Graph Mining and its applications to Web Search

Ilaria Bordino
Ilaria Bordino

Graph Mining and its applications to Web Search

Thesis Committee
Prof. Stefano Leonardi (Advisor)
Prof. Ricardo Baeza-Yates (Advisor)
Prof. Paolo Ferragina (External examiner)
Prof. Paolo Boldi
Dr. Aristides Gionis
Prof. Vicente Lopez

Reviewers
Prof. Tanya Berger-Wolf
Prof. Gerhard Weikum
Author's address:
Ilaria Bordin
Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma
Via Ariosto 25, I-00198 Roma, Italy
e-mail: bordinodis.uniroma1.it
www: http://www.dis.uniroma1.it/~bordin/
For my mother, Anna Serena, and for my niece, Giulia,  
who are my pride and my hope.
Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisors, Prof. Stefano Leonardi and Prof. Ricardo Baeza-Yates, who have guided me throughout my thesis with their precious knowledge whilst allowing me the room to work in my own way.

I am heartily thankful to Debora Donato, who has supported me with unvaluable encouragement in my studies and in life.

I am also deeply indebted to my other collaborators: Luca Becchetti, Paolo Boldi, Carlos Castillo, Aris Gionis, Massimo Santini and Sebastiano Vigna. This thesis would not have been written without their inspiring advice.

Finally, I want to thank my family, my friends and my colleagues who have blessed my life with their constant love and support.
Abstract

Graphs allow to model interactions among entities in many real-world scenarios. Web Mining uses graph-based techniques to characterize the Web topology and to gather useful information for applications relying on social networks. We provide several contributions in this area.

We consider critical issues arising in the analysis of Web graphs, such as the huge volumes and the distributed nature of data. We present streaming algorithms for frequent-subgraph mining and decentralized computation of aggregates.

Characterizing the evolution of the Web is also important in Web search. We extensively study one year of data about the .uk Web, represented as a graph provided with temporal information.

Web-usage mining leverages information from query logs to improve search-engine services. Graphs provide compact and navigable representations of query-related information. We extend the work on query-log analysis by exploring two novel ideas: joint mining of multiple graphs with different semantics; analyzing query reformulation through spectral methods.

Resumen

Los grafos permiten modelar las interacciones entre las entidades en muchos escenarios del mundo real. La minería de datos utiliza técnicas basadas en grafos para caracterizar la topología de Internet y para recoger informaciones útiles para las aplicaciones que se basan en las redes sociales.

Ofrecemos varias contribuciones en este ámbito considerando cuestiones críticas que surgen en el análisis de grafos del Web, como los grandes volúmenes y la naturaleza distribuida de los datos. Presentamos algoritmos de streaming para contar subgrafos frecuentes y para calcular medidas de agregación descentralizadamente.

La caracterización de la evolución de la Web también es importante en búsquedas en internet. Estudiamos ampliamente un año de datos sobre el Web del Reino Unido, representados como un grafo con información temporal.

La minería de los datos de uso de la Web en base a la información de las bitácoras de consulta permite mejorar los servicios de los motores de búsqueda. Los grafos ofrecen representaciones compactas y navegables de la información sobre las bitácoras de consulta. Ampliamos el trabajo de análisis de las bitácoras de consulta mediante la exploración de dos nuevas ideas: análisis conjunta de múltiples grafos con semánticas diferentes; analizar bitácoras de consulta a través de métodos espectrales.
Contents

1 Introduction  
1.1 Graph mining in Web search  
  1.1.1 Web mining  
  1.1.2 Graph-mining applications in Web and social networks.  
1.2 Tools for large-scale graph mining  
  1.2.1 Properties and models of large networks  
  1.2.2 Matrix algebra  
  1.2.3 Graph clustering  
  1.2.4 Tensors for mining graphs over time.  
  1.2.5 Spectral graph theory  
  1.2.6 Streaming algorithms  
  1.2.7 Tera/peta-scale graph mining  
  1.2.8 External-memory algorithms  
1.3 Contributions of the thesis  
  1.3.1 Algorithmic methods for graph mining  
  1.3.2 Temporal characterization of the UK Web  
  1.3.3 Mining query-log graphs  

2 Counting Subgraphs in Large Networks  
2.1 Introduction  
2.2 Related work  
2.3 Algorithms for counting subgraphs of three and four nodes  
2.4 Implementation  
2.5 Experimental results  
2.6 Clustering networks  
  2.6.1 Swap randomization  
  2.6.2 Feature selection  
2.7 Network classification  

v
3 Fully Decentralized Computation of Aggregates over Data Streams 53
3.1 Introduction ............................................. 54
  3.1.1 Our Contribution ..................................... 55
  3.1.2 Related work ....................................... 57
3.2 Model and Preliminaries ................................. 57
3.3 Overview of the approach ............................... 58
3.4 Decentralized computation of $F_0$, $F_1$, and $F_2$ ... 59
  3.4.1 Composable, duplicate-insensitive sketches .............. 59
  3.4.2 Maintaining $F_0$ and $F_1$ .......................... 61
  3.4.3 Maintaining $F_2$ with duplicates ...................... 61
3.5 Distributed implementation and communication tradeoffs ... 65
  3.5.1 Distributed implementation .......................... 66
  3.5.2 Communication and Tradeoffs ........................ 67
3.6 Experimental analysis ................................. 67
  3.6.1 Accuracy of the estimation ......................... 68
  3.6.2 Communication and tradeoffs ....................... 69
4 Temporal Characterization of the UK Web 73
4.1 Introduction ............................................. 73
4.2 Related work ........................................... 75
4.3 Dataset .................................................. 76
  4.3.1 Crawling ........................................... 77
  4.3.2 Aligning the snapshots .............................. 78
  4.3.3 Creating a graph annotated with temporal labels ..... 79
4.4 General definitions and notation ....................... 80
4.5 Data assessment ........................................ 81
  4.5.1 Preprocessing: Classifying URLs ..................... 81
  4.5.2 Measuring page persistence .......................... 83
  4.5.3 Experiments related to deletion of pages .............. 84
  4.5.4 Adjusting the graph ................................ 89
4.6 Host evolution over time ............................... 90
  4.6.1 Clustering evolution patterns ....................... 91
  4.6.2 Using ODP as external source of quality information . 93
  4.6.3 Correlations among different properties .............. 95
  4.6.4 Predicting high dinamicity .......................... 95
  4.6.5 Correlations among neighbors ....................... 96
  4.6.6 Macroscopic dynamics .............................. 98
  4.6.7 Birth and death rates .............................. 99
4.7 Page and link dynamics ............................... 103
  4.7.1 Birth rate of Web pages ............................ 103
  4.7.2 Birth, death and replacement of Web pages .......... 103
6.5.2 Similarity with graph projections . . . . . . . . . . . . . . . 177
6.5.3 Query-dependent subgraphs . . . . . . . . . . . . . . . . . . . 179
6.5.4 Query-independent sub-graphs . . . . . . . . . . . . . . . 180
6.5.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 182
6.6 Application to query recommendations . . . . . . . . . . . . . . . 183

7 Conclusions and Future Work . . . . . . . . . . . . . . . . . . . . 187
Chapter 1

Introduction

The present thesis is the result of a joint doctoral program between the Department of Computer and System Sciences at Sapienza University of Rome and the Department of Information and Communication Technologies at Pompeu Fabra University of Barcelona.

The work has focused on the development of new graph-mining techniques and their application to Web search.

Graphs are general mathematical constructs that allow an effective modeling of the interactions occurring among disparate entities. The great flexibility of these data structures make them a suitable choice for representing virtually any natural structure, including those undergoing dynamical changes of topology. Thus, not surprisingly graphs have nowadays become ubiquitous representations for modeling complex relationships between data in a wide variety of real-world scenarios. Prominent examples are the Internet, the World Wide Web, social networks, protein-protein interaction networks in biochemistry, food Webs in ecology.

We specifically focus on the field of Web Mining, where graph-based techniques are applied to characterize the structural properties of the Web topology, and also to discover communities or detect outliers.

The analysis of the underlying graph structure has even greater importance for all the Web-based systems and applications where the users exchange information across a social network. Traditional sociological domains widely study network data, such as friendship and informal contacts among people, as well as collaboration/influence in companies, organizations, professional communities, markets, political movements. A rich social structure has also emerged in several computing applications: content creation, online communication,
2

1. INTRODUCTION

blogging, social media, electronic markets. Nowadays, people seek information from other people via Facebook, Yahoo! Answers, MySpace, LinkedIn, Flickr, del.icio.us, instead of using more formal channels.

All the cases mentioned above are examples of complex networks. Complex networks [32, 70, 191] are networks that exhibit non-trivial topological properties, such as power-law degree distributions, assortativity or disassortativity among nodes, short paths and community structure. These features don’t occur in simple networks, such as lattices or random graphs [103]. The study of complex networks is an active and multi-disciplinary research area, which has been inspired, to a large extent, by the empirical analysis or real-world networks, like social networks, chemical/biological networks, and technological networks.

Hence, the problems of designing algorithms for the mining of network data and developing applications on top of them have gained increasing interest in the last few years.

In the remainder of this chapter, we briefly overview the crucial issues that arise in the mining of large-scale graph structures. We devote particular attention to graph applications in the area of Web search, describing the most relevant findings obtained in this research field. We then present the problems that we have chosen to address in the thesis.

Section 1.1 discusses the importance of graph mining in the field of Web search, and describes several applications arising in the context of Web and social media. Section 1.2 contains a brief overview of the most powerful tools for the analysis of massive graph data. Finally, section 1.3 presents the main contributions of the thesis.

1.1 Graph mining in Web search

Huge volumes of data make critical challenges arise in the area of Web Mining. Since its inception, the World Wide Web has had a deep impact on almost every aspect of our lives. The Web is the biggest and most widely known information source that is easily accessible: It has really brought everything just a few clicks away from the comfort of our home or office. It has become an important channel for conducting business. It also provides us with convenient means to communicate with each other, to express our views and opinions on anything, and to discuss with people from anywhere in the world.

The Web has many unique characteristics, which make mining useful information and knowledge a fascinating and challenging task. Some of these characteristics include the huge volumes and the extreme heterogeneity and dynamicity of the data.
The amount of information stored in the Web is enormous, and is growing and changing constantly. At the same time, the coverage of information is very wide and diverse. The data often contain a lot of noise, and they are always characterized by a great diversity in the quality and authoritativeness of their content, as well as in the way such a content is organized: it can be structured tables, semi-structured Web pages, unstructured text or multimedia files.

The Web is also extremely dynamic. Although a significant part of it can be viewed as a static collection of pages whose content is rarely modified, there is a large and constantly increasingly part of the Web that is very changeable. People use blogs, news, and other live Web applications for real-time discussions about anything. New information is posted continuously, and the data become obsolete at a very rapid rate.

Another fundamental characteristic of the Web is its hyperlinked structure. Hyperlinks exist among Web pages within a site and across different sites. Within a site, links serve as information organization mechanisms. Across different sites, the links represent implicit endorsements of authority to the target Web pages.

Moreover, the biggest revolution introduced by Web 2.0 has certainly been that of turning the Web into a virtual society. The Web is not only about data, information and services, but it is also about interactions among people, organizations and automated systems. The Web provides us with convenient means to communicate with people from anywhere in the world via Internet forums, blogs and review sites.

1.1. Web mining

Web mining aims at discovering useful knowledge from all the wealth of information available in the Web, be it the content of Web pages, the hyperlink structure or the usage data stored in the logs of search engines.

Web mining uses many data-mining techniques, but it is not merely an application of traditional data mining, mostly because of the heterogeneity and the semi-structured or unstructured nature of Web data. Based on the primary kinds of data used in the mining process, Web mining tasks can be divided into three main categories [147, 190]: Web content mining, Web structure mining and Web usage mining.

Web content mining extracts useful information from the content of Web pages [34, 147, 167]. For example, we can automatically cluster or classify Web pages according to their topics. These tasks are similar to those in traditional data mining. However, we can also discover patterns in Web pages to extract useful data such as descriptions of products, postings of forums, etc. Furthermore, we can mine customer reviews and forum postings to discover
customers’ sentiments. The task of opinion mining is technically challenging, because it requires the analysis of huge amounts of unstructured content. At the same time, it is very useful in practice because businesses and organizations always want to know consumers’ opinions about their products and services. Besides the traditional collections of text documents, Web content mining also targets the knowledge discovery from the wealth of multimedia resources, such as images, videos, audio, which are embedded in or linked to Web pages.

**Web structure mining** mines useful information from the hyperlink structure of the Web. This includes the organization inside a Web page, internal and external links and the site hierarchy. Some relevant link-mining tasks include link-based classification and cluster analysis, which can be used to discover hidden patterns in the data, and predicting the existence, the type or the purpose of a link between two entities.

Link mining [114] is a key technology used by search engines to create notions of authority, and thus to discover relevant pages. The simple existence of a link between two pages can be considered as a conveyance of authority from the former to the latter. However, search-engine designers soon realized that indegree is not the proper basis to create a robust measure for authoritativeness of Web pages, because it is prone to adversarial manipulation.

More effective approaches are those proposed with two famous ranking algorithms, such as *PageRank* and *HITS*. PageRank [177] combines the authoritative information of both source and target page in an iterative way to determine the rank. One advantage of PageRank is that it computes a global score, which is query independent and thus can be computed offline, in a preprocessing phase. On the other hand, one of its major drawbacks is to be found in the fact that it can be manipulated by spammers through the creation of *link farms* [37], which are dense substructures that are artificially built to modify the Web topology, in the attempt of increasing the ranking of spam pages.

In the HITS [141] algorithm, introduced by Kleinberg, every page is associated with a hub and an authority score, which mutually and iteratively reinforce each other. Differently from PageRank, the HITS algorithm computes a query-dependent score, which captures better the relevance of a page with respect to the peculiar information need expressed by a query. However, the dependance on the query makes HITS more expensive than PageRank.

A large body of research has worked on improved ranking algorithms for more efficient computation of PageRank, and for providing refined definitions that address specific issues, such as personalization and topic-specific search [47, 109, 126].
Web usage mining analyzes usage logs to capture and model behavioral patterns of users who interact with Web sites and search engines. The logs of Web sites can be examined [124, 167, 190] to gain useful insights on the navigation patterns followed by the users to access the informative contents made available on the sites themselves. Analyzing how users visit Web sites has numerous practical applications.

First of all, it can provide guidelines for Web-site reorganization, helping designers to place important information where the users look for it.

Moreover, understanding user behavioral patterns is crucial for generating user profiles, which are employed to increase customization and create dynamic recommendations of products and services, providing users with pages and advertisements of interest. In fact, targeted advertising is probably the most important application of Web usage mining, given that ads are a major source of revenue for Web portals and e-commerce sites.

Mining Web-site usage information also finds applications in detecting frauds: commercial portals maintain a signature for each user based on buying patterns on the Web (e.g., amount spent, categories of items bought), and they analyze such patterns to discover anomalies and malicious behavior.

Search-engine logs represent another valuable source of information about users’ preferences and interests. The interaction logs between users and search engines enclose a wealth of implicit and explicit knowledge about how users refine their queries, and what kind of strategies they are using to locate the information they need. Hence, the analysis of such data is a task of critical importance for search engines in order to improve the Web-search experience.

A central problem here is how to discover and represent the information need associated with a query. For various reasons queries themselves, as lists of keywords, are often not good descriptors of the users’ latent needs. For example, they are generally very short, and they may be ambiguous. In many cases, they are formulated by people who are not familiar with the specific terminology of a given knowledge domain.

Query-log analysis [20, 39, 206, 212] attempts to model the semantics of queries using the wealth of information stored in the search-engine logs. This information include session data, as well as the results returned by the search engine and the documents clicked by users.

Analyzing query-log data and understanding the semantic relations between queries are crucial tasks for a number of applications that are very relevant from a search-engine point of view. Examples of these applications include boosting ranking algorithms, improving advertising algorithms through discovery of better bidding keywords, and recommending related queries.
1. INTRODUCTION

1.1.2 Graph-mining applications in Web and social networks.

Many graph data appear in the context of Web and social media. In this subsection, we describe relevant examples of graph applications in the context of Web content mining, Web structure mining, and Web usage mining.

Applying graph mining to analyze Web content. Web content mining applies graph-based techniques to discover patterns in the content and organization of Web pages. This research line is related to the semantic Web developments. The semantic Web aims at discovering automated schemes to learn relevant information from Web pages and associate documents with machine-understandable semantic annotations.

The general idea is to take advantage of the fact that nowadays a considerable fraction of Web text has a semi-structured nature. Models like Object Exchange Model [7] are used to represent the data as labeled graphs.

XML markups (and new generation HTML as well) bear information that concerns not only the layout of a document, but also its logical structure. This information can be extracted through the usage of DOM\(^1\), a standard language that takes a Web page as input and shows it in a structured tree, which represents interfaces, objects, and relations between them.

The main application of this kind of research is that of extracting schemas and structured summaries of the data [6, 118, 204]. The task of detecting content patterns in large document collections has numerous practical applications. For example, pattern discovery can be used for detection and tracking of topics, which can serve to identify critical events and trends. Other objectives are information extraction and integration, learning relations between Web documents, and site (re)construction.

Mehta et al. [162] recently presented a page-segmentation algorithm that combines visual and content information to create a semantic tree structure whose leaves represent segments having a unique topic. This structure can be useful for information extraction and automatic page adaptation. Zheng et al. [214] proposed a method for filtering noises in Web pages based on building DOM parsing trees and applying domain-specific extraction rules.

Applying graph mining to analyze Web structure. A straightforward, but prominent example of graph data that is relevant in the area of Web search is the Web graph, which is the graph induced by the Web documents and the hyperlinks connecting them.

The study of the topological properties of the Web graph was started by Barabási and Albert [33] and by Kumar et al. [150]. A more complete analysis of the Web graph was later presented by Broder et al. [63], who reported many measures of the Web together with the bow-tie picture, a macroscopic

---

\(^{1}\)http://www.w3.org/DOM/
characterization of the Web structure. Later, the bow-tie structure was refined by Donato et al. [100] in an extensive study of a large sample of the Web provided by the WebBase project.

Link mining is also used by search engines to improve the quality of search results. The discovery of the possibility of using the Web hyperlink structure to define notions of authority and identify relevant pages determined the transitions from first- to second-generation search engines. The idea was introduced by Brin and Page [177], who presented the PageRank algorithm. The HITS algorithm, developed by Kleinberg [141], also leverages the link structure of the Web, but uses a different, two-fold notion of authority, as we already mentioned when introducing Web structure mining.

Later, link-based techniques were shown to be effective methods for spam detection. Web spam can significantly deteriorate the quality of search-engine results. Thus, there is a large incentive for commercial search engines to detect spam pages efficiently and accurately.

Castillo et al. [75] developed a system for Web-spam detection by exploiting the link dependencies among the Web pages. They showed several ways of combining information about the Web topology with the predictions obtained through a text-based classifier.

Becchetti et al. [37] built Web-spam classifiers that only consider the link structure of the Web, obtaining a performance comparable to that of state-of-the-art spam classifiers based on content attributes.

Chung et al. [80] presented a study of the temporal evolution of link farms over a period of two years. Their results show that almost all large link farms do not grow anymore while some of them shrink, and many large link farms are created in one year.

**Applying graph mining to Web-usage data.** Many other graph applications have recently emerged in the virtual society of the Web. An excellent survey is provided by Donato and Gionis [99]. Interesting examples are social networks, online communities, expertise-sharing groups. People create content and share opinions and thoughts through a wide collection of media, like blogs, forums, wikis, newsletters, newsgroups, question/answering systems.

Social media are usually characterized by a very rich structure and by a great heterogeneity in the quality of the content made available by the users’ contributions. The various applications share a peculiar characteristic: the data can be represented by an underlying graph structure, where the nodes represent users who have personalized profiles and exchange information, while the links represent many different types of activities and interactions between users. Understanding the structure of such graphs and designing algorithms for extracting the latent knowledge, also called the *wisdom of the crowds*, introduce new challenges in the field of graph mining.
Popular problems for all the networks representing on-line communities are the following: identifying experts for interesting topics; finding and ranking relevant items; predicting future successful items; detecting trends that indicate a surge or decline of interest in certain topics.

Castillo et al. [74] studied the problem of predicting popular items in dynamic settings, such as photo-sharing systems, blogs and co-citation networks.

Su et al. [192] analyzed the feasibility of using question/answering portals to build automated collections of human-reviewed data.

Important graph applications arising in the context of Web usage mining are based on the usage of query-log data. Query logs record the queries that the users make, the results returned by search engines, and the documents that the users click on within the set of results. Analyzing these data provides useful insights for many practical applications. Relevant examples are the following: improving tools for error correction; recommending related queries; using clickthrough data to refine search-engine results; improving advertising algorithms.

The idea of inferring graph structures from query logs has attracted increasing interest in the past few years. Many different definitions can be used to mine different semantic information. Baeza-Yates [21] has raised the attention on the possibility of building many different graphs, based on disparate information sources, such as the text of the queries or the content of common clicked pages.

Click graphs [39, 89, 206] are query-document bipartite graphs, where a query is connected to the documents that were clicked within the set of results returned by the search engine for that query. Click graphs have been extensively used for different purposes, like suggesting related queries [88], identifying multi-topic documents [27], finding high-quality and relevant results for a query, annotating documents with compact, query-based descriptions [29, 178], finding a set of diverse queries that cover different aspects of the original query [51].

Alternatives to click graphs are the view graph and the anti-click graph [73]. The view graph generalizes the click graph relating queries to documents whose URLs have been viewed in the result list returned to the user, but not necessarily clicked. The anti-click graph intends to capture the negative feedback that users implicitly give to a top-ranked document when they ignore it by clicking on documents ranked below. Analyzing the view graph and the anti-click graph provides useful information for the task of detecting spam.

All the graphs defined above do not consider the notion of time. However, temporal aspects must be taken into account in order to gain a better understanding of the patterns followed by the users in the attempt of satisfying more complex information goals.
The time notion is crucial in the definition of the *query-flow-graph*, introduced by Boldi et al. [42]. The *query-flow graph* models user behavioral patterns and query dependencies. The main focus here is on the phenomenon of the sequentiality of similar queries: intuitively, in this graph a directed edge from one query to another means that the two queries are likely to be part of the same search mission. In the query-flow-graph, each link is weighted with a probability that indicates if the two queries corresponding to the endpoints of the transition are related. Any path over the query-flow graph may be seen as a searching behavior, whose likelihood is given by the strength of the edges along the path. The features used to learn the edge weights include textual features, session features, such as the number of sessions in which the transition appears, the average session length, the average position of the queries in the sessions, and time-related features, like, for instance, the average time difference between the two queries in the sessions that contain the transition.

Interesting applications of the query-flow graph are segmenting and assembling chains in user sessions, and generating query recommendations [42]. In the latter case, the advantage of using the query-flow-graph instead of the click graph is that the method selects queries that actually followed the starting query into one or more user sessions. Thus, the recommendations are more likely to be natural continuations of the same information need. An interesting variant that was also investigated is that of proposing recommendations that depend on the *history* of the last submitted queries, and not only on the last query. This approach may be very helpful when one has to cope with sparse or ambiguous data.

### 1.2 Tools for large-scale graph mining

Advances in technology have enabled us to collect vast amounts of data from various sources, whether they be from experimental observations, simulations, sensors, credit-card transactions, or from networked systems. Organizations and researchers have then to face the ever-growing problem of how to manage and analyze data collections of very large scale. Finding patterns, summaries and anomalies in large-scale graphs is a serious challenge. The problem becomes even more difficult if the data taken into consideration evolve over time.

This section presents a brief overview of the most powerful tools for the analysis of large, real graphs. In our review, we are only able to report examples of the most popular and/or recent contributions that have brought significant advances in research. Thus, the presentation provided below should by no means be considered exhaustive.
1.2.1 Properties and models of large networks

Each complex network (or class of networks) presents specific topological features that characterize its connectivity and highly influence the dynamics of the processes executed on the network.

Real-world networks exhibit peculiar structural properties, such as power-law degree distributions, community structure and small diameter, which cannot be explained by the uniformly random connectivity of the Erdős-Rényi model [103]. Random graphs have short paths, but their degree distributions follow a Poisson distribution.

