntax for aliasing is `f.myalia.qf=realfield`. A user query for `myalias:foo` will be queried as `realfield:foo`.

The alias may also refer to multiple fields, with boost factors, by listing the field names with a space between them, and the optional boost factor immediately following the field name and the caret ('^') operator. Let's imagine I have a schema with fields `name`, `namealias`, `address`, `city`, `state`, and I want to provide a "who" and "where" search. I could then configure aliases like this: `&f.who.qf=name^5.0+namealias^2.0&f.where.qf=address^1.0+city^10.0+state`. Any user query for `who:foo` would expand to a DisMax query across fields name and namealias. If I further want to hide the real field names, I can combine this with "User Fields" feature, and say `&uf=who,where` to only allow fielded search for those two aliases. The following parameters are supported, either as regular request params, or as local params:

`q.alt` — If specified, this query will be used (and parsed by default using standard query parsing syntax) when the main query string is not specified or blank. This comes in handy when I need something like a match-all-docs query (don't forget &rows=0 for that one!) in order to get collection-wise faceting counts.

`qf` (Query Fields) — List of fields and the "boosts" to associate with each of them when building DisjunctionMaxQueries from the user's query. The format supported is `fieldOne^2.3 fieldTwo fieldThree^0.4`, which indicates that fieldOne has a boost of 2.3, fieldTwo has the default boost, and fieldThree has a boost of 0.4 ... this indicates that matches in fieldOne are much more significant than matches in fieldTwo, which are more significant than matches in fieldThree.

Local parameters provide a way to "localize" information about a specific argument that is being sent to *Solr*. In other words, LocalParams provide a way to add meta-data to certain argument types such as query strings. LocalParams are expressed as prefixes to arguments to be sent to *Solr*. For example, assume I have the existing query parameter

```
q=solr rocks
```

I can prefix this query string with LocalParams to provide more information to the query parser, for example changing the default operator type to “AND” and the default field to “title” for the query parser:

```
q={!q.op=AND df=title}solr rocks
```

To indicate a LocalParam, the argument is prefixed with curly braces whose contents begin with an exclamation point and include any number of `key=value` pairs separated by whitespace. So if the original argument is foo, applying LocalParams would look something like `{!k1=v1 k2=v2 k3=v3}foo`. There may only be one LocalParams prefix per argument, preventing the need for any escaping of the original argument. Values in the key-value pairs may be quoted via single or double quotes, and backslash escaping works within quoted strings.

Example:

```
q={!type=dismax qf=”myfield yourfield”}solr rocks
```

4.1.2 VELOCITY RESPONSE WRITER

V*elocityResponseWriter* enables *Solr* to respond with content generated from *Velocity* templates. Along with technologies like *SolrJ*, this makes *Solr* itself capable of driving sophisticated search interfaces without the need for an intermediate application server between the browser and *Solr*.

Thus, first to make a faceted query some configurations for this *ResponseWriter* have to be added to `solrconfig.xml` ,like this:

```
<queryResponseWriter
      name="velocity"
      class="solr.VelocityResponseWriter"
      startup="lazy"/>
```

Set up a RequestHandler in solrconfig.xml:

```
<requestHandler name="/browse" class="solr.SearchHandler">
      <lst name="defaults">
      <str name="v.properties">velocity.properties</str>
      <str name="echoParams">explicit</str>
      <!-- VelocityResponseWriter settings -->
      <str name="wt">velocity</str>
      <str name="v.template">browse</str>
      <str name="v.layout">layout</str>
      <str name="title">MushroomSolr</str>
      <!-- Query settings -->
      <str name="defType">edismax</str>
      <str name="qf">
              CapShape CapSurface CapColor Bruises Odor GillAttachment
GillSpacing GillSize GillColor StalkShape StalkRoot StalkSurfaceAboveRing
StalkSurfaceBelowRing StalkColorAboveRing StalkColorBelowRing VeilType
VeilColor RingNumber RingType SporePrintColor Population Habitat Class
```

```
      </str>
       <str name="df">text</str>
       <str name="mm">100%</str>
       <str name="q.alt">*:*</str>
       <str name="rows">5</str>
       <str name="fl">*,score</str>
 ...
</requestHandler>
```

4.1.3 FACETED QUERY

These are the parameters used to drive the Faceting behavior, grouped by the type of faceting they support.

Note that many parameters may be overridden on a per-field basis with the following syntax:

`f.<fieldName>.<FacetParam>=<value>`

eg.

```
facet.limit=10
f.category.facet.limit=5
```

Indicating that 10 terms will be returned for all specified facet.fields, excepting category. Which if specified will return 5 terms.

`facet`	Set to "true" this parameter enables facet counts in the query response. Any blank or missing value, or "false" will disable faceting. None of the other parameters listed below will have any effect without setting this param to "true". The default value is blank.
`facet.query` (Arbitrary Query Faceting)	This parameter allows me to specify an arbitrary query in the default query syntax to generate a facet count. By default, faceting returns a count of the unique terms for a "field", while facet.query allows me to determine counts for arbitrary terms or expressions. This parameter can be specified multiple times to indicate that multiple queries should be used as separate facet constraints. It can be particularly useful for numeric range based facets, or prefix based facets (i.e. price: `[* TO 500]` and `price:[501 TO *]`). To specify facet queries not in the default query syntax, prefix the facet query with the name of the query notation. For example, to use the hypothetical myfunc query parser, send parameter `facet.query={!myfunc}name~fred`

Several params can be used to trigger faceting based on the indexed Terms of a field. When using this param, it is important to remember that "Term" is a very specific concept in *Lucene* -- it relates to the literal field/value pairs that are indexed after any analysis occurs. For text fields that include stemming, or lowercasing, or word splitting you might not get what you expect.

`facet.field`	Term in the field and generate a facet count using that Term as the constraint. This parameter can be specified multiple times to indicate multiple facet fields. None of the other params in this section will have any effect without specifying at least one field name using this param.
`facet.prefix`	Limits the terms on which to facet to those starting with the given string prefix. Note that unlike fq, this does not change the search results -- it merely reduces the facet values returned to those beginning with the specified prefix.
`facet.sort`	This param determines the ordering of the facet field constraints. • `count` - sort the constraints by count (highest count first) • `index` - to return the constraints sorted in their index order (lexicographic by indexed term). For terms in the ascii range, this will be alphabetically sorted. The default is count if facet.limit is greater than 0, index otherwise.
`facet.limit`	This param indicates the maximum number of constraint counts that should be returned for the facet fields. A negative value means unlimited. The default value is 100.

The LocalParams syntax provides a method of adding meta-data to other parameter values, much like XML attributes. One can tag specific filters and exclude those filters when faceting. This is generally needed when doing multi-select faceting.

Consider the following example query with faceting:

```
q=mainquery&fq=status:public&fq=doctype:pdf&facet=on&facet.field=doctype
```

Because everything is already constrained by the filter `doctype:pdf`, the `facet.field=doctype` facet command is currently redundant and will return 0 counts for everything except `doctype:pdf`.

To implement a multi-select facet for doctype, a GUI may want to still display the other doctype values and their associated counts, as if the `doctype:pdf` constraint had not yet been applied. Example:

```
=== Document Type ===
    [ ] Word (42)
    [x] PDF  (96)
    [ ] Excel(11)
    [ ] HTML (63)
```

To return counts for doctype values that are currently not selected, tag filters that directly constrain doctype, and exclude those filters when faceting on doctype.

```
q=mainquery&fq=status:public&fq={!tag=dt}doctype:pdf&facet=on&facet.field={
                              !ex=dt}doctype
```

Filter exclusion is supported for all types of facets. Both the tag and ex local params may specify multiple values by separating them with commas.

To change the output key for a faceting command, specify a new name via the key local param. For example,

```
facet.field={!ex=dt key=mylabel}doctype
```

will cause the results to be returned under the key "mylabel" rather than "doctype" in the response. This can be helpful when faceting on the same field multiple times with different exclusions.

Therefore, to enable a multifaceted query I have to combine these tags:

TAG	**MEANING**
/solr/collection1/browse?	the url that show the *Solr* browsing web page
&q=	tag that identify a main query
&fq=	tag that identify a subquery
{!tag=dt}CapColor:brown	tag used to select a specific faced field and its facet value
%22OR%22	tag needed when are chosen more facet value of the same facet field
facet=on	tag to enable the faceted browsing
&facet.field=	tag to choose a facet field when faceted browsing is enabled
{!ex=dt}Bruises	tag to select a specific facet field to show when faceted browsing is enabled

The following example shows how to write a well format query for faceted browsing in *Solr*. Considering three facet fields, which they have different numbers of facet values each. I would like to perform a query that select two facet values of a same facet field and a single value from another facet field.

```
CAPCOLOR              (79)
     [x] Brown    (23)
     [x] Gray     (45)
     [ ] Red      (11)
CAPSURFACE            (18)
     [ ] Scaly     (7)
     [ ] Smooth   (11)
ODOR                 (183)
     [ ] None     (34)
     [x] Pungent  (80)
     [ ] Spicy    (65)
     [ ] Anise     (4)
```

So, in order to select these facet values in a faceted browsing context, is needed a query like this

```
http://localhost:8081/solr/collection1/browse?
   &q=&fq={!tag=dt}CapColor:brown%22OR%22{!tag=dt}CapColor:gray
      &q=&fq={!tag=dt}Odor:pungent
         &facet=on
            &facet.field={!ex=dt}CapColor
            &facet.field={!ex=dt}CapSurface
            &facet.field={!ex=dt}Odor
```

4.2 CODING

The previous section shows how to perform a multi facet query in *Solr*. In order to automatically enable it, the frond-end layout of *Velocity* has be to modified and a new *Javascript* that manages the query to be sent to the *Solr* server has to be written.

4.2.1 VELOCITY SIDE

To provide a web browsing like I have described and shown in chapter 2 I need to modify the `facet.field.vm` file that is located into `/example/solr/collection1/conf/velocity`.
As a first step, I have added checkboxes on the front-end layout of Solr instead of the hyper-links, whichs comes by default. In order to show checkboxes I can simple add html tags on `facet.field.vm`

```
<div style="display:none;">
$i
#set($url_2="{!tag=dt}$field.name:$facet.name")
$url_2
</div>

<input type="checkbox" id="facet_value$i" class="multifacet"
      value="$url_2" onchange="relocator(this)"/>
<label for="$i">
      <acronym id="facet_value$i" class="$url_2"
         onmouseover="showPopup(this)"
         onmouseout="closePopup(this)">
            $facet.name($facet.count)
      </acronym>
</label>
```

I have also added others lines of code into the velocity file to call the *Javascript* that will handle the user request and others codes that will be used by *Javascript* to perform a query.

```
<script type="text/javascript"
src="#{url_for_solr}/admin/file?file=/velocity/MULTIFACET.js&contentType=te
xt/javascript">
</script>
```

4.2.2 JAVASCRIPT SIDE

Rather than the Velocity code I have described above (that just shows a faceted layout), the *Javascript* code truly enables the faceted browsing and makes a multifaceted query. The script starts automatically when the DOM of the page is loaded and the first task is to make:

- One array that contains all the checkboxes
- Another one array that contains just the checkboxes already chosen
- Mark with true each checkbox chosen

```
/** Get all the elements with name "facet-field" in the web page.
 *  for each element, if its name is "facet-field" --> get its value.
 *  Only the <span className="facet-field">
 *  have also name="facet-   field".
 *  After this loop,
 *  I'll generate an array of elements that contains
 *  into each cell a field_value of my facet.
 */
for (var j=0, length = spanTags.length; j < length; j++) {
   field_arr[j] = spanTags.item(j).className;
}

/** Create a map of checkboxes (checkbox[id] = [value])
 *  but only these used for the facet_value
 */
for (var j=0, length = inputTags.length; j < length; j++) {
   if (inputTags[j].type == "checkbox" &&
          inputTags[j].className == "multifacet") {
      checkbox_map[z] = inputTags.item(j).value;
      z++;
   }
}

...
```

```
   /** Search into url_cut which checkboxes values are chosen or not
    *  and I'll mark it to true or false.
    */
   for (var j=0; j < number_of_checkbox; j++) {
      if (url_cut.search(checkbox_map[j]) != -1) {
         document.getElementById("facet_value"+j).checked=true;
      }
      else { document.getElementById("facet_value"+j).checked=false; }
   }
```

So, now the script will wait an "onchange" event from the checkboxes. When it will happen the script will bring the value of that checkbox chosen all it will call a function that handle the url query. The checkbox value contains the facet-field and the facet-value.

```
/** Function called from facet_field.vm
 *  that calls createUrl() function to allows multiface browsing.
 */
function relocator(selected) {
   var str= "facet_value";
   var id_checkbox = selected.id.substr(str.length);
   var variable_velocity = selected.value;

   var goTo = createUrl(id_checkbox, variable_velocity);

   /** window.location.replace("---") change the web page.
    */
   window.location.replace(goTo);
}
```

The core of this script is the `createUrl` function. This function is essentially split in three main pieces. The first piece of code set all the variables I need to perform a query and calculate other variables.

```
   var id_checkbox = id;
   var variable_velocity =  variable;
   var checkbox_status = false;
   ...
   var brws = "browse";
   var append1_1 = "?&q=";
   var append1_2 = "&q=";
   var append2 = "&fq=";
```

```
   var append3 = "&facet=on";
   var append4 = "&facet.field={!ex=dt}";

   var count_tag = 0;

   /** This is the first part of my new_url.
    *  see the url got from the command "window.location.href"and
    *  when I find this sequence of chars "{!tag=dt}",
    *  first thing I'll increase the variable count_tag.
    *  In this way I can know if I've already handled the url.
    */
   new_url = url;
   /** If I use new_url.search("{!tag=dt}") instead new_url.charAt(j)
    *  I can't count how many times {!tag=dt} is written
    *  into the url query.
    *  It's used at the end of this code to check url.
    */
   for (var j=0, length = new_url.length; j<length; j++) {
      if (new_url.charAt(j) == "{" &&
            new_url.charAt(j+1) == "!" &&
             new_url.charAt(j+2) == "t" &&
                new_url.charAt(j+3) == "a" &&
                   new_url.charAt(j+4) == "g" &&
                      new_url.charAt(j+5) == "=" &&
                         new_url.charAt(j+6) == "d" &&
                            new_url.charAt(j+7) == "t" &&
                              new_url.charAt(j+8) == "}") {
         count_tag ++;
      }
   }
...
```

The second or the third piece of this script makes really the url query. The difference between these two parts of code is that one is used to make the url when a checkbox is selected and the other one makes the url when a checkbox is unselected.

```
...
   /** Second thing.
    *  For each element into field_arr I've found above,
    *  I'll concat it with append4.
    *  FOR EXAMPLE:
    *  url_append       &facet=on
    *  url_append
    *  &q=facet=on
```

```
     *  &facet.field={!ex=dt}Bruises&facet.field={!ex=dt}CapColor...
     *  This loop 'll generate a long string but it'sn't a issue.
     */
       for (var j=0, length = field_arr.length; j<length; j++) {
          url_append += append4 + field_arr[j];
       }

    /** This is the last part of my new_url,
     *  I'll append url_append to new_url.
     *  FOR EXAMPLE:
     *  new_url    /solr/collection1/browse?&q=&fq={!tag=dt}CapColor:brown
     *  url_append
     *  &q=facet=on
     *  &facet.field={!ex=dt}Bruises&facet.field={!ex=dt}CapColor...
     *  --------------------
     *  new_url    /solr/collection1/browse?&q=&fq={!tag=dt}CapColor:brown
     *  &q=facet=on
     *  &facet.field={!ex=dt}Bruises&facet.field={!ex=dt}CapColor...
     */
     new_url +=  url_append;
...
```

There are also a lot of other parts of code that handle queries like, for example, to perform a disjunction

```
new_url = new_url.substr(0,pos2+old_variable.length) +
             "%22OR%22" +
                variable_velocity +
                   new_url.substr(pos2+old_variable.length);
```

but, essentially, this is how my script works.

In the end, now, I have provided a complete faceted browsing in *Solr*.