Thus, other models were proposed to explain the generative processes of these networks and the laws of their evolution.

Barabási and Albert [33] introduced an algorithm for generating scale-free networks using a preferential attachment mechanism, which favors nodes with higher degree in attracting the links that are added to a network. This model leads to building graphs that have power-law degree distributions and short paths. However, the networks generated through this process are still characterized by a small clustering coefficient. Hence, the model is not able to capture the community structure that many real data sets exhibit.

The model proposed by Watts and Strogatz [205] overcomes this limitation through the expedient of inserting into a highly structured network, which is characterized by a high clustering coefficient, further random links to create shortcuts and obtain small diameter.

Another model that is able to capture communities is the copying model proposed by Kleinberg [142]. In this model, whenever a new node has to be inserted in the network, a random number $k$ of edges to be added is chosen; with some probability $\beta$ these edges are linked to random nodes, and with probability $1 - \beta$ they are copied from a randomly chosen node.

In complex network analysis, a very interesting problem consists of measuring the structural properties of evolving networks in order to understand how the connectivity of graphs changes along the process.

Measuring properties of the networks, and their dynamics, is important to understand how networks reflect and affect the processes involving the entities comprising the network. A large body of work has deeply investigated the subject of network measurements. A good survey is the one provided by Costa et al. [87]. The set of measures proposed and analyzed includes vertex degree, clustering coefficient, shortest-path length. These simple properties have been used to define various notions of centrality, in the attempt of capturing the relevance of nodes within a network.

One of the most basic tools for analysing graph structures and revealing the properties of the underlying data is finding frequent patterns in graphs. Milo et al. [165, 166] observed that different families of networks can be character-
ized by network motifs, which are specific subgraph patterns whose frequency is significantly higher than the frequency that one would expect in random networks with the same degree distribution. Therefore, the distribution of the (frequent) subgraphs can be used to characterize the networks and classify them into families.

Besides studying static topological properties of graphs, researchers soon realized that a characterization of the network dynamics is also important, and they started to investigate how networks evolve over time. Leskovec et al. [104] studied a wide range of real graphs, including Web graphs, online communities, and click streams. They observed unexpected phenomena, such as graph densification and shrinking diameters over time. The authors provided a new graph model, based on a forest fire spreading mechanism, to capture this behavior.

Social networks play a fundamental role as a medium for the spread of information, ideas, and influence among its members. Behaviors tend to cascade from node to node like an epidemic. There are many examples in the real world: news, opinions, rumors, and urban legends; virus and disease propagation; topic diffusion among bloggers, or cascading failures in financial markets. In social networks, there is a long history about studying diffusion processes and mechanisms of influence spreading. Diffusion processes were studied for viral marketing effects in the success of new products [65, 98, 152, 159].

Kempe et al. [139] studied some models for social-network dynamics, and they developed approximation algorithms for maximizing the spread of influence in a social network.

Backstrom et al. [19] analyzed two large data sets with user-defined communities. They observed that the propensity of individuals to join networks, and of networks to grow rapidly, depend on properties such as the structure of the underlying social network.

Anagnostopoulos et al. [13] considered the problem of identifying social influence, and how to distinguish it from other factors that can induce statistical correlation between the actions of friends in a social network.

1.2.2 Matrix algebra

The analysis of large graphs requires to discover patterns, summaries and anomalies in spite of the high dimensionality of the data. This is a difficult task, which becomes even more challenging when the data evolve over time. Specifically for applications that generate huge volumes of data with high speed, the methods have to be fast in order to be effective.

Graphs and matrices are closely related: in many cases, the information captured with a graph can be also represented with a matrix. For example,
consider a network administrator, who has to monitor IP flows over time. For a given time window, the traffic information can be represented as a matrix, where the sources are the rows and the destinations are the columns. In this setting, we may want to find interesting patterns for a given window, as well as across multiple windows. Many powerful tools for graph mining come from matrix algebra. Below we recall some popular methods for summarizing and approximating matrices.

Singular Value Decomposition (SVD) is a powerful tool for graph analysis: it discovers concepts in a document-term matrix (LSI [153]); it can also find fixed points or steady-state probabilities. Examples are two popular ranking algorithms, PageRank [62] and HITS [141]. The PageRank vector is given by the first eigenvector of the transition matrix (modified with teleporting). The hub and authority scores computed by Kleinberg’s algorithm correspond to the strongest left and right singular vectors of the adjacency matrix. SVD can be also used for reducing dimensionality (KL [111]) and finding rules (PCA [161]).

Although SVD and PCA are very successful in general for large sparse graphs, these methods have the major drawback of requiring huge amounts of space, because the matrices resulting from these transformations are not sparse anymore. Large, real graphs, with prominent example the Web graph, are often very sparse: Thus, they need matrix decompositions that are able to avoid the loss-of-sparsity problem.

Other low-rank approximations address the above issue and also provide a more intuitive interpretability. Drineas et al. [101] introduced the CUR matrix-decomposition method. CUR is a low-rank approximation explicitly expressed in terms of a small number of rows and columns of the original matrix: It is a decomposition in terms of the data, thus it is much more understandable. CUR was applied in recommendation systems [102] to reconstruct user-product relationships based on a small number of samples about users and products.

Sun et al. [194] presented CMD, a method for computing low-rank approximations that are provably equivalent to CUR, but are more efficient because they require less space. The authors also showed how to use CMD for detecting anomalies and monitoring time-evolving graphs.

1.2.3 Graph clustering

An interesting feature characterizing many real-world networks is the community structure, that is, the gathering of vertices into groups such that there is a higher relatedness of vertices within groups than between them. The problem of clustering networks to detect such communities is well studied and has extreme importance in a wide range of scenarios.
In any communication network, graph clustering serves as a tool for analysis, modelling and prediction of the function, usage and evolution of the network. Applications include business analysis, marketing, improving the infrastructure, and identifying anomalous use.

In the field of bioinformatics, graph clustering tasks typically deal with classification of gene expression data and protein interactions, as well as epidemic spreading.

In social networks, cluster analysis is relevant to market studies, as it helps to identify the formation of trends.

Traditional approaches adopted for graph clustering are k-means [158] and spectral clustering [172]. METIS [164] is a standard algorithm for finding disjoint communities. The above methods are affected by significant limitations: they require user-given parameters, they don’t scale well to large data sets, and they compute a one-way clustering, meaning that they cluster data by taking into account similarities along one dimension only. The latter feature makes them a not proper choice for the analysis of multi-dimensional data.

In many cases it is desirable to co-cluster, or simultaneously cluster multiple dimensions of the input data, to capture the entire semantics of their relations. Various approaches were proposed for the co-clustering problem, exploring different methods: information-theoretic co-clustering [96], cross-associations [76], or bayesian co-clustering [186].

Faloutsos et al. [196] recently introduced another problem that is relevant in the area of clustering/community-finding. The problem is that of identifying, given a graph, the centerpiece, that is, the node that has directed or indirected connections to all the nodes in a set. This has several applications in law enforcement (find the mastermind criminals, connected to all the suspected), gene regulatory networks, (find the protein that interacts with most of the other proteins), and viral marketing.

### 1.2.4 Tensors for mining graphs over time.

Matrices and matrix operations are crucial tools for finding patterns in two-dimensional data. However, powerful as they may be, they can only handle problems where there are two dimensions or aspects of interest. In many real-world problems, there are more aspects to take into consideration: for example, consider the problem of analyzing a collection of Internet data where each record has sender, receiver, port number, and timestamp.

A tensor is a multi-dimensional array: it can be viewed as a generalization of a matrix associated with three or more aspects. A matrix describes a graph, while a tensor can describe a graph with nodes and edges of various types, time-evolving graphs, time series. Tensors allow to face a wider
1. INTRODUCTION

range of applications that graph cannot even touch: they can be used whenever one has to process real data that are in high dimensions with multiple modes. Relevant examples include: rich, time-evolving data about network traffic; social networks provided with edge labels that represent different semantic relations among the entities involved (for example, who called whom, who texted whom, who emailed whom); microarray data representing gene expressions evolving over time. In general, tensors are useful for every kind of multidimensional data. The third dimension can be something different than time: e.g., sales data may consider as relevant dimensions customer, product, and branch; treatment data records may include patient, diagnosis, and drug.

Tensors have elegant theory and algorithms [72, 144]. The two most popular tensor decompositions are Tucker [199] and PARAFAC [40]. Both find row-, column-, and tube-groups. Recently, Kolda et al. [145] proposed DTA, a space-efficient method for dynamic tensor analysis, and STA, a streaming, tensor-analysis method, which provides a fast streaming-approximation to DTA.

1.2.5 Spectral graph theory

Spectral graph theory [79] uses the spectra of specific matrices associated with the graph, such as the adjacency matrix, the Laplacian matrix, or their normalized forms, to provide information about the graph. For certain families of graphs it is possible to characterize a graph by the spectrum (of one of these matrices). More generally, this is not possible, but useful information can still be extracted for important applications in various fields.

Examples of parameters that are determined by the spectrum of the adjacency matrix are the number of paths and the number of triangles in the graph. One use of spectral graph theory is also to assist in determining whether two graphs are isomorphic. Examples of graphs that are spectrally determined by the adjacency matrix are complete graphs, empty graphs, and some classes of regular graphs.

Other results concerning vertex connectivity, diameter, sparse cuts and normalized cuts involve the Laplacian or the normalized Laplacian. Sparse cuts are important for a number of different applications, like clustering and community detection, as well as telephone network design, VLSI layout, and parallel computation.

1.2.6 Streaming algorithms

A natural approach that can be used to perform computations on huge data sets is the one based on the adoption of the data-stream model. In this model, data arrive in a stream, one item at a time, and the algorithms are required
to use very little space and per-item processing time, and they are often not allowed to perform multiple passes of computation. This restriction is motivated by the need to process data of so large size that one can only afford to store them in secondary memory and make a small number of passes.

The model was first formalized by Henziger et al. [128], who gave several algorithms and complexity results related to graph-theoretic problems and their applications. The interest in the model started from the results of Alon et al. [12], who proved bounds on the memory requirements of one-pass algorithms for computing statistics over data streams. Streaming solutions were provided for many problems, including frequency estimation, norm estimation, order statistics, synopsis structures, signal reconstructions.

Recently, two excellent surveys [17, 168] discussed many algorithmic tools for data streams, among which we highlight two powerful primitives: sampling and sketches. Sampling is a simple and effective method to deal with massive data. Many sampling algorithms have been proposed in the streaming setting, such as reservoir sampling [201], concise samples and counting samples [116], and distinct sampling [115]. Sketching is another powerful technique that achieves dimensionality reduction using random projections that are fast to compute and still preserve the distances between nodes. Sketching has been used to compute many important statistics, like, for instance, $L_p$-norm [130].

**Graph streaming problems** were considered since the beginning of the data-stream model [128]. Unfortunately, many results showed that a large amount of space is required for these types of problems. Subsequent early work considered counting the number of triangles in a graph [31] and estimating common neighborhoods [66]. Again, a large component of these results were negative, proving that many complicated computations are not possible in this model using only a small amount of space. Most graph algorithms need to access the data in a very adaptive fashion. Since the entire graph can not be stored, emulating a traditional algorithm may necessitate an excessive number of passes over the data.

The semi-streaming model [105] is characterized by an $O(n \text{ polylog } n)$ space restriction, that is, space proportional to the number of nodes rather than the number of edges. For dense graphs this represents considerably less space than that required to store the entire graph. This restriction was identified as a *sweet spot* for graph streaming in a survey article by Muthukrishnan [168], and it was first explored by Feigenbaum et al. [105]. The W-Stream and Stream-Sort models, presented by Demetrescu et al. [94] and Aggarwal et al. [9] respectively, extend the semi-streaming model introducing the usage of intermediate, temporary streams to obtain effective space-passes trade-offs for natural graph problems.
1.2.7 Tera/peta-scale graph mining

MapReduce [93, 175] is a programming model introduced by Google that has recently gained great popularity for the processing of data sets of very large scale. This model allows users to write map/reduce components with functional type code. The user first specifies a map function that is used for processing the data; the results are then merged together using a reduce operation. The MapReduce run-time system schedules these components to distributed resources for processing, and at the same time it guarantees a transparent handling of many problems, like parallelization, network communication, and fault tolerance. Typical MapReduce computations are distributed sorting, document clustering, machine learning, inverted index construction.

Although a user can employ MapReduce on a single machine, MapReduce frameworks are designed to support operations on a cloud of computers. Distributing the computation on a large collection of commodity computers allows the technique to gain a great increase in terms of scalability, by having a large number of independent instantiations of the MapReduce job components on different partitions. However, MapReduce processes are interesting beyond the cloud as well. Once a problem has been factored in terms of MapReduce primitives, the same primitives may be also useful for streaming computations or on a single computer equipped with a large disk.

Graph problems have not been deeply explored in MapReduce. This is due to the fact that iterative computations and graph traversing are very inefficient in such a model, because a mapper only reads a random record for each map operation, thus causing the need for a large number of iterations. Coehn [81] proposed implementations for several graph problems, such as augmenting edges with degrees, enumerating triangles and rectangles, finding trusses and components. In the above work, each graph operation is decomposed into a series of MapReduce processes. The obtained implementations are not efficient, because they require many iterations, not to mention the fact that the initialization of every job is very expensive. To make graph algorithms really feasible, the MapReduce model must be enriched with easier support for iterative computations and graph traversing. J. Ekanayake [5] has very recently developed \textit{i-MapReduce}, a streaming-based runtime environment that supports iterative MapReduce computations efficiently.

1.2.8 External-memory algorithms

Another possibility that has been deeply investigated to develop efficient solutions for data-intensive applications is that of external memory algorithms [202]. When the data to be processed are too massive to fit completely into the internal memory of computers, the resulting input/output communication between
faster internal memory and slower external memory becomes a critical bottleneck. The external memory model [202] incorporates data placement and movement directly into the algorithm design.

These algorithms are typically based on the adoption of the parallel disk model (PDM) introduced by Vitter and Shriver [203]. The two key mechanisms for efficient algorithm design in PDM are locality of memory references (which takes advantage of block transfer) and parallel disk access (which takes advantage of multiple disks). Substantial gains in performance are obtained through an explicit management of the contents of each level of the memory hierarchy, therefore bypassing the virtual memory system.

A variety of paradigms have been proposed for solving batched or online problems efficiently in external memory. For the batched problem of sorting and related problems like permuting and fast Fourier transform, the key paradigms include distribution and merging. The paradigm of disk striping offers an elegant way to use multiple disks in parallel. Other useful techniques for batched EM problems involve matrices, geometric data, and graphs.

Research on external memory algorithms took into consideration several graph problems, including shortest paths and graph traversals. With the advent of multi-core architectures, there has been an increased realization that parallelism is the primary mechanism for achieving orders-of-magnitude improvements in performance. Arge et al. [15] introduced the parallel external memory model, which aims at modeling the modern chip-level multiprocessor architectures. In this model, Arge et al. studied the problems of connected components, biconnected components and minimum spanning tree in a connected graph.

1.3 Contributions of the thesis

This thesis addresses several problems that have arisen great interest in the fields of graph mining and Web search. We provide contributions in three main areas, which are, respectively, algorithmic methods for graph mining, analysis of the temporal evolution of the UK Web graph, and mining graphs extracted from query logs.

1.3.1 Algorithmic methods for graph mining

Characterizing real-world networks requires to deal with critical challenges, such as the huge volumes of data collections and the distributed nature of many sources of information. Streaming algorithms have proven to be very
effective tools to handle these difficulties. For this reason, we have adopted this computational model to study two specific problems: counting frequent subgraphs in large networks and computing aggregating measures in a fully decentralized setting. These works were done during the time spent at Sapienza University of Rome.

**Counting subgraphs in large-scale networks**

One of the most basic tools for analysing graph structures and revealing the properties of the underlying data is finding frequent patterns in graphs. The task of mining frequent subgraphs finds numerous applications in various domains, including: network characterization \[165, 166\], modeling complex networks \[150, 149\], detecting anomalies and malicious activity \[37\], and indexing graph databases \[211\].

Despite the numerous applications of counting subgraphs, and despite the fact that many interesting graph data have really large scale, a lot of existing work have been restricted to the analysis of small networks. Counting subgraphs on large data sets turns out to be a challenging computational task, due to the explosion of the number of candidates involved.

In chapter 2, we present a class of data-stream algorithms that approximate the number of all subgraphs of three and four vertices in directed and undirected graphs. We extend the techniques of Buriol et al.\[68\] for counting triangles in data streams. Our algorithms are based on random sampling and they operate in the *incidence* data-stream model with at most three passes on the data. We show that our algorithms guarantee to compute an approximation of the true number of subgraphs of arbitrary quality, with high probability. The number of samples depends only on the required precision of the approximation and on the structure of the network.

We demonstrate the practical applicability of our algorithms by developing optimized implementations and evaluating their performance on networks from many real application domains, such as Web graphs, Wikipedia graphs, protein-protein interaction networks, food-Webs, graphs representing the US road system or the topology of the Internet. For comparison, we also created a large number of synthetic networks using graph generators that obey the models of Barabási and Albert \[33\], and Kleinberg \[142\] respectively. The graphs analyzed have very different sizes, up to one billion edges.

We perform extensive experiments that demonstrate the relevance and usefulness of our subgraph counting algorithm for the task of recognizing families of complex networks. We show that the precision obtained by clustering and classification algorithms that use as features the distributions of small subgraphs in the networks cannot be achieved with simpler topological features as for instance degree, assortativity and eigenvalue distribution. This work is presented in \[57\].
Fully decentralized computation of aggregates over data streams

In several emerging applications, e.g. sensor networks and network monitoring, data is collected in massive streams at several distributed points of observation. A basic and challenging task is to allow every node to monitor a neighbourhood of interest by issuing continuous aggregate queries on the streams observed in its vicinity. This class of algorithms is fully decentralized and diffusive in nature: collecting all data at few central nodes of the network is infeasible in networks of low-capability devices or in the presence of massive data sets. The main difficulty in designing diffusive algorithms is to cope with duplicate detections. These arise both from the observation of the same event at several nodes of the network and/or receipt of the same aggregate information along multiple paths of diffusion.

In chapter 3, we describe compact distributed data structures and a communication scheme to keep track of statistic aggregates over distributed data streams. We are able to answer at any node of the network continuous queries on the number of distinct events, total number of events and the second frequency moment over the streams observed within a suitably defined neighborhood of each node. These algorithms require only polylogarithmic storage space at each node of the network for the number of distinct events, and the total number of events. For the second frequency moment, we show a polylogarithmic bound on the storage size at each node when the distribution of events follows a Zipf’s law.

We show optimized implementations and tests on a realistic network topology. The graph used in the experiments is a snapshot of (part of) the Internet at the level of the Autonomous Systems collected by DIMES [187] in December 2008. The traffic streams were obtained from large-scale real or synthetically generated data. Real data were extracted from a collection of HTTP requests submitted to the 1998 World Cup Web site, made available\(^2\) by the Internet Traffic Archive [16]. We also generated synthetically a large stream of skewed data, which contains approximately 17 million records.

Experimental results show that our fully decentralized algorithms achieve in practice very good bounds on storage, computation and communication costs while providing a good approximation of the statistics of interest. We characterize the trade-off between the worst-case communication cost and accuracy for the sketch schemes we consider. Moreover, we experimentally evaluate the performance of a conservative strategy to trade off communication costs and accuracy. This work is described in [36].

\(^2\)URL: http://ita.ee.lbl.gov/html/contrib/WorldCup.html
1.3.2 Temporal characterization of the UK Web

The Web is characterized by an extremely dynamic nature, as it is proved by the rapid and significant growth it has experienced in the last decade and by its continuous evolution through creation or deletion of pages and hyperlinks. Consequently, analyzing the temporal evolution of the Web has become a crucial task that can provide search engines with valuable information for refining crawling policies, improving ranking models or detecting spam.

Understanding and analyzing the structure of the Web is an important and delicate challenge that requires theoretical efforts of modelization to be always corroborated by empirical findings.

The problem becomes even more elusive if time evolution is taken into account, because one would like to have not only different snapshots of the same portion of the Web to be available, but also some guarantee on their mutual consistency (for example, to be sure that the same crawling policies have been followed) is in that case of imperative importance.

Recently, a new temporal dataset has been made public: it is made of a series of twelve 100M pages snapshots of the .uk domain [48], which were gathered at a monthly rate. The Web graphs of the twelve snapshots have been merged into a single time-aware graph that provides constant-time access to temporal information.

In chapter 4, we present an extensive analysis that was conducted on this collection, which represents the first public data on the evolution of the Web that use a large scale and a significant diversity in the sites considered. This study was started at Sapienza University of Rome, and completed while visiting Yahoo! Research Labs in Barcelona for the joint program with Pompeu Fabra University.

We start our work by assessing the data obtained through crawling. Next, we analyze the structure of this huge time-aware graph at the level of interconnection between hosts, studying the temporal evolution of 3500 sites with respect to a number of topological properties, including degrees, number of degree supporters and eigenvector distributions. We then characterize the dynamics of the data considering the level of interconnection between pages. We first quantify the turnover rate of pages and links, observing a highly dynamic behavior: existing pages are removed from the Web and replaced by new pages at a very rapid rate. The evolution of the hyperlink structure is even faster than that of Web pages.

We also investigate how the content of Web pages evolves over time. We study several measures for the frequency and the degree of change of the content of a page over a series of consecutive snapshots. Our results show that most of pages change very rarely, or they are affected by minor changes during the time interval taken into consideration. Our analysis also shows that past
degree of change is a very good predictor of the future change. For a complete presentation of the results of this work, see [53, 55, 56].

1.3.3 Mining query-log graphs

Information extracted from query logs has shown to be effective for many different applications in the services provided by a search engine. Graphs can be used to provide a compact and navigable representation of the query-related information extracted from query logs. In this thesis, we extend the work that has been done about studying query-log graphs by exploring two novel ideas: (i) analyzing query-log data through the joint mining of multiple graph representations that capture different kinds of semantic information (chapter 5); (ii) applying spectral methods to the problem of studying query-reformulations (chapter 6). These works were done during the time spent at Yahoo! Research, Barcelona and they focused on the analysis of large-scale query-log data provided by Yahoo!.

Multiple-graph mining for query-log analysis

In chapter 5, we present a general framework for query-log analysis, which aims at providing an effective support for the execution of many tasks that are relevant from a search-engine point of view. The fundamental building blocks of our framework consist of (i) a collection of graph projections extracted from a query log according to different definitions, each capturing a peculiar semantic relation between queries, and (ii) a set of operations that are provided for the mining and maintenance of the graphs.

We build and study three different graphs, which are defined by Baeza-Yates [21] and relate queries accordingly to various types of information: common words, common clicked results, session information. We used MapReduce [93] to extract the graphs from a large piece of log provided by Yahoo!. The graphs were stored using WebGraph [49], a framework for graph representation that exploits modern compression techniques.

The selection of operations was essentially application driven. Our toolbox contains set operations and more complex graph operations, like extraction of subgraphs, connected components and articulation points. Although interesting results can be obtained by applying operations as simple as set operations, we show that more complex graph algorithms are needed to exploit all the wealth of information that is enclosed in the graphs. Interestingly, the operations are also used for the maintenance of the framework: for example, the union of two graphs can be computed to merge the data extracted from subsequent snapshots of a query log. The usage of very compact and navigable graph representations allowed us to develop main-memory implementations for all the operations introduced in the framework. We also investigated some
possible alternative implementations.

It is important to observe that the approach we follow is completely unsupervised: it does not require the usage of any knowledge bases or other expensive editorial resources. This characteristic makes the method very general and applicable to different data, without any kind of linguistic and culture-specific issues.

Although we aim at showing that our approach allows to devise a general off-the-shelf toolbox for many different problems, we show how the method can be customized in practice for two specific applications, which are, respectively, classifying query-reformulation types and detecting spam queries, achieving good performance. This work is described in [52].

**Query similarity by projecting the query flow graph**

In this work, we exploit the information present in the query-flow graph in order to develop a measure of semantic similarity between queries. Query similarity is not easily quantifiable as it strongly depends on user intent. Queries similar from a syntactic perspective may be motivated by completely different intents. When attempting to define query similarity, there are many issues arising due to polysemy, synonymy, and the very small amount of available information. In such a complex setting, methods based on reformulation paths have shown to be effective.

In chapter 6, we show that graph-projection techniques can boost the ability of recommendation-systems to approximate the human notion of relatedness between queries. We describe a method for measuring similarity between queries by projecting a query-reformulation graph (or appropriate subgraphs extracted from it) on a low-dimensional Euclidean space; the graph was built using a set of sessions extracted from a query log of the Yahoo! search engine.

We show that our similarity measure captures the human notion of related queries better than other measures on the original graph. We apply this method to the task of producing diverse and useful recommendations, and we show how to improve its efficiency further by applying the projection locally in the neighborhood of the input query. This work is presented in [54].
Chapter 2

Counting Subgraphs in Large Networks

The problem of mining frequent patterns in networks has many applications, including analysis of complex networks, clustering of graphs, finding communities in social networks, and indexing of graphical and biological databases. Despite this wealth of applications, the current state of the art lacks algorithmic tools for counting the number of subgraphs contained in a large network.

In this work we develop data-stream algorithms that approximate the number of all subgraphs of three and four vertices in directed and undirected networks. Thus, we extend significantly existing work that has focused mostly on counting triangles in undirected graphs. The algorithms we propose are based on random sampling. The number of samples depends only on the required precision of the approximation and the structure of the network. Our algorithms require at most three passes, when we assume that the network is stored as a sequence of incidence lists.

As a practical application, we use the frequency of occurrence of all subgraphs to prove their significance in order to characterize different kinds of networks: we achieve very good precision both in clustering and classifying networks with similar structure. The significance of our method is supported by the fact that such high precision cannot be achieved through the usage of simpler topological properties, such as degree, assortativity, and eigenvector distributions. We have also tested our techniques using swap randomization.
2.1 Introduction

Graphs are ubiquitous data representations that are used to model complex relations in a wide variety of applications, including biochemistry, neurobiology, ecology, social sciences, and information systems. One of the most basic tools for analysing graph structures and revealing the properties of the underlying data is finding frequent patterns in graphs. The task of mining frequent subgraphs finds numerous applications in various domains, including:

**Network characterization.** As Milo et al. [165, 166] observed, different families of networks can be characterized by specific subgraph patterns occurring at numbers that are significantly higher than those observed in randomized networks with the same single-node characteristics (such as degree distributions). Therefore, the distribution of the (frequent) subgraphs in the network can be used to characterize the networks and classify them into families.

**Modeling complex networks.** Assuming that the distributions of subgraphs in a network are “signatures” of the underlying generating process, any realistic model for such network should produce networks with similar subgraph distributions. As an example, consider the “copying model” for the growth of the Web [149], which was obtained by devising a process that generates small bipartite cliques. Such bipartite clique structures have been observed to occur frequently in the Web graph [150], and they have been hypothesized to be signatures of web communities.

**Anomaly detection in graphs.** Mining frequent subgraphs allows to detect anomalies, outliers, and activity of malicious users. For instance, the topology of the Web is adversarially modified by spammers, who introduce a number of local structures (“link farms” [37]) attempting to increase the PageRank score of certain pages.

**Indexing graph databases.** Consider the graph query problem, which consists of finding all the graphs of a database that contain a given query graph as a subgraph. Yan et al. [211] proposed a solution for this problem based on indexing the most frequent subgraphs, up to a maximum size, which occur in the graphs belonging to the database, thus, making the frequent-subgraph mining problem a crucial component.

Despite the numerous applications of counting subgraphs, and despite the fact that many interesting graph datasets have really large scale (e.g., the Web graph), a lot of the existing work has been restricted to the analysis of small networks [166, 200]. Counting subgraphs on large datasets is a very challenging computational task, due to the explosion of the number of candidate subgraphs involved. Even the most basic problem, counting the number of triangles in an undirected graph, reduces to matrix multiplication.
A natural approach that can be used to perform computations on huge data sets is the one based on the adoption of the data-stream model [168]. In this model, data arrive in a stream, one item at a time, and the algorithms are required to use very little space and per-item processing time. This restriction is motivated by the need to process data of so large size that one can only afford to store them in secondary memory and make a small number of passes. The most basic subgraph-counting problem, counting the number of triangles in an undirected graph, has indeed been studied in the data-stream model, and many algorithms have been proposed on different variants of the stream model with different types of approximation guarantees [31, 68, 135]. However, to the best of our knowledge, the problem of designing efficient data-stream algorithms for counting subgraphs other than undirected triangles in large-scale graphs has not been studied before.

In this work, we generalize the techniques of counting undirected triangles in large graphs and we develop data-stream algorithms that approximate the number of all subgraphs of three and four vertices in directed and undirected graphs. Thus, we extend significantly the existing work in the area of subgraph mining. More specifically, our contributions can be summarized as follows.

- We extend the techniques of Buriol et al. [68] for counting triangles in data streams, and we develop a general framework for approximating the number of all subgraphs of three and four vertices in directed and undirected graphs. Our algorithms are based on random sampling and they operate with at most three passes on the data. We show that our algorithms guarantee to compute an \((1 \pm \epsilon)\) approximation of the true number of subgraphs, with probability at least \((1 - \delta)\), where \(\epsilon\) and \(\delta\) are respectively representing the error and the confidence parameters that can be arbitrarily chosen by the users. The number of samples depends only on the required precision of the approximation and on the structure of the network.

- We demonstrate the practical applicability of our algorithms by developing an optimized implementation and evaluating their performance on networks from many real application domains of size up to one billion edges. We extensively evaluate the algorithms by comparing various sampling strategies and reporting their running-time performance and the approximation achieved.

- We perform a wide set of experiments that demonstrate the relevance and usefulness of our subgraph-counting algorithm for the task of recognizing families of complex networks. We provide experimental evidence that further supports the results of Milo et al. [165, 166], and we extend their work. We demonstrate that the precision obtained by clustering
and classifying algorithms that use as features the distributions of three- and four-node patterns in the networks cannot be achieved with simpler topological features as for instance degree, assortativity and eigenvalue distribution. We also assess the statistical significance of our method by swap randomization [117].

2.2 Related work

The problem of discovering frequent subgraphs has been studied extensively in the area of data mining [95, 129, 151, 210]. The algorithmic techniques for this problem are mostly based on the a-priori principle [10]. The problem we study is different than the above line of research. A key difference between our work and these studies is that we focus on counting the occurrences of subgraphs in one single large graph. A second significant difference is that the papers above consider the nodes of the graph to be labeled, and the complexity of the problem is due to the explosion of different patterns considered (which are pruned with the use of a support threshold and the a-priori strategy) rather than to the explosion of possible positions in the graph that a given subgraph can occur.

Our algorithm for clustering networks is based on the distribution of three- and four-node subgraphs and it is, to a large extent, inspired by the work of Milo et al. [165, 166], who search for those patterns whose frequency is significantly higher than the frequency that one would expect in random networks with the same degree distribution. A similar idea of testing data-mining results against random networks with a given degree distribution was also proposed by Gionis et al. [117]. Kashtan et al. [136] have proposed an algorithm that uses a randomly sampled set of subgraphs to estimate subgraph concentrations and to detect network motifs. However, this algorithm has been shown to have a bias for sampling certain subgraphs more often than others by Weranieke [207], who proposes an approach based on randomized enumeration for determining the significance of subgraphs.

The problem of finding graphlets of three and four nodes in protein-interaction networks was studied recently by Pržulj et al. [179]. Differently from our algorithms, the heuristics proposed in [179] do not have any performance guarantee on the processing time and storage requirements. Moreover, their algorithm is not scalable for large graphs. Other sampling schemes that have recently been proposed by Borgwardt et al. [59] are not directly relevant to our work.
The streaming model [128] admits an ideal approach to processing massive data sets on a disk. The input arrives sequentially and the algorithms are required to process them using very little space and processing time. More precisely, a common requirement for streaming computations is that the per-item processing time, the storage space and the global computing time are simultaneously linear or, preferably, polylogarithmic in the size of the input stream. In some cases, a larger global computation time can be accepted. The input must be read only once or a small number of times. A key point is that no random access to the data stream is allowed: once a given item is read, the algorithm cannot go back to a previous item again. However, in some cases the algorithm is allowed to make multiple passes of computation over the input: after it reads the last item, it may start a new pass by reading again the stream starting from the first item.

The data-stream model was first formalized by Henziger et al. [128], who gave several algorithms and complexity results related to graph-theoretic problems and their applications. The interest in the model started from the results of Alon et al. [12], who proved bounds on the memory requirements of one-pass algorithms for computing statistics over data streams. Streaming solutions were provided for many problems, including frequency estimation, norm estimation, order statistics, synopsis structures, signal reconstructions [17, 115, 116, 128, 130, 168, 201].

The large body of work on data-stream algorithms contrasts with a lack of efficient solutions for many natural graph problems. Graph streaming problems were considered since the beginning of the data-stream model [128]. Unfortunately, many results showed that a large amount of space is required for these types of problems. Most graph algorithms need to access the data in a very adaptive fashion. Since the entire graph can not be stored, emulating a traditional algorithm may necessitate an excessive number of passes over the data. Previous to our work, streaming algorithms for counting triangles have been presented in [31, 135, 68, 69]. In this work we extend the techniques of counting triangles to counting all subgraphs of size 3 and 4 for directed and undirected graphs. Contemporaneously with our work, Tsourakakis [198] has proposed another method for fast triangle counting based on the connection between the number of triangles and the eigenvalues of the adjacency matrix.

The semi-streaming model [105] is characterized by an $O(n \text{polylog } n)$ space restriction, that is, space proportional to the number of nodes rather than the number of edges. For dense graphs this represents considerably less space than that required to store the entire graph. This restriction was identified as a sweet spot for graph streaming in a survey article by Muthukrishnan [168], and it was first explored by Feigenbaum et al. [105]. The W-Stream and Stream-Sort models, presented by Demetrescu et al. [94] and Aggarwal et
al. [9] respectively, extend the semi-streaming model introducing the usage of intermediate, temporary streams to obtain effective space-passes trade-offs for natural graph problems, like graph connectivity, minimum spanning tree and geometrical problems. However, the semi-streaming assumptions are in practice prohibitive for large graphs.

2.3 Algorithms for counting subgraphs of three and four nodes

Let $G = (V, E)$ be a graph, which can be either directed or undirected. Our basic model for the representation of $G$ is the “incidence” stream model, in which we assume that all edges incident to the same vertex appear subsequently in the stream. In the case of undirected graphs, every edge appears twice—in the incidence list of both incident nodes.

Notice that if the dataset contains the edges of the graph in arbitrary order, it can be converted to the incidence format described above with an additional sorting step, which, in practice, can be performed efficiently even for very large datasets stored in secondary memory. The ordering $v_1, \ldots, v_n$ of the vertices can be arbitrary. We denote by $d_i$ the number of vertices incident to vertex $v_i$.

We present a general three-pass sampling algorithm for counting the number of occurrences of a specific subgraph $M$ of $c$ vertices in the graph $G$. We denote by $M = (X_M, Y_M)$ a simply connected subgraph of $G$ having no multiple edges and no vertex loops. Here $X_M = \{\overline{x}_1, \ldots, \overline{x}_c\}$ and $Y_M$ are the vertices and edges of $M$, respectively.

The subgraphs of size 3 and 4 for undirected and directed graphs are shown in Figures 2.1, 2.2, 2.3, and 2.4. In the figures all the possible subgraphs are shown, except for the case of directed patterns of four nodes, where only four subgraphs are shown out of 199 possible ones. The complete list of the directed four-node patterns is reported in Appendix 7. Observe that we only consider the 190 subgraphs containing a pair of edges with the same source node, as this is the basic substructure that we use for sampling.

![subgraphs](image)

Figure 2.1: All subgraphs of size 3 for undirected graphs.
2.3. Algorithms for counting three- and four-node subgraphs

We now describe in detail the algorithm for counting subgraphs. First we fix a subgraph $M = (X_M, Y_M)$, whose number of occurrences we want to count in the graph $G$. The algorithm uses two basic concepts: (i) the concept of a \textit{prototype} $S_M$ for $M$, and (ii) the concept of the \textit{sample space} $S$ for $M$ with respect to $S_M$.

The prototype $S_M = (X_M, Y_M')$, $Y_M' \subseteq Y_M$ is simply a subgraph of $M$ defined on the same set $X_M$ of $c$ vertices as $M$. For example, if $M$ is the subgraph representing an undirected triangle, then $S_M$ can be a path of length 2. The sample space $S$ is defined to be the set of all distinct subgraphs $S$ of $G$ that are \textit{isomorphic} to $S_M$.

At a very intuitive level, $S$ defines the set of “candidate places” in which $M$ can potentially appear in $G$. The algorithm samples such candidate places from $S$, checks if $M$ actually appears in those places, and then uses the count of the occurrences of $M$ in the sample to estimate the number of occurrences of $M$ in $G$.

More formally, we define $S$ to be the set of subgraphs $S = (X, Y)$ of $G$ for which there is a bijection $f : X \rightarrow X_M$, such that for each $x_1, x_2 \in X$ it is $(x_1, x_2) \in Y$ if and only if $(f(x_1), f(x_2)) \in Y_M'$. Given a subgraph $S = (X, Y)$ in $S$, we define $\bar{Y}(X)$ to be the edges that are needed to extend $S$ to an occurrence of $M$, that is, $\bar{Y}(X) = \{(f^{-1}(\bar{x}_1), f^{-1}(\bar{x}_2)) : (\bar{x}_1, \bar{x}_2) \in Y_M/Y_M'\}$. Finally, we denote by $|S|$ the size of the sample space $S$. 

Figure 2.2: All subgraphs of size 4 for undirected graphs.

Figure 2.3: All subgraphs of size 3 for directed graphs.
2. SUBGRAPH COUNTING

The general three-pass algorithm is the following:

**SampleSubgraph**

1st. Pass: Compute the size $|S|$ of the sample space.

2nd. Pass: Uniformly choose a member $S = (X,Y)$ of the sample space.

3rd. Pass: Run the following test:
   - if all edges in $\bar{Y}(X)$ are in the graph then $\beta = 1$
   - else $\beta = 0$
   return $\beta$

**Example.** Consider the problem of counting the number of length-four cycles in an undirected graph. Let us choose the prototype $S_M$ to be a length-two path $(a,b), (b,c)$ and a fourth vertex $d$ different from $\{a,b,c\}$. The sample space $S$ is the set of the subgraphs of $G$ that are isomorphic to $S_M$.

In the first pass of the algorithm, we compute the number of length-two paths $P = \sum_{i \in V} \frac{d_i(d_i-1)}{2}$ and the number of vertices $|V|$ of the graph. One can see immediately that for this particular choice of $S_M$, the size of the sample space is given by $P \times (|V| - 3)$.

In the second pass, a random member of the sample space is uniformly sampled. This step requires to enumerate all length-two paths in the graph and to provide an efficient method to retrieve the length-two path corresponding to a given position in the order.

In the third pass, it is returned $\beta = 1$ if the graph contains edges $(a,d)$ and $(c,d)$, $\beta = 0$ otherwise. Observe that every cycle of length 4 can be detected from 4 different elements of the sample space, formed by the combination of the 4 length-two paths of a length-four cycle and, for each of them, the corresponding missing vertex. Therefore, every length-four cycle has the same probability to be sampled.

The accuracy and the performance of the algorithms we propose rely crucially on the structure of the sample space and, consequently, on the choice of the prototype $S_M$. We observe that in order to provide a uniform sampling of
all occurrences of $M$, we need to ensure that every single occurrence of $M$ in the graph can be detected by extending the same number of distinct elements in the sample space. This number, which we denote by $n_M$, is defined to be the number of isomorphic mappings of the prototype $S_M$ to the subgraph $M$. The specified requirement is clearly guaranteed by our method since we check if all the edges needed to extend a sampled subgraph to an occurrence of the subgraph $M$ exist in the graph.

We make a few remarks on the SAMPLESUBGRAPH algorithm. The first pass requires to design a sample space whose size can be easily determined in a streaming pass over the graph. Thus, the choice of the prototype $S_M$ should be made in a way that ensures that the size of the sample space depends only on simple parameters, like the number of vertices, the number of edges, and the degree of every vertex.

The second pass of the algorithm requires to list all samples of the sample space in linear order, and to efficiently identify the sample corresponding to some position in the order. This task is easy if, say, the sample space is formed by the set of all edges of the graph. In general, more complex enumeration schemes might be needed.

Now, let us denote by $T_M$ the number of occurrences of subgraph $M$ in the graph $G$. We recall that the objective of the algorithm is to provide a good estimation of $T_M$. The following Lemma gives an exact estimation of the expected value of the parameter $\beta$, which is returned by the SAMPLESUBGRAPH algorithm.

**Lemma 2.3.1.** The algorithm SAMPLESUBGRAPH outputs a value $\beta$, which has expected value

$$E[\beta] = \frac{n_M \cdot T_M}{|S|}$$

**Proof.** The algorithm chooses a random element $S(X)$ of the sample space defined on a subset $X$ of $c$ vertices. Each of the $T_M$ subgraphs can be obtained in $n_M$ different ways, i.e., starting from $n_M$ different samples. Since each choice of a $S(X)$ has the same probability, the probability of choosing a sample that can be extended to a subgraph is $\frac{n_M \cdot T_M}{|S|}$. \(\square\)

A single run of the SAMPLESUBGRAPH algorithm returns a binary value. To obtain an estimation of the expectation of the parameter $\beta$, we perform multiple runs of the SAMPLESUBGRAPH algorithm. In particular, we start

$$s \geq \frac{3}{\epsilon^2} \cdot \frac{|S|}{n_M \cdot T_M} \cdot \ln\left(\frac{2}{\delta}\right)$$

(2.1)
parallel instances of SAMPLESUBGRAPH and return the value

\[ \tilde{T}_M := \left( \frac{1}{s} \cdot \sum_{i=1}^{s} \beta_i \right) \cdot \frac{|S|}{n_M} \]

as an estimate of \( T_M \), the number of occurrences of the subgraph \( M \) in the graph. For the quality of approximation provided by \( \tilde{T}_M \) we can show the following.

**Lemma 2.3.2.** With probability \( 1 - \delta \) the following statement holds:

\[ (1 - \epsilon) \cdot T_M < \tilde{T}_M < (1 + \epsilon) \cdot T_M. \]

**Proof.** (Sketch) We use the Chernoff bounds

\[ \Pr \left[ \frac{1}{s} \sum_{i=1}^{s} \beta_i \geq (1 + \epsilon) \mathbb{E}[\beta] \right] < e^{-\epsilon^2 \cdot \mathbb{E}[\beta] \cdot s/3} \]

and

\[ \Pr \left[ \frac{1}{s} \sum_{i=1}^{s} \beta_i \leq (1 - \epsilon) \mathbb{E}[\beta] \right] < e^{-\epsilon^2 \cdot \mathbb{E}[\beta] \cdot s/2}, \]

and the definition of \( s \) from Equation (2.1) for the number of repetitions of the algorithm. \( \square \)

We complete this section with the analysis of the running time of the algorithm. The time complexity of the first two passes of the algorithm cannot be stated in general, since they depend on the specific sampling strategy. However, if there is a constant-time method to access the \( i \)-th element of the sample space, the uniform selection of a sample can be implemented in constant time per edge of the graph by reservoir sampling \[201\], as we will discuss in the next section. For the third pass of the algorithm, if we implement the different instances of the SAMPLESUBGRAPH algorithm independently of each other, we require \( O(\frac{1}{\epsilon^2} \cdot \log(\frac{1}{\delta}) \cdot (\frac{|S|}{n_M T_M})) \) time to process each edge. However, as we will also describe in the next section, the cost of checking for \( M \) in all instances of \( S_M \) can be reduced to expected constant time per edge of the graph \( \tilde{G} \) via hashing. We therefore conclude with the following theorem.
Theorem 1. There is a three-pass streaming algorithm to count the number of subgraphs in incidence streams up to a multiplicative error of $1 \pm \epsilon$, with probability at least $1 - \delta$, which needs $O(s)$ memory cells and amortized expected update time $O(1 + s \cdot \frac{|V|}{T_M})$, where

$$s \geq \frac{3}{\epsilon^2} \cdot \frac{|S|}{n_M \cdot T_M \cdot \ln(\frac{2}{\delta})}.$$  

### 2.4 Implementation

In this section we discuss specific choices and optimization issues for our counting algorithms. We start by the choice of the sampling prototype we use. For patterns of size 3, for directed graphs, we use two prototypes, $S^3_{1d}$ and $S^3_{2d}$, shown in Figure 2.5. We sample $S^3_{1d}$ if this substructure is included in the subgraph to count. We use $S^3_{2d}$ otherwise. We next consider subgraphs of size 3 in undirected graphs. The only two patterns of this kind are the path of length two and the triangle. It is straightforward to count the paths of length two in the incidence stream model. The path of length two $S^3_{1u}$ is also used as prototype sample to count triangles, as shown in Figure 2.5.

For subgraphs of size 4, for undirected graphs, we use three prototypes, $S^4_{1u}$, $S^4_{2u}$, and $S^4_{3u}$, shown in Figure 2.6. $S^4_{1u}$ is implemented for all the subgraphs, $S^4_{3u}$ for the subgraph that have it as a substructure and $S^4_{2u}$ for the ones that do not contain $S^3_{1u}$.

For subgraphs of size 4, for directed graphs, we use only one prototype, $S^4_{1d}$, shown in Figure 2.6. Hence, we count only subgraphs that contain $S^4_{1d}$; these are 190 out of the 199 possible ones. Appendix 7 shows the complete list of such patterns.

![Figure 2.5: Prototypes for 3-node subgraphs: $S^3_{1d}, S^3_{2d}, S^3_{1u}$](image)

![Figure 2.6: Prototypes for 4-node subgraphs: $S^4_{1u}, S^4_{2u}, S^4_{3u}, S^4_{1d}$](image)
Our algorithm is a three-pass algorithm. The first pass is used to compute the size of the sample space, which must be known to perform uniformly random sampling. Computing the size of the sample space is straightforward given the sampling strategies that we adopt. We only need to know the number of nodes and edges in the graph, and the degree of every node. These properties can be immediately computed with a simple scan of the incidence stream.

The second pass of the algorithm is used to build the sample set. To choose a random member of the sample space, we define an order on the elements of the sample space. Next, we generate a random index and we select the sample corresponding to that particular index. Thus, we must be able to enumerate the elements of the sample space. The $S_3^{3d}$ and $S_3^{4u}$ prototypes require to select random edges, which is immediate in our incidence-stream model.

In the other cases, we use the procedures described by Buriol et al. [68, 69] for the uniform sampling of a length-two path or a $K_{13}$. These procedures identify the positions of the edges involved in a sample using the fact that the incidence list of a node with degree $d_i$ includes $d_i(d_i - 1)/2$ paths of length two and $d_i(d_i - 1)(d_i - 2)/6$ $K_{13}$. The edges involved a sample with index $k$ can be identified with a scan of the incidence stream: we start reading the incidence list of a node, and, if the node is involved in enough paths of length two or $K_{13}$, then we choose the proper sample, otherwise we search for the $k$-th sample in the next incidence list.

In the third pass the algorithm checks whether the samples collected in the first two phases can be completed to instances of the subgraph $M$ taken into consideration.

We are actually able in most of the cases to merge in the implementation the first two passes of the algorithm. The input graph is read twice: the former scan is used to build the sample set, whereas the latter is needed for checking the existence of subgraphs.

We now explain how the merging of the first two passes can be executed. Subsequently, we describe how the third pass can be implemented efficiently.

**Merging the first two passes into one.** In order to combine the first two passes of our algorithm, we need to choose elements at random from a sample set whose size is not known in advance. The application of *reservoir sampling* [201] makes it possible to find a random element within a given set in one pass. The idea is to sample the first element appearing in the stream and subsequently replace with probability $1/i$ the currently sampled item with the $i$-th element of the input stream. We can perform reservoir sampling in parallel for all executions of the algorithm by generating according to a specific distribution the index of the next item to be sampled. This idea for prototypes $S_1^{3d}$, $S_2^{3d}$, $S_1^{4u}$, $S_2^{4u}$, or $S_1^{4d}$ is explained in [68, 69]. For the $S_3^{4u}$ prototype we
retain the three-pass algorithm given the difficulties of implementing the more sophisticated sampling strategy.