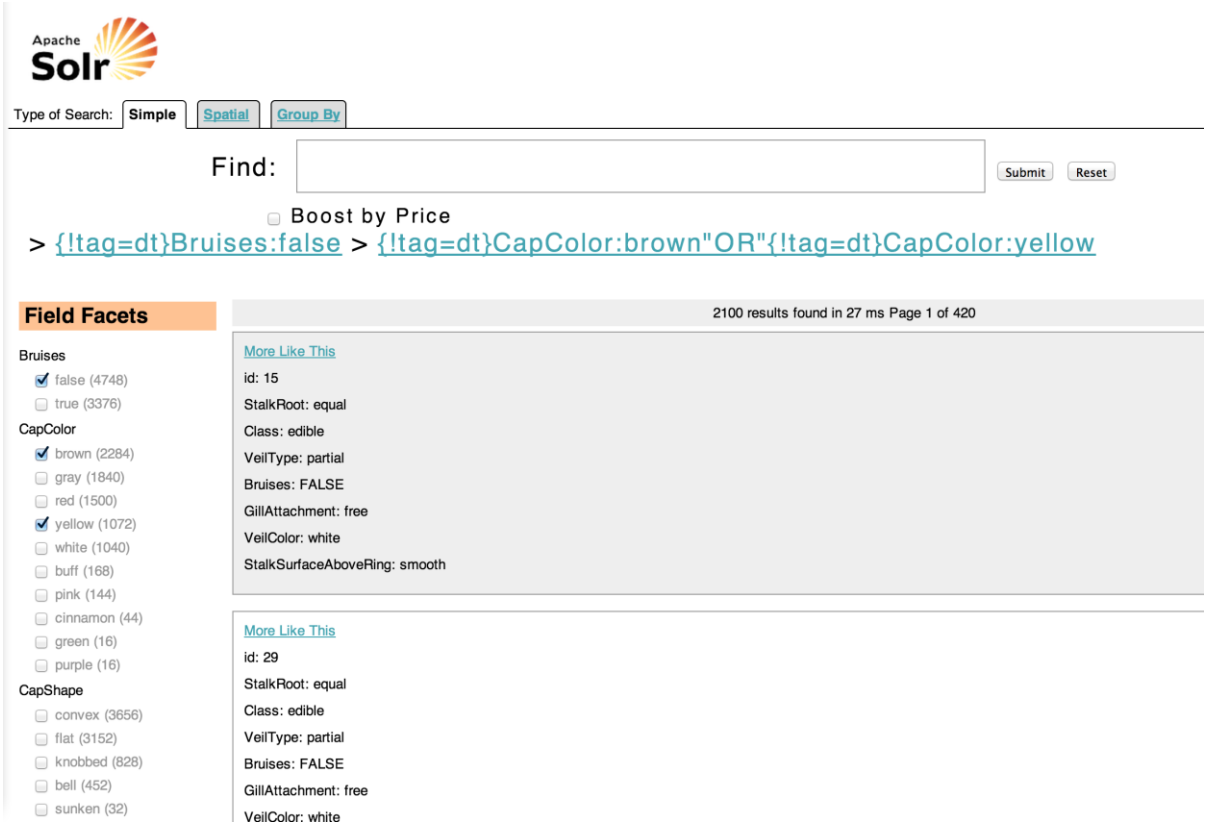


Figure 4.2 – Multifaceted browsing over Solr.

5 BAYESIAN NETWORKS

5.1 WHAT CAN ONE DO WITH A BAYESIAN NETWORKS ?

Similar to any modeling language, the value of *Bayesian Networks* is mainly tied to the class of queries they support.

Consider the network in Figure 5.1 for an example and the following queries: Given a male that came out positive on both tests, what is the probability he has the condition? Which group of the population is most likely to test negative on both tests? Considering the network in Figure 5.2: What is the overall reliability of the given system? What is the most likely configuration of the two fans given that the system is unavailable? What single component can be replaced to increase the over-all system reliability by 5%? Consider Figure 5.3: What is the most likely channel input that would yield the channel output 1001100?

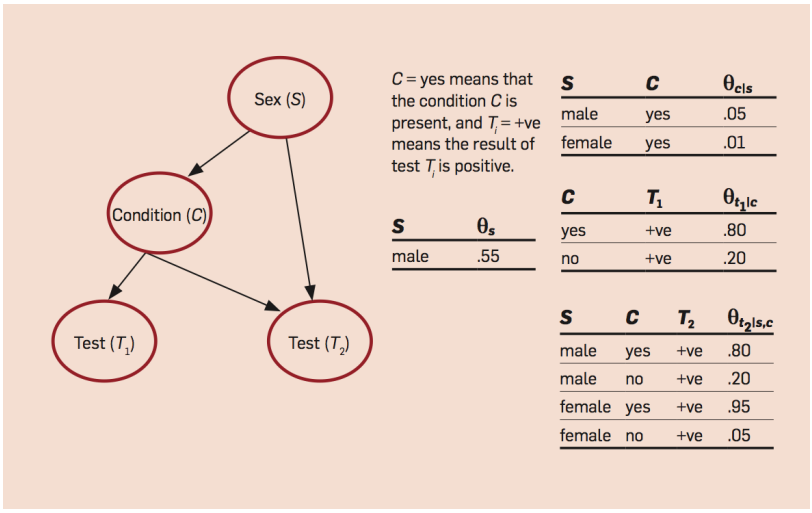
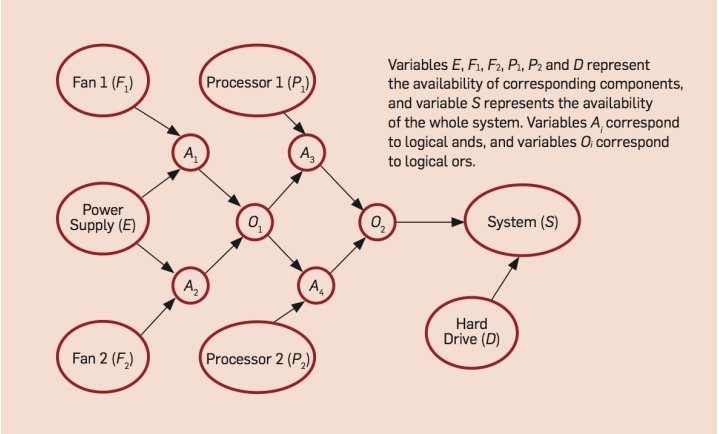


Figure 5.1 – A Bayesian network that models a population, a medical condition, and two corresponding tests.

These are example questions that would be of interest in these application domains, and they are questions that can be answered systematically using three canonical Bayesian network queries [11]. A main benefit of using Bayesian networks in these application areas is the ability to capitalize on existing algorithms for these queries,



instead of having to invent a corresponding specialized algorithm for each application area.

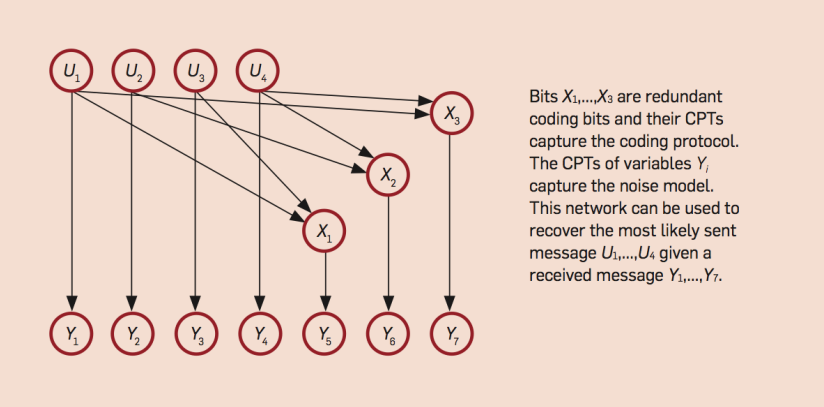


Figure 5.2 – A Bayesian network generated automatically from a reliability block diagram.

Figure 5.3 – A Bayesian network that models a noisy channel with input (U_1,...,U_4, X_1,...,X_3) and output (Y_1,...,Y_7).

Probability of Evidence

This query computes the probability Pr(e), where e is an assignment of values to some variables E in the Bayesian network— e is called a variable instantiation or evidence in this case. For example, in Figure 5.1, I can compute the probability that an individual will come out positive on both tests using the probability-of-evidence query *Pr*(T_1 = `+ve`, T_2 = `+ve`). I can also use the same query to compute the overall reliability of the system in Figure 5.2, *Pr*(S = `avail`). The decision version of this query is known to be PP-complete. It is also related to another common query, which asks for computing the probability *Pr*(`x|e`) for each network variable X and each of its values x. This is known as the node marginals query.

Most Probable Explanation (MPE)

Given an instantiation e of some variables E in the Bayesian network, this query identifies the instantiation q of variables Q = E that maximizes the probability *Pr* (`q|e`). In Figure 5.1, I can use an MPE query to find the most likely group, dissected by sex and condition, that will yield negative results for both tests, by taking the evidence e to be T_1 = `-ve;` T_2 = `-ve` and Q={S,C}. I can also use an MPE query to restore images as shown in Figures 5.4. Here, I take the evidence e to be C_{ij} = `true` for all i,j and Q to include P_i for all i. The decision version of MPE is known to be NP-complete and is therefore easier than the probability-of-evidence query under standard assumptions of complexity theory.

Maximum a Posteriori Hypothesis

Given an instantiation e of some variables E in the Bayesian network, this query identifies the instantiation q of some variables Q ⊆ E that maximizes the probability *Pr*(`q|e`). Note the subtle difference with MPE queries: Q is a subset

(MAP) of variables E instead of being all of these variables. MAP is a more difficult problem than MPE since its decision version is known to be NP^{PP} -complete, while MPE is only NP- complete. As an example of this query, consider Figure 5.2 and suppose I are interested in the most likely configuration of the two fans given that the system is unavailable. I can find this configuration using a MAP query with the evidence e being S = `un_avail` and Q = {F1, F2}.

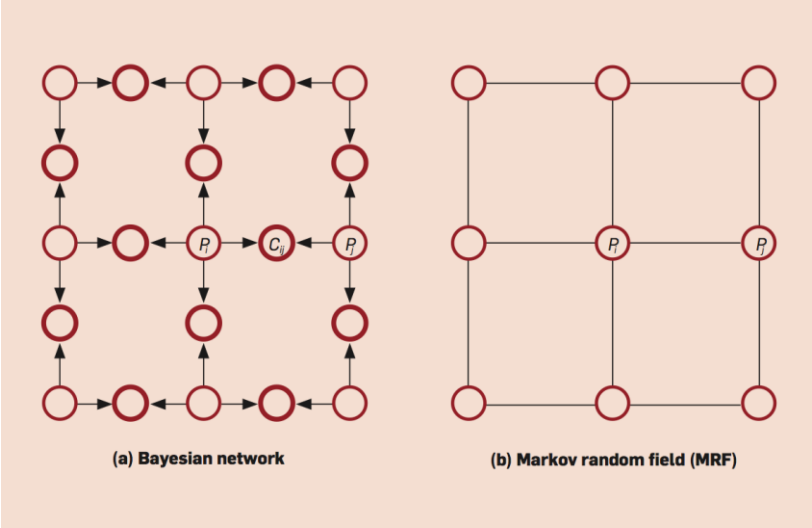
One can use these canonical queries to implement more sophisticated queries, such as the ones demanded by sensitivity analysis. Sensitivity analysis can also be used for automatically revising these parameters in order to satisfy some global constraints that are imposed by experts, or derived from the formal specifications of tasks under consideration. Suppose for example that I compute the overall system reliability using the network in Figure 5.2 and it turns out to be 95%. Suppose I wish this reliability to be no less than 99%: $Pr(S = \texttt{avail}) \geq 99\%$. Sensitivity analysis can be used to identify components whose reliability is relevant to achieving this objective, together with the new reliabilities they must attain for this purpose. Note that component reliabilities correspond to network parameters in this example.

5.2 BEYOND BAYESIAN NETWORKS

Viewed as graphical representations of probability distributions, Bayesian networks are only one of several other models for this purpose. In fact, in areas such as statistics and in AI, Bayesian networks are studied under the broader class of probabilistic graphical models (PGMs), which include other instances such as Markov networks and chain graphs (for example, Edwards [12] and Koller and Friedman [13]). Markov networks correspond to undirected graphs, and chain graphs have both directed and undirected edges. Both of these models can be interpreted as compact specifications of probability distributions, yet their semantics tend to be less transparent than Bayesian networks. For example, both of these models include numeric annotations, yet one cannot interpret these numbers directly as probabilities even though the whole model can be interpreted as a probability distribution. Figure 5.4b depicts a special case of a Markov network, known as a Markov random field (MRF), which is typically used in vision applications. Comparing this model to the Bayesian network in Figure 5.4a, one finds that smoothness constraints between two adjacent pixels P_i and P_j can now be represented by a single undirected edge $P_i - P_j$ instead of two directed edges and an additional node, $P_i \rightarrow C_{ij} \leftarrow P_j$. In this model, each edge is associated with a function $f(P_i, P_j)$ over the states of adjacent pixels. The values of this function can be used to capture the smoothness constraint for these pixels, yet do not admit a direct probabilistic interpretation.

Bayesian networks are meant to model probabilistic beliefs, yet the interest in such beliefs is typically motivated by the need to make rational decisions. Since such decisions are often contemplated in the presence of uncertainty, one needs to know the likelihood and utilities associated with various decision outcomes. A classical example in this regard concerns an oil wildcatter that needs to decide whether or not to drill for oil at a specific site, with an additional decision on whether to request seismic soundings that may help determine the geological structure of the site. Each of these decisions has an associated cost. Moreover, their potential outcomes have associated utilities and probabilities. The need to integrate these probabilistic beliefs, utilities and decisions has lead to the development of Influence Diagrams, which are extensions of Bayesian networks that include three types of nodes: chance, utility, and decision. Influence diagrams, also called decision networks, come with a toolbox

that allows one to compute optimal strategies: ones that are guaranteed to produce the highest expected utility [13].



Bayesian networks have also been extended in ways that are meant to facilitate their construction. In many domains, such networks tend to exhibit regular and repetitive structures, with the regularities manifesting in both CPTs and network structure. In these situations, one can synthesize large Bayesian networks automatically from compact high-level specifications. A number of concrete specifications have been proposed for this purpose. For example, template-based approaches require two components for specifying a Bayesian network: a set of network templates whose instantiation leads to network segments, and a specification of which segments to generate and how to connect them together [13]. Other approaches include languages based on first-order logic, allowing one to reason about situations with varying sets of objects.

Figure 5.4 – Modeling low-level vision problems using two types of graphical models: Bayesian networks and mRfs.

6 IMPROVING SOLR WITH BAYESIAN NETWORKS

In chapters 2 and 5 I have described that a Big Data context could need more sophisticated solutions than the usual context. These features are due to the enormous amount of data that could generate confusion to the user but also because efficient solutions can improve al lot the user experience.

The current chapter of this thesis is devoted to join the two previous solutions implemented on *Solr* (Big Data scenario and Faceted Browsing) with a search aware path for a better user experience, provided by Bayesian Networks.

6.1 BAYESIAN NETWORKS TOOLS

Probabilistic Graphical models (PGMs) are a way to represent conditional independence assumptions using graphs. Nodes represent random variables and lack represent conditional independencies. Specially, they consists of a probability distribution, defined on a set of variables, and a graph, such that each node in the graph represents a variable and the graph represents (some of) the independencies of the probability distribution. The graph is a useful visual representation of complex stochastic system. The graphical structure is also the basis of efficient inference algorithm. There are many kind of different graphical models, but the two most popular ones are based on *directed acyclic graphs* (also called "Bayesian Networks") and on *undirected graphs* (also called "Markov Random Fields").

Moreover, the rapid development of Bayesian network research over the past 15 years has been accompanied by a proliferation of BN software tools. A comparison of the most popular tools can be found at http://www.cs.ubc.ca/~murphyk/Software/bnsoft.html.

The choice of a tool among the most popular ones has been based on these following concerns:

- source code open and free
- API available
- GUI available
- code written in Java

After have tried the tools which corresponded to these characteristics, the chosen was fall on *Open Markov* [14].

6.1.1 OPEN MARKOV

Open Markov is an open source software tool developed at the *Research Center on Intelligent Decision-Support Systems* [15] of the *Universidad Nacional de Educación a Distancia* [16], in Madrid, Spain.

The project started in 2003 and its original name was *Carmen*, but in 2010 it was renamed as *OpenMarkov*. It departed from the experience in the construction of *Elvira*, an open-source tool begun in 1997 as a join project of several Spanish universities, but everything in the new software was redesigned and the code of *OpenMarkov* was built from scratch. The development language for *OpenMarkov* is Java, mainly in order to allow it to run it on different platforms.

A probabilistic network is represented in *OpenMarkov* as a generic data structure consisting mainly of a graph, a set of variables, and a set of potentials. Each type of network is defined by a set of constraints. It is possible to implement new types of networks easily by combining the existing constraints and, if necessary, by adding new ones.

OpenMarkov accepts three types of variables: finite-states, numerical, and discretized, and two types of links: directed and undirected. It also has several types of potentials: uniform (mainly used to assign a default potential to each node in networks that only contain directed links), table (the most frequently used potential), delta (i.e., Kronecker delta for finite-state variables and Dirac delta for numeric variables), tree/ADD, several canonical models (OR, AND, MAX, MIN, etc.), sum, product, linear combination, conditional Gaussian, exponential, mixture of exponentials, and logistic regression. There are also three potentials for dynamic models: same as previous, cycle length shift, and Weibull distribution.