Efficient implementation of the third pass. As we mentioned in the previous section, if we check for the existence of a given subgraph in each pattern sample in a naive way, we need $O(\frac{1}{\epsilon} \cdot \log(\frac{1}{\delta}) \cdot (\frac{|S|}{n^M \log |M|}))$ processing time for each edge of the graph. We now describe how to perform this task in expected constant time per edge. The main idea is the following: before invoking the third pass, we collect all the samples selected by the instances of the SampleSubgraph algorithm. For each sample $S = (X, Y')$, we need to test whether the edges $\bar{Y}(X)$ are present in the graph. We can test edge membership by maintaining a set $U$ of missing edges, i.e., $U$ contains the edges $\bar{Y}(X)$ for all the samples selected. We construct a hash table for $U$ using a uniform hash function that requires linear space, as proposed in [176]. Now we can implement the third pass in the following way. For each edge $e$, we look up whether it is in the set $U$. If $e \in U$ we mark it. Both these steps can be done in expected constant time. In a post-processing step we test, for every sample $S$, whether all the edges $\bar{Y}(X)$ have been marked.

We remark the fact that the amount of space required by our algorithms depends only on the required precision of the approximation, thus on the number of samples to be collected. The procedures we use to collect the samples scan the input at most twice, and they not need to keep the graph into memory. For each sample selected, we just store into the hash table a constant number of edges. The hash table is then used in the last pass to count how many samples are corresponding to actual instances of the subgraph taken into consideration.

2.5 Experimental results

We evaluate our algorithms on an extensive collection of real and synthetic datasets, which we summarize in Table 2.1. All the datasets are freely available 1.

Directed graphs.
- Wikipedia graphs: seven graphs extracted from the link structure of Wikipedia (from an old dump of June 13th, 2004[67]), considering seven different languages.
- Webgraphs: fivelarge webgraphs, extracted from crawls of Web domains like .cnr, .eu, .in, .uk. These data are made available by the Laboratory of

\footnote{www.dis.uniroma1.it/~bordino/counting_subgraphs/datasets}
Table 2.1: The datasets used for experimental evaluation.

- **Cellular networks**: 43 graphs, collected by Jeong et al. [132], representing the metabolic networks of 43 different organisms.
- **Citation graphs**: two graphs [113] representing citations among physics papers, plus a third instance extracted from the U.S. patent dataset [122].
- **Food webs**: six instances, collected by the COSIN Project [3], representing the predatory interactions among different species in a given environment.
- **Word adjacency networks**: four graphs are extracted from texts written in different languages [165]: Each node corresponds to a word, and two nodes are connected when the word represented by the target node directly follows the word corresponding to the source node in the text.
- **Synthetic graphs**: we created 39 instances using two graph generators that obey some well-known models [149, 33] for the growth of the Web. The graph generators are included in the COSIN library, developed within the COSIN project.

### Undirected graphs.
- We have generated the undirected versions of the Wikipedia graphs, the Web graphs and the synthetic instances.
- **Autonomous systems graphs**: 12 instances, collected within the COSIN project, representing the Internet topology.
- **Collaboration graphs**: one coauthorship network based on the Internet Movie Database, collected by Barabási and Albert ([33]), and 4 networks of coauthorships between scientists posting preprints on different archives, provided by M.E.J. Newman [171, 170].

---


2.5. Experimental results

| Subgraph | Graph | Nodes (M) | Edges (M) | Exact count | $|S| = 10,000$ | $|S| = 100,000$ | $|S| = 1,000,000$ |
|----------|-------|-----------|-----------|-------------|----------------|----------------|----------------|
|          | WikiEN | 0.34      | 5.3       | 5.2         | -4.7           | 3.63           | 2.70           |
|          | WikiDE | 0.12      | 6.2       | 24.15       | 10.46          | 0.55           | -2.19          |
|          | cn-2000| 0.32      | 3.2       | 9.4         | -5.43          | 1.02           | -1.70          |
|          | eu-2005| 0.96      | 19        | 140         | -6.50          | 4.85           | -2.84          |
|          | in-2004| 1.38      | 16.9      | 790         | 4.84           | 4.00           | 2.01           |
|          | uk-2002| 15        | 100       | -           | -              | 77.43          | -              |
|          | uk-2005| 39        | 936       | -           | -              | 302.64         | -              |
| M_{11}   |           |           |           |             |                |                |                |
| M_{12}   |           |           |           |             |                |                |                |

Table 2.2: Results obtained for subgraph $M_{11}$ in the graphs extracted from crawls of Wikipedia and of Web domains.

- **Protein interaction networks**: 3 graphs collected in 2000-2001 and maintained by the DIP database at the University of California\(^4\). They represent the interactions among proteins for different organisms.
- **US-road-graphs**: 12 graphs representing USA road networks, used for the 9th DIMACS Implementation Challenge about Shortest Paths\(^5\).

**Quality of approximation**

Our first set of experiments aims at showing that our methods produce very accurate counts. To evaluate the quality of the approximation, we compare the counts computed by our methods with counts computed by an exact method. We are able to compute the exact counts of subgraphs of 3 nodes for all our datasets. However, the exact computation of 4-node subgraphs is a very expensive task and thus we are not able to perform such computation for the largest instances in our collection.

**Directed 3-node subgraphs**: For the sake of concreteness and clarity of presentation, we present approximation results on counting the occurrences of two particular directed 3-node subgraphs, $M_{11}$ and $M_{12}$. We have verified that the quality of approximation is similar for many other subgraphs.

Table 2.2 and 2.3 report the results obtained for two subgraphs, namely, $M_{11}$ and $M_{12}$, in two Wikipedia graphs and five large Web graphs. These graphs represent the largest instances in our data collection. For each input network, the tables indicate the size of the graph, expressed in terms of number

\(^4\)http://dip.doe-mbi.ucla.edu/

\(^5\)http://www.dis.uniroma1.it/~challenge9/
### 2. Subgraph Counting

| Subgraph | Graph | Nodes | Edges | Exact count | $|S| = 10,000$ | $|S| = 100,000$ | $|S| = 1,000,000$ |
|----------|-------|-------|-------|-------------|---------------|----------------|------------------|
| $M_{12}$ | WikiEN | 0.34  | 5.3   | 0.69        | 7.44          | 2.49           | 1.11             |
|          | WikiDE | 0.12  | 2.1   | 0.96        | 6.30          | 2.92           | 0.08             |
|          | en-2000| 0.32  | 3.2   | 1.5         | 11.2         | 4.64           | 4.31             |
|          | en-2005| 0.86  | 19.3  | 23.92       | 8.67          | 5.32           | 3.67             |
|          | in-2004| 1.38  | 16.9  | 130         | 2.83          | 1.65           | 0.29             |
|          | uk-2002| 1.8 $\cdot 10^7$ | 1.9 $\cdot 10^8$ | - | 1.92 | - | 0.15 |
|          | uk-2005| 3.9 $\cdot 10^7$ | 5.36 $\cdot 10^8$ | - | - | - | 3.0 |

Table 2.3: Results obtained for subgraph $M_{12}$ in the graphs extracted from crawls of Wikipedia and of Web domains.

of nodes and number of edges, the exact number of occurrences $N$ of the subgraph taken into consideration, and the time (in seconds) that was needed to complete the exact counting.

In both tables, the three rightmost columns report the quality of the approximations that we obtained experimenting with three different values for the size of the sample set: $10^K$, $100^K$ and $1M$. For each graph and for each sample size, Table 2.2 and Table 2.3 report: (i) the quality $Qlt(\%)$ of the estimation of $N$ returned by our approximation algorithm, expressed in terms of percentage deviation from the exact count (a positive value indicates an overestimation and a negative value indicates an underestimation); (ii) the running time in seconds.

For the two largest datasets, uk-2002 and uk-2005, which have 200 and 940 million edges respectively, we do not indicate the quality of the approximation since we are not able to compute the exact value.

It can be observed from Table 2.2 and Table 2.3 that, as expected, the larger the set of samples collected by our algorithm, the more the quality of results improves. The method obtains an approximation as good as 5% even when using only $10^K$ samples.

When $100^K$ samples are used, we get an error no larger than 10%; this means that our subgraph-counting algorithm is able to compute a very good estimation of the number of occurrences of a small subgraph even when building a sample set of such limited size.

Also, it is worth to observe that, with this size of the sample set, the running time of our algorithm is always one order smaller than that required to complete the exact computation. The difference becomes more and more significant as the graph size increases.

When $1M$ samples are used, the algorithm returns excellent estimations of the number of occurrences of the subgraph considered: in the examples reported in the tables, the average percentage deviation from the exact count.
2.5. Experimental results

is always below 5%, and in many cases it is below 2%.

We have verified that the quality of approximation is similar for many other subgraphs and for different families of input networks. The very good performance obtained by our algorithm motivates us in using it for the analysis of graphs of huge size, for which the exact counting cannot be executed using trivial methods. This is the case of the two crawls of the .uk domain, which are the two largest instances in our data set: for these networks, the time required by our algorithm for estimating the number of occurrences of a given subgraph is at most 5 or 6 minutes.

**Undirected 4-node subgraphs:** For each undirected subgraph of 4 nodes we implement two of the three sampling strategies $S_{4u}^1$, $S_{4u}^2$, $S_{4u}^3$, presented in Section 2.4. Strategy $S_{4u}^1$ is used for all subgraphs. We then implement $S_{4u}^3$ as alternative strategy for the subgraphs that contain $S_{4u}^3$, while we use $S_{4u}^2$ for the rest of subgraphs.

| Graph  | $\frac{\left|S_{4u}^1\right|}{\left|S_{4u}^3\right|}$ | $S_{4u}^1$ | $S_{4u}^3$ | $S_{4u}^2$ | $S_{4u}^3$ |
|--------|-------------------------------------------------|-------------|-------------|-------------|-------------|
| WikiCS | 62.052                                          | -2.27       | -1.59       | -2.41       | -1.15       | 5.06        | -2.59       |
| WikiDE | 216.649                                         | 4.02        | 3.18        | 37.62       | 4.78        | 25.13       | 2.17        |
| WikiEN | 35.802                                          | 43.26       | 14.2        | 44.81       | -8.59       | -           | 7.01        |
| WikiES | 78.202                                          | -4.65       | 2.8         | -2.97       | 0.7         | -24.5       | 6.08        |
| WikiIT | 69.424                                          | -4.43       | 1.91        | 5.60        | 1.25        | 25.37       | -1.29       |
| WikiPT | 20.447                                          | -16.75      | 7.46        | -8.50       | 2.02        | -10.93      | 3.81        |
| AS-19980402 | 19.447                                      | -22.49      | 2.47        | 25.38       | -8.03       | 27.15       | 9.43        |
| AS-19980703 | 19.078                                      | -31.06      | 0.89        | 22.16       | 0.12        | 24.39       | -1.52       |
| AS-19981002 | 20.315                                      | -24.67      | 1.21        | -25.82      | -0.45       | -           | -7.60       |
| AS-19990114 | 18.873                                      | -35.01      | 0.29        | -24.35      | -3.97       | 33.10       | 8.30        |
| AS-19990402 | 19.161                                      | -26.63      | -0.41       | -10.01      | 0.71        | 9.16        | -1.30       |
| AS-19990702 | 19.903                                      | -31.50      | -0.65       | -35.01      | -1.45       | -21.45      | -3.39       |

Table 2.4: Quality of the estimations obtained on 6 Wikipedia graphs and 6 AS graphs by sampling $S_{4u}^1$ or $S_{4u}^3$.

None of the above sampling strategies can be a-priori considered as the best one for all cases. The reason is that, according to our analysis, the smaller the size of the sample space, the smaller is the variance of the estimation, the better the strategy, since we need fewer samples to obtain a good approximation.
Thus we test the quality of approximation obtained from the three strategies by comparing the sizes of the corresponding sample spaces.

| Graph  | \(|\{S^4_{1}\}\)| | Subgraph | Subgraph |   |
|-------|-----------------|--------|--------|---|
| WikiCS | 2.330           | 45.29  | 23.90  | 49.01  | 26.76 |
| WikiDE | 4.640           | 32.02  | 20.01  | 44.20  | 26.18 |
| WikiEN | 37.224          | 46.23  | -10.2  | 19.09  | -1.95 |
| WikiES | 4.190           | 31.72  | 23.76  | 41.55  | 26.89 |
| WikiIT | 3.063           | 35.52  | -59.10 | 24.14  | 15.35 |
| WikiPT | 6.553           | 40.91  | 28.20  | 23.73  | -16.75|
| AS-19980402 | 14.549     | 52.96  | 32.70  | 40.71  | 21.32 |
| AS-19980703 | 15.472     | 50.45  | -38.07 | 30.92  | 20.41 |
| AS-19981002 | 15.374     | 45.80  | -33.93 | 22.83  | 13.65 |
| AS-19990114 | 17.472     | 35.34  | -30.61 | 43.06  | 22.80 |
| AS-19990402 | 17.721     | 45.54  | -35.60 | 33.91  | 23.45 |
| AS-19990702 | 17.918     | 36.10  | -29.47 | 20.16  | 13.63 |

Table 2.5: Quality of the estimations obtained on 6 Wikipedia graphs and 6 AS graphs by sampling \(S^4_{1}\) or \(S^4_{2}\).

Tables 2.4 and 2.5 report the accuracy of different sampling strategies (in terms of percentage deviation from the exact count) for 6 Wikipedia graphs and 6 AS graphs. In Table 2.4 we consider the subgraphs that contain \(S^4_{3}\) as a substructure, while in Table 2.5 we consider the remaining subgraphs. For both tables, the second column provides the ratio between the sizes of sample spaces that are computed by

\[ |\{S^4_{1}\}| = P \cdot (V - 3), \quad |\{S^4_{2}\}| = E^2, \quad \text{and} \quad |\{S^4_{3}\}| = k_{13}, \]

where \(P\) is the number of paths of length two, \(V\) is the number of nodes, \(E\) is the number of edges and \(k_{13}\) is the number of cliques \(K_{13}\).
2.6 CLUSTERING NETWORKS

<table>
<thead>
<tr>
<th>Wikipedia Graph</th>
<th>Nodes</th>
<th>Edges</th>
<th>AS Graph</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>WikiCS</td>
<td>5 561</td>
<td>85 398</td>
<td>AS-19980402</td>
<td>3 522</td>
<td>12 648</td>
</tr>
<tr>
<td>WikiDE</td>
<td>116 251</td>
<td>3 815 782</td>
<td>AS-19980703</td>
<td>3 797</td>
<td>13 872</td>
</tr>
<tr>
<td>WikiEN</td>
<td>339 833</td>
<td>9 622 786</td>
<td>AS-19981002</td>
<td>4 180</td>
<td>15 536</td>
</tr>
<tr>
<td>WikiES</td>
<td>27 262</td>
<td>492 632</td>
<td>AS-19990114</td>
<td>4 517</td>
<td>16 752</td>
</tr>
<tr>
<td>WikiIT</td>
<td>13 132</td>
<td>268 670</td>
<td>AS-19990402</td>
<td>4 885</td>
<td>18 552</td>
</tr>
<tr>
<td>WikiPT</td>
<td>8 645</td>
<td>84 166</td>
<td>AS-19990702</td>
<td>5 357</td>
<td>20 656</td>
</tr>
</tbody>
</table>

Table 2.6: Graphs used for comparing sampling strategies.

We notice that $S^4_u$ outperforms $S^4_1$, achieving a very good approximation with error smaller than 5% in most of cases. We also notice that $S^2_u$ behaves better than $S^4_1$, although the results are not as good as in the previous case: the errors are around 20 – 30% on average. This result is somehow expected, since we are sampling 2 random edges instead of a more structured sub-pattern like a $K_{13}$ clique.

In both cases the experimental results match the theoretical analysis: for all graphs the winning strategy is the one that uses the smallest sample space.

**Directed 4-node subgraphs:** The algorithms for counting directed subgraphs of 4 nodes exhibit similar behavior to that of the algorithms for undirected subgraphs.

### 2.6 Clustering networks

Inspired from previous work [166, 165], we use the distribution of subgraph counts for characterizing different types of networks. Our working hypothesis is that subgraphs can be interpreted as components that play a fundamental role for the functions of the network. Thus, we decide to use clustering to study the significance of these patterns. The objective is to investigate whether features based on subgraphs can be used to partition the input networks into meaningful families (e.g., web graphs, food-chain networks, protein-interaction networks, etc.) and if they outperform simpler features based on classical measures used in literature to characterize complex networks [90].

Our approach is to represent graphs by vectors, where each coordinate cor-
responds to the number of occurrences of one particular subgraph. Let $G$ be a graph and let $t_n$ be the number of distinct subgraphs of $n$ nodes. Let $|M_j|$ be the number of instances of the $j$-th subgraph with $j = 0, \ldots, t_n$. We then represent $G$ by the vector $m(G) = \langle m_1, \ldots, m_{t_n} \rangle$, where the coordinate $m_j$ is the normalized number of occurrences of subgraph $M_j$. Since we compare vectors by the Euclidean distance we normalize the subgraph-occurrence vectors by their Euclidean norm.

Then, given a set of graphs from different families we cluster the graphs by clustering the vectors by which the graphs are represented. For the clustering task we use Weka, a standard software that provides a wide collection of machine-learning algorithms and data-analysis tools. We use two clustering algorithms, the Expectation-maximization (EM) and the $k$-means algorithms [125], varying the number of clusters from 4 to 8.

We compare how well the computed clusters match with the original classes, adopting the classes to cluster evaluation method implemented in Weka. We label each instance with the type of network it belongs to. During the clustering, this label is ignored. In a second phase, the majority class in each cluster is determined.

**Undirected graphs.** In the undirected case the best result is achieved with the EM method for $k = 7$ clusters: more than 75% of the instances are correctly classified, as shown in the matching matrix of Table 2.7. During the clustering phase, 6 out of 7 classes are correctly identified: clusters 0, 1, 2, 3, 5, 6 match the classes AS graph, road-graph, collaboration, protein, wikipedia, and synthetic, respectively.

Cluster 4 does not correspond to any class; it contains only 2 wikipedia graph instances that correspond to two of the smallest wikigraphs, the Italian and the Czech subsets. This could be a further sign of transient regime (growth) in these two graphs (similar to the observations made in [67]), which might explain why they are not clustered together with more mature instances.

The 13% of the wrongly classified instances are synthetic graphs that are grouped together with the AS graphs. A possible interpretation of this fact is that AS graphs are collections reconstructed by merging information provided by BGP routing tables. Since the graphs can be considered the result of the automatic routing determined by BGP strategies, one can argue that AS graphs share some characteristics with graphs built using some predefined methodology as in the case of synthetic graphs.

**Directed graphs** For the directed graphs we compare the following 5 groups of features:

1. **Standard topological properties:** We consider 16 different measures for every node, including indegree, outdegree, average indegree of succes-
2.6. Clustering networks

<table>
<thead>
<tr>
<th>assigned to</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS graph</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>collaboration</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>protein</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>road-graph</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>wikipedia</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>synthetic</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>Webgraph</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.7: Matching Matrix for the clustering using EM method with k=7 clusters in the undirected case.

sors, average outdegree of predecessors, assortativity, edge reciprocity, PageRank, and local number of triangles.

These are the most common measures used to characterize complex networks (see [90]). We then consider the well-known PageRank algorithm [177] and two modifications, which respectively are Sigma PageRank, i.e., PageRank scores computed introducing a standard deviation of the PageRank scores of predecessors of each node, and Truncated PageRank, i.e., PageRank scores computed without considering the mass-rank contribution of nodes till distance \( d \in \{1, 2, 3, 4\} \) (for further details see [38]). Following [38], we call \( x \) a supporter of \( y \) a distance \( d \) iif there is a path of length \( d \) connecting \( x \) and \( y \). We count the number of supporters at distance \( d \in \{1, 2, 3, 4\} \) for each node.

In order to assign the considered microscopic measures to each graph, we compute the mean, the variance, the median, the 10-percentile and 90-percentile of them, for a total number of 81 features.

2. Subgraphs with 3 nodes, for a total number of 13 features.

3. Subgraphs with 4 nodes containing a \( K_{12} \) clique, for a total number of 190 features.

4. All the subgraphs of size 3 and 4, for a total number of 203 features.

5. All the properties listed above, for a total number of 81 + 203 = 284 features.

For all the listed cases, we run the \( k \)-means algorithm imposing \( k = 7 \) clusters.

The matching matrices for the first 4 cases are shown in Table 2.8. In all the
Table 2.8: Matching Matrix for the clustering using $k$-means method with $k=7$ clusters using (1) classical topological properties; (2) only 3 nodes subgraphs; (3) only 4 nodes subgraphs; (4) 3 and 4 nodes subgraphs.

cases at least 5 out of 7 classes are correctly identified. We number from 0 to 6 the clusters that match the classes cellular, food-web, word, citation, wikipedia, Webgraph and synthetic and we use the numbers greater than 7 for the clusters matching no class.

In the first case, i.e., using standard topological properties, the $k$-means algorithm is able to correctly classify the $74\%$ of the instances. Webgraphs and citation graphs are not recognized at all: the instances of these two classes are indeed assigned to the wikipedia cluster. This observation confirms some previous findings concerning the similarity between Webgraphs and wikigraphs [67]. The $16\%$ of error is due to a misclassification of synthetic graphs.

In the second case, i.e., using only subgraphs of size 3, $77.78\%$ of the instances are correctly classified. Once again, Webgraphs are not recognized but there is a considerable improvement for what concerns synthetic graphs, with the $97\%$ of the instances clustered together. An error is introduced with respect to the previous case for the cellular networks, with $30\%$ of the instances wrongly classified.

In the third case, i.e., using only subgraphs of size 4, we correctly classify $84.26\%$ of the instances. Almost all the synthetic instances are clustered together, that is, the clustering algorithm separates synthetic and real networks.

In the fourth case, i.e., using all the subgraphs, $90.74\%$ of the instances are correctly classified. Cluster 7 does not correspond to any class and it contains only 2 food-web instances. The citation class is not identified and its 2 instances are assigned to the wikipedia class. The $4\%$ of the misclassified instances are Webgraphs. It is interesting to note that the two crawls of the .uk domain (performed in 2002 and 2005 respectively) are assigned to the synthetic class, while the two smallest Webgraphs, extracted from a crawl of the CNR Italian domain and a crawl of the .eu domain, are considered in the
2.6. Clustering networks

It is worth observing that using all the 284 features together (classical + subgraphs) does not improve the result achieved using only the subgraphs-counting methodology.

The results show that subgraphs of small size outperform standard topological features and play a fundamental role for the characterization of complex networks.

Subgraph significance. To further understand the significance of each subgraph with respect to each cluster, we perform a $t$-test for each pair of clusters, and for each given subgraph. As in the EM algorithm, we assume that the normalized number of occurrences of each subgraph within a cluster follows a normal distribution. Let $\bar{x}_{i,m}$ and $s_{i,m}$ be the mean and the standard deviation of subgraph $m$ within the cluster $i$. Then the $t$-test statistic for two clusters $i$ and $j$, with respect to subgraph $m$ is given by

$$
t_{(i,j),m} = \frac{\bar{x}_{i,m} - \bar{x}_{j,m}}{s_{\bar{x}_{i,m} - \bar{x}_{j,m}}}
$$

where

$$s_{\bar{x}_{i,m} - \bar{x}_{j,m}} = \sqrt{\frac{(n_i - 1)s^2_{i,m} + (n_j - 1)s^2_{j,m}}{n_i + n_j - 2} \left( \frac{1}{n_i} - \frac{1}{n_j} \right)}.
$$

For each cluster $i$, we compute the minimum $t$ distance $T_{i,m} = \min_j t_{(i,j),m}$ from every other cluster $j$ and we choose as the best subgraph to describe the cluster $i$, the one with the maximum $T_{i,m}$ value. The results for the undirected graphs are given in Table 2.6. The clusters that correspond to the AS graphs, the wikipedia graphs, and the synthetic graphs are best characterized by the same pattern, a dense subgraph of four nodes—a clique minus one edge. The collaboration graphs are characterized by a star of size 4, perhaps a demonstration of the effect of having a few prolific authors.