OpenMarkov is able to represent several types of networks, such as Bayesian networks, Markov networks, influence diagrams, LIMIDs, and decision analysis networks (DANs), as well as several types of temporal models: dynamic Bayesian networks, simple Markov models, MDPs, POMDPs, Dec- POMDPs, and dynamic LIMIDs [1]. Currently *OpenMarkov* can only evaluate Bayesian networks, influence diagrams, and simple Markov models, but it can be used to build several types of models, such as (PO)MDPs, that might be read by any other tool able to parse the *ProbModelXML* format.

OpenMarkov has integrated the most usual algorithms for Bayesian networks and influence diagrams, such as *variable elimination* and *clique tree propagation*. There are some algorithms for evaluating MDPs, such as value iteration, policy iteration, and modified policy iteration.

In *OpenMarkov* it is possible to learn Bayesian networks interactively from databases by applying the two most popular methods: *search-and-score*, with several metrics, and the *PC algorithm* [17].

6.2 LEARNING A BAYESIAN NETWORK

There are two main ways to build a Bayesian network. The first one is to do it manually, with the help of a domain expert, defining a set of variables that will be represented by nodes in the graph and drawing causal arcs between them, as explained in Section. The second method is to do it automatically, learning the structure of the network (the directed graph) and its parameters (the conditional probabilities) from a dataset, as explained in Section 2.2.

There is a third approach, interactive learning, in which an algorithm proposes some modifications of the network, called *edits* (typically, the addition or the removal of a link), which can be accepted or

[1] Three of these models have been proposed originally by CISIAD group: DANs, simple Markov models, and dynamic LIMIDs.

rejected by the user based on their common sense, their expert knowledge or just their preferences; additionally, the user can modify the network at any moment using the graphical user interface and then resume the learning process with the edits suggested by the learning algorithm. It is also possible to use a model network as the departure point of any learning algorithm, or just to indicate the positions of the nodes in the network learned, or to impose some links, etc. However, this approach is not deeply explained as for the goal of the project the best choice was the second approach.

When learning any type of model, it is always wise to gain insight about the dataset by inspecting it visually. The networks used in this thesis are in the format *Comma Separated Values (CSV)*. They can be opened with a text editor, but this modality it is very difficult to see the values of the variables. A better alternative is to use a spreadsheet, such as *OpenOffice Calc* or *LibreOffice Calc*. *Microsoft Excel* does not open these files properly because it assumes that in .csv files the values are separated by semicolons, not by commas; a workaround to this problem is to open the file with a text editor, replace the commas with semicolons, save it with a different name, and open it with Microsoft Excel.

In this thesis I will learn the *Mushroom network* with the *Hill Climbing* algorithm, also known as *search-and-score* [18]. As a dataset, I will use the file *MushroomFullNamewithID.csv*, which contains about 5000 of a mushroom dataset. The following are the steps for the learning activity.

1. Download the runnable jar from http://www.openmarkov.org/org.openmarkov.full-0.1.3.jar.
2. Run it on *Terminal*

```
java – jar org.openmarkov.full-0.1.3.jar
```

3. In *OpenMarkov*, select the Tools. Learning option. This will open the Learning dialog, as shown in Figure 6.2.
4. In the Database field, select the file *MushroomFullNamewithID.csv*.
5. Observe that the default learning algorithm is Hill climbing and the default options for this algorithm are the metric K2 and a value of 0.5 for the α parameter, used to learn the numeric probabilities of the network (when α = 1, we have the Laplace correction). These values can be changed with the Options button.
6. In *OpenMarkov* there are also three options for preprocessing the dataset:
   - selecting the variables to be used for learning,
   - discretizing the numeric variables (or some of them)
   - treating missing values

   In this project I will use only the first one. In the tab Preprocessing, select Use selected variables and uncheck the box of the variable ID, as shown in Figure 6.1. (Many databases contain administrative variables, such as the ID, etc., which are irrelevant for diagnosis and therefore should be excluded when learning a model.) If I had specified a model network, the option Use only the variables in the model network would be enabled.

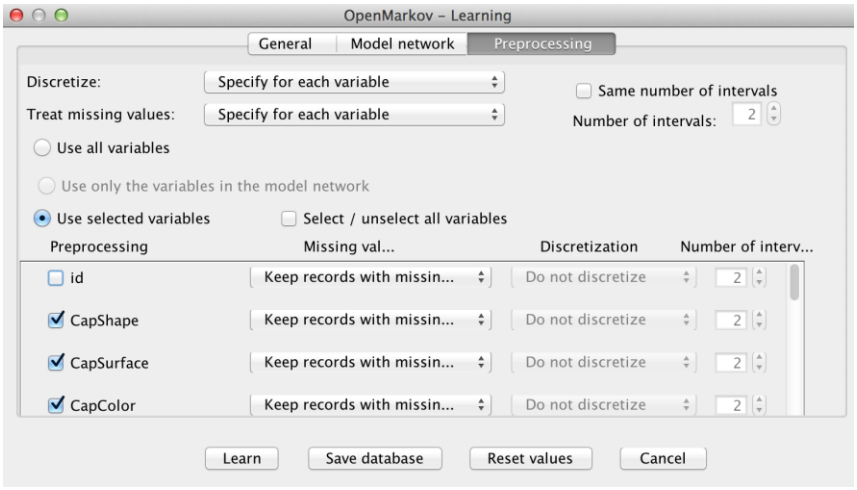


Figure 6.1 – Selection of variables.

7. Select Automatic learning and click Learn.

Figure 6.2 – OpenMarkov's Learning dialog.

The learning algorithm builds the networks and *OpenMarkov* arranges the nodes in the graph in several layers so that all the links point downwards—see Figure 6.3.

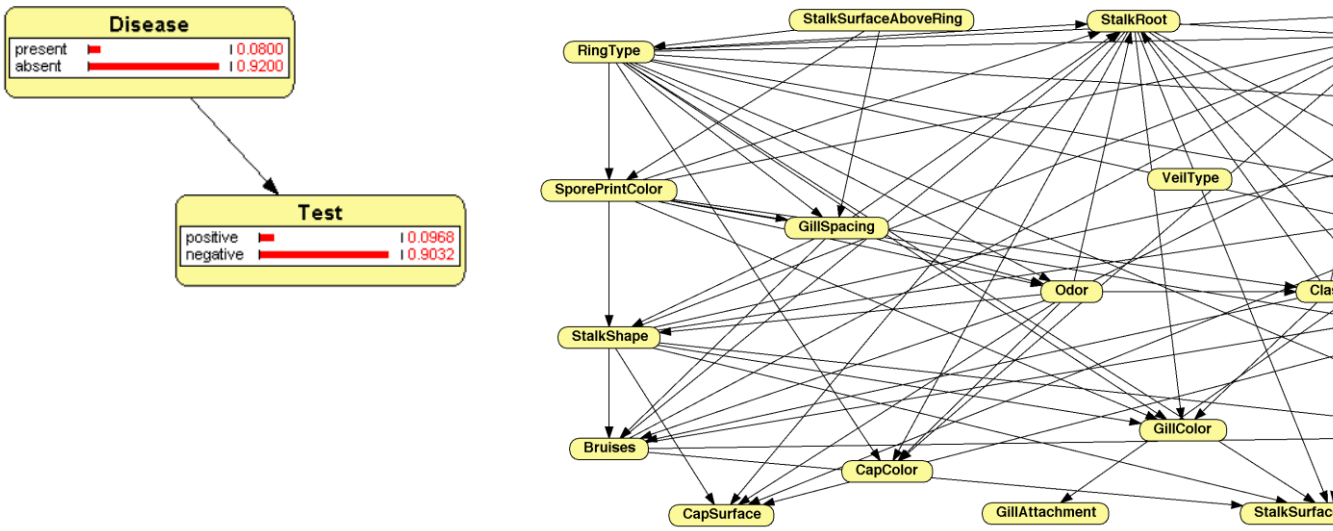
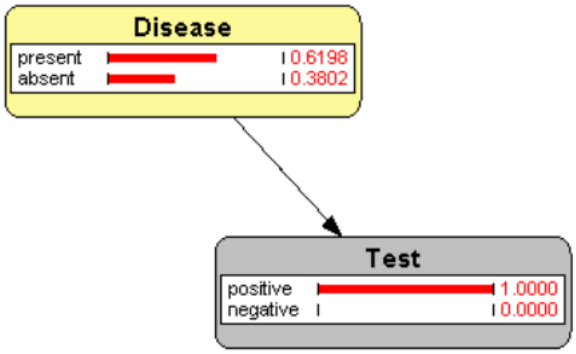


Figure 6.3 – Network Mushroom learned automatically.

6.2.1 INFERENCE AND FINDING



The Mushroom file contains so many variables so, for this section, I will use a simple example. Remind that a *finding* consists of the assignment of a value to a variable as a consequence of an observation. A set of findings is called an *evidence* case. The probabilities conditioned on a set of findings are called *posterior probabilities*.

Click the inference button ("bolt" button) to switch from edit mode to inference mode. As the option propagation is set to automatic by default, *OpenMarkov* will compute and display the prior probability of the value of each variable, both graphically (by means a horizontal bars) and numerically – see Figure 6.4. Note that the Edit toolbar has been replace by the Inference toolbar, whose button as described below.

Figure 6.4 – Prior probabilities for the network disease-test.pgmx.

For example, a finding may be the test has given a positive result: `Test = positive`. Introduce it by double-clicking on the state positive of the node Test (either on the string, or on the bar, or on the numerical value) and observe that the result is similar to Figure 6.5: the node Test is colored in gray to denote the existence of a finding and the probabilities of its states have changed to 1.0 and 0.0 respectively. The probabilities of the node Disease have changed as well, showing that

P(`Disease=present | Test=positive`) = 0.6198 and P(`Disease=absent | Test=positive`) = 0.3802.

Therefore, I can answer that the *positive predictive value* (PPV) of the test is 61.98%.

Figure 6.5 – Posterior probabilities for the evidence case.

An alternative way to introduce a finding would be to right-click on the node and select the Add finding option of the contextual menu. A finding can be removed by double-clicking on the corresponding value or by using the contextual menu. It is also possible to introduce a new finding that replaces the old one.

6.3 OPEN MARKOV API

OpenMarkov API is written in Java as a *BitBucket* repository. Thus, in order to develop and modify the API, I need to clone it on my computer using *Mercurial* as a *Maven* project.

The *ExampleAPI* can be found at https://bitbucket.org/cisiad/org.openmarkov.exampleapi/src.

ExampleAPI comes like a runnable program that calculates probabilities of a given Bayesian Network, read from a .pgmx file. Essentially, two main parts structure it:

1. The first function set all the variables, read the BN file and, mostly important,
   - add, modify or remove finding
   - call the algorithm to compute the posterior probabilities. I will use the *Variable Elimination* algorithm
   - call the print function
2. The second function calculates all the posterior probabilities and prints the results

In order to use it for my project, I had to modify the API adding code to implement algorithms and modifying few lines of code to allow me to use it inside a servlet container.

6.4 CODING

The algorithm proposed below has the purpose to notify the user about the current search path and the possible next search path that could be became true if the user hovers on a facet value meanwhile he/she are making a search on *Solr*.

To allow me to call this algorithm I have to write a client side and a server side code.

6.4.1 CLIENT SIDE: JAVASCRIPT

When a user hovers on facet value, the Velocity file of the web page call a script that it sends a *Ajax* request to a servlet and, when the script will have received the response from the servlet, it will show a popup that it will show some useful information about the current research path. The code is very simple like is shown below.

```
...
/** Create the url where servlet lives.
 */
 var url_servlet = window.location.href;
 url_servlet = url_servlet.substr(0,url_servlet.search(solr)) + path1;
```

```
/** Call java servlet and return a html string to show the summmary.
 */
pageRequest.open("POST", url_servlet, false);
var string_to_servlet =
   "NUMBER"+choices.length+
      "PRE"+choices+
         "POST"+variable_velocity;
pageRequest.send(string_to_servlet);
var pageStringHtml = pageRequest.responseText;
popup = window.open("","","width="+w+",
                      height="+h+",top="+t+",left="+l+",scrollbars=no
                      menubar=no status=no titlebar=no location=no");
popup.document.write(pageStringHtml);
...
```

The script sends to the servlet the currents facet values chosen by the user and the possible, next, facet value, which the user has hovered.

6.4.2 SERVER SIDE: BAYESIAN SERVLET

The servlet brings the body of the *Restful* request done by the Javascript and it calls the Bayesian code. The Bayesian code, which is been made from the *Open Markov* API, makes a result that it will be resend to the servlet. At this point, the servlet will format the result and send back the result to the script and it will show the final result.

```
public class PopupServlet extends HttpServlet {

    run.external.jar.RunJar read;
    open.markov.OpenMarkovPopup popup;

protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException, ParserException,
InterruptedException {
...
   payloadRequest = getBody(request);

...
   /**
    * Get the path Servlet lives
    */
```

```
   String path = getServletContext().getRealPath("/");

   popup = new open.markov.OpenMarkovPopup();
   popup.OpenMarkovPopup(preField, preField, postField, postField);
   String done = popup.getResult();

   /**
    * Make the html response web page
    */
...

   try {
        toClient.println(html);
   } finally {
        toClient.close();
   }
}

    protected static String getBody(HttpServletRequest request) throws
IOException {
...
    }

@Override
    protected void doGet(...) {
...
    }

@Override
    protected void doPost(...) {
...
    }
}
```

A full discussion about the algorithms inside the API code will be show in the next section.

6.5 ALGORITHMS

I come now to the second optimization of the user experience made during the research project: the inclusion in the selection panel of *Solr* of an interactive recommendation system which can guide you

in your choice of querying more in line with its search requirements. To complicate the exposure of the products and the increase of the number and characteristics of the same, the user search experience can become complicated if you do not have a complete knowledge of the product you are looking for, so the novice user can choose one product over another look and you will not be able to understand where this choice will lead him.

To solve this problem and to make the process of selection of search queries easy and efficient as possible the process of selection of search queries I have developed an interactive system of recommendation that acts in this way: hover on a facet in *Solr* selection panel will communicate a Javascript function a servlet to the current query and the facets that the user intends to select by hovering over them with the mouse, then the servlet API calls the law of *OpenMarkov* reading this information and loads the previously calculated Bayesian network through the GUI and calculates *OpenMarkov* global probability twice, once for the current query and insert each value in the PRE array and one for the query which is added as a featured selection of the facet on which the user is passed over it with the mouse and the values will be inserted into the array POST. So for each probability the standard deviation in the matrix PRE and POST is calculated, whereupon the values in the matrix are compared with the SD and if exceed a certain threshold value, I can define if the facet can fit into the category ADDED, UNALTERD or DELETED. Obtained from the three arrays ADDED, DELETED and UNALTERED, I order them by the greatest probability to the lowest, and for each category I select the top 5 values that will be sent to the servlet through another Javascript function that will display these facets obtained in tabular format in a popup.

The user then is facilitated in his request because every hover over each facet will have real-time knowledge of how the eventual selection will affect the search.

Fugure 6.6 the final result and shows how the algorithm works.

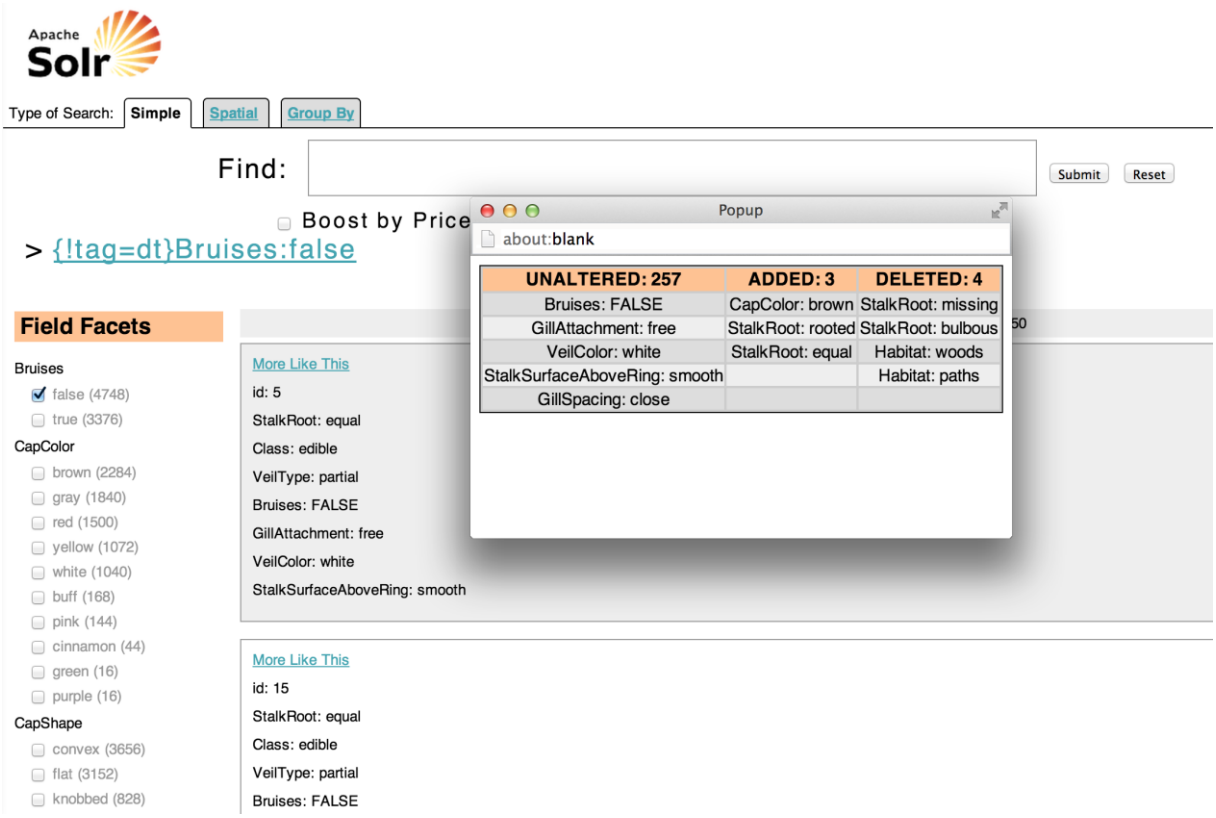


Figure 6.6 – Bayesian algorithm.

6.5.1 STANDARD DEVIATION

I have used the *Standard Deviation* (SD) because a its useful property is that, unlike the variance, it is expressed in the same units as the data.

```
postProbabilities = new double[variablesOfInterest.size()][maxNumValues];
postAverage = new double[variablesOfInterest.size()];
postStandardDeviation = new double[variablesOfInterest.size()];
meanStandardDeviation = new double[variablesOfInterest.size()];
double[] values;
double postDelta = 0;
for (int i = 0; i < variablesOfInterest.size(); i++) {
   for (int j = 0; j < maxNumValues; j++) {
      postProbabilities[i][j] = 0;
    }
}
/**
 * Set the 2 previous matrix and calculate the post mean and the
 * post standard deviation for each field
 */
for (int i = 0; i < variablesOfInterest.size(); i++) {
   Variable variable = variablesOfInterest.get(i);
   TablePotential posteriorProbabilitiesPotential =
                              posteriorProbabilities.get(variable);
   values = posteriorProbabilitiesPotential.getValues();
   for (int j = 0; j < values.length; j++) {
      if (j == 0) {
         postAverage[i] = 0;
      }
      postProbabilities[i][j] = values[j];
      postAverage[i] += values[j];
   }
   postAverage[i] = postAverage[i] / values.length;
   for (int j = 0; j < values.length; j++) {
      postDelta = values[j] - postAverage[i];
      postStandardDeviation[i] += Math.pow(postDelta, 2);
   }
   postStandardDeviation[i] =
            Math.sqrt(postStandardDeviation[i] / (values.length - 1));
```

```
  /**
   * Compute also the mean from the two standard deviation for
   * each field
   */
   meanStandardDeviation[i] =
                  (preStandardDeviation[i] + postStandardDeviation[i]) / 2;
}
```

6.5.2 THRESHOLD

This section explains how I can choice if a probability belongs to ADDED, DELETED or UNALTERED. As I have described above, I have used the standard deviation (SD) (represented by the Greek letter sigma, σ) because in statistics and probability theory, it shows how much variation or dispersion from the average exists. A low standard deviation indicates that the data points tend to be very close to the mean (also called expected value); a high standard deviation indicates that the data points are spread out over a large range of values. To understand better why I have chosen SD, Figure 6.6 helps me showing a plot of a normal distribution (or bell-shaped curve) where each band has a width of 1 standard deviation.

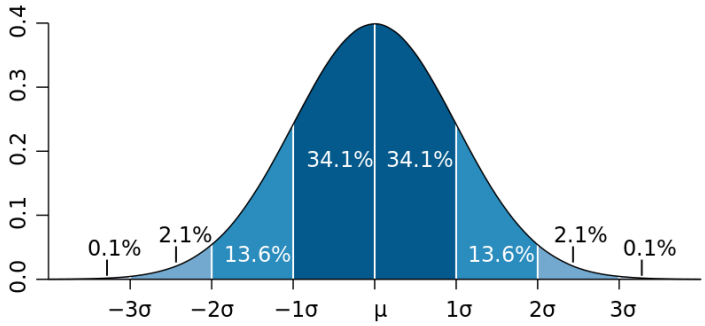


Figure 6.7 – Standard deviation.

```
/**
 * For each value I'll choice if it belongs to unaltered, added or
 * deleted. UNALTERED means that the facet_field is present at the
 * same time on the current choice and also on the possible next
 * choice the user could do when hover on a checkbox. ADDED means
 * that the facet_field is present only on the possible next choice
 * and on the current choice its value is insignificant. DELETED on
 * the contrary of Added, means that the facet_field is present only
 * on the current choice and on the possible next choice its value
 * is insignificant.
 */
 for (int i = 0; i < postProbabilities.length; i++) {
    for (int j = 0; j < postProbabilities[i].length; j++) {
       if (((postProbabilities[i][j] - preProbabilities[i][j]) <
```

```
                meanStandardDeviation[i]) &&
                ((preProbabilities[i][j] - postProbabilities[i][j]) <
                      meanStandardDeviation[i])) {
            unalteredKey.add(i + "," + j);
            unalteredValue.add(postProbabilities[i][j]);
        }
        if ((postProbabilities[i][j] - preProbabilities[i][j]) >
                meanStandardDeviation[i]) {
            addedKey.add(i + "," + j);
            addedValue.add(postProbabilities[i][j]);
        }
        if ((preProbabilities[i][j] - postProbabilities[i][j]) >
                meanStandardDeviation[i]) {
            deletedKey.add(i + "," + j);
            deletedValue.add(postProbabilities[i][j]);
        }
    }
}
```

7 KEYWORDS SEARCH OVER RELATIONAL DATABASE

Keyword-based search has been become highly popular thanks to its easy to use interface that is immediately understood by non-expert users [19]. Its user friendly characteristics have been exploited by web search engines, which have become, in the last years, the principal mean to browse and find information on the web. However, there is a great deal of structured information available on the web, for example online databases, forum/logs, e-commerce sites, that cannot be retrieved by traditional search engines. The advantages of enabling keyword-based search on structured data sources have been well understood by the research community who has focused on this topic in the last few years. Structured data sources, such as relational databases, are not easily accessible by non-expert users since they require users to learn a structured query language and to understand the database schema, which might be of considerable size and complexity.

Keyword-based search on databases differs from keyword search over textual documents under many aspects. The techniques developed by the information retrieval community for keyword search over textual documents cannot be applied straightforward to databases. The semantic information contained in textual documents can be easily understood by human users but it is extremely difficult to extract automatically. Traditional IR techniques consider textual documents as "bags of words": the keywords present in the documents are indexed separately and the semantics represented by the order of the words and phrases in the text is completely lost. At query time, the index is used to retrieve all documents that contain the given keywords. However, there is no guarantee that all the retrieved documents are relevant to the keyword query submitted by the user, and that all relevant documents have actually been retrieved. Indexing keywords in a document is not sufficient alone to capture the semantics of a textual document, as a result, documents concerning a different topic may also be retrieved while documents about the correct topic but using different keywords, for example synonyms, may not be retrieved. In order to measure the relevance of results retrieved by keyword-based search tools, two measures, named precision and recall have been introduced.

Precision: *the fraction of retrieved documents that are relevant to the keyword query submitted*

$$Precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|}$$

Recall: *the fraction of relevant documents, with respect to the given keyword query, that are successfully retrieved*

$$Recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|}$$

On the other side, information in a database is modeled with a well defined structure that guarantees that the stored data are consistent and normalized. Querying a database implies submitting a structured query which contains logical predicates that are evaluated against the data. The tuples in the database either satisfy the logical conditions expressed in the query and are retrieved, or do not satisfy the query are not returned as result. Hence, precision and recall are always equal to 1.

Enabling keyword-based search on databases is a challenging task. First of all, allowing users to query a database with keyword queries implies the introduction of ambiguity in the query process, since keyword queries have a poor semantics and cannot express the intended meanings of the users in a precise way. Hence, techniques for ranking the relevance of the obtained results become necessary. Top-K result ranking also helps users in browsing the results and focusing on the most pertinent tuples, especially in cases where the database contains a huge volume of data and the query result may be potentially large as well. Another key challenge to enable keyword search over databases regards how the information is organized inside a database. While in textual keyword search it is sufficient to identify documents, as the information units containing all the data relevant to the query, in databases the relevant information to a query may be spread out across multiple tables due to normalization and design choices. Answering a keyword query over a database implies, first, identifying all the tables containing the keywords and, second, joining these tables on the fly to obtain the query answer. Depending on the database schema, there may be multiple possible join paths that connect the tables containing the keywords, each of them expressing a different query semantics. Again, techniques to rank all obtained join paths based on their estimated relevance are necessary since the poor semantic of the keyword queries doesn't allow to distinguish which join path is the correct one. Algorithms for ranking the possible join paths are necessary also for performances reasons since executing all possible resulting structured queries on the database will lead to an unnecessary workload and a poor time response.

In the next section the problem of keyword-based search over relational databases is formally defined and explained using an example, while the rest of this chapter describes in detail the related work on keyword-based search over databases.

7.1 PROBLEM STATEMENT

A first step in enabling keyword-based search over databases requires modeling the database as a graph structure which is later used at query time to compute all possible joining paths between the database tables containing relevant information for the keyword query. A database can in principle be

represented using two different graphs: a schema graph and an instance graph. The schema graph represents only the structure of the database and contains the relations and the foreign key between the database relations. The instance graph represents the information content of the database and contains the tuples and the foreign keys relations between tuples. The two graphs are formally defined as follows [20].

Schema graph: *a database schema can be represented using a directed graph $G_S(V,E)$, where V is the set of relation (table) schemas $\{R_1, R_2, ..., R\}$ and E is the set of edges between the relation schemas. An edge in the schema graph exist between two relation schemas R_i and R_j, if R_j has a foreign key referencing the primary key in R_i. Note that multiple edges may exist between R_i and R_j in case R_j has multiple foreign keys referencing R_i.*

An example of a schema graph is given below in Figure 7.1. The figure illustrates a small portion of the DBLP database schema.

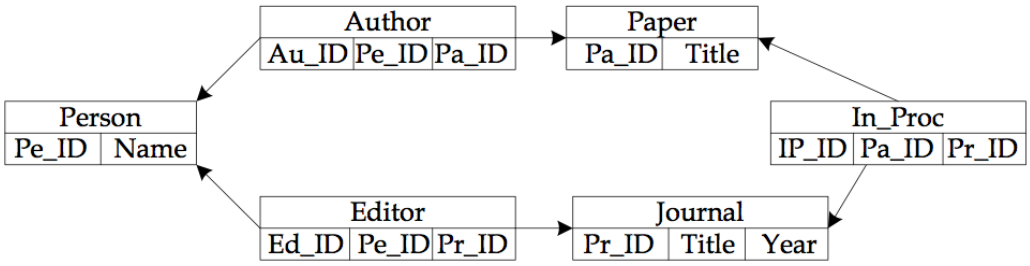


Figure 7.1 – A portion of the DBLP schema graph.

A relation on the relation schema R_i is a set of tuples $r(R_i)$ that conforms to the relation schema. A database instance is a set of relations over the database schema. Figure 7.2 shows an example of database instance that conforms to the schema in Figure 7.1. The instance graph, which represents a particular database instance, is formally defined as follows.

Instance graph: *a database instance can be modeled using an undirected instance graph $G_I(V,E)$, where V is the set of tuples in the database instance, and E is the set of edges between tuples. Two tuples t_i and tj in G_I are directly connected, i.e., there exist an edge between the two, if there exist at least one foreign key reference from t_i to t_j or vice versa. Two tuples t_i and t_j are reachable if there exist a connected path between the two in G_I.*

An example of instance schema is shown in Figure 3.3. For simplicity of visualization, the graph represents the tuples using their tuple identifiers. As shown in the figure, the instance graph can potentially have a much bigger size compared to the schema graph.

Author		
Au_1	Pe_1	Pa_1
Au_2	Pe_1	Pa_3
Au_3	Pe_3	Pa_2
Au_4	Pe_3	Pa_4

Editor		
Ed_1	Pe_2	Jo_1
Ed_2	Pe_2	Jo_2
Ed_3	Pe_1	Jo_3
Ed_4	Pe_1	Jo_1

In_Proc		
IP_1	Pa_1	Jo_2
IP_2	Pa_2	Jo_1
IP_3	Pa_3	Jo_3
IP_4	Pa_4	Jo_1

Person	
Pe_1	Josh
Pe_2	Maria
Pe_3	Ron

Journal		
Jo_1	Science	2009
Jo_2	Databases	2010
Jo_3	Databases	2011

Paper	
Pa_1	Top-K rank
Pa_2	Keyword search
Pa_3	XML Databases
Pa_4	Online Databases

Figure 7.2 – An example of database instance.

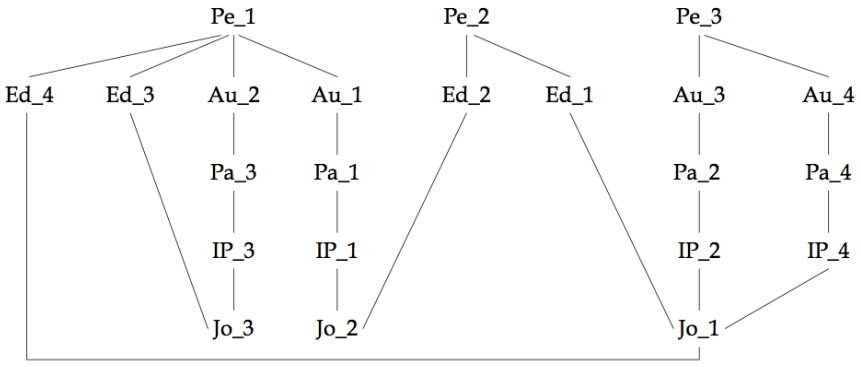


Figure 7.3 – An example of instance graph

A keyword query is considered, by the current approaches in keyword-based search over databases, as a "bag of words", meaning that the semantics represented by the keywords ordering in the query is lost. Hence, the formal definition of keyword query is as follows.

Keyword query: *a keyword query is a set of l keywords, with* $l \geq 1$ *and usually small* ($\approx \leq 10$), *denoted as* $Q = (k_1, k_2, ..., k_l)$.

The keyword query "Josh Databases" is an example of keyword query over the portion of database instance considered in the previous example. An l-keyword query executed on a database returns a set of results, where a result is a portion of the instance graph, named Minimal Total Joining Network of Tuples (MTJNT) [21]. Given a schema graph G_S and an instance graph G_I, the keyword relations and the associated Candidate Networks (CN) have to be identified in order to generate the MTJNTs for a given keyword query. A candidate network is a portion of graph that contains relations either containing the given keywords, or serving as connection between relations actually containing keywords. The relations that are part of a candidate network are called keyword relations. The formal definitions are given below.

Keyword relation: *given a keyword query Q and a schema graph* G_S, *a keyword relation* $R_i\{K'\}$ *is a subset of the relation* R_i *where the tuples contains a subset of the keywords in the query* $K' \subseteq Q$, *formally:*

$R_i\{K'\} = \{t | t \in r(R_i) \wedge \forall k \in K', t \text{ contains } k \wedge \forall k \in (Q - K'), t \text{ doesn't contain } k\}$

An empty keyword relation is a relation that does not contain any keyword, i.e., $K' = \emptyset$.

Candidate network: *given a keyword query Q and a schema graph* G_S*, a candidate network is a tree of keyword relations, i.e., for each two adjacent relations* $R_i\{K_1\}$ *and* $R_j\{K_2\}$, $(R_i, R_j) \in E(G_S)$ *or* $(R_j, R_i) \in E(G_S)$*. A candidate network needs to satisfy the following conditions:*

Total: a candidate network must contain each keyword in at least one keyword relation

Minimal: the total condition is not satisfied anymore if any keyword relation is removed from the candidate network

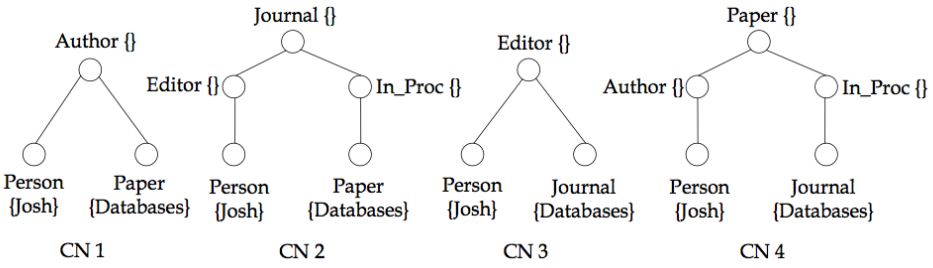


Figure 7.4 – An example of Candidate Networks.