<table>
<thead>
<tr>
<th>cluster</th>
<th>best subgraph</th>
<th>cluster</th>
<th>best subgraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 : asgraph</td>
<td>△</td>
<td>4 : no class</td>
<td>△</td>
</tr>
<tr>
<td>1 : road-graph</td>
<td>△</td>
<td>5 : wikipedia</td>
<td>△</td>
</tr>
<tr>
<td>2 : collaboration</td>
<td>△</td>
<td>6 : synthetic</td>
<td>△</td>
</tr>
<tr>
<td>3 : protein</td>
<td>△</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.9: The subgraphs that best characterize each cluster in the undirected case.
2.6.1 Swap randomization

We assess the statistical significance of our method by comparing the number of subgraphs found in a network with the number of subgraphs observed in random networks with the same degree sequence. This method is called swap randomization [117] since a random network with a given degree sequence is generated through a sequence of swaps between pairs of edges of the graph. The swap operations maintain the degree of every vertex.

We generate 5 randomized networks for each of the 43 real directed cellular graphs. For each graph we perform $1000 \cdot \#\{\text{edges}\}$ swaps, a number much larger than what has been empirically considered sufficient to obtain a random graph [117]. We compute the exact number of occurrences of the directed minors of 3 and 4 nodes in the 43 real networks and in the 258 randomized ones. The application of the EM algorithm with two clusters correctly separates all but one of the randomized networks from all the real ones. If we let EM choose the number of clusters, the randomized networks are split into two clusters of 172 and 43 elements each, while the real networks are all classified in a third cluster.

Overall, since the real networks can be separated so easily from the randomized ones, we conclude that 3- and 4-node subgraphs contain valuable information about the structure of the real networks.

2.6.2 Feature selection

We have shown that our algorithm based on subgraph counting achieves very good precision in clustering networks that belong to different families. However, the number of features used by the method is quite large: the set of properties computed for every graph includes the occurrences of all the directed connected subgraphs of three and four nodes, which means 203 features altogether.

In practice, we noticed during experimental evaluation that the graphs analyzed contain no occurrences of many subgraphs, especially the most dense and complicated patterns of four nodes. Hence, many of the properties taken into consideration do not carry significant information about the structure of the networks considered.

For this reason, we chose to apply feature selection techniques to extract from our feature set the subset of the properties that have a concrete usefulness for network characterization.

Feature selection is commonly used in data mining to remove from a data set those features that are irrelevant with respect to the task to be performed. This process can be extremely useful in reducing the dimensionality of the data that must be handled by a clusterer or a classifier, as well as in reducing
execution time and improving accuracy: relevant features can introduce noise by obscuring the relevant ones.

Weka provides an implementation of a large collection of methods for feature selection and ranking. We experiment with two techniques: the former is Correlation based Feature Selection [123], while the latter is Chi-Squared Feature evaluation [156], which we combine with a greedy hill-climbing search to identify the appropriate set of attributes to be ranked.

The first method selects a set of possible features for the optimal subset. When applied to our set of networks, the CFS algorithm selects a set of 14 features, which is shown in Table 2.6.2.

We perform a new clustering experiment, in which we use only these features for every graph in the data set. We run the K-Means algorithm with \( k = 7 \) clusters, which is the case that provided the best results when all the features available were taken into consideration.

![Table 2.10: Feature selection: CFS, K-Means with \( k = 7 \).](image)

The algorithm is able to correctly classify 75% of the instances (see the matching matrix in Table 2.11).

<table>
<thead>
<tr>
<th>assigned to</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>cellular</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>webgraph</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>synthetic</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>word</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>food-web</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>citation</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>wikipedia</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.11: Feature selection: CFS, K-Means with \( k = 7 \) - Matching Matrix.

This precision is comparable to the one obtained with the standard topological properties, but is significantly lower that the one achieved with the
whole set of features based on subgraph counting (> 90%).

Next, we apply the Chi-Squared method. This algorithm does not return a subset of features. Instead, it computes a ranking of the features. We select the features in the top 15 positions of such a ranking.

![Subgraph Counting](image)

Table 2.12: Feature selection: Chi-Squared, K-Means with \( k = 7 \).

This time, the K-Means algorithm with \( k = 7 \) clusters achieves very good performance: 88% of the instances are correctly classified (the matching matrix is shown in Table 1.13). This results is comparable to the one obtained with the whole set of features.

<table>
<thead>
<tr>
<th>assigned to</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>cellular</td>
<td>0</td>
<td>43</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>webgraph</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>synthetic</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>word</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>food-web</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>citation</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>wikipedia</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.13: Chi-squared, K-Means with \( k = 7 \) clusters: Matching Matrix.

### 2.7 Network classification

The very good precision achieved by our subgraph-counting method in clustering networks suggests that frequent patterns capture important information about the structural properties of networks. This motivates us in the attempt of using the same methodology for classifying networks. Hence, we run a second set of experiments with the aim of verifying whether the frequencies of occurrences of all the subgraphs of three and four nodes can be used as...
significant features to perform a real classification task.

**Data sets**

For the classification experiment, we use a collection made of 640 graphs, which are divided in two groups. The first group is formed by 320 graphs representing single web domains. These graphs were extracted from the UK-2005 web Graph, which was obtained from a large crawl of the .uk domain and was also used in the clustering experiments.

The second class comprises 320 synthetic graphs that we created using a graph generator following the well known *copying model* proposed by Kumar et al. [149]. The synthetic graphs have sizes similar to those of the real web graphs.

**Classification.** As we explained in the previous section, we represent graphs by vectors, where each coordinate corresponds to the number of occurrences of one particular subgraph. The vectors are normalized by their Euclidean norm. We use the Weka software to perform the classification task. We experiment with the J48 algorithm, which is the Weka implementation of C4.5 decision trees. We do not use pruning and we let Weka generate as many rules as possible as long as there are at least 2 graphs per leaf (this is the M parameter in the weka.classifiers.trees.J48 implementation).

<table>
<thead>
<tr>
<th>Class</th>
<th># Instances</th>
<th># Correctly classified</th>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>web</td>
<td>112</td>
<td>110</td>
<td>0.982</td>
<td>0</td>
<td>1</td>
<td>0.982</td>
<td>0.991</td>
</tr>
<tr>
<td>synthetic</td>
<td>106</td>
<td>106</td>
<td>1</td>
<td>0.018</td>
<td>0.981</td>
<td>1</td>
<td>0.991</td>
</tr>
</tbody>
</table>

Table 2.14: Classification with test split: Summary.

<table>
<thead>
<tr>
<th>Class</th>
<th># Instances</th>
<th># Correctly classified</th>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>web</td>
<td>320</td>
<td>315</td>
<td>0.984</td>
<td>0.003</td>
<td>0.997</td>
<td>0.984</td>
<td>0.991</td>
</tr>
<tr>
<td>synthetic</td>
<td>320</td>
<td>319</td>
<td>0.997</td>
<td>0.016</td>
<td>0.985</td>
<td>0.997</td>
<td>0.991</td>
</tr>
</tbody>
</table>

Table 2.15: Classification with bagging on cross validation: Summary.

**Test options.** We use different test options to evaluate the output of the classifier: *percentage split*, *cross validation* and *bagging* with cross validation. In the first case, we use 66% of the data is used as training set, and the remaining 34% for testing.

We then apply 10-fold cross validation. This methods partitions the data into 10 sub-samples. A single sub-sample is retained as validation data for testing the model, while the remaining subsamples are used as training data.
Table 2.16: Classification with feature selection and test split: Summary.

<table>
<thead>
<tr>
<th>Class</th>
<th># Instances</th>
<th># Correctly classified</th>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>web</td>
<td>112</td>
<td>111</td>
<td>0.991</td>
<td>0</td>
<td>1</td>
<td>0.991</td>
<td>0.996</td>
</tr>
<tr>
<td>synthetic</td>
<td>106</td>
<td>106</td>
<td>1</td>
<td>0.009</td>
<td>0.991</td>
<td>1</td>
<td>0.995</td>
</tr>
</tbody>
</table>

Table 2.17: Classification with feature selection and cross validation: Summary.

<table>
<thead>
<tr>
<th>Class</th>
<th># Instances</th>
<th># Correctly classified</th>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>web</td>
<td>320</td>
<td>316</td>
<td>0.988</td>
<td>0.006</td>
<td>0.994</td>
<td>0.988</td>
<td>0.991</td>
</tr>
<tr>
<td>synthetic</td>
<td>320</td>
<td>318</td>
<td>0.994</td>
<td>0.013</td>
<td>0.988</td>
<td>0.994</td>
<td>0.991</td>
</tr>
</tbody>
</table>

The cross-validation process is repeated 10 times, and each of the 10 sub-samples is used exactly once as the validation data. The 10 results from the folds are then averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once.

Additionally, we use **bagging**, a technique that creates many classifiers (in our case, 10), and then uses majority voting for deciding the class which an element belongs to. The classifiers that use bagging perform in general better than the individual classifiers they are composed of.

For what concerns the error metrics, we use the TP Rate (True Positive Rate) and the FP Rate (False Positive Rate). We use Precision, Recall and the F-measure, which represents the harmonic mean of Precision and Recall.

**Classification results** Tables 2.14 and 2.15 present the outcome of the classifier for the two cases in which percentage split and bagging with cross validation are used as test options.

Observe that our features based on subgraph counts allow to achieve an excellent precision in separating the web graphs from the synthetic networks. This result makes us claim that the predefined rules which the Copying model relies on can not give a conclusive explanation of how the Web graph of single domains is formed.

**Feature selection** We already observed in the clustering experiments that our feature set is large, but many of the features are not conveying useful information, as they are related to very complicated patterns that hardly ever appear in the networks. We showed that the feature set can be significantly reduced to 15 features by applying a feature-selection algorithm, while preserving the excellent performance of the method.

We now adopt the same approach in our classification experiment: We use
the Chi-Square method to rank the features and we retain the subset formed by the 15 top significant properties. We then run the classifier again: Once again, we obtain excellent results (2.16,2.17). We are able to obtain the same precision achieved with the whole set of features.

In the case of clustering, we are able to correctly cluster more than 91% of the instances. The precision we obtain in the classification experiment is equal to 99%.

We then conclude that different families of networks are well described by a limited set of small substructures that carry meaningful information about their generative processes, as well as their structure and functionalities.
Chapter 3

Fully Decentralized
Computation of Aggregates
over Data Streams

In several emerging applications, data is collected in massive streams at several distributed points of observation. A basic and challenging task is to allow every node to monitor a neighbourhood of interest by issuing continuous aggregate queries on the streams observed in its vicinity. This class of algorithms is fully decentralized and diffusive in nature: collecting all data at few central nodes of the network is unfeasible in networks of low capability devices or in the presence of massive data sets. The main difficulty in designing diffusive algorithms is to cope with duplicate detections. These arise both from the observation of the same event at several nodes of the network and/or receipt of the same aggregated information along multiple paths of diffusion.

In this work, we consider fully decentralized algorithms that answer locally continuous aggregate queries on the number of distinct events, total number of events and the second frequency moment in the scenario outlined above. The proposed algorithms use in the worst case or on realistic distributions sublinear space at every node. We also propose strategies that minimize the communication needed to update the aggregates when new events are observed. We finally present experimental analysis providing evidence for the efficiency and accuracy of our algorithms on realistic simulated scenarios.
3. Distributed Streaming

3.1 Introduction

A variety of emerging network applications are based on spreading a large number of network devices over a broad area. Examples are the continuous and distributed monitoring of IP traffic flows, in which data is collected at multiple points of observation and real-time analysis is performed on aggregated streams, or the use of sensing devices, connected in a wireless sensor network, for environmental monitoring. Sensing devices observe large amounts of events in their surroundings. These events are recorded and processed by nodes in the form of data streams. In sensor databases we are required to answer aggregate queries over the streams of data, while in distributed monitoring applications we are interested in monitoring the events observed by the sensing devices, for instance the movement of objects or measurements of environment parameters.

A number of constraints is imposed in these applications by the limited resources available at the sensing devices:

1. Communication is the most power-consuming operation. Transmission of data to neighbor nodes must be carefully optimized in order to ensure a longer life to battery-operated devices. A direct consequence is that large streams of raw data cannot be transmitted between nodes.

2. Sensing devices are limited in computational power. The data received from the nodes can often only be processed on the fly in a streaming fashion.

3. Sensing devices have limited storage capacity. Data can be stored, even locally, only in aggregate form.

It is a basic and challenging task to provision the network with primitives that allow every node to monitor its surrounding by issuing continuous aggregate queries on the streams of events observed by all nodes reachable within a few hops.

A number of specific issues need to be addressed when querying and monitoring distributed streams of data:

Continuous queries. Differently from traditional databases, these applications must answer long-running queries, notifying the application whenever a new answer is found. This occurs for instance in applications that raise an alert whenever a specific event happens, or some measurement of interest exceeds a given threshold.

Distributed streams. One first difficulty to handle when processing aggregate queries on distributed streams is to discriminate events that have been observed at multiple nodes of the network. For instance, this problem arises naturally for terrain-monitoring applications, where sensing devices need to monitor objects that move across an area, or in point-to-point communication when packets are observed along all the nodes of a path.
One second difficulty to handle is that nodes are inherently unreliable: they switch on and off in order to optimize power consumption, they are prone to communication faults and to power outage. In several important cases it is therefore impossible to aggregate information using hierarchical structures, e.g. trees, where faults at single nodes of the network may disrupt the whole structure. It is also impossible to rely on very few nodes for collecting the aggregates on raw data produced in the network.

Local queries. In sensor databases applications, aggregate queries may be issued by any node of the network and the answer restricted to the observations made in a neighborhood of interest. Power consumption is optimized if the nodes can decide to operate only when their observations differ substantially from other nodes in their vicinity.

Decentralized algorithms addressing the issues above are inherently diffusive in nature. The problem of detecting duplicates in aggregate information is now magnified, since a node may receive the same data as part of aggregates received through multiple paths of propagation. In this work we show how a set of continuous aggregate queries can be processed from fully decentralized algorithms on distributed streams of data.

3.1.1 Our Contribution

The design of algorithms for the computation of aggregates within suitable neighbourhoods of all nodes of a network entails two fundamental aspects:

1. The algorithm used to summarize the scenario of interest. For any node \( u \) it requires a compact summary, or sketch in the sequel, of the stream of events observed from \( u \) that is composable and duplicate insensitive [82, 84]: it is possible to merge the sketches of a set of nodes \( U \) to form a sketch of the union of the streams of events observed by all nodes of \( U \). In the sequel, we consider the natural case in which \( U \) is the set of neighbors within \( l \) hops from a node \( u \), for some \( l \geq 0 \).

2. How and when information is propagated. This in turn requires i) addressing communication costs and ii) estimating the (further) error introduced by delay in communication.

Given the massive data sets we are interested in, we impose constraints on the memory requirements for the computation and the overall amount of data exchanged to perform the task. Namely, i) the amount of memory used at every node of the network should be polylogarithmic in the number of nodes of the network and in the size of the streams. ii) The communication complexity measured in terms of the overall number of messages exchanged among nodes to perform the task. Below is a list of the main contributions of our work:
1. We consider a very general distributed streaming scenario and we describe compact distributed data structures and a communication scheme to keep track of statistic aggregates over distributed data streams. We are able to answer at any node of the network continuous queries on the number of distinct events, total number of events and the second frequency moment (size of Self-join) over the streams observed within a suitably defined neighborhood of each node. These algorithms require only polylogarithmic storage space at each node of the network for number of distinct events, and the total number of events. Observe that our methods apply to a very general, fully-decentralized setting, whereas existing works consider either a centralized scenario with a single aggregating point or simple models for decentralized computation where a direct communication is assumed to be available between any pair of nodes in the network.

2. For the second frequency moment, we provide an important contribution by showing a polylogarithmic bound on the storage size at each node when the distribution of events follows a Zipf’s law. Observe that in [85] no polylogarithmic upper bound is given on the amount of space needed to estimate the second frequency moment on data streams with duplicates, whereas a $\Omega(\sqrt{n})$ lower bound is provided in the worst case for algorithms based on uniform sampling.

3. We show optimized implementations and tests on large scale real and synthetic datasets and realistic network topologies. Experimental results show that the fully decentralized strategies we consider achieve in practice very good bounds on storage, computation and communication costs while providing a good approximation of the statistics of interest.

4. We characterize the trade-off between the worst case communication cost and accuracy for the sketch schemes we consider. Moreover, we experimentally evaluate the performance of a conservative strategy to trade off communication costs and accuracy, proposed in [86] for a special case of the scenario considered in this work.

Organization of the work. In Section 3.2 we introduce the model and notation we use in the rest of the work. In Sections 3.3 and 4 we outline our overall approach and discuss sketch schemes for fully decentralized estimation of $F_0$, $F_1$ and $F_2$ in the presence of duplicates. For $F_2$, we also present the analysis for the important case of skewed data. In Section 3.5 we address the fully decentralized implementation of the sketches we consider and show how to optimize the amount of communication exchanged among nodes of the network. In Section 3.6 we show the experimental evaluation of the algorithms. Finally, Section ?? offers our concluding remarks.
3.1.2 Related work

An excellent survey of distributed streaming techniques has recently been provided in [83]. They distinguish between one-shot queries and continuous queries. Tree-based aggregation for approximately answering of one shot holistic queries use composable data synopses, as for instance CM sketches [84] for point query, range queries, quantiles, AMS sketches [11, 12] for frequency moments, FM [108] sketches for counting distinct items. Greenwald and Khanna [120] also propose decomposable synopses for computing quantiles that can be adapted to tree-based aggregation.

Alternative to tree-based aggregation is broadcasting to all neighbor vertices till the summaries reach the aggregation node of the network. These proposals need to exploit synopses that are order and duplicate insensitive as those proposed from Considine et al. [82], FM sketches [108], CM sketches [84], and duplicate-insensitive aggregation schemes [85] that propose duplicate insensitive estimation of the second frequency moment for a single aggregating point. However, even in the restricted case addressed in [85], no polylogarithmic upper bound on the needed space is given.

Decentralized computation with every node in the graph taking part in the computation and receiving the computed value has been proposed in [137, 138] for computing max, sum, avg and distinct items. This model is similar in spirit to our proposal with two main differences: a. in our setting we cannot assume direct communication between any pair of nodes in the network; b. we are interested in holistic aggregates at the local neighbor of any node of the network.

Main issue addressed so far in processing continuous queries on distributed data streams is to reduce the communication overhead between remote nodes and the aggregating point by allowing some slack in the estimation. In [18] the problem of distributing the slack between the participating sites is addressed for Top-k computation on distributed streams. In [86] several strategies of communication between the coordinating site and the distributed participants are studied for duplicate-insensitive aggregates as FM sketches and distinct sampling [115].

3.2 Model and Preliminaries

Model. We assume an undirected graph $G = (V, A)$, representing entities connected over a communication network. Without loss of generality we assume all communication edges bidirectional. Every node in the network observes a stream of data over time. We denote by $S(u)$ the stream at node $u$. Each element of $S(u)$ is an (item, attribute) pair $(i, a)$, where $i$ is a value that we assume to be in $[n] = \{0, 1, \ldots, n-1\}$, and $a$ is an attribute from a discrete domain $\mathcal{A}$. 
Since all statistics we consider are order insensitive, $S(u)$ may be regarded as a multiset of pairs, each pair with a multiplicity equal to the number of times it appears in the stream. Considered any stream $S$, we also define the base set $B(S)$, i.e., the set of distinct pairs appearing in $S$. In the sequel, loosely speaking, we often use the expression event to mean the observation of an (item, attribute) pair at some node of the network. Given a subset $U \subset V$, we define $S(U)$ in the natural way as $\bigcup_{u \in U} S(u)$, where the multiplicity of a pair $(i, a)$ in $S(U)$ is the sum of its multiplicities in $S(u)$, $\forall u \in U$. For $l \geq 0$, $N_l(u)$ denotes the subset of $G$’s vertices within distance $l$ from $u$. In the sequel, we write $B(U)$ instead of $B(S(U))$ for the sake of coinisness.

Statistics. The general problem we are interested in is the following: Given $l \geq 0$, for every $u \in V$, compute some statistics over $B(N_l(u))$. Depending on the choice of the attribute, we’ll be able to estimate different aggregates of interest. Considered a stream $S$, we denote by $m_i(S)$ (or simply $m_i$ when no confusion arises) the number of distinct pairs $(i, a)$ in $S$: $m_i = |B(S)|$, for every $i \in [n]$. The $p$-th frequency moment of $S$ is $F_p = \sum_{i \in S} m_i^p$. When considering $S(u)$ for some vertex $u$ or $S(U)$ for some subset $U$ of vertices, abusing notation we write $m_i(u)$, $F_p(u)$, $m_i(U)$ and $F_p(U)$ for $m_i(S(u))$, $F_p(S(u))$, $m_i(S(U))$ and $F_p(S(U))$ respectively. In the sequel, we consider $F_0$, the number of distinct items, $F_1$, the total number of distinct pairs in the stream, and $F_2$, the second frequency moment over the set of distinct pairs.

Communication. We make the minimal assumption that in a round of communication a node can only broadcast a message to the set of its immediate neighbours. We make no assumptions as to the topological structure of the neighbourhood of a node or the way in which communication is performed. Our model has an immediate practical counterpart in wireless networks in which communication occurs over a broadcast channel, while in other cases a round of communication may actually involve multiple transmissions along point-to-point connections.

### 3.3 Overview of the approach

Each node in the network observes a local stream over time and keeps a compact summary or sketch thereof, which depends on the application of interest. Consider a streaming algorithm $A$. We denote by $Sk^A$ the function used by $A$ to compute sketches and by $Sk^A(S)$ the sketch computed by $A$ at the end of the observation of a stream $S$. In the setting we consider, every node $u$ locally maintains two sketches $Sk(S(u))$ and $Sk(S(N_l(u)))$ and performs the following actions:
3.4. DECENTRALIZED COMPUTATION OF $F_0$, $F_1$, AND $F_2$

i. when a pair $(i, a)$ is observed at $u$, the node updates $Sk(S(u))$ and $Sk(S(N_i(u)))$ in a way that depends on the statistics of interest and the algorithm used;

ii. if $u$ receives a sketch $Sk(S(v))$ originating from a node $v$ (i.e., an update of $Sk(S(v)))$, $u$ updates $Sk(S(N_i(u)))$ and forwards the update to its neighbours, as long as its distance from $v$ is less than $l$;

iii. $u$ continuously monitors the current estimate of the statistics of interest over $S(u)$. If this value differs from the most recently propagated value significantly, the current value of $Sk(S(u))$ is propagated to $u$’s immediate neighbours.

Step ii. is achieved using messages that contain a TTL (Time To Live) field, which is decremented at each node reached by the message, as is customary in gossip-based network protocols.

Clearly, not all sketches are suitable for our setting. Following previous work, we consider sketches that are composable and duplicate insensitive [169, 82, 121]. Considered an order insensitive statistics of interest, a sketching algorithm $A$ for its estimation is composable if, given two streams $S_1$ and $S_2$, $Sk^A(S_1 \cup S_2) = merge(Sk(S_1), Sk(S_2))$, where $merge(\cdot)$ is a suitable sketch aggregation function that depends on $A$ and the statistics of interest [82]. A sketching algorithm is duplicate insensitive if, considered any stream $S$, $Sk(S) = Sk(B(S))$ (where $B(S)$ is regarded as a multiset). In next Section, we address the problem of designing accurate, composable and duplicate insensitive sketches for the primitives of interest in this work.