Figure 7.4 shows, as example, the four candidate networks obtained from the schema graph in Figure 7.1 for the keyword query Josh Databases.

A candidate network can be easily translated into the correspondent structured query that joins the sequence of relations represented in the CN, and returns the (possibly empty) set of MTJNTs generated by the CN. The formal definition of MTJNT is given below.

Minimal Total Joining Network of Tuples (MTJNT): *given a keyword query Q and a schema graph* G_S*, a Joining Network of Tuples is a tree formed by adjacent tuples* $t_i \in r(R_i)$ *and* $t_j \in r(R_j)$ *that are joined using the foreign key references contained in the relation schemas* R_i *and* R_j *in* G_S*. A joining network of tuples is a MTJNT if it satisfies the total and minimal conditions:*

Total: the joining network of tuples must contain each keyword in at least one tuple

Minimal: the total condition is not satisfied anymore if any tuple is removed from the joining network of tuples

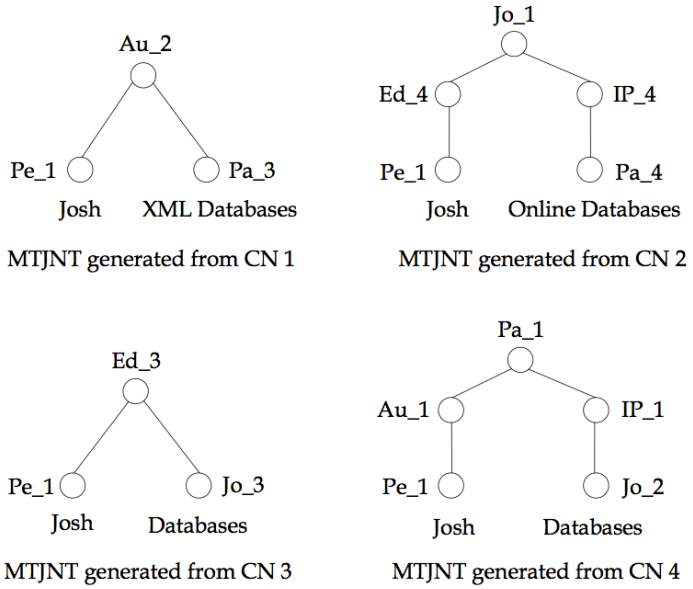


Figure 7.5 – An example of MTJNTs.

Following the previous examples, Figure 7.5 shows the four MTJNTs resulting from the four candidate networks described before. The MTJNTs needs to be ranked before presenting them as results to the user.

7.2 SPECIFIC TOPICS IN KEYWORD SEARCH

With the growing number of publications on keyword-based search over relational databases in the last few years, the researchers started to focus on specific problems. As described in section 7.1, enabling keyword search over databases requires different steps, each of which is non-trivial and demands the developing of specific techniques. This chapter section enumerate and describes the most relevant steps, and, for each step, reviews the most interesting and closely related approaches.

7.2.1 KEYWORD QUERY CLEANING

Keyword query cleaning is the first necessary step to execute before processing further the keyword queries. A keyword query may potentially contain typos that needs to be identified and corrected before attempting to match the keywords against the database terms. Moreover, the query may contain stop words such as “a”, “the”, that need to be removed since they don't represent potential matches but are part of the natural language. Many approaches do not consider the problem of keyword query cleaning and simply assume, by hypothesis, that the queries are already clean from stop words and correctly spelled. FRISK [22] and [23] formalizes the keyword cleaning problem and proposes an algorithm that clean and segment the queries. Query segmentation is a problem closely related to query cleaning. Since keyword queries are usually composed of plain words separated by white spaces, there isn’t a clear division between the keywords. A keyword can potentially be composed of many words, for example a person’s full name, a city name, a movie name etc. Query segmentation techniques aim to identify which tokens, i.e., words, belongs to which keyword in a query. The algorithm studied in FRISK is able to clean and segment the keyword queries in a stream and to provide the top-K clean queries for each original query given by the user. In a successive work [24],

instead of a fixed cost function, the cleaning problem is modeled using a Hidden Markov Model. The HMM can learn the correct segmentation on a query from query logs or from user feedback, achieving a better accuracy.

7.2.2 COMPUTING AND RANKING THE TOP-K MTJNTS

To efficiently identify and rank the most relevant MTJNTs for a given query is a key problem in keyword-based search over databases. Since structured data presents very different characteristics compared to textual documents, techniques developed by the information retrieval community cannot be straightforward applied and new algorithms needs to be developed. Structured data is usually represented using a graph there the tuples are connected by their foreign key relationships, and the goal for the algorithms is to find the MTJNT trees inside the data graph in an efficient way. Moreover, the resulting MTJNTs need to be ranked in order to present to the user only the most prominent results, since the number of resulting tuples can be potentially large. Traditional information retrieval techniques are suitable to rank single documents but provide poor results on structured data since the information to answer a query is spread out in different database relations. Finding a successful scoring function for structured data has become a popular research topic and many approaches have been presented. V. Hristidis [25] proposes a new algorithm, named the Global Pipelined algorithm, that process the candidate networks and outputs the top-k MTJNTs. The algorithm process the CNs in round robin and for each candidate network it calculate if it can produce any MTJNT with a top-K score. Since the exact scores can be calculated only after querying the database, the algorithm estimates the maximum possible score of MTJNTs extracted from a particular CN, and if the estimate doesn’t exceed the actual score of K already produced MTJNTs, the candidate network can be safely pruned and doesn't need to be executed on the database. In S. Chakrabarti [26] the database attributes are considered as separate documents and the keywords are identified using full text search features provided by the DBMS. The documents are retrieved in the decreasing order of their scores, while upper and lower bound scores are maintained for the related MTJNTs. The upper and lower bounds over the final score are used to identify a superset of the top K MTJNTs to be retrieved. Then, in a second phase, the best candidates are executed on the database in order to calculate their exact score while other documents are pruned until the exact top-K results have been returned to the user. [27] focus on the effectiveness and accuracy of the scoring function for MTNTs. The novel scoring function proposed by Liu is an adaptation of the traditional TF-IDF score used for textual documents for databases. A MTJNT is considered as a collection of documents, where each document represent a node being part of the MTJNT. A score is computed for each MTJNT as the normalized score of the component documents. The approach also keeps into account a simple query segmentation technique that recognizes, utilizing an index, if there are keywords composed of multiple words. In B. Ding [28] the problem of finding the top-K best answer is addressed in the form of finding the top-K minimum cost group Steiner trees in a graph. The work of Ding concentrate on efficiency, scalability and computational complexity of the novel algorithm introduced. The approach presented in Jagadish [29] considers querying the database and ranking the results as two separate problems and focus on ranking. The new concept of Qunit is introduced to represent a semantic unit of information in a database. Qunits are conceived as views of the database that contains all the relevant information to a user query and that reflects the user's perceived organization of the information in the database, which may not correspond to the actual database structure. When a keyword query is submitted, the top-K Qunits that best represent the answer are firstly identified and then executed on the database to return the results. The key issue in the technique proposed by Nandi is how to identify the Qunits of a database. Alternative solutions have been proposed, such as manually identifing the Qunits while designing the database structure, or using various information sources, such as query logs, the database schema, or web forms, to identify them automatically. Another interesting approach is

presented in L. Qin [30] where instead of finding MTJNTs, sub-graphs called communities are identified as query results. While MTJNTs are tuple trees containing a single answer, communities are sub-graphs containing multiple MTJNTs. A community contains all keywords, defined as center nodes, plus additional joining nodes up to a threshold radius. From a semantic point of view, a community groups together all MTJNTs that contains similar information, in order to give to the user an aggregated result.

7.2.3 SUMMARIZE AND DIVERSIFY QUERY RESULTS

An interesting aspect of the keyword-based search over databases that only few works keep into account is how to best organize the results returned to the user. Imagine, for example, the query "Clint Eastwood movies" over the well known IMDB[2] database. The number of matching tuples is large and returning the top 10 movies may not be satisfying for the user. The user may want to be able to browse the movies, or organize them, for instance, by genre. An interesting approach is presented in G. Koutrika [31] and implemented in CourseRank[3] at the University of Standford. The novel idea is to use tag clouds, a well-known technique utilized to browse information in social networks, to better present and browse keyword queries results. Given the set of MTJNTs resulting from a keyword query search, the terms present in the results are ranked and the high-ranked ones are selected to be presented, as a summary over the results, in the form of a data cloud. The cloud can then be utilized to easily browse the results and refine the initial keyword query. E. Demidova [32] presents DivQ, an approach that attempts to balance between the relevance and the novelty of keyword query results. Since keywords query are intrinsically ambiguous, they can be interpreted in different ways, leading to different answers. DivQ employs a ranking function that keeps into account the similarity between query results and score higher results that are different between each other. The idea is to give to the user a quick glance of the major plausible interpretations of a keyword query in the underlying database. The user can then effectively select the intended interpretation or simply have a better idea of the information contained in the database, as a starting point for browsing its content.

7.2.4 INDEXING USING DBMSS FEATURES

The majority of approaches to enable keyword-based search over databases necessitate to build an index over the database in order to retrieve the keywords. The index is usually built as an external structure, but recently, due to the introduction of full text search features into the commercial DBMSs, new techniques that exploits the capabilities of the DBMSs have been developed. In Widom [33] the DB2's Net Search Extender is used to build an inverted index over the database. Using a crawler, the database content is extracted offline and stored as a collection of virtual textual documents. The documents are then indexed using an inverted index and, at query time, the documents are retrieved and ranked using standard information retrieval techniques. The approach has good performances in cases where the database content is not updated frequently. The approach developed in L. Qin X. Y. [34] is another work that exploits the built in full text indexes present in DBMSs. The keyword queries are translated into SQL queries using the indexes and executed directly on the database.

[2] http://www.imdb.com/
[3] https://www.courserank.com/w/home

7.2.5 EXTENDING KEYWORD SEARCH CAPABILITIES

Recent works have focused on keyword-based search over databases from the point of view of the user, trying to enhance the user experience and provide an even more user friendly interface. One research direction regards the use of forms to query the database. Forms typically allows the users to specify the terms of a underlying pre-defined query. The advantage of a form over a keyword query approach is that the form has more expressive power and can be used to generate complex SQL queries that are not easily extracted from keyword queries. However, since a form represent a pre-defined query, a huge number of forms is necessary to satisfy all the possible search needs of the users. In the approach proposed in E. Chu [35], a keyword search interface is used to help the user in browsing and finding the correct form for her needs, choosing from a large collection of available forms. Then, the user can query the database through the form she selected. The drawback of this approach is that generating the forms requires the manual work of an experienced database designer who needs to understand well the information present in the database and try to predict what queries might the future users need to pose on the database. To address this problem, Jagadish [36] presents a technique to automatically generate forms. The approach uses heuristic rules and the schema of the database to predict which parts of the database are more relevant for querying.

Other approaches have focused on the developing of auto-complete techniques for keyword search over databases. Auto-complete is a technique that allows the user to receive a feedback at each keystroke she types in. The feedback could be either a suggested word or directly a query result. S. Ji [37] studies efficient algorithms that process the queries incrementally using previously computed and cached results in order to achieve an interactive speed. Moreover, to improve the results and correct possible typos in the keyword query, the system also tries to find the tuples that include words similar to the keywords in the query, even if they do not match exactly. The TASTIER system proposed in G. Li [38] and G. Li S. J. [39] uses a partitioned instance graph in order to achieve interactive performances. In the approach, the most relevant sub-graphs for the query are firstly individuated, and then the possible successive keywords are predicted by looking at the links between the tuples in the sub-graphs.

Another way to improve the user experience is by personalizing the keyword search results. Keeping into account the user preferences or feedback increases the effectiveness of the search. In K. Stefanidis [40] the proposed ranking algorithm keeps into account both the relevance to the query, and a set of preferences indicated by the user. A different solution to personalize the search is proposed in M. Lu [41], where the semantics of each data value is enhanced through tags that are provided by the users and inserted into the database together with the original value. When querying the database, the tags, that may be present in the keyword query, are utilized to retrieve the results.

8 KEYWORDS SEARCH & BAYESIAN NETKORKS

The more the relational data complexity is increasing and the user base is shifting towards the less technically skilled, the more the keyword searching is becoming an attractive alternative to traditional SQL queries, mainly due to its simplicity [19]. Unfortunately, this simplicity comes with the price of inherent ambiguity. Thus, the challenge of answering a keyword query over a relational database is to discover the database structures that contain the keywords and explore how these structures are inter-connected to form an answer. The discovered structures, alongside their inter-connections, are actually representing in relational terms the semantic interpretation of the keyword query.

Numerous studies and tools can already be found in the scientific literature. Generally, these works consider the database as a network of interconnected tuples, they detect those containing the keywords in the query, they generate connected components based on how these tuples are associated, and they return these connected tuples as an answer to the query. To do so, specialized structures that index the database content are used. By using these indices, they may directly retrieve the tuples of interest, or they may instead construct the queries expressions that retrieve these tuples when evaluated. This is the basic idea followed by the modern commercial database management systems (DBMs) supporting full-text search over their relational database.

Unfortunately, existing techniques suffer from two main limitations. The first is that they require a-priori access to the data instance in order to build the indices that will locate the tuples related to the given keywords at run time. This seriously limits their applicability if such access is not possible. Examples of such situations include databases on the hidden web and sources located behind wrappers in data integration systems that typically expose only their schema information and lack notification mechanisms for their data updates. The second limitation is that no considerable attention has been paid to the inter-dependencies among the query keywords. The likelihood that a specific data structure represent the same semantics as a keyword in a user query does not only depend on the relationship between the keyword and the data structure, but also on the data to which the other keywords in the

query are mapped. This is because despite the fact that a keyword query is a flat list of keywords, the meaning of each keyword is not independent of the meaning of the others, but they all collectively represent the intended concepts the user had in mind posing the query. Furthermore, not all the keywords represent instance values. Many are used as meta-data specification of the adjacent keywords. Although there are already keyword based approaches on relational data that take into consideration metadata, they provide only partial solutions to the problem, and they only use the metadata as a way to improve their technique.

In this chapter, I propose a software for answering keyword queries over relational databases. This work was born by the knowledge about Bayesian Networks that I have developed during the research project and so, using they, and leaning Keyword Search from the previous chapter, I have written a software that permits me to generate query without joins operations between tuples, which are the most requiring database performance killer.

8.1 SCENARIO

To this purpose I have used a small portion of a real DB that contains data about authors, editors and publications. The DBLP database is the same that I have already used like example in the previous chapter, more specifically, the DB described in section 7.1. So, the Figure 8.1 shows the DBLP schema and which table I have used.

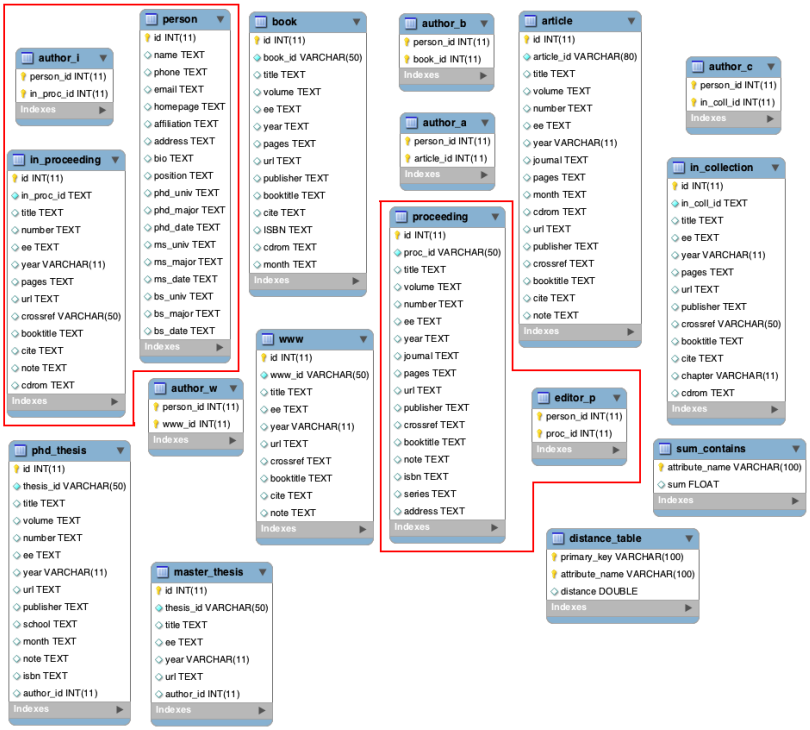


Figure 8.1 – DBLP schema.