3.4 Decentralized computation of $F_0$, $F_1$, and $F_2$

3.4.1 Composable, duplicate-insensitive sketches

We start the presentation of our algorithms with a brief overview of the most important characteristics of counting sketches, which we use as building blocks of our methods for computing aggregate statistics in a fully-decentralized setting. A counting sketch is a composable and duplicate insensitive counter of the number of distinct $(i, a)$ pairs appearing in a stream $S$. In the sequel, we consider two approaches: the first is the original approach of Flajolet and Martin, considered in [86] and modified in [12]. The second is a slightly different technique proposed in [30]. Since these are well established techniques, we only give an overview to make the work self-contained, referring the reader to [108, 12, 86, 30] for details. The use of composable and duplicate insensitive sketches has been considered previously for restricted distributed settings, especially for data aggregation at the sink of a sensor network [169, 82, 121].
3. DISTRIBUTED STREAMING

**FM sketches.** In the sequel, we use the phrase “FM sketch” to refer to any implementation of the original counting sketch of [108]. FM sketches [108] use a simple approach in which each sketch is a vector of \( m \) entries, each entry being a bitmap of length \( k = O(\log M) \), with \( M \) an upper bound on the size of the universe. In our setting, the universe is the set of possible \((i,a)\) pairs, so that \( M \leq n|A| \) (in practice, \( M = 2^k \), e.g., \( k = 64 \)). Considered the \( s \)-th bitmap of the sketch. Every pair \((\text{item, attribute}) (i,a)\) is hashed onto the bitmap bits using a (independently chosen) hash function \( H_s(\cdot) : [n] \times A \rightarrow \{0,\ldots, \log_2 M - 1\} \), such that the probability of hashing onto the \( h \)-th bit is \( 2^{-h} \). The bit under consideration is set to 1 if it was 0. After processing the stream, let \( r_s \) denote the position of the least significant bit that is still 0 in the \( s \)-th bitmap: \( r_s \) is a good estimator for \( \log_2 F_0 \), the logarithm of the number of distinct pairs observed. To improve accuracy, we consider \( m \sum_{s=1}^m r_s \) as an estimator of \( \log_2 F_0 \), where \( m = O \left( \frac{1}{\epsilon^2 \log \frac{1}{\delta}} \right) \).

**Bar-Yossef et al. [30].** In this approach, the sketching algorithm maps every pair \((i,a)\) to an integer using a pairwise independent hash function \( h(\cdot) \) and at any point in time it maintains the list of the \( L \) smallest distinct values observed so far, where \( L = \lceil 96/\epsilon^2 \rceil \), \( \epsilon \) being the required precision. If \( v \) is the \( L \)-th smallest distinct value maintained by the algorithm, \( LM/v \) is an estimator of \( F_0 \), where \( M = n^3 \) (see [30] for details). Precision can be increased by the standard trick of considering \( m \) independent and parallel copies of the algorithm and taking the median of the corresponding estimations, where again \( m = O \left( \frac{1}{\epsilon^2 \log \frac{1}{\delta}} \right) \). In the rest of the work, we will use the phrase “BY sketch” to refer to any implementation of this counting sketch.

**Composability.** Clearly, both sketches are composable and duplicate insensitive: Considered two streams \( S_1 \) and \( S_2 \) over the same universe and their FM sketches \( SkFM(S_1) \) and \( SkFM(S_2) \), \( Sk(S_1) \ OR \ Sk(S_2) \) is the sketch corresponding to \( S_1 \cup S_2 \). As for the BY sketches, every such sketch is an array of \( m \) lists, each maintained according to the algorithm described above. Merging of two such sketches is simply achieved as follows: for every \( s = 1,\ldots,m \), merge the \( s \)-th lists of the two sketches and keep the \( L \) smallest values of the merged list. Both approaches achieve similar bounds in terms of efficiency and precision, as stated by the following

**Theorem 2 ([86, 108, 30]).** Given a stream \( S \) of \((\text{item, attribute})\) pairs, it is possible to maintain an estimate \( \hat{C} \) of the number \( C \) of distinct pairs in \( S \) using \( O \left( \frac{1}{\epsilon^2 \log \frac{1}{\delta}} \right) \) memory words, such that:

\[
P \left[ |\hat{C} - C| > \epsilon C \right] \leq \delta.
\]
3.4. DECENTRALIZED COMPUTATION OF $F_0$, $F_1$, AND $F_2$

3.4.2 Maintaining $F_0$ and $F_1$

Any of the schemes outlined above can be used to maintain, for a stream $S$, an accurate and duplicate insensitive sketch for $F_0(S)$ or $F_1(S)$. For $F_0$, we want to count the number of distinct items observed in $S$, so we set $a = \text{null}$. As to $F_1$, this is the basic problem of counting the number of distinct pairs, therefore it can be maintained with the guarantees mentioned above.

3.4.3 Maintaining $F_2$ with duplicates

We now present our algorithm for computing $F_2$ in a fully-decentralized setting. We adapt the method proposed by Achlioptas [8] to maintaining $F_2$ with duplicates in a decentralized scenario. We show that the method achieves polylogarithmic space on skewed data.

Maintaining $F_2$ using random projections

The most effective approaches to efficiently maintain $F_2$ in a centralized setting are based on random projections of the frequency vectors over a space of smaller dimension. A similar approach, explicitly designed to maintain $F_2$ over a data stream, was proposed in [12] and extended in [106, 130]. Independently, a number of techniques have been proposed to achieve the more general goal of maintaining pairwise euclidean distances of a set of vectors in lower dimensional space, mostly extending or modifying a key result by Johnson and Lindenstrauss (see [92] for a relatively simple proof). A substantial simplification of the original scheme that we adopt in this work was proposed by Achlioptas [8].

Maintaining $F_2(S)$ in the absence of duplicates. In this case, $(i, a), (i, b) \in S$ implies $a \neq b$. Then, the approach described in the above paragraphs immediately applies to $\hat{m}(S)$ as done in [12]: assume $\hat{m}$ is the state of the frequency vector after the first $t$ elements in $S$ have been observed. At any time $t$, we maintain $\bar{m} = mR$ as follows: If the value of the $t + 1$-th item observed is $i$, $\bar{m}_j = \bar{m}_j + e_i \cdot R$ for $j = 1, \ldots, d$, where $e_i$ is the $n$-dimensional row vector whose $i$-th component is 1, all other components being 0. So, observation of $i$ determines the addition or subtraction of 1 in every component of $\bar{m}$.

Maintaining $F_2(S)$ in the presence of duplicates. When duplicates are present in $S$, we use the “tug-of-war” sketch considered in [11, 85]. In particular, we notice that $\bar{m}_j(S) = I_j(S) - D_j(S)$, where $I_j(S)$ (respectively, $D_j(S)$) counts the number of distinct $(i, a)$ pairs observed in $S$ that determine the addition of +1 (addition of −1) to $\bar{m}_j(S)$. Hence, for every $j = 1, \ldots, d$, we can estimate $I_j(S)$ and $D_j(S)$ using the techniques described in Section 3.3. More in detail, for every $j$, we maintain a counting sketch to estimate
$I_j(S)$ (respectively $D_j(S)$). The overall tug-of-war sketch obtained this way consists of $2d$ counting sketches and it is clearly duplicate insensitive and composable. In particular, to compose two tug-of-war sketches, the component counting sketches are pairwise composed in the obvious way. Following the above paragraph on maintaining the $2$-norm, if $\tilde{I}_j(S)$ (respectively, $\tilde{D}_j(S)$) denotes the estimation of $I_j(S)$ (respectively, $D_j(S)$), our estimator of $F_2(S)$ is $\frac{1}{2} \sum_{j=1}^{d} (I_j(S) - D_j(S))^2$.

**Maintaining $R$.** Maintaining $R$ explicitely requires $O(nd)$ space at every node, which is unfeasible. Furthermore, if we want to compute $F_2$ over the union stream of a subset of the vertices, it is necessary that all nodes use the same random projection. In practice, all nodes generate the entries of $R$ whenever needed using the same pseudorandom generator (and thus polylogarithmic space). To generate $R_{ij}$, $u$ computes the $j$-th value of the random generator with initial seed $i$. This way, all vertices generate the same value for $R_{ij}$.

1

**Analysis for skewed data**

The best algorithm to maintain $F_2$ with duplicates under general distributions requires space $O\left(\frac{1}{\epsilon^2} \sqrt{n} \log n\right)$ [85]. In the same work, the authors prove an $\Omega(\sqrt{n})$ lower bound for algorithms based on uniform sampling and it is an open question whether this bound indeed holds in general.

In the present work, we show that the algorithm described above provides an accurate estimation of $F_2$ using polylogarithmic space for skewed data, which is the case in most applications of interest. We assume that $m(S)$ is distributed according to a Zipf law with parameter $\alpha > 1$, i.e.: $m_i = M/i^\alpha$. The analysis for $\alpha \leq 1$ (showing decreasing accuracy as $\alpha$ decreases) proceeds along similar lines and will be given in the full version of the work. Under these assumptions we are able to state the following theorem:

**Theorem 3.** If $m(S)$ is distributed according to Zipf law with parameter $\alpha > 1$, it is possible to compute an estimation $\tilde{F}_2(S)$ such that:

$$P\left[|\tilde{F}_2(S) - F_2(S)| > 2\epsilon F_2(S)\right] \leq \delta,$$

by using an amount of $O\left(\left(\frac{1}{\epsilon^2(\alpha-1)^2} \log \frac{1}{\delta}\right)(\log \frac{1}{\epsilon} + \log \frac{1}{\delta})\right)$ memory words per node.

1It is clear that the matrix generated this way no longer satisfies the independence assumptions of [8], but choosing the pseudorandom generators independently for every column is in practice enough and is close to the requirements of [12].
3.4. DECENTRALIZED COMPUTATION OF $F_0$, $F_1$, AND $F_2$

Proof. We consider $F_2(S)$ under the assumption that $m(S)$ is distributed according to a Zipf law with parameter $\alpha$. We consider the case $\alpha > 1$ in the sequel. We drop $S$ in the rest of this subsection, and we assume without loss of generality that $m_1 \geq \cdots \geq m_n$:

$$m_i = \frac{M}{i^\alpha},$$

where $M$ is a positive, integer constant. It is clear that by this definition the components of $m$ will be fractional. Assuming integer components does not affect the result, but it makes proofs much more involved, so we do not consider this issue here. The proof of the following simple lemma is deferred to the full work.

Lemma 3.4.1. If $m$ follows a Zipf law with parameter $\alpha > 1$, for every $n \geq 2$:

$$F_2 \geq \frac{M^2}{4\alpha - 2}.$$

We next characterize the error achieved by the overall algorithm. Note that this error has two sources: i) the random projection; ii) the propagation algorithm.

In particular, recall that $\hat{m} = mR$ and $\hat{m}_j = I_j - D_j$. In fact, $I_j$ and $D_j$ are estimated at $u$ from the corresponding FM sketches, so that $u$ actually only computes two estimates $\tilde{I}_j$ and $\tilde{D}_j$ of $I_j$ and $D_j$. In particular, consider realizations of $I_j$, $D_j$, $\tilde{I}_j$ and $\tilde{D}_j$ and assume that $\tilde{I}_j = (1 + \epsilon_1(j))I_j$ and $\tilde{D}_j = (1 + \epsilon_2(j))D_j$. The estimation of $F_2$ using the projected vector would be $\frac{1}{d} \sum_{j=1}^{d} \tilde{m}_j^2$. In practice, $u$ computes a vector $\tilde{m}$ that is itself an estimation of $\hat{m}$. In particular, $\tilde{m}_j = \tilde{I}_j - \tilde{D}_j$. This implies that our actual estimation of $F_2$ is $\tilde{F}_2 = \frac{1}{d} \sum_{j=1}^{d} \tilde{m}_j^2$.

Lemma 3.4.2.

$$\tilde{F}_2 = \frac{1}{d} \sum_{j=1}^{d} (I_j - D_j)^2 + E,$$
where $|E| \leq \frac{3}{4} \sum_{j=1}^{d} \max\{|\epsilon_1(j)|, |\epsilon_2(j)|\}(I_j + D_j)^2$.

In the sequel, we show that i) $\frac{1}{d} \sum_{j=1}^{d} (I_j - D_j)^2$ (i.e., the squared 2-norm of the projected vector) is a close approximation of $F_2$ with high probability; ii) $|E|$ can be compensated by forcing the $\epsilon_1(j)$’s and $\epsilon_2(j)$’s to be small enough. The first claim is just a restatement of Lemma 5.1 in [8].

**Lemma 3.4.3** ([8]). Let $S = \frac{1}{\|m\|^2} \sum_{j=1}^{d} (I_j - D_j)^2$. With probability at least $1 - \frac{\delta}{2}$ the following holds:

$$|S - d| \leq \epsilon d,$$

whenever $d \geq \frac{24}{3\epsilon^2 - 2\epsilon} \ln \frac{4}{\delta}$.

**Lemma 3.4.4.** If $d$ dimensions are used for the random projection, using memory $O\left(\frac{d}{\epsilon^2(\alpha - 1)^4} \log \frac{4}{\delta}\right)$ it is possible to ensure that with probability at least $1 - \frac{\delta}{2}$ the following holds:

$$|E| \leq \epsilon F_2.$$

**Proof.** First note that, for every $j = 1, \ldots, d$:

$$I_j + D_j = \sum_{i=1}^{n} \frac{M}{m^i} < M + M \int_{2}^{n-1} \frac{1}{i^\alpha} di < \frac{\alpha + 1}{\alpha - 1} M,$$

which implies $(I_j + D_j)^2 < \left(\frac{\alpha + 1}{\alpha - 1}\right)^2 M^2$ for every $j$.

Now, we represent every counter (2$d$ counters) using counting sketch (see Theorem 2) consisting of $O\left(\left(\frac{1}{\epsilon}\right)^2 \left(\frac{1}{\alpha - 1}\right)^4 \ln \frac{4}{\delta}\right)$ bitmaps. Choosing constants suitably, Theorem 2 allows to prove that:

$$P\left[|\tilde{I}_j - I_j| > \frac{\epsilon}{12\alpha - 6} \left(\frac{\alpha - 1}{\alpha + 1}\right)^2 I_j \right] < \frac{\delta}{4d}.$$

The same holds for $\tilde{D}_j$ and $D_j$. This implies:

$$P\left[\exists j : \max\{|\epsilon_1(j)|, |\epsilon_2(j)|\} > \frac{\epsilon}{12\alpha - 6} \left(\frac{\alpha - 1}{\alpha + 1}\right)^2 \right] < \frac{\delta}{2}.$$
As a result, with probability at least $1 - \frac{\delta}{2}$:

$$|E| \leq \frac{3}{\alpha} \sum_{j=1}^{d} \max\{|\epsilon_1(j)|, |\epsilon_2(j)|\}(I_j + D_j)^2$$

$$< \frac{3}{d} \sum_{j=1}^{d} \frac{\epsilon}{12\alpha - 6} \left(\frac{\alpha - 1}{\alpha + 1}\right)^2 (I_j + D_j)^2 < \epsilon F_2,$$

where the third inequality follows from $(I_j + D_j)^2 < \left(\frac{\alpha + 1}{\alpha - 1}\right)^2 M^2$ and from Lemma 3.4.1.

We can now prove the theorem.

Again, we drop $S$ since clear from context. We apply Lemmas 3.4.3 and 3.4.4 with $d = \frac{24}{3\epsilon^2 - 2\epsilon^3} \ln \frac{4}{\delta}$. Setting

$$\hat{S} = \frac{1}{d} \sum_{j=1}^{d} (I_j - D_j)^2,$$

Lemmas 3.4.3 and 3.4.4 imply that $(|\hat{S} - F_2| \leq \epsilon F_2) \wedge (|E| < \epsilon F_2)$ with probability at least $1 - \delta$, so that we have:

$$P\left[|\hat{F}_2 - F_2| > 2\epsilon F_2\right] = P\left[|\hat{S} + E - F_2| > 2\epsilon F_2\right]$$

$$\leq P\left[|\hat{S} - F_2| + |E| > 2\epsilon F_2\right] \leq P\left[|\hat{S} - F_2| > \epsilon F_2\right]$$

$$+ P\left[|E| > \epsilon F_2\right] \leq \delta.$$

The bound of Theorem 3 depends on the value of $\alpha$ and it becomes unbounded as $\alpha \to 1$. In fact, for $\alpha = 1$ we obtain a polylogarithmic bound independent of $\alpha$ and which is stronger than the one of Theorem 3 for values of $\alpha$ close to 1.

### 3.5 Distributed implementation and communication tradeoffs

In this section, we describe a distributed implementation of the primitives described in Section 3.4. Given the composability and duplicate insensitivity of the sketches we consider, the results on the accuracy immediately carry over, so we do not discuss this issue again.
3. DISTRIBUTED STREAMING

PROCESSITEM(LS, GS, i, a, l)

**Input:** LS, GS: SKETCH, i: item, a: attribute, l: distance

1: LS = UPDATESKETCH(LS, i, a)
2: GS = UPDATESKETCH(GS, i, a)
3: Stat = GENSTAT(LS)
4: if DIFF(Stat, Stat_0) > threshold then
   5: MS = BUILDMESSAGE(LS, l)
   6: Send MS to u’s neighbours
   7: Stat_0 = Stat

PROCESSMESSAGE(GS, MS)

**Input:** GS, SKETCH, MS: message

1: S = SKETCH(MS)
2: GS = MERGESKETCH(GS, S)
3: h = TTL(MS)
4: if h > 0 then
   5: h = h - 1
   6: MS = BUILDMESSAGE(S, h)
   7: Send MS to u’s neighbours
   8: else
   9: Drop MS

Figure 3.1: **Generic algorithm** for continuous monitoring of statistics within distance l.

3.5.1 Distributed implementation

**Generic node behaviour (Figure 1).** Upon observing a pair \((i, a)\), node \(u\) invokes \(\text{PROCESSITEM}(S, i, a, 1)\) to update its local and global sketches \(LS\) and \(GS\) (lines 1 and 2) and the estimate of the statistics of interest over \(S(u)\) (line 3). If this exceeds the last value sent\(^2\) by more than a given threshold (line 4), a message \(MS\) containing \(LS\)’s update is built and sent, with initial \(TTL = 1\). Whenever \(u\) receives a message from some neighbour \(v\), containing \(v\)’s update of \(Sk(S(v))\), \(\text{PROCESSMESSAGE}(S, MS)\) extracts the sketch and updates \(u\)’s global sketch (lines 1 and 2). The \(TTL\) is decremented and, if larger than 0, the message is forwarded to \(u\)’s neighbours. Note that this generalizes the “no sharing” update scheme of [86], where they consider a star network topology with a single, central coordinator maintaining a global sketch (See Subsection 3.5.2). It remains to show i) how sketches are updated and merged and ii) how the threshold is defined. The former issue is briefly discussed in the next paragraphs for \(F_0\) (and \(F_1\)) and for \(F_2\), while the latter is addressed in Subsection 3.5.2.

**Fully decentralized distinct counting.** In this case, \(LS\) and \(GS\) in Figure 1

\(^2\)More precisely, whose corresponding sketch was sent.
are counting sketches for the estimation of $F_0(u)$ and $F_0(N_l(u))$ respectively. Counting sketch update and merge operations and count estimation have been succinctly described in Subsection 3.4.1.

**Fully decentralized estimation of $F_2$.** In this case, $LS$ and $GS$ in Figure 1 are composite sketches for the estimation of $F_2(u)$ and $F_2(N_l(u))$ respectively. Following Subsection 3.4.3, $LS$ and $GS$ each consist of $2d$ counting sketches, where $d$ is chosen according to Lemma 3.4.3. Considering $LS$ ($GS$ has exactly the same structure), for every $j = 1, \ldots, d$, the $j$-th sketch $LS[j]$ consists of two counting sketches $ILS[j]$ and $DLS[j]$, to keep track of counters $I_j(S(u))$ and $D_j(S(u))$, as described in Subsection 3.4.3. Update, merge and $F_2$ estimation routines occur as described in Subsection 3.4.3.

### 3.5.2 Communication and Tradeoffs

In this section, we describe in more detail how continuous monitoring of statistics is performed; in particular, when and how information is propagated within the network. The general approach has been described in Section 3.3. In the experiments we considered the *Threshold triggered updates* approach investigated of [86] ("No sharing" policy). Carrying this policy over to our scenario, node $u$ sends an update, i.e., its local sketch $LS$, whenever the estimate $C$ of the statistics of interest over the local stream changes, with respect to the last propagated value $C_0$, i.e., whenever $C > (1 + \theta/|N_l(u)|)C_0$, for some $\theta > 0$.

We also considered an orthogonal approach in which, when a new observation occurs, only the portion of the sketch that eventually changes is propagated. This approach allows a slight improvement in the overall (worst case) number of bits transmitted and brings to messages of smaller size, which can be important in some applications, e.g., sensor networks. Since we used the approach of [86] in the experiments, we omit the presentation of this part of the work.

### 3.6 Experimental analysis

The graph used in the experiments is a real topology of (part of) the Internet at the level of the Autonomous Systems collected by DIMES [187] in December 2008. The symmetrized version of this graph consists of 65,512 nodes and 148,364 edges, and its diameter is equal to 8.

Nodes were assigned real and synthetic traffic data. Real data consist of HTTP requests sent to the 1998 World Cup Web site, made available\(^3\) by

\(^3\)URL: http://ita.ee.lbl.gov/html/contrib/WorldCup.html
the Internet Traffic Archive [16] and spanning three months (May-July 1998). These data were also used in [86]. We considered a week of data, consisting of 10 million tuples. Each HTTP request record contains clientID, objectID, serverID and a timestamp. In our experiments, we did not consider serverID and we chose to focus on the triples (clientID, objectID, timestamp). The data tuples were assigned to nodes of the graph using a hash function to map clientIDs onto graph vertices. Stream tuples then consisted of (objectID, timestamp) pairs.

We generated synthetic data considering a universe of items of size 1,000,000. We imposed \( m_i \) (number of distinct pairs with \( i \) the item) to follow a Zipf distribution with parameter \( \alpha = 2.0 \). Every node in the graph was assigned a stream of length uniformly distributed in \([500,1000]\), consisting of tuples \(<i,a>\), where \( i \) was chosen with probability proportional to \( m_i \) and \( a \in [m_i] \). Globally, we obtained a data stream of approximately 17 million tuples.

### 3.6.1 Accuracy of the estimation

A first set of experiments only concerned the accuracy of the estimation provided by the sketches. We used counting sketches corresponding to a precision \( \epsilon = 0.1 \) with probability at least \( 1 - \delta \), where \( \delta = 0.1 \). For each node in the network, statistics were computed on the streams observed within neighborhoods formed by i) all nodes at distance 1 and ii) all nodes within distance 2. For every statistics of interest we considered the average of the estimates computed over i) the 100 nodes with highest degrees and ii) a set of 100 nodes chosen uniformly at random. For each node and for each statistic, we computed the average error with respect to the exact value.

In estimating \( F_0 \) (see Figure 3.2), we considered the two counting sketches described in Subsection 3.4.1, i.e., the FM sketch originally proposal by Flajolet and Martin [108], and the BY sketch proposed by Bar-Yossef et al [30].

We observed that, when FM sketches are used, the counting sketch provides approximations with error lower than 5% in most cases of the number of distinct items observed by every node within its 1(2)-neighborhood. BY sketches perform extremely well: their accuracy is comparable or slightly better than that of FM sketches. The only issue to consider is that the estimator adopted in [30] (see Subsection 3.4.1) is \( L \cdot M/v \), with \( L = 96/\epsilon^2 \), that corresponds to a value of 9600 in our case. This implies that when using BY sketches, the algorithm always returns an estimate no lower than this value. Since \( L = 96/\epsilon^2 \) is a constant for fixed \( \epsilon \), we simply keep the exact count as long as this remains below this value. In the rest of this section we retain FM sketches, which are simpler to implement and provide an accuracy that is totally adequate for our purposes. Extended results will appear in the full
work.

Accuracy for $F_1$ is always below 4%.

In the estimation of $F_2$ (see Figure 3.3), we used a value of $d$ (number of dimensions of the projected space) equal to 200, which is much lower than the one requested by theoretical analysis (2567) for the values $\epsilon$ and $\delta$ we consider, which may suggest that the analysis can be improved. Complete results will be reported in the full work. In general, as predicted by our analysis, the quality of results is very good for synthetic data, which were generated according to a very skewed distribution following a Zipf’s law with parameter $\alpha = 2$. On the converse, larger errors are observed on real data that exhibit a much lower skewness than that requested by our theoretical analysis: we observed very small values (in the range $[0.6, 0.8]$) of the Zipf’s parameter $\alpha$ for the streams at all the nodes involved in the measurements.