I have chosen these tables because I would like a circle path because I can have two different way to select a record.

Like it already shown in figure, the tables chosen are *person*, *author_i*, *in_proceeding*, *proceeding*, *editor_p*. Thus, the join columns are the following:

- person.id = author_i.person_id
- author_i.in_proc_id = in_proceeding.id
- in_proecedind.crossref = proceeding.proc_id
- proceeding.id = editor_p.proc_id
- editor_p.person_id = person.id

After I have focused which are the join columns I have chosen to make a full outer join to have a complete view of the tables. The SQL query is shown below.

For a better understanding I have chosen to select just five columns for each table and I have renamed they (a total of fifteen columns). The columns were chosen based on their semantic importance. The results are also limited at only 1000 rows, either because the DB is so big and this is a first test (a non-limited result showed more than 3.500.000 rows), either I didn't have enough memory. This issue will be explained better in the next section. Note also that the DBLP database run on *MySql*, which it hasn't the full outer join so, I have done a union between left and right join.

```
SELECT p.id as p_id, p.name as p_name, p.homepage as p_homepage, p.address
       as p_address, p.bio as p_bio,
       ip.title as ip_title, ip.ee as ip_ee, ip.year as ip_year, ip.pages as
       ip_pages, ip.booktitle as ip_booktitle,
       pr.title as pr_title, pr.ee as pr_ee, pr.year as pr_year, pr.pages as
       pr_pages, pr.booktitle as pr_booktitle
FROM dblp.person as p
left join dblp.author_i as ai
on p.id = ai.person_id
left join dblp.in_proceeding as ip
on ai.in_proc_id = ip.id
left join dblp.editor_p as e
on e.person_id = p.id
left join dblp.proceeding as pr
on pr.id = e.proc_id
union all
SELECT p.id as p_id, p.name as p_name, p.homepage as p_homepage, p.address
       as p_address, p.bio as p_bio,
       ip.title as ip_title, ip.ee as ip_ee, ip.year as ip_year, ip.pages as
       ip_pages, ip.booktitle as ip_booktitle,
       pr.title as pr_title, pr.ee as pr_ee, pr.year as pr_year, pr.pages as
       pr_pages, pr.booktitle as pr_booktitle
FROM dblp.person as p
right join dblp.author_i as ai
on p.id = ai.person_id
right join dblp.in_proceeding as ip
on ai.in_proc_id = ip.id
right join dblp.editor_p as e
```

```
on e.person_id = p.id
right join dblp.proceeding as pr
on pr.id = e.proc_id
limit 1000
```

8.2 LEARNING A BAYESIAN NETWORKS

How to learn a Bayesian Networks from a .csv file is already explained in section 6.2. So, here I will explain just the concerns that could be useful in this chapter.

Once I have exported the query results on a csv file, is needed to know that Open Markov GUI accepts only csv file that are formatted following these characteristics:

1. column are separated only with comma (,) or semicolon (;)
2. comma or semicolon must be used only to separate different columns. It's not allowed use they for a different scope
3. each record must be on a single row. It's not allowed that a record is saved on more than one row of the csv file

This concerns just described are very important because if I don't want to follow they, I can't learn a network or, worst, I could learn a wrong network. Thus, if a csv file doesn't follow these characteristics, is possible to format it with a simple python script. This issue is due because, currently, the GUI code has these restrictions but the source code is open and so, in a future, can be possible to modify it and write a better parser.

There is also another aspect that, as I have described a beat previously, could be an issue. When I run the GUI I have to add this command `-Xmx<size>`

```
java –Xmx8g – jar org.openmarkov.full-0.1.3.jar
```

This issue is due because the *Heap Size* of the *Java Virtual Machine* (JVM) is setted by default at ¼ of the PC memory. I have 8GB of RAM memory so, the default heap size is setted at 2GB. This issue makes san important consequence:

> *A very heterogeneous database (like can be that I have used) increases a lot the heap memory used and not less, if the size of each attribute is big (for example if the attribute is a text field) also increases the heap memory used.*

When it happens, Open Markov GUI shows this message while it's trying to learn the network: *"Open Markov run Out of Memory"*.

Thus, after these tasks, is possible to learn a network of the full outer join I have queried. The Figure 8.1 shows it.

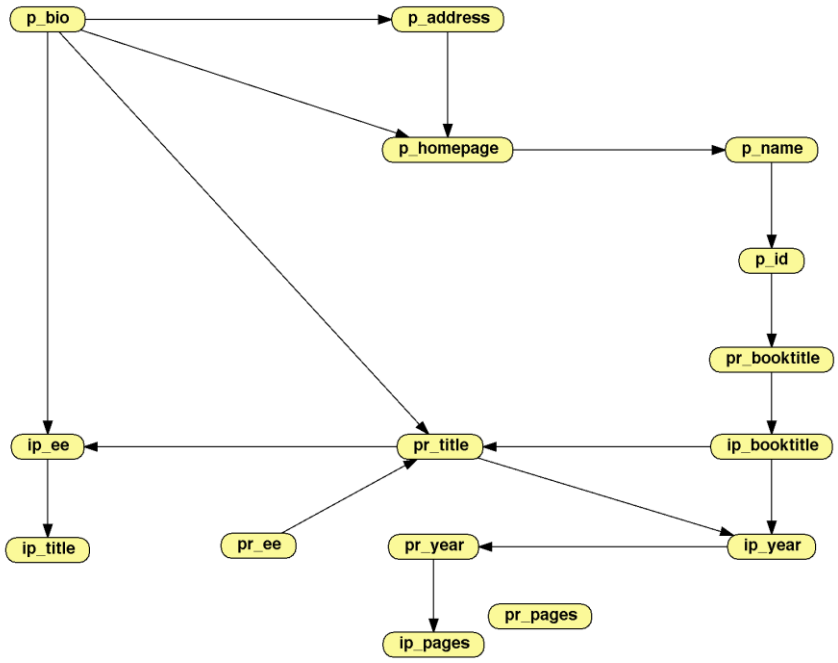


Figure 8.2 – Bayesian network learned from the full outer join.

8.3 SOFTWARE

I have developed a software that, reading the bayesian network, can discovered a record that I would like to find, without making a Sql query on a DB but just knowing one or more attributes that belong to it. The software is essentially structured into two main parts:

1. A first part where I have to add the attribute (evidence) for which I would like to find the record
2. A second part (called from the previous part) that computes all the probabilities and shows the results

Now, let me explain better the software.

The first part is simple and, in addition to what I have already written, just read the BN file, set all the variables and call the algorithm to compute the posterior probabilities (I have used *Variable Elimination*).

The second part is the most important. It is also splitted into three different sections. The first one computes all the posterior probabilities and makes a matrix that contains they. However, differently the code I have already described in Section 6.5, this part runs just one time because I don't have to calculate the PRE and the POST likelihood. The portion of code below shows how the probabilities are computed.

```
...
for (int i = 0; i < variablesOfInterest.size(); i++) {
   Variable variable = variablesOfInterest.get(i);
   double[] values;
```

```
   TablePotential posteriorProbabilitiesPotential =
                              posteriorProbabilities.get(variable);
   values = posteriorProbabilitiesPotential.getValues();
   for (int j = 0; j < values.length; j++) {
      if (maxNumValues < values.length) {
         maxNumValues = values.length;
      }
   }
}
...
```

The second section is the sort algorithm. I need a descending sort for all likelihoods of the same column in order to find the most probability attribute that it belongs to the record I would like discover. I have chosen a quick sort algorithm because is one of the most fast and I have modified it because I had to sort two array at the same time (one array of keys and one array of values). In the end, the last part of the code print the most probability for each column ad so, I would have the record I would like to find.

8.4 TEST

Since it is so probably that a single attribute has more than one row that belongs to it, I have chosen to use for this test at least two attributes, which belongs to different tables. Thus, in this way, it is more probably to identify a single, non-repeatable, record.

The tests has been done using two or three attributes (evidence) for each record I would like to search and I have repeated they fifty times, either for two attributes, either for three attributes. So, this is an example.

```
evidence.addFinding(probNet, "p_name", "Rainer Stiefelhagen");
evidence.addFinding(probNet, "ip_year", "2007");
evidence.addFinding(probNet, "pr_booktitle", "CLEAR");
```

I have also decided to show the Top 3 probabilities instead of only the Top 1 so I can see better how my software works. The figure 8.3 shows a beat of the result.

```
Evidence:
  ip_year: 2007
  pr_booktitle: CLEAR
  p_name: Rainer Stiefelhagen
THRESHOLD: 0.1

0)COLUMN: p_id
    1) PROB: 0.7634527922972593 NAME: 1473
    2) PROB: 0.004558727597138533 NAME: 245
    3) PROB: 0.004558727597138533 NAME: 241
COLUMN: p_id   PROB: 0.7634527922972593  NAME: 1473

1)COLUMN: p_name
    1) PROB: 1.0 NAME: Rainer Stiefelhagen
COLUMN: p_name   PROB: 1.0  NAME: Rainer Stiefelhagen

2)COLUMN: p_homepage
    1) PROB: 0.24313351020933327 NAME: http://isl.ira.uka.de/~stiefel/
    2) PROB: 0.020228836009389337 NAME: http://ls1-www.cs.uni-dortmund.de/~tick/homepage.html
    3) PROB: 0.01997983852276458 NAME: http://haegar.informatik.uni-wuerzburg.de/index-eng.html
COLUMN: p_homepage   PROB: 0.24313351020933327  NAME: http://isl.ira.uka.de/~stiefel/

3)COLUMN: p_address
    1) PROB: 0.4019235098156571 NAME: NULL
    2) PROB: 0.06603624382825045 NAME: Adenauerring 20ffice 230 (Geb. 50.20)76131 Karlsruhe
    3) PROB: 0.05023792307613514 NAME: Institut f??r Theoretische InformatikUniversit?t HannoverAp
COLUMN: p_address   PROB: 0.4019235098156571  NAME: NULL

4)COLUMN: p_bio
    1) PROB: 0.7743047405300342 NAME: NULL
    2) PROB: 0.05040981653684825 NAME: I am the chair of Combinatorial Optimization (CO) at the TU
    3) PROB: 0.03205331855058954 NAME: Bernhard Nebel received his first degree in Computer Scienc
COLUMN: p_bio   PROB: 0.7743047405300342  NAME: NULL

5)COLUMN: ip_title
    1) PROB: 0.012027653139816213 NAME: NULL
    2) PROB: 0.010730924737099688 NAME: Introduction.
    3) PROB: 0.010514334053551636 NAME: Practical Epistemic Entailment Checking in SROIQ.
```

Figure 8.3 – Keyword search result example.

It's important to note that the most likelihood of a column does not mean that it was a high probability like, for example, 80% because it is depend on the closeness or remoteness (in terms of bayesian network) between the evidence and the attribute found.

Table below shows a brief description of the results.

	2 EVIDENCE		**3 EVIDENCE**	
	TOP 1	**TOP 3**	**TOP 1**	**TOP 3**
	6/13=0.461	10/13 = 0.769	6/12 = 0.5	9/12 = 0.692
	8/13 = 0.615	10/13 = 0.769	8/12 = 0.666	9/12 = 0.692
	...	...	...	...
	10/13 = 0.769	12/13=0.923	9/12 = 0.692	11/12=0.916
	8/13 = 0.615	10/13 = 0.769	7/12=0.583	9/12 = 0.692
MEAN OF ALL THE RESULTS	0.615	0.7228	0.633	0.7832

The mean shows that the probability to rebuild a single record fluctuates around 62% at least and it becomes about 73% if I use three evidences.

Note that more rows I can use to learn the network, more accurate will be the results. This consequence is caused by the DB which is highly heterogeneous and more rows or columns can spread better the probabilities of each record.

9 CONCLUSIONS AND FUTURE WORK

Information visualization, especially by utilizing web-based platforms, is becoming an increasingly common medium for making decisions in many research and industrial areas. Thus, a large class of tools enabling web-based, interactive platforms for visualizing data is emerging.

In the context of Big Data, the availability of tools able to generate useful, accessible information is of prime importance. Moreover, dynamic visualization to facilitate the extraction, interpretation, and dissemination of data in this context is a key requirement. In order to manage the volume of big data, programmers are pushed to rapidly developing dynamic visual tools able to manage, analyze, and understand very large datasets.

This thesis, designed and developed a dynamic visual tool for big data, by extending *Apache Solr*, the most popular and open source enterprise search engine. *Apache Solr* main features include full-text search, dynamic clustering, database integration, rich document and *NoSQL* database handling.

In particular, this thesis aimed to provide an integrated and ready-to-use solution to visualize Big Data. So, in order to make it possible I endowed *Solr* with features (such as sharding and replication, distributed file system, etc.) that allow it to be performant in a Big Data context. Moreover, I modified its front-end equipping it with a new *faceted* search interface. Finally, in order to obtain a dynamic interface, I added, a bayesian suggestion component that allows a user to see how his/her search path might change by choosing an item instead of another one.

The provided extended Apache *Solr*, however, might be improved in the future. In fact, at present the facet fields are manually chosen. A future relevant improvement might be to implement a *machine-learning* (ML) algorithm that automatically choices and classifies facets. I have already thought about this improvement and, in the following, I will make a brief description of the simple method I devised.

The predominant source of data, in the context of the domain I analyzed, is in the form of text descriptors about a product, thus, I decided to implement a simple *Naïve Bayes* classifier. One issue for this classification scheme was that, often, an item has not enough information (enough words) to be properly automatically classified (or manually by a human). In this case, the idea is just to remove such an item from the set under consideration for classification. As the human generated text contents are used to define a product, I decided to adopt the bag of words model as features for each item. However, because the text descriptors are human generated, the problem of synonyms has to be faced. Thus, I can incorporate some domain specific area to help with classification.

I can also apply some standard techniques in text processing to clean up the feature set such as stemming/lemmatization, lowercasing, and stop word, punctuation, and number removal.

Secondly, I can use domain specific knowledge about the product catalog and what is contained in it. In addition to the title, items have brands and other descriptions such as specifications that we may use. Anyway, In order to 1) understand what features I can add and 2) weighting the features I have by how important they are in distinguishing between categorizations. Using the conventional *Mutual Information* formula [42], the skewness in the distribution of categories and the modification is not applicable as the distribution of categories is more uniform and is needed to modify it.

This is just a simple example and we could also try to use different machine-learning algorithm like k-nearest algorithm (kNN) or Tree Classifier.

This thesis also presents a different approach to keyword search over relational databases. This additional task has been possible thanks to the knowledge gained on Bayesian networks during the previous project. The current approaches in this research area heavily rely on indexes built over the database content in order to identify the location of the keywords inside the database. The use of indexes allows a fast and precise search on the database content but, at the same time, makes it unsuitable for many real world scenarios where it is not possible to access a-priori the database content to build an index. Examples of such scenarios are deep web databases, mediator systems, and third party databases. The technique presented in this thesis uses a bayesian approach to learn a network from the database and a software for keyword search using these networks has been developed.

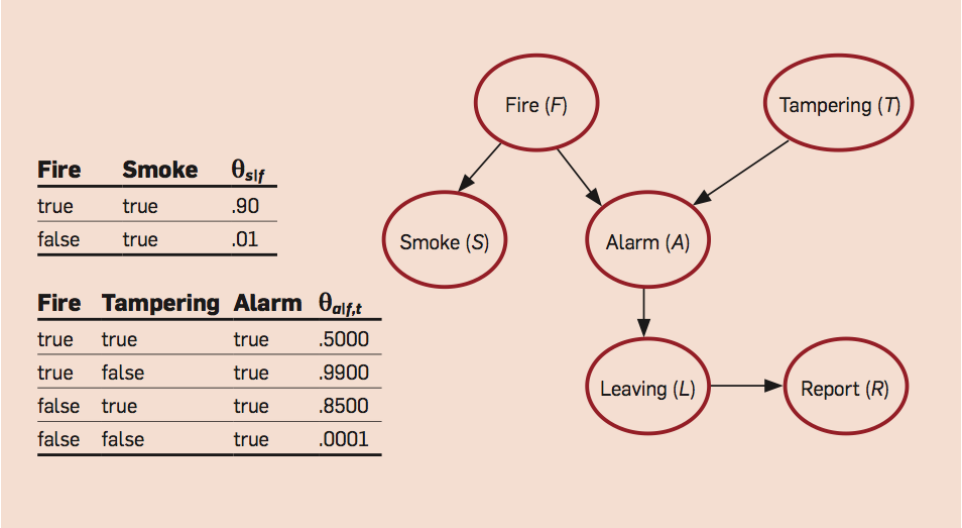
10 APPENDIX

10.1 BAYESIAN NETWORKS

Bayesian Networks have been receiving considerable attention over the last few decades from scientists and engineers across a number of fields, including computer science, cognitive science, statistics, and philosophy [43]. In computer science, the development of Bayesian networks was driven by research in artificial intelligence, which aimed at producing a practical framework for commonsense reasoning. Statisticians have also contributed to the development of Bayesian networks, where they are studied under the broader umbrella of probabilistic graphical models [12].

Interestingly enough, a number of other more specialized fields, such as genetic linkage analysis, speech recognition, information theory and reliability analysis, have developed representations that can be thought of as concrete instantiations or restricted cases of Bayesian networks. For example, pedigrees and their associated phenotype/genotype information, reliability block diagrams, and hidden Markov models (used in many fields including speech recognition and bioinformatics) can all be viewed as Bayesian networks. Canonical instances of Bayesian networks also exist and have been used to solve standard problems that span across domains such as computer vision, the Web, and medical diagnosis.

So what are Bayesian networks, and why are they widely used, either directly or indirectly, across so many fields and application areas?



Intuitively, Bayesian networks provide a systematic and localized method for structuring probabilistic information about a situation into a coherent whole. They also provide a suite of algorithms that allow one to automatically derive many implications of this information, which can form the basis for important conclusions and decisions about the corresponding situation (for example, computing the overall reliability of a system, finding the most likely message that was sent across a noisy channel, identifying the most likely users that would respond to an ad, restoring a noisy image, mapping genes onto a chromosome, among others). Technically speaking, a Bayesian network is a compact representation of a probability distribution that is usually too large to be handled using traditional specifications from probability and statistics such as tables and equations. For example, Bayesian networks with thousands of variables have been constructed and reasoned about successfully, allowing one to efficiently represent and reason about probability distributions whose size is exponential in that number of variables (for example, in genetic link-age analysis, low-level vision, and networks synthesized from relational models).

For a concrete feel of Bayesian networks, Figure 10.1 depicts a small network over six binary variables. Every Bayesian network has two components: a directed acyclic graph (called a structure), and a set of conditional probability tables (CPTs). The nodes of a structure correspond to the variables of interest, and its edges have a formal interpretation in terms of probabilistic independence. I will discuss this interpretation later, but suffice to say here that in many practical applications, one can often interpret network edges as signifying direct causal influences. A Bayesian network must include a CPT for each variable, which quantifies the relationship between that variable and its parents in the network.

Figure 10.1 – A Bayesian network with some of its conditional probability tables (CPTs).

For example, the CPT for variable A specifies the conditional probability distribution of A given its parents F and T.

According to this CPT, the probability of A = `true` given F = `true` and T = `false` is *Pr*(A=`true`|F =`true`; T =`false`) = .9900 and is called a network *parameter*[4].

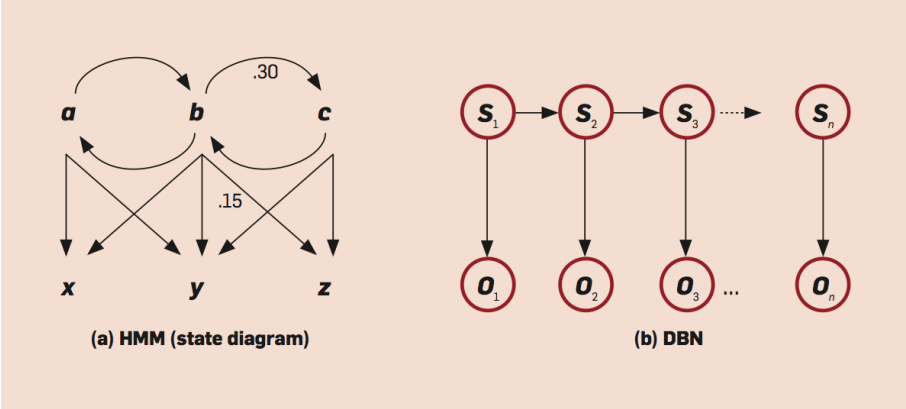
A main feature of Bayesian networks is their guaranteed consistency and completeness as there is one and only one probability distribution that satisfies the constraints of a Bayesian network. For example, the network in Figure 10.1 induces a unique probability distribution over the 64 instantiations of its variables. This distribution provides enough information to attribute a probability to every event that can be expressed using the variables appearing in this network, for example, the probability of alarm tampering given no smoke and a report of people leaving the building.

[4] Bayesian networks may contain continuous variables, yet my discussion here is restricted to the discrete case.

Another feature of Bayesian networks is the existence of efficient algorithms for computing such probabilities without having to explicitly generate the underlying probability distribution (which would be computationally infeasible for many interesting networks). These algorithms, to be discussed in detail later, apply to any Bayesian network, regardless of its topology. Yet, the efficiency of these algorithms—and their accuracy in the case of approximation algorithms—may be quite sensitive to this topology and the specific query at hand. Interestingly enough, in domains such as genetics, reliability analysis, and information theory, scientists have developed algorithms that are indeed subsumed by the more general algorithms for Bayesian networks. In fact, one of the main objectives of this chapter is to raise awareness about these connections. The more general objective, however, is to provide an accessible introduction to Bayesian networks, which allows scientists and engineers to more easily identify problems that can be reduced to Bayesian network inference, putting them in a position where they can capitalize on the vast progress that has been made in this area over the last few decades.

10.1.1 CASUALITY AND INDEPENDENCE

I will start by unveiling the central insight behind Bayesian networks that allows them to compactly represent very large distributions. Consider Figure 10.1 and the associated CPTs. Each probability that appears in one of these CPTs does specify a constraint that must be satisfied by the distribution induced by the network. For example, the distribution must assign the probability .01 to having smoke without fire, *Pr*(S=`true`|F=`false`), since this is specified by the CPT of variable S. These constraints, however, are not sufficient to pin down a unique probability distribution. So what additional information is being appealed to here?



The answer lies in the structure of a Bayesian network, which specifies additional constraints in the form of probabilistic conditional independencies. In particular, every variable in the structure is assumed to become independent of its non-descendants once its parents are known. In Figure 10.1, variable L is assumed to become independent of its non-descendants T, F, S once its parent A is known. In other words, once the value of variable A is known, the probability distribution of variable L will no longer change due to new information about variables T, F and S. Another example from Figure 10.1: variable A is assumed to become independent of its non-descendant S once its parents F and T are known. These independence constraints are known as the Markovian assumptions of a Bayesian network. Together with the numerical constraints specified by CPTs, they are satisfied by exactly one probability distribution.

Does this mean that every time a Bayesian network is constructed, one must verify the conditional independencies asserted by its structure? This really depends on the construction method. I will discuss three main methods in the section entitled “How Are Bayesian Networks Constructed?” that include subjective construction, synthesis from other specifications, and learning from data. The first method is the least systematic, but even in that case, one rarely thinks about conditional independence when constructing networks. Instead, one thinks about causality, adding the edge X → Y whenever X

is perceived to be a direct cause of Y. This leads to a causal structure in which the Markovian assumptions read: each variable becomes independent of its non-effects once its direct causes are known.

Figure 10.2 – A hidden markov model (hmm) and its corresponding dynamic Bayesian network (DBn).

The ubiquity of Bayesian networks stems from the fact that people are quite good at identifying direct causes from a given set of variables, and at deciding whether the set of variables contains all of the relevant direct causes. This ability is all that one needs for constructing a causal structure.

The distribution induced by a Bayesian network typically satisfies additional independencies, beyond the Markovian ones discussed above. Moreover, all such independencies can be identified efficiently using a graphical test known as *d-separation* [44]. According to this test, variables X and Y are guaranteed to be independent given variable Z if every path between X and Y is blocked by Z. Intuitively, a path is blocked when it cannot be used to justify a dependence between X and Y in light of our knowledge of Z. For an example, consider the path $\alpha : S \leftarrow F \rightarrow A \leftarrow T$ in Figure 5.1 and suppose I know the alarm has triggered (that is, I know the value of variable A). This path can then be used to establish dependence between variables S and T as follows. First, observing smoke increases the likelihood of fire since fire is a direct cause of smoke according to path α. Moreover, the increased likelihood of fire explains away tampering as a cause of the alarm, leading to a decrease in the probability of tampering (fire and tampering are two competing causes of the alarm according to path α). Hence, the path could be used to establish a dependence between S and T in this case. Variables S and T are therefore not independent given A due to the presence of this unblocked path. One can verify, however, that this path cannot be used to establish a dependence between S and T in case I know the value of variable F instead of A. Hence, the path is blocked by F.

Even though I appealed to the notion of causality when describing the d-separation test, one can phrase and prove the test without any appeal to causality—I only need the Markovian assumptions. The full d-separation test gives the precise conditions under which a path between two variables is blocked, guaranteeing independence whenever all paths are blocked. The test can be implemented in time linear in the Bayesian network structure, without the need to explicitly enumerate paths as suggested previously.

The d-separation test can be used to directly derive results that have been proven for specialized probabilistic models used in a variety of fields. One example is hidden Markov models (HMMs), which are used to model dynamic systems whose states are not observable, yet their outputs are. One uses an HMM when interested in making inferences about these changing states, given the sequence of outputs they generate. HMMs are widely used in applications requiring temporal pattern recognition, including speech, handwriting, and gesture recognition; and various problems in bioinformatics. Figure 10.2a depicts an HMM, which models a system with three states (a, b, c) and three outputs (x, y, z). The figure depicts the possible transitions between the system states, which need to be annotated by their probabilities. For example, state b can transition to states a or c, with a 30% chance of transitioning to state c. Each state can emit a number of observable outputs, again, with some probabilities. In this example, state b can emit any of the three outputs, with output z having a 15% chance of being emitted by this state.

This HMM can be represented by the Bayesian network in Figure 10.2b. Here, variable S_t has three values a, b, c and represents the system state at time t, while variable O_t has the values x, y, z and represents the system output at time t. Using d-separation on this network, one can immediately derive the characteristic property of HMMs: once the state of the system at time t is known, its states and outputs at times > t become independent of its states and outputs at times < t.

I also note the network in Figure 10.2b is one of the simplest instances of what is known as dynamic Bayesian networks (DBNs). A number of extensions have been considered for HMMs, which can be viewed as more structured instances of DBNs. When proposing such extensions, however, one has the obligation of offering a corresponding algorithmic toolbox for inference. By viewing these extended HMMs as instances of Bayesian networks, however, one immediately inherits the corresponding Bayesian network algorithms for this purpose.

10.1.2 HOW ARE BAYESIAN NETWORKS CONSTRUCTED ?

One can identify three main methods for constructing Bayesian networks [11]. According to the first method, which is largely subjective, one reflects on their own knowledge or the knowledge of others (typically, perceptions about causal influences) and then captures them into a Bayesian network.

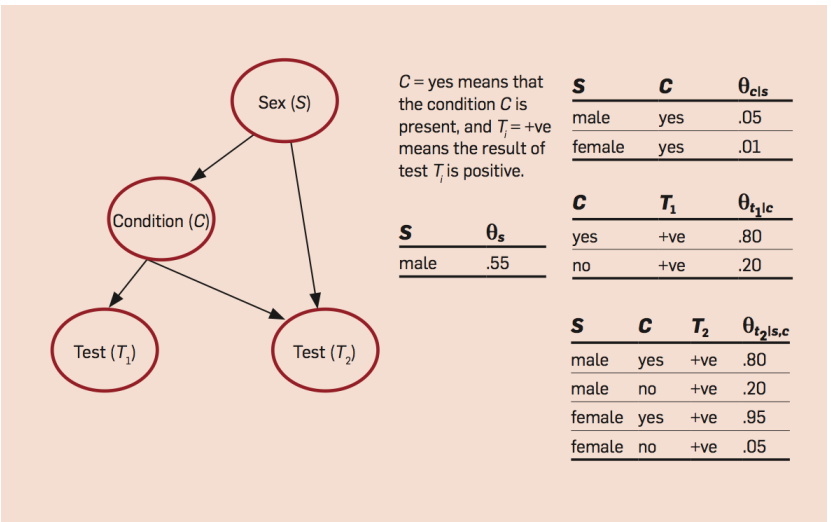
Figure 10.3 – A Bayesian network that models a population, a medical condition, and two corresponding tests.

The network in Figure 10.1 is an example of this construction method. The network structure of Figure 10.3 depicts another example, yet the parameters of this network can be obtained from more formal sources, such as population statistics and test specifications.

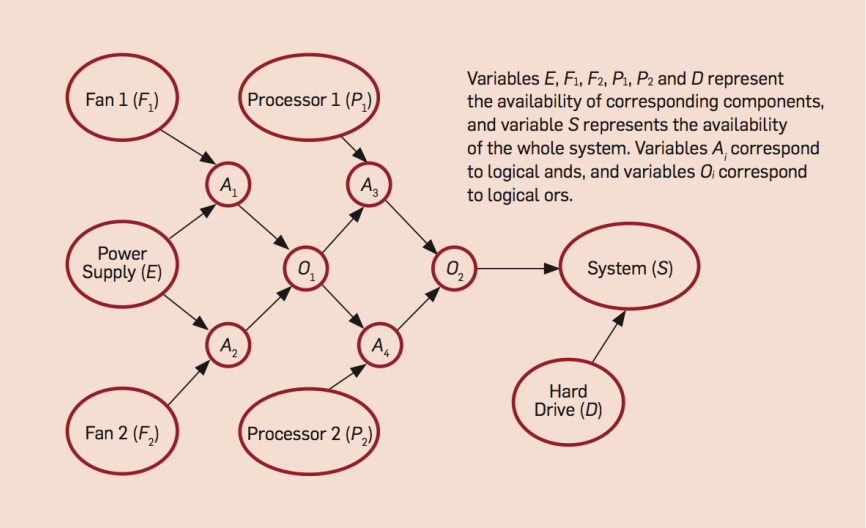
According to this network, I have a population that is 55% males and 45% females, whose members can suffer from a medical condition C that is more likely to occur in males. Moreover, two diagnostic tests are available for detecting this condition, T1 and T2, with the second test being more effective on females. The CPTs of this net- work also reveal that the two tests are equally effective on males.

The second method for constructing Bayesian networks is based on automatically synthesizing them from some other type of formal knowledge.

For example, in many applications that involve system analysis, such as reliability and diagnosis, one can synthesize a Bayesian network automatically from a formal system design. Figure 10.4 depicts a reliability block diagram (RBD) used in reliability analysis. The RBD depicts system components and the dependencies between their availability. For example, Processor 1 requires either of the fans for its availability, and each of the fans requires power for its availability. The goal here is to compute the overall reliability of the system (probability of its availability) given the reliabilities of each



of its components. Figure 10.4 shows also how one may systematically convert each block in an RBD into a Bayesian network fragment, while Figure 10.5 depicts the corresponding Bayesian network constructed according to this method. The CPTs of this figure can be completely constructed based on the reliabilities of individual components (not shown here) and the semantics of the transformation method [12].



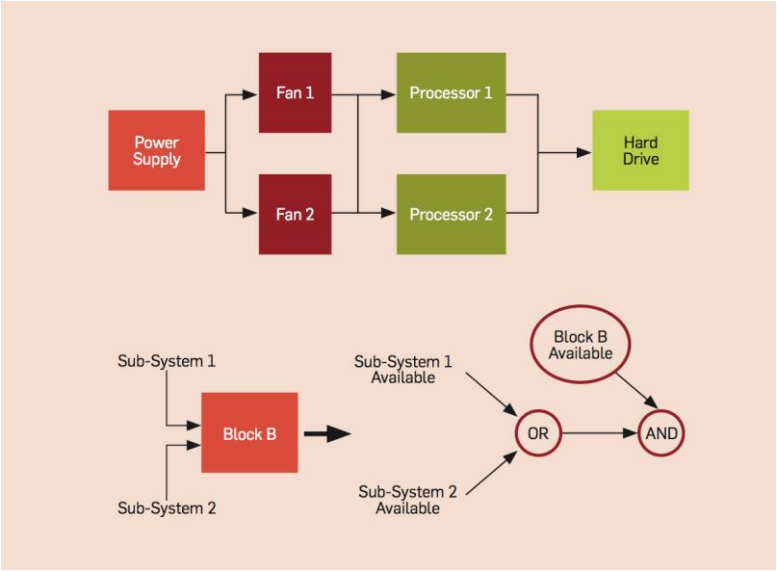


Figure 10.4 – A reliability block diagram (top), with a systematic method for mapping its blocks into Bayesian network fragments (bottom).

The third method for constructing Bayesian networks, which is what I will use during the two next developments of this thesis, is based on learning them from data, such as medical records or student admissions data. Consider Figure 10.3 and the data set depicted in the table here as an example.

Sex S	**Condition C**	**Test T_1**	**Test T_2**
male	no	?	-ve
male	?	-ve	+ve
female	yes	+ve	?
.	.	.	.
.	.	.	.
.	.	.	.