3.6.2 Communication and tradeoffs

The approach we adopt to optimize communication was described in the previous section. It basically consist of the following: every node $u$ sends a message update for a given counter to its immediate neighbors whenever the local estimation computed for that counter exceeds the last update that was sent by more than a factor $1 + \theta/N_l(u)$. 

In order to evaluate the communication costs of the developed methods, we simulated a scenario in which the nodes in the network start to process their local stream simultaneously. We computed $F_0$ and $F_2$ keeping track of the total number of message updates sent by the algorithm at ten equally spaced checkpoints, which were fixed by identifying ten intervals of uniform length within the stream of events observed at each node. We considered different values of $\theta$ ($\theta = 0.1, \theta = 0.2$ and $\theta = 0.4$ and $\theta = 0.8$) to vary the local threshold used by every node to decide whether send or not to send a new message update after processing a new event. We present and discuss below accuracy/communication tradeoffs for the critical $F_0$ estimation primitive for distance-2 neighbourhoods. Results for distance 1 are similar and will be given in the full work.

Figures 3.4 and 3.5 shows the results concerning the estimation of $F_0$ within distance $l = 2$. The general trend is clear: at the beginning, so for the first checkpoints, the number of first-time observations of items grow faster. Accordingly, we observe a steep increase in the number of messages sent. The estimations stabilize after few checkpoints and the amount of exchanged messages drop dramatically. Also, since most distinct items are observed in the first part of the stream, nodes gain very early a very good accuracy in their estimations of $F_0$: for $\theta \in [0.1, 0.2, 0.4, 0.8]$ average error at the first checkpoint
is already around 3–4%.

We experimented with higher values of $\theta$ ($\theta \in [2, 10, 20]$), in order to trade communication with degrading accuracy too much. While $\theta = 2$ does not change the picture, for higher values of $\theta$ we observed a significant drop in the total number of message updates (by a factor in $[1.7, 2]$). Interestingly, the quality of approximation still remains very good: With $\theta = 10$, the average error gets lower than 10% as soon as the third checkpoint is reached, and it is around 7% at the end of the observation. For $\theta = 20$, the error gets lower than 15% at the third checkpoint, and it is not greater than 10% at the last checkpoint.

**Memory requirements.** We implemented sketches for $F_0$ and $F_1$ using at most 575 bytes per sketch. This figure is compatible with commercial sensor networks platforms, although packet sizes have typically smaller values\(^4\). In the case of $F_2$, memory requirements are between 145 and 224 KB per sensing node. This is compatible with applications that use modern sensors such as Intel’s Imote2, which are powerful but are also characterized by higher energy consumption, and thus are not a proper choice in several scenarios where long-life guarantees are a crucial issue, like wireless sensor networks for environmental monitoring, which may be required to work for more than one year. In these cases, application designers typically prefer less powerful devices such as TelosB, MicaZ and TinyNode184, which are provided with less memory ($\sim 10$ KB RAM) but have smaller consumption.

<table>
<thead>
<tr>
<th>LEGENDA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FM</td>
<td>Sketches by Flajolet et Martin</td>
</tr>
<tr>
<td>BY</td>
<td>Sketches by Bar-Yosseff et al.</td>
</tr>
<tr>
<td>R</td>
<td>Real data</td>
</tr>
<tr>
<td>S</td>
<td>Synthetic data</td>
</tr>
<tr>
<td>T</td>
<td>Nodes with highest degree</td>
</tr>
<tr>
<td>RA</td>
<td>Random nodes</td>
</tr>
<tr>
<td>$l$</td>
<td>Radius of the neighborhood</td>
</tr>
</tbody>
</table>

Table 3.1: Legenda

\(^4\)This can be overcome by having a constant number of packets per update message or by reducing the size of update messages, as we show in the full work.
Figure 3.2: Accuracy of the estimation computed for $F_0$ (See Legenda in Table 3.1)

Figure 3.3: Accuracy of the estimation computed for $F_2$
Figure 3.4: Average error in estimating $F_0, l = 2$ at ten checkpoints for different triggering thresholds.

Figure 3.5: Messages sent in estimating $F_0, l = 2$ at ten checkpoints for different triggering thresholds.
Chapter 4

Temporal Characterization of the UK Web

4.1 Introduction

The Web is characterized by an extremely dynamic nature, as it is proved by the rapid and significant growth it has experimented in the last decade and by its continuous evolution through creation or deletion of pages and hyperlinks. Consequently, analyzing the temporal evolution of the Web has become a crucial task that can provide search engines with valuable information for refining crawling policies, improving ranking models or detecting spam.

Understanding and analyzing the structure and the evolution of the Web is an important and delicate challenge that requires theoretical efforts of modelization to be always corroborated by empirical findings. The latter, in turn, are costly to obtain, because they require bandwidth, computation time and human intervention, besides the robust software to gather the data and to provide easy access to the collected information. Indeed, apart for commercial search engines, there have been only very few attempts to perform such a task in an academic setting, and to make the data publicly available. The only previous project with this aim is the Internet Archive\(^1\), a non-profit organization that is trying to provide a sort of time collection of the Web; their dataset cannot be easily accessed for batch analysis, and although socially and

\(^{1}\text{http://www.archive.org/}\)
4. CHARACTERIZATION OF THE UK WEB

historically important, it is of scarce interest for those who aim at studying structural properties.

The problem becomes even more elusive if time evolution is taken into account, because one would like to have not only different snapshots of the same portion of the Web to be available, but also some guarantee on their mutual consistency (for example, to be sure that the same crawling policies have been followed) is in that case of imperative importance.

Recently, a new temporal dataset has been made public: it is made of a series of twelve 100M pages snapshots of the .uk domain [48], which were gathered at a monthly rate. The Web graphs of the twelve snapshots have been merged into a single time-aware graph that provide constant-time access to temporal information.

In this work, we present an extensive analysis that was conducted on the above collection, which represents the first public data on the evolution of the Web that use a large scale and a significant diversity in the sites considered.

• First, we focus our attention on the problem of assessing the data obtained through crawling: Understanding all its possible anomalies found is an issue of uttermost importance if one wants to understand correctly how the data evolve over time. For this purpose, we perform several tests to check whether the information contained in the graph is reliable, that is, whether it depends essentially on appearance and disappearance of pages and links, or on the crawler behavior. The results of this analysis show that the graph is actually reliable.

• Next, we afford the issue of providing a characterization that also takes temporal evolution into account.

• As a first attempt, we analyze the structure of this huge time-aware graph at the level of interconnection between hosts. We study the host-graph, i.e., the graph in which every node corresponds to a site, whereas a directed edge represents the existence of hyperlinks between pages belonging to two different hosts. Understanding the structure of the Web at this macroscopic level can provide valuable insights for improving site accessibility and navigation, or discovering related hosts.

• We study the temporal evolution of 3500 sites with respect to a number of topological properties, including degrees, number of degree supporters and eigenvector distributions. The sites taken into consideration mostly exhibit a very stable behavior. However, a non negligible percentage of hosts is characterized by increasing or decreasing evolution patterns. We notice a positive correlation in the growth of different properties. Interestingly, the growth of the measures analyzed is assortative, that is, Web sites mostly grow together with their neighbors in the host graph.
4.2 Related work

Several previous works [60, 77, 107, 119, 143] focused on Web evolution, tracing a set of pages to compile statistics about their frequency and rate of change over time. The search engine perspective is dominant in these studies. Cho and Garcia Molina [77] performed a daily collection of around 720,000 pages from 270 popular sites during a period of 4 months. They counted how many days each URL was accessible and proposed estimators for the frequency of change of Web pages.

Brewington and Cybenko [60] collected pages observed over an average interval of 37 days and used the recording of the last-modified timestamp and the downloading time of Web pages to study their rate of change.

Fetterly et al. [107] crawled on a weekly basis a set of 150 million URLs obtained from Yahoo! Directory, spanning a time interval of 11 weeks in 2002. This work aimed at studying the frequency and the degree of change of Web pages. The authors observed that a significant amount of changes on the Web consists of small modifications, like html tags.
Ntoulas et al. [173] collected for one year exhaustive weekly downloads of Web pages belonging to 154 popular sites gathered from the Google Directory. They observed high birth and death rates for Web pages, and an even higher turnover rate for the hyperlinks. At the same time, they noticed that newly created pages tend to borrow their content heavily from existing pages, and most of the pages that persist over time exhibit only minor changes in their content.

Koheler [143] considered a collection of 361 URLs selected at random from a Web crawl during a period of 4 years. The author performed accessibility tests, observing a periodic resurrection of Web pages and sites. He claimed that once a collection has reached a considerable age, it tends to stabilize.

Gomes and Silva [119] examined a set of 51 million pages, using a number of snapshots gathered from a national community web, spanning a period of 3 years. The authors measured the persistence of both URLs and Web content. They found that most URLs have a short life, while a minor fraction of pages persist for long periods of time. Gomes and Silva observed that persistent URLs are static, short and tend to be linked from other sites. They also noticed that URL persistence is not influenced by the depth.

Toyoda and Kitsuregawa [197] analyzed data from the Japanese Web Archive and proposed a novelty measure for estimating the certainty that a newly created page newly appeared within a series of unstable snapshots. This measure can be used to extract novel pages for further analysis with reasonable precision.

The notion of hostgraph was proposed by Bharat et al. [41]. A few studies analyzed hostgraphs ([23, 155, 188]). Baeza-Yates et al. [22] presented a comparison among the results of twelve characterization studies of several national domains.

### 4.3 Dataset

The data consist of one year of data about the UK Web (.uk ccTLD), which were collected by Laboratory of Web Algorithmics\(^2\), at University of Milan.

The .uk domain was chosen because it contains pages that are written in the English language, which is really necessary since understanding the content is actually needed for any kind of assessment by human judgements. At the same time, the chosen domain is characterized by a size that is significant for a national domain but still reasonable enough to manage.

\(^2\)\url{http://law.dsi.unimi.it/}
4.3. Dataset

4.3.1 Crawling

The data were gathered by crawling the .uk domain from June, 2006 to May, 2007. Twelve monthly snapshots of about 100M pages each were collected. Each snapshot was taken at the beginning of the correspondent month, during a period of 7-10 days, using UbiCrawler [45], a scalable and distributed crawler developed at University of Milan. Basic information about the snapshots is shown in Table 4.1. A more complete description of how the data was built is provided by Boldi et al. [48].

<table>
<thead>
<tr>
<th></th>
<th>Pages</th>
<th>GZip'd Size (GB)</th>
<th>Nodes</th>
<th>Arcs</th>
<th>Graph Size(GB)</th>
<th>bit/arc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jun</td>
<td>112,386,763</td>
<td>402</td>
<td>80,644,902</td>
<td>2,481,281,617</td>
<td>0.89</td>
<td>3.07</td>
</tr>
<tr>
<td>Jul</td>
<td>136,956,559</td>
<td>477</td>
<td>96,395,298</td>
<td>3,030,665,444</td>
<td>1.16</td>
<td>3.30</td>
</tr>
<tr>
<td>Aug</td>
<td>141,395,895</td>
<td>507</td>
<td>100,751,978</td>
<td>3,250,153,746</td>
<td>1.23</td>
<td>3.25</td>
</tr>
<tr>
<td>Sep</td>
<td>148,965,298</td>
<td>546</td>
<td>106,288,541</td>
<td>3,871,625,613</td>
<td>1.32</td>
<td>2.93</td>
</tr>
<tr>
<td>Oct</td>
<td>129,558,491</td>
<td>478</td>
<td>93,403,772</td>
<td>3,130,910,405</td>
<td>1.03</td>
<td>2.83</td>
</tr>
<tr>
<td>Nov</td>
<td>150,146,132</td>
<td>546</td>
<td>106,783,458</td>
<td>3,479,400,938</td>
<td>1.16</td>
<td>2.86</td>
</tr>
<tr>
<td>Dec</td>
<td>144,489,446</td>
<td>525</td>
<td>103,098,631</td>
<td>3,768,836,665</td>
<td>1.34</td>
<td>2.77</td>
</tr>
<tr>
<td>Jan</td>
<td>151,578,113</td>
<td>553</td>
<td>108,563,230</td>
<td>3,929,837,236</td>
<td>1.38</td>
<td>2.72</td>
</tr>
<tr>
<td>Feb</td>
<td>153,966,540</td>
<td>564</td>
<td>110,123,614</td>
<td>3,944,932,566</td>
<td>1.39</td>
<td>2.74</td>
</tr>
<tr>
<td>Mar</td>
<td>151,427,461</td>
<td>545</td>
<td>107,565,084</td>
<td>3,642,701,825</td>
<td>1.34</td>
<td>2.84</td>
</tr>
<tr>
<td>Apr</td>
<td>150,606,689</td>
<td>559</td>
<td>106,867,191</td>
<td>3,790,305,474</td>
<td>1.36</td>
<td>2.79</td>
</tr>
<tr>
<td>May</td>
<td>150,054,551</td>
<td>556</td>
<td>105,896,555</td>
<td>3,738,733,648</td>
<td>1.30</td>
<td>2.69</td>
</tr>
</tbody>
</table>

Table 4.1: Per-snapshot full-text and web-graph stats.

The crawler started from a seed of 150,000 urls obtained from the Open Directory Project and applied a breadth-first visiting policy, visiting at most 50,000 pages and 16 levels per host.

Crawling parameters A careful definition of the crawling parameters is crucial, as in any limited-size crawl. The stopping criterion is that of reaching about 100M pages, without counting duplicates. The main features are listed below.

Crawl policy. We use UbiCrawler’s [46] built-in per-host breadth-first visit. A number of threads scan in parallel distinct hosts, and newly discovered URLs are added to a queue. When a thread completes its visit, it extracts from the queue the first URL whose host has an IP address that is not currently being visited, and starts visiting that host in a breadth-first fashion.
4. CHARACTERIZATION OF THE UK WEB

Seed. The seed is a large (190,000 elements) set of URLs obtained from the Open Directory Project. The reason for such a large seed is that of making the crawl more stable and repeatable, and reducing the amount of spam (as links in the Open Directory Project are judged by humans).

Maximum number of pages per host. We limited each host to a maximum of 50,000 pages. This guarantees that we shall crawl at least 2,000 hosts, and limits the impact of web traps and database-driven sites.

Maximum inter-host depth. We do not delve more than 16 levels in a host. The main reason for a limit in depth is avoiding traps and also badly configured 404 pages, which sometimes generate an infinite number of links by prefix buildup.

URL normalisation. URLs are normalised following the strategy explained in the BURL Java class. We apply all safe normalisations, escape all illegal characters, and treat in a special way square brackets as they are ubiquitously (although erroneously) used in an unescaped form.

Duplicate detection. Many pages are duplicates, and to detect their presence we maintain a set of 64-bit fingerprints obtained after stripping attributes (of HTML elements) and other non-relevant parts of the page. When a duplicate is detected we just store a pointer to the original page. About 25\% of the overall pages happen to be duplicates (so, to collect 100M distinct pages we had to download some 130M pages at each crawl).

4.3.2 Aligning the snapshots

The first important step in getting a temporally labelled collection is alignment: identifying URLs in different snapshots that correspond to the same Web page. Alignment is a non-trivial issue because if a URL is not static it might contain session-generated data (e.g., a session ID) that makes de facto identical URLs appear as syntactically distinct. For the present collection, the radical choice was made of considering only static URLs (i.e., URLs that do not contain a question mark). Table 4.2 shows that, as a first attempt, our choice is not unreasonable. However, one of the open problems is to develop some sensible alignment technique for dynamic URLs.

3www.dmoz.org
4The class is available in bundle with the LAW software, downloadable at
4.3. Dataset

<table>
<thead>
<tr>
<th></th>
<th>Jun</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jun</td>
<td>31.3</td>
<td>19.0</td>
<td>18.3</td>
<td>17.2</td>
<td>15.3</td>
<td>15.0</td>
<td>13.8</td>
<td>13.7</td>
<td>13.2</td>
<td>12.3</td>
<td>11.9</td>
<td>11.1</td>
</tr>
<tr>
<td>Jul</td>
<td>35.2</td>
<td>23.3</td>
<td>21.7</td>
<td>18.5</td>
<td>18.3</td>
<td>16.2</td>
<td>16.4</td>
<td>16.0</td>
<td>15.2</td>
<td>14.5</td>
<td>13.6</td>
<td></td>
</tr>
<tr>
<td>Aug</td>
<td>37.3</td>
<td>24.3</td>
<td>19.4</td>
<td>19.4</td>
<td>17.1</td>
<td>17.3</td>
<td>16.7</td>
<td>15.5</td>
<td>15.0</td>
<td>15.0</td>
<td>13.8</td>
<td></td>
</tr>
<tr>
<td>Sep</td>
<td>39.9</td>
<td>21.2</td>
<td>21.2</td>
<td>18.7</td>
<td>19.0</td>
<td>18.1</td>
<td>16.9</td>
<td>16.3</td>
<td>14.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oct</td>
<td>33.8</td>
<td>22.2</td>
<td>19.0</td>
<td>19.1</td>
<td>18.3</td>
<td>16.6</td>
<td>16.4</td>
<td>15.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov</td>
<td>37.3</td>
<td>22.3</td>
<td>21.9</td>
<td>21.3</td>
<td>18.8</td>
<td>18.3</td>
<td>16.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec</td>
<td>36.6</td>
<td>23.5</td>
<td>21.0</td>
<td>19.0</td>
<td>17.9</td>
<td>16.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan</td>
<td>39.0</td>
<td>23.7</td>
<td>20.8</td>
<td>19.9</td>
<td>18.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feb</td>
<td>37.7</td>
<td>23.1</td>
<td>22.2</td>
<td>20.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mar</td>
<td>38.1</td>
<td>22.4</td>
<td>20.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apr</td>
<td>36.9</td>
<td>24.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>May</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>36.9</td>
</tr>
</tbody>
</table>

Table 4.2: Static URLs overlap, in millions.

4.3.3 Creating a graph annotated with temporal labels

The Web graphs of the twelve snapshots were created and compressed using WebGraph [50], a framework for managing very large graphs that currently provides the best compression available in terms of bits per link.

Aligning URLs, we obtained a global graph $G$ that includes all static pages (and related links) appearing in each snapshot. Correspondently, the Web graphs of the single snapshots were merged into a single, global time-aware graph that provides constant time access to temporal information. For the merged graph, WebGraph was augmented with the possibility of storing highly compressed labels on the arcs. Given that there are twelve crawls, one per month, the graph must provide twelve bits of information per node and per arc. The labelling facilities of WebGraph, combined with a compression technique based on Huffman codes, make it possible to store a label in just 2.16 bits per label (for more details, see [48]). The label assigned to each node or link allows to detect whether it was present or not in any given snapshot.

Using the same framework, we also extracted from the Web graph of each

http://law.dsi.unimi.it/.

*This choice, unfortunately, cannot prevent opaque session-dependent URLs from generating noise in the collection.*
snapshot the corresponding host graph. Moreover, we created a global host graph representing the union of the host graphs. This graph is provided with labels that maintain temporal information regarding the single graphs which every element (node or edge) belongs to.

4.4 General definitions and notation

**Snapshot representation.** Let $T$ be the number of snapshots collected, numbered from 1 to $T$; the $t$-th snapshot consists of:

- a set $S_t$ of **seen URLs** that includes all URLs ever found by the crawler;
- a set $C_t$ of **crawled URLs**: $C_t \subseteq S_t$ this is the set of URLs that have a WARC record in the WARC store saved by the crawler; such set is divided into:
  - a set $V_t$ of **existing URLs**, which form the nodes of the $t$-th crawl graph;
  - a set $F_t$ of **failure URLs**: this is the set of URLs that resulted in a 4xx response;
  - a set $D_t$ of **duplicate URLs**: the crawler saved them as duplicate of some other existing URL (an element of $V_t$), called its **archetype**; in other words, there is a map $a_t : D_t \rightarrow V_t$ that maps each duplicate URL to its archetype. For convenience, let us extend it on $D_t \cup V_t$ by the identity.
- for every crawled URL $u \in C_t$, we have a stored page $P_t(u)$, that is, the content and HTTP headers received and saved in the WARC file (note that the crawler does not save the content for duplicate URLs, so for the sake of definiteness we let $P_t(u) = P_t(a_t(u))$ whenever $u \in D_t$).

The $t$-th crawl graph is the directed graph $G_t = (V_t, E_t)$ where $(x, y) \in E_t$ if $P_t(a_t(x))$ contains an anchor $y'$ with $y = a_t(y')$.

We also let $C = \cup_t C_t$, $F = \cup_t F_t$, $V = \cup_t V_t$, $D = \cup_t D_t$, $E = \cup_t E_t$ and $G = (V, E)$.

We also build the host graph of each snapshot. We model the host graph extracted from the Web crawled at time $t$ as a directed graph $H_t = (W_t, A_t)$. The set of nodes $W_t$ corresponds to Web hosts, whereas a directed edge $(u, v) \in A_t$ indicates the existence of hyperlink(s) between nodes $u$ and $v$. We represent the evolving Web at the level of interconnection between hosts as a sequence
4.5. DATA ASSESSMENT

of graphs \( H_t = (W_t, A_t) \), where \( t = t_1, \ldots, t_{12} \). The total set of nodes in this sequence of graphs is aligned. Again, let \( W = \bigcup_t W_t \), \( A = \bigcup_t A_t \) and \( H = (W, A) \).

Node (Site) appearance trace (PAT). For every \( u \in V \), consider the map \( f_u : \{1, \ldots, T\} \to \{0, 1\} \) such that \( f_u(t) = 1 \) iff \( u \in V_t \): this is what we will refer to as page appearance trace (PAT).

The persistence of a URL between times \( i \) and \( j \) is the fraction of 1s in its PAT between positions \( i \) and \( j \).

Similarly, for every \( u \in W \), consider the map \( f_u : \{1, \ldots, T\} \to \{0, 1\} \) such that \( f_u(t) = 1 \) iff \( u \in V_t \): this is what we will refer to as host appearance trace (PAT). The persistence of a host between times \( i \) and \( j \) is the fraction of 1s in its PAT between positions \( i \) and \( j \).

4.5 Data assessment

In this section we present a set of experiments performed to assess the reliability of our dataset. This assessment phase is instrumental in the subsequent usage of the crawled data, and aims at establishing how confident we can be that a snapshot itself is a faithful representation of the .uk domain. We first provide details about the definitions and the notation used in the experiments. Subsequently, we present the experiments themselves and the relative results.

4.5.1 Preprocessing: Classifying URLs

This experiment aims at classifying the URLs in \( u \in V \), according to the following parameters:

- **Crawl depth** \((D_C)\): the length of the shortest directed path from a(ny) URL in the seed to \( u \); more precisely, \( u \) has depth 0 if it is in the seed\(^7\), and it has depth \( k + 1 \) if there is some \( u' \) of depth \( k \) such that \((u', u) \in E\);

- **Syntactic depth** \((D_S)\): the number of slashes “/” in the path part of the URL.

The left side of Table 4.3 shows the head of the crawl depth distribution, reporting, for the most common values of the parameter, the percentage of nodes with that specific depth value. It is worth to note that the fourth most common value is equal to infty. Nodes that have an infinite crawl depth cannot

\(^6\)To simplify notation, we shall write a PAT as the ordered sequence of its values.

\(^7\)The seed was the same for all snapshots.
be reached from the seed. There are almost 16 million such nodes, that is, a fraction equal to 11% of the total nodes. This means that, by considering only the static URLs, we are significantly disconnecting the graph.

Figure 4.1 shows the crawl depth distribution (only finite values are considered). We notice that the 70% of the nodes have a crawl depth not greater than 10, and the 60% of the nodes have a crawl depth not greater than 7. Hence, a major part of the nodes in the dataset are reachable from a URL in the seed with a small number of hops.

The right side of Table 4.3 shows the head of the syntactic depth distribution, which is plotted in Figure 4.2. We observe that 95% of the nodes have a syntactic depth not greater than 9, and 85% of the nodes correspond to URLs with no more than 5 syntactic levels. This is as expected, because relevant content is often located at the top of a site’s hierarchy, rather than in a deeper location within the site.