Figure 10.5 – A Bayesian network generated automatically from a reliability block diagram.

Each row of the table corresponds to an individual and what I know about them. One can use such a data set to learn the network parameters given its structure, or learn both the structure and its parameters. Learning parameters only is an easier task computationally. Moreover, learning either structure or parameters always becomes easier when the data set is complete— that is, the value of each variable is known in each data record.

Since learning is an inductive process, one needs a principle of induction to guide the learning process. The two main principles for this purpose lead to the *maximum likelihood* and *Bayesian* approaches to learning (see, for example, the work of [12] and [13]). The maximum likelihood approach favors Bayesian networks that maximize the probability of observing the given data set. The Bayesian approach on the other hand uses the likelihood principle in addition to some prior information, which encodes preferences on Bayesian networks.

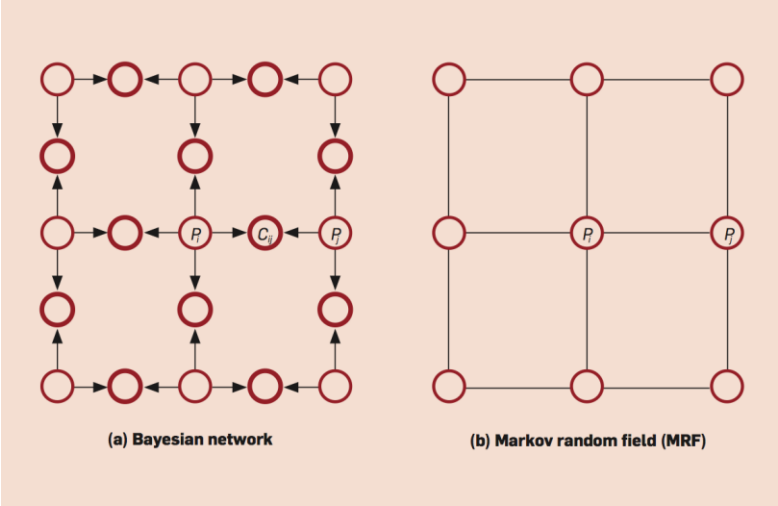
Suppose I are only learning network parameters. The Bayesian approach allows one to put a prior distribution on the possible values of each network parameter. This prior distribution, together with the data set, induces a posterior distribution on the values of that parameter. One can then use this posterior to pick a value for that parameter (for example, the distribution mean). Alternatively, one can decide to avoid committing to a fixed parameter value, while computing answers to queries by averaging over all possible parameter values according to their posterior probabilities.

It is critical to observe here that the term "Bayesian network" does not necessarily imply a commitment to the Bayesian approach for learning networks. This term was coined to emphasize three aspects: the often subjective nature of the information used in constructing these networks; the reliance on Bayes conditioning when reasoning with Bayesian networks; and the ability to perform causal as well as evidential reasoning on these networks.

These learning approaches are meant to induce Bayesian networks that are meaningful independently of the tasks for which they are intended. Consider for example a network which models a set of diseases and a corresponding set of symptoms. This network may be used to perform diagnostic tasks, by inferring the most likely disease given a set of observed symptoms. It may also be used for prediction tasks, where I infer the most likely symptom given some diseases. If I concern with only one of these tasks, say diagnostics, I can use a more specialized induction principle that optimizes the diagnostic performance of the learned network. In machine learning jargon, I say I am learning a *discriminative model* in this case, as it is often used to discriminate among patients according to a predefined set of classes (for example, has cancer or not). This is to be contrasted with learning a *generative model*, which is to be evaluated based on its ability to generate the given data set, regardless of how it performs on any particular task.

I finally note that it is not uncommon to assume some canonical network structure when learning Bayesian networks from data, in order to reduce the problem of learning structure and parameters to the easier problem of learning parameters only. Perhaps the most common such structure is what is known as naïve Bayes: $C \rightarrow A_1,...,C \rightarrow A_n$, where C is known as the class variable and variables $A_1,...,A_n$ are known as attributes. This structure has proven to be very popular and effective in a number of applications, in particular classification and clustering.

10.1.3 CANONICAL BAYESIAN NETWORKS



A number of canonical Bayesian networks have been proposed for modeling some well-known problems in a variety of fields. For example, genetic linkage analysis is concerned with mapping genes onto a chromosome, utilizing the fact that the distance between genes is inversely proportional to the extent to which genes are linked.

Canonical models also exist for modeling the problem of passing information over a noisy channel, where the goal here is to compute the most likely message sent over such a channel, given the channel output. For example, Figure 10.6 depicts a Bayesian network corresponding to a situation where seven bits are sent across a noisy channel (four original bits and three redundant ones).

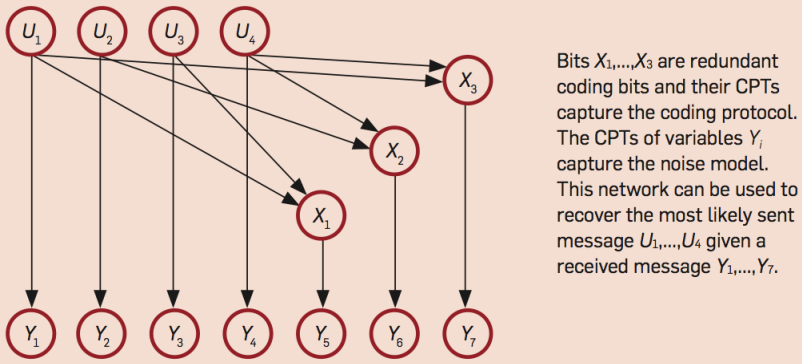


Figure 10.6 – A Bayesian network that models a noisy channel with input (U_1,...,U_4, X_1,...,X_3) and output (Y_1,...,Y_7).

Another class of canonical Bayesian networks has been used in various problems relating to vision, including image restoration and stereo vision, for examples images that were restored by posing a query to a corresponding Bayesian network. Figure 10.7a depicts the Bayesian network in this case, where we have one node P_i for each pixel i in the image—the values p_i of P_i represent the gray levels of pixel i. For each pair of neighboring pixels, i and j, a child node C_{ij} is added with a CPT that imposes a smoothness constraint between the pixels.

That is, the probability $Pr(C_{ij} = \texttt{true} \mid P_i = p_i, P_j = p_j)$ specified by the CPT decreases as the difference in gray levels $| p_i - p_j |$ increases. The only additional information needed to completely specify the Bayesian network is a CPT for each node P_i, which provides a prior distribution on the gray levels of each pixel i. These CPTs are chosen to give the highest probability to the gray level vi appearing in the input image, with the prior probability $Pr(P_i = p_i)$ decreasing as the difference $| p_i - v_i |$ increases. By simply adjusting the domain and the prior probabilities of nodes Pi, while asserting an appropriate degree of smoothness using variables C_{ij}, one can use this model to perform other "pixel-labeling" tasks such as stereo vision, photomontage, and binary segmentation.

Figure 10.7 – Modeling low-level vision problems using two types of graphical models: Bayesian networks and mRfs.

Canonical Bayesian network models have also been emerging in recent years in the context of other important applications, such as the analysis of documents, and text. Many of these networks are based on topic models that view a document as arising from a set of unknown topics, and provide a framework for reasoning about the connections between words, documents, and underlying topics. Topic models have been applied to many kinds of documents, including email, scientific abstracts, and newspaper archives, allowing one to utilize inference on Bayesian networks to tackle tasks such as measuring document similarity, discovering emergent topics, and browsing through documents based on their underlying content instead of traditional indexing schemes.

GLOSSARY

BIG DATA	Big Data is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications. The challenges include capture, curation, storage, search, sharing, transfer, analysis and visualization.
FACETED SEARCH	Faceted search, also called faceted navigation or faceted browsing, is a technique for accessing information organized according to a faceted classification system, allowing users to explore a collection of information by applying multiple filters. A faceted classification system classifies each information element along multiple explicit dimensions, called facets, enabling the classifications to be accessed and ordered in multiple ways rather than in a single, pre-determined, taxonomic order.
FACETED CLASSIFICATION	Faceted Classification is an analytic-synthetic classification scheme. It classifies objects using multiple taxonomies that express their different attributes or facets rather than classifying using a single taxonomy. A faceted classification system allows the assignment of an object to multiple taxonomies (sets of attributes), enabling the classification to be ordered in multiple ways, rather than in a single, predetermined, taxonomic order.
BAYESIAN NETWORK	A Bayesian network, Bayes network, belief network, Bayes(ian) model or probabilistic directed acyclic graphical model is a probabilistic graphical model (a type of statistical model) that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG).
KEYWORD SEARCH	In computer science, a key search algorithm is an algorithm for finding an item with specified properties among a collection of items. The items may be stored individually as records in a database.
CLUSTERING	Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.
SOLR	Apache Solr (pronounced “solar”) is an open source enterprise search platform from the Apache Lucene project. Its major features include full-text search, hit highlighting, faceted search, dynamic clustering, database integration, and rich document (e.g., Word, PDF) handling. Providing distributed search and index replication, Solr is highly scalable. Solr is the most popular enterprise search engine. Solr 4 adds NoSQL features.
LUCENE	Apache Lucene is a free/open source information retrieval software library developed by the Apache Software Foundation.

OPEN MARKOV	OpenMarkov is a software tool for probabilistic graphical models (PGMs) developed by the Research Centre on Intelligent Decision-Support Systems of the UNED in Madrid, Spain.
PGM	A graphical model is a probabilistic model for which a graph denotes the conditional dependence structure between random variables. They are commonly used in probability theory, statistics—particularly Bayesian statistics—and machine learning.
HADOOP FILE SYSTEM	The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file-system written in Java for the Hadoop framework. Apache Hadoop is an open-source software framework for storage and large-scale processing of data-sets on clusters of commodity hardware. Hadoop is an Apache top-level project being built and used by a global community of contributors and users.
ZOOKEEPER	Apache ZooKeeper is a software project of the Apache Software Foundation, providing an open source distributed configuration service, synchronization service, and naming registry for large distributed systems.
TOMCAT	Apache Tomcat (or simply Tomcat, formerly also Jakarta Tomcat) is an open source web server and servlet container developed by the Apache Software Foundation (ASF).
JETTY	Jetty is a pure Java-based HTTP (Web) server and Java Servlet container.
SERVLET CONTAINER	Servlet container is the component of a web server that interacts with Java servlets. A web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.
CARROT²	Carrot² is an open source search results clustering engine. It can automatically cluster small collections of documents, e.g. search results or document abstracts, into thematic categories. Apart from two specialized search results clustering algorithms, Carrot² offers ready-to-use components for fetching search results from various sources.
VELOCITY	Apache Velocity is a Java-based template engine that provides a template language to reference objects defined in Java code.
JAVASCRIPT	JavaScript (JS) is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed.
AJAX	Asynchronous JavaScript and XML: is a group of interrelated web development techniques used on the client-side to create asynchronous web applications. With Ajax, web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page. Data can be retrieved using the XMLHttpRequest object. Despite the name, the use of XML is not required (JSON is often used instead), and the requests do not need to be asynchronous.

DBMS	Database management systems (DBMSs) are specially designed software applications that interact with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is a software system designed to allow the definition, creation, querying, update, and administration of databases.
IDE	An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger.
NETBEANS	NetBeans is an integrated development environment (IDE) for developing primarily with Java, but also with other languages, in particular PHP, C/C++, and HTML5. It is also an application platform framework for Java desktop applications and others.

BIBLIOGRAPHY

[1] J. C. a. P. Z. D. Te'eni, XHuman Computer Interaction: Developing Effective Organizational Information Systems, Hoboken: John Wiley & Sons, 2007.

[2] B. S. a. C. Plaisant, Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th edition), Boston: Pearson/Addison-Wesley, 2004.

[3] J. Nielsen, Usability Engineering, San Francisco: Morgan Kaufman, 1994.

[4] D. Te'eni, Human-Computer Interaction and Management Information Systems: Foundations, Armonk: M.E. Sharpe, 2006.

[5] H. Simon, Designing organizations for an information-rich world. In Computers, Communication, and the Public Interest, Baltimore: MD: The Johns Hopkins University Press., 1971.

[6] A. Solr. [Online]. Available: https://lucene.apache.org/solr/index.html.

[7] Jetty. [Online]. Available: http://www.eclipse.org/jetty/.

[8] A. Tomcat. [Online]. Available: https://tomcat.apache.org/.

[9] A. ZooKeeper. [Online]. Available: https://zookeeper.apache.org/.

[10] A. Hadoop. [Online]. Available: https://hadoop.apache.org/.

[11] Darwiche, A. Modeling and Reasoning with Bayesian Networks., Cambridge: Cambridge University Press , 2009.

[12] Edwards, D. Introduction to Graphical Modeling, Springer, 2000.

[13] D. a. F. Koller, N. Probabilistic Graphical Models: Principles and Techniques., Cambridge: MIT Press, 2009.

[14] O. Markov. [Online]. Available: http://www.openmarkov.org/.

[15] CISIAD. [Online]. Available: http://www.cisiad.uned.es/.

[16] UNED. [Online]. Available: http://portal.uned.es/portal/page?_pageid=93,1&_dad=portal&_schema=PORTAL.

[17] R. E. Neapolitan, Learning Bayesian Networks, Upper Saddle River, NJ: Prentice-Hall, 2004.

[18] E. Herskovits, Computer-based probabilistic-network construction., Stanford University: PhD thesis, Dept. Computer Science, 1990.

[19] E. D. F. G. R. T. L. Y. V. Sonia Bergamaschi, Keyword Search over Relational Databases: A Metadata Approach, Athens: SIGMOD Conference, 2011, pp. 565-576.

[20] F. G. S. R. Sonia Bergamaschi, Keyword Search over Relational Databases: a Hidden Markov

Model approach, Modena: DBGroup UniMoRe, 2011.

[21] L. C. L. Qin, Keyword Search in Databases. Synthesis Lectures on Data Management., Morgan & Claypool, 2010.

[22] K. Q. P. a. X. Yu., Keyword query cleaning., Proceedings of the VLDB endowment , 2008, pp. 09 - 920.

[23] K. Q. P. a. X. Yu., FRISK: keyword query cleaning and processing in action., Proceedings of ICDE, 2009, pp. 531 - 1534.

[24] K. Q. Pu., Keyword query cleaning using hidden markov models., 2009, p. 27–32 .

[25] H. H. a. Y. P. V. Hristidis, Authority-bases keyword search in databases. ACM Transactions on Database Systems, 2008.

[26] V. G. J. H. a. D. X. S. Chakrabarti, Ranking objects by exploiting relationships: computing top-K over aggregation., 2006, pp. 371 - 382 .

[27] C. Y. W. M. a. A. C. F. Liu, Effective keyword search in relational databases, 2006, pp. 563 - 574 .

[28] S. W. L. Q. X. Z. B. Ding, Finding top-K min-cost connected trees in databases., 2007, pp. 836 - 845.

[29] A. N. Jagadish, Quints: queried units in database search, 2009.

[30] L. C. a. Y. T. L. Qin, Querying communities in relational databases., 2009, pp. 724 - 735.

[31] Z. M. Z. a. H. G. M. G. Koutrika, Data clouds: summarizing keyword search results over structured data, 2009, pp. 391 - 402 .

[32] P. F. X. Z. a. W. N. E. Demidova, DivQ: diversification for keyword search over structured databases., 2010, p. 331–338 .

[33] Q. S. a. J. Widom., Indexing relational database content offline for efficient keyword-based search, 2005, pp. 297 - 306 .

[34] X. Y. a. L. C. L. Qin, Keyword search in databases: the power of rdbms., 2009, pp. 681 - 694 .

[35] A. B. X. C. A. D. a. J. N. E. Chu, Combining keyword search and forms for ad hoc querying of databases., 2009, pp. 349 - 360 .

[36] M. J. Jagadish., Automated creation of a form-based database query interface., 2008, pp. 695 - 709.

[37] C. L. a. J. Feng., Efficient interactive fuzzy keyword search., 2009, pp. 371 - 380.

[38] X. Z. J. F. a. J. W. G. Li, Progressive keyword search in relational databases, 2009, pp. 1183 - 1186 .

[39] C. L. Feng., Efficient type-ahead search on relational data: a tastier approach., 2009, pp. 695 - 706.

[40] M. D. a. E. P. K. Stefanidis, Perk: personalized keyword search in relational databases through preferences., 2010.

[41] D. A. B. T. D. a. A. K. H. T. M. Lu, Schema- as-you-go: on probabilistic tagging and querying of wide tables, 2011, pp. 181 - 192 .

[42] M. Information. [Online]. Available: https://en.wikipedia.org/wiki/Mutual_information.

[43] A. Darwiche, What are Bayesian networks and why are their applications growing across all fields?, 2010.

[44] J. Pearl, Probabilistic Reasoning in Intelligent System: Networks of Plausible Inference, Morgan Kaufmann, 1988.