The computation of the Pearson correlation coefficient between the two depth distributions returned a value equal to $-0.32$, which basically means that such distributions are not significantly correlated. We might expect some
4.5. Data assessment

correlation very likely to be found, if we thought that, the deeper a page within the tree directory of a site, the longer the path to reach it starting from a page in the seed set. We believe that the observed evidence can partially be explained with the way our crawler works: it enters a site as soon as it finds some links to it, not necessarily from its homepage.

4.5.2 Measuring page persistence

The persistence of a page is the fraction of 1s in its PAT. We compute such a measure for the pages in the intersection between the first and the last crawl. Figure 4.3 shows the distribution of persistence values.

If the crawling were perfect, a page existing both in the first month and in the last one should be present in every snapshot, so we would expect its persistence to be equal to 1. This does not happen in every case: sometimes the crawler failed to crawl existing URLs. One explanation for this fact might be a temporary unavailability of a site. In some other cases, the crawler did not reach the URL because the per-host limit imposed had been exceeded.

Figure 4.2: Syntactic depth distribution.
4. CHARACTERIZATION OF THE UK WEB

| $D_C$ | $|V|$ | $D_S$ | $|V|$ |
|-------|-------|-------|-------|
| 6     | 15    | 1     | 19    |
| 7     | 14    | 2     | 21    |
| 5     | 13    | 3     | 19    |
| $\infty$ | 11  | 4     | 15    |
| 8     | 11    | 5     | 10    |
| 4     | 8     | 6     | 6     |
| 9     | 8     | 7     | 4     |
| 10    | 5     | 8     | 2     |
| 3     | 4     | 9     | 1     |
| 11    | 3     | 10    | 1     |

Table 4.3: Crawl depth and syntactic depth distribution.

Hence, there exist nodes in the intersection between the first and the last month that are not appearing in some intermediate snapshots.

However, we observe that around 90% of the considered nodes have a persistence in the range $(0, 0.8, 1)$. These nodes have at most two missing zeros in their PATs. We feel confident in the fact that such pages existed also in the months in which the crawler failed to capture them.

4.5.3 Experiments related to deletion of pages

This section provides details about the experiments we conducted to evaluate how faithfully the crawled data is able to capture the event of a page deletion.

No-Resurrection Assumption Experiment We say that a node $u$ breaks the no-resurrection assumption at time $t$ if and only if there are $t_1 < t < t_3$ such that $f_u(t_1) = f_u(t_3) = 1$ and $f_u(t) = 0$. We believe that if a URL breaks the no-resurrection assumption at time $t$ then it is the crawler that failed to crawl the URL for the two reasons that were mentioned in the previous subsection. A PAT breaking the no-resurrection assumption is referred to as an anomalous PAT.

We use presence/absence information provided by node labels to compute the number of URLs that break the no-resurrection assumption. We start our analysis of the results by dividing the appearance traces of the nodes into four groups:

- **Monotone non descending ($A$)**: the nodes that appeared at some time and then never disappeared. The PAT of these nodes has the form $0^*1^+$. 
4.5. Data assessment

Figure 4.3: Persistence distribution for the nodes in the intersection between the first and the last snapshot.

- **Monotone descending (D):** the nodes that appeared in the first snapshot and at some point disappeared forever. These nodes are characterized by a PAT like $1^+0^+$.

- **Plateaux (P):** the nodes that appeared at some time and then disappeared forever. Such nodes exhibit a PAT with the form $0^+1^*0^+$.

- **Bad nodes (B):** the nodes that break the no-resurrection assumption. These pages disappeared at some point and then reappeared.

The upper section of Table 4.4(a) shows the percentage number of occurrences of each type of pattern. All the possible $2^{12} - 1$ patterns actually appear in the data. We observe that 12.35% of the nodes exhibit monotone non descending appearance traces, while 11.40% of the Web pages in the union graph have a PAT characterized by a monotone descending behavior. Monotone PATs are reported in Tables 4.4(b) and 4.5(a). The 44.46% of the nodes are classified as plateaux: they appear at a given time, they exist in a number
4. CHARACTERIZATION OF THE UK WEB

<table>
<thead>
<tr>
<th>Pattern type</th>
<th>%</th>
<th>V</th>
<th>Non-desc. PAT</th>
<th>%</th>
<th>V</th>
<th>%</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>12.35</td>
<td>000000000001</td>
<td>4.74</td>
<td>38.41</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>11.40</td>
<td>111111111111</td>
<td>3.27</td>
<td>26.48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>46.44</td>
<td>000000000011</td>
<td>0.98</td>
<td>7.91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>29.82</td>
<td>000000000111</td>
<td>0.72</td>
<td>5.82</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anomalous PAT</td>
<td>0.35</td>
<td>000000001010</td>
<td>0.98</td>
<td>7.91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000000101</td>
<td>0.30</td>
<td>011111111111</td>
<td>0.38</td>
<td>3.10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110111111111</td>
<td>0.29</td>
<td>000000111111</td>
<td>0.33</td>
<td>2.70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111110111111</td>
<td>0.28</td>
<td>000111111111</td>
<td>0.33</td>
<td>2.70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110100000000</td>
<td>0.27</td>
<td>000011111111</td>
<td>0.22</td>
<td>1.81</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000001010</td>
<td>0.27</td>
<td>001111111111</td>
<td>0.21</td>
<td>1.69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010100000000</td>
<td>0.25</td>
<td>000001111111</td>
<td>0.18</td>
<td>1.45</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.4: (a) Appearance traces for all the nodes in the graph and common anomalous PATs. (b) Monotone non-descending appearance traces.

of consecutive crawls, then they disappear forever. The number of bad nodes is less than 30%. The lower section of Table 4.4(a) reports the percentage number of occurrences of the most common anomalous PATs. It is worth to note that these patterns contain only one misplaced $0$. The pages characterized by such appearance traces do exist in a number of consecutive crawls, except for one month.

We also investigate how the fraction of URLs that break the no-resurrection assumption at time $t$ changes as $t$ changes. The results of this experiment are presented in Table 4.5(b), which reports, for each time $t$, the number of nodes that break the no-resurrection assumption at time $t$, expressed as percentage with respect to the number of bad nodes and to the total number of nodes. We remind here that, by definition, a node $u$ breaks the no-resurrection assumption at time $t$ if and only if there exist $t_1 < t < t_3$ such that $u$ appears in months $t_1$ and $t_3$, while it is not present in the $t$-th snapshot. There cannot be nodes breaking the assumption neither in month 1 nor in month 12. The number of bad nodes is higher in the intermediate snapshots, while it is significantly lower in months 2 and 11. This finding is not in contrast with the results provided in Table 4.4(a), where we observe that the first and the third most common bad traces are the ones characterized by a single bad zero in the eleventh and in the second position. The reason is that the results presented in Table 4.5(b) are obtained by aggregating, for each time $t$, the occurrences of
4.5. Data assessment

Table 4.5: (a) Monotone descending appearance traces. (b) Change of the fraction of nodes breaking the no-resurrection assumption as $t$ changes.

We complete the present experiment by studying how the fraction of bad nodes changes depending on the URL classification parameters. Table 4.6 provides results for some frequent values of the crawl depth. For a given depth value $k$, and for every $t$, the percentage number of nodes breaking the no-resurrection assumption is reported. If we analyze the results obtained for a specific value of the crawl depth, we observe that the fraction of bad nodes keeps basically stable over the twelve snapshots. On the converse, if we take into account a single snapshot, we notice that, in many cases, the fraction of bad nodes is higher for higher crawl depth values. We think this is completely reasonable, because a bad behavior is more likely to be expected when we go far from the seed set, which contains well maintained pages obtained from ODP.

**Gone-Is-Gone Assumption Experiment** Every non-anomalous PAT $f_u$ has the form $0^k1^h0^\ell$: if $\ell > 0$ then we say that $u$ disappeared at time $k + h$. Although there is in general no way to see (from the WARC files) if the URL actually disappeared, we may have a clue in some cases as follows:

- suppose that $u$ disappeared at time $t$;
- let $A = \{v \in V_t \cap V_{t-1} \mid (v,u) \in E_{t-1}\}$ be the set of nodes that existed
4. CHARACTERIZATION OF THE UK WEB

<table>
<thead>
<tr>
<th>Depth</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>6</th>
<th>7</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = 1</td>
<td>0.04</td>
<td>0.14</td>
<td>0.77</td>
<td>1.74</td>
<td>1.46</td>
<td>0.60</td>
<td>0.37</td>
</tr>
<tr>
<td>t = 2</td>
<td>0.05</td>
<td>0.25</td>
<td>0.87</td>
<td>1.86</td>
<td>1.54</td>
<td>0.83</td>
<td>0.48</td>
</tr>
<tr>
<td>t = 3</td>
<td>0.04</td>
<td>0.23</td>
<td>0.79</td>
<td>2.01</td>
<td>1.73</td>
<td>0.85</td>
<td>0.56</td>
</tr>
<tr>
<td>t = 4</td>
<td>0.05</td>
<td>0.25</td>
<td>0.83</td>
<td>2.49</td>
<td>2.04</td>
<td>0.93</td>
<td>0.62</td>
</tr>
<tr>
<td>t = 5</td>
<td>0.04</td>
<td>0.21</td>
<td>0.76</td>
<td>2.04</td>
<td>1.64</td>
<td>0.77</td>
<td>0.48</td>
</tr>
<tr>
<td>t = 6</td>
<td>0.04</td>
<td>0.24</td>
<td>0.85</td>
<td>2.38</td>
<td>1.97</td>
<td>0.89</td>
<td>0.60</td>
</tr>
<tr>
<td>t = 7</td>
<td>0.03</td>
<td>0.21</td>
<td>0.83</td>
<td>1.13</td>
<td>0.88</td>
<td>0.85</td>
<td>0.53</td>
</tr>
<tr>
<td>t = 8</td>
<td>0.04</td>
<td>0.24</td>
<td>0.86</td>
<td>1.85</td>
<td>1.64</td>
<td>1.02</td>
<td>0.54</td>
</tr>
<tr>
<td>t = 9</td>
<td>0.04</td>
<td>0.25</td>
<td>0.92</td>
<td>2.38</td>
<td>1.97</td>
<td>0.93</td>
<td>0.57</td>
</tr>
<tr>
<td>t = 10</td>
<td>0.04</td>
<td>0.25</td>
<td>0.83</td>
<td>2.16</td>
<td>1.74</td>
<td>0.88</td>
<td>0.55</td>
</tr>
<tr>
<td>t = 11</td>
<td>0.04</td>
<td>0.22</td>
<td>0.81</td>
<td>2.13</td>
<td>1.76</td>
<td>0.84</td>
<td>0.53</td>
</tr>
<tr>
<td>t = 12</td>
<td>0.04</td>
<td>0.24</td>
<td>0.87</td>
<td>2.07</td>
<td>1.72</td>
<td>0.83</td>
<td>0.43</td>
</tr>
</tbody>
</table>

Table 4.6: Change of the fraction of bad nodes for different values of the crawl depth.

both at time $t$ and time $t - 1$, and that were in-neighbors of $u$ at time $t - 1$ (we assume that both $v$ and $u$ do not break the no-resurrection assumption);

- if $P_{t-1}(v) = P_t(v)$ (that is: if the content of page $v$ did not change from time $t - 1$ to time $t$), then page $P_t(v)$ should still contain an anchor to $u$, which means that we should have requested $u$ at time $t$ and should have obtained a 4xx as answer. That is, $u \in F_t$.

A URL that does not satisfy the property above is said to break the gone-is-gone assumption.

We compute the number of URLs breaking the gone-is-gone assumption as follows. Let $d(u)$ be the disappearance time of $u$: in particular, let this be $T + 1$ if $u$ did not disappear. We first output a list of triples $(v, u, t)$ such that $(d(v) > d(u) = t$ and $(v, u) \in E_{t-1}$). For each such triple, we check whether it satisfies the constraint $(u \in F_t$ or $P_t(v) \neq P_{t-1}(v))$. This can be done by accessing the WARC files to see if $u$ was actually requested but produced a 4xx, or if the content of $v$ was changed. If the specified constraint is not satisfied, then $u$ breaks the gone-is-gone assumption.

Table 4.7 presents the outcome of the experiment. The fraction of triples for which $u$ produced a 4xx is very small, and the number of triples breaking the gone-is-gone assumption is also small. In most of cases, the results are what we expected: whenever the edge from node $v$ to node $u$ does not exist anymore, we find that the content of the page has changed. We believe this
is a sign of the good quality of the adopted crawling scheme. Clearly, due to
the choice of a time granularity of one month, we cannot have full confidence
in the fact that our time-aware graph captures all the changes that actually
occurred in the real data: some intermediate changes might be not represented.
Moreover, we cannot really know how fast the content of Web pages changed
from one snapshot to the subsequent one: the chosen granularity makes us
unable to tell which week (day) of the considered month occurred in. The
choice of a finer granularity would improve the ability of providing a faithful
representation of the data.

\begin{table}[!h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
Time & $\% u \in F_t$ & $\% u \notin F_t$ & $\%$ others \\
\hline
1 & 0.55 & 88.37 & 11.07 \\
2 & 0.39 & 82.87 & 16.74 \\
3 & 0.40 & 81.91 & 17.69 \\
4 & 0.16 & 89.51 & 10.33 \\
5 & 0.15 & 89.15 & 10.70 \\
6 & 0.69 & 92.59 & 6.71 \\
7 & 0.04 & 88.24 & 11.73 \\
8 & 0.10 & 82.99 & 16.91 \\
9 & 0.15 & 84.41 & 15.44 \\
10 & 0.26 & 77.57 & 22.17 \\
11 & 0.89 & 65.02 & 34.09 \\
\hline
\end{tabular}
\caption{Results for the gone-is-gone experiment.}
\end{table}

4.5.4 Adjusting the graph

The experiments presented in the previous subsection show that the fraction
of URLs that break the no-resurrection assumption is small, in particular
at small depths. Also, the most common anomalous PATs contain a single
anomalous extra 0.

Given that these results make us confident in the good quality of the
crawling, we decide to “adjust” the graph by eliminating all anomalous PATs
through the adoption of a \textit{fill-in-the-gap} approach, that is, we make all anoma-
lous PATs of the form $0^k1\alpha10^h$ into $0^k1^{T-(k+b)}0^h$: in other words, we assume
that the URL existed also if it was not crawled. We perform the adjustment
## 4. CHARACTERIZATION OF THE UK WEB

| Pattern type | % $|V|$ |
|--------------|------|
| A            | 16.18 |
| D            | 14.19 |
| P            | 50.54 |
| B            | 19.09 |

| Anomalous PAT | % $|V|$ |
|---------------|------|
| 000000001001  | 0.22 |
| 000100100000  | 0.17 |
| 000001001000  | 0.16 |
| 100100000000  | 0.15 |
| 001001000000  | 0.14 |
| 000010010000  | 0.13 |
| 010010000000  | 0.12 |

Table 4.8: Appearance traces for the all the nodes in the adjusted graph and common anomalous PATs.

using the labelling facility implemented in WebGraph to generate new labels\(^8\) for our time-aware Web graph. We then investigate how much the applied adjustment changes the outcomes of the experiments presented in the previous subsection. The upper section of Table 4.8 shows the number of occurrences for the various pattern types in the patched graph. We observe that the fraction of bad nodes is now equal to 19.09%: thanks to the applied adjustment, around 10% of the nodes, that were originally charaterized by an anomalous PAT, are now good nodes belonging to a different category (monotone non descending, monotone descending or plateaux). The lower section of Table 4.8 shows the most common appearance traces for the nodes that still break the no-resurrection assumption in the adjusted graph. Once the patterns with a single bad zero have been patched, the most common patterns become the ones with two bad zeros.

### 4.6 Host evolution over time

We start the analysis of this huge time-aware graph at a coarse granularity. We study the hostgraph, investigating how 3 500 sites evolve with respect to

\(^8\)The new labels will be freely available.
several topological properties, including degrees, number of degree supporters and eigenvector distributions. Characterizing the dynamics of the Web at this macroscopic level can provide valuable information for improving site accessibility and navigation, as well as discovering related hosts.

Given an instance of the host graph $H_t = (W_t, A_t)$ crawled at time $t$, we extract a set of features $f_i, i = 1, \ldots, F$ for each node $v \in W_t$. We compute in-degree, out-degree, PageRank, estimation of the number of degree supporters within distance 3. We denote by $f_i(v, t)$ the value of feature $f_i$ for the node $v \in W_t$ extracted from the graph $H_t = (W_t, A_t)$.

We now consider how the computed features evolve over time. For a given feature $f_i$ and a node $v$ we denote by $f_i(v) = (f_i(v, 1), \ldots, f_i(v, T_v))$ the time series consisting of the values of feature $f_i$ during the life of node $v$.

For convenience we assume that in the time series $f_i(v)$, the first value $f_i(v, 1)$ corresponds to the first time that the node $v$ appears in the Web. Thus the length $T_v$ of the time series $f_i(v)$ varies among different nodes; it is shorter for nodes that appear later in the Web, however, for convenience again, we assume that $T_v = T'$ for all nodes. Such a uniformity on the length can be imposed by truncating time series of length greater than $T'$ and pruning away time series with length smaller than $T'$.

It is also important to note that the values of the features can vary significantly in magnitude among different nodes: a no popular host might have a handfull of in-links, while a popular one can have hundreds or thousands. One way to account for such a difference is to focus on the growth of features values with respect to the value at the point of appearence of a node. We define $g_i = (1, \frac{f_i(v, 2)}{f_i(v, 1)}, \ldots, \frac{f_i(v, T_v)}{f_i(v, 1)})$ to be the normalized time series for a feature $f_i$ and a node $v$, where each component is divided by $f_i(v, 1)$.

### 4.6.1 Clustering evolution patterns

Our first experiment aims at identifying nodes with similar evolution patterns. We cluster the nodes of the host graph given the normalized feature time series $g_i(v)$. Let $C_i = \{C_i, C_i, \ldots, C_i, C_i\}$ be a clustering of the nodes of the host graph using feature $f_i$. Since each node is represented by its time series $g_i(v)$, the clustering task is a standard time-series clustering problem. We look for meaningful trends in the evolution patterns of the considered properties by applying EM clustering [125].

In order to make meaningful inferences about how Web sites evolve over time, we want to track the behavior of one host starting from the beginning of its existence. For this reason, we choose to focus on the appearing sites, i.e., the ones that are not present in a first portion of consecutive snapshots, while they are always appearing in the subsequent fraction of consecutive crawls.
We select a subset of 3509 sites that make their first appearance within the dataset in the third month or later. We feel confident in the fact that such hosts were not captured by the crawler before because they were actually not existing. Applying the truncations that are needed to obtain fixed-length time series, we introduce a sliding time window of seven months, associating every node with feature time series of seven points.

We cluster the evolution patterns obtained for the selected set of hosts with respect to the different measures computed.

Figure 4.4: Clustering results for in-degree.

Figures 4.4, 4.5, 4.6 present the results. The plots show mean value and standard deviation of the considered properties over the seven-month time window, with a data point for each month.

In all the experiments, we notice that a major part of the sites (80-90%) are collected into clusters characterized by the fact that the mean value of the analyzed property does not change significantly during the period in which we observe data.

However, there is always a non negligible fraction of sites for which the mean value of the feature considered for clustering increases considerably with respect to the initial value, growing by a factor in $[2, 10]$ (and much larger for
4.6. Host evolution over time

In the cases of in-degree and PageRank, around (5−6\%) of the sites belong to a cluster for which the mean value of the specified property decreases over time.

4.6.2 Using ODP as external source of quality information

We use data from the Open Directory Project to have external information about the quality of the sites. We downloaded the RDF dump of the ODP database available at June 29th, 2007 and we extracted the list of .uk sites mentioned in it. The 35.9\% of the sites within the selected collection appear in ODP. We consider the set of clusters related to each of the above mentioned properties and we compute, for each cluster the percentage of hosts that are mentioned in the ODP directory. The natural assumption, here, is that being mentioned in the ODP directory is a sign of the good quality of a site. The table 4.9 presents the results : for what concerns the supporters at distance three, we observe that cluster 1, which is the largest one (it collects about 80\% of the dataset) and the most stable with respect to the variations of the
property over the seven months, has a percentage of ODP sites that is exactly equal to the expected value (35.9%). Cluster 2 and cluster 3, which collect about 5% of the sites in the dataset, exhibit the most significant growth of the mean value of the specified property. The percentage of sites that are mentioned in the ODP data is lower than the expected value for both these clusters (21% and 19%). Cluster 4, for which we observe a growth in the number of supporters that is still significant, though not as large as the one of the above mentioned clusters, comprises a large fraction of ODP sites. The results we got for pagerank look quite similar to the ones presented above: the clusters for which the average pagerank keeps stable during the seven months are characterized by a percentage of ODP sites that is close to the expected value, while such a percentage is lower in the clusters for which we observe a fast growth of the mean value, like cluster 8 (25% of ODP sites). Cluster 4 is characterized by an increasing trend and 44% of its hosts are mentioned in ODP. In the case of the indegree clusters, there is a low percentage of ODP sites in the clusters for which we observed a very fast growth, like cluster 1 and cluster 9 (about 15% of sites in ODP), while such a percentage becomes larger than the expected value in the clusters that show a decreasing trend,
like cluster 3 and cluster 6, that respectively have 45% and 42% of their sites being mentioned in the ODP data.

4.6.3 Correlations among different properties

In order to investigate the existence of a correlation among the clusters obtained for the various features studied, we consider all the pairs given by two clusters related to two distinct properties, and for each such pair we compute the \textit{lift coefficient}, which represents the probability that any Web site in the dataset appears in both the clusters belonging to the specified pair, in relation to the probability that the site independently appears in the two clusters. Let $N$ be the number of sites in the collection. Given two features $f_i$ and $f_j$, the lift coefficient for $C_{i,k} \in C_i = \{C_{i,1}, C_{i,2}, \ldots, C_{i,l_i}\}$ and $C_{j,h} \in C_j = \{C_{j,1}, C_{j,2}, \ldots, C_{j,l_j}\}$ is expressed as follows:

$$L(<C_{i,k}, C_{j,h}>) = \frac{|C_{i,k} \cap C_{j,h}|}{\frac{|C_{i,k}|}{N} \frac{|C_{j,h}|}{N}}.$$ 

Tables 4.10 and 4.11 show the lift coefficients for the two cases of in-degree versus PageRank and in-degree versus the estimated number of in-degree supporters. The interesting cases are those in which the lift coefficient takes a value greater than 1: this indicates the existence of a positive correlation between the two considered clusters. It is worth to notice that such positive correlation can be observed between the clusters comprising sites for which the average PageRank experiments a considerable increase over time and the clusters collecting the hosts characterized by the most significant growth of the average value of in-degree. A positive correlation also exists between the clusters collecting sites for which the time series of in-degree and PageRank exhibit a decreasing trend over time. Similar results are obtained when considering in-degree versus estimated number of in-degree supporters at distance not greater than 3.

4.6.4 Predicting high dinamicity

The experiments conducted to analyze the temporal evolution of .uk Web hosts have led us to discover non negligible fractions of hosts characterized by significant increasing or decreasing trends in the time series of PageRank, in-degree or number of in-degree supporters.

For each feature $f_i$, we now try to figure out whether the Web hosts belonging to the cluster characterized by the most significant growth can be predicted using the knowledge of the values assumed for the same hosts by all the properties considered either in the first month or in the first two months of
their existence. We handle this classification problem using Weka\textsuperscript{9}, a software providing the implementation of a large collection of machine learning tools.

Table 4.6.4 presents the output of the experiment for the case of in-degree supporters, which is the one for which we get the best results. We run the J48 algorithm, using a cost sensitive classifier that is evaluated using different test options. The first technique chosen for building the training set and the test set is \textit{percentage split}: 66\% of the data is used to train the classifier, whereas the remaining part forms the test set. We then apply \textit{cross validation}, which requires the data to be divided into ten approximately equal partitions. Each part is held out in turn and the learning scheme is trained on the remaining nine folds. The overall error estimate is the average of the ten error estimates.

The last technique we use, \textit{bagging}, consists of creating many classifiers and using majority voting for deciding the class which an elements belongs to. The classifiers that use bagging usually perform better than the individual classifiers they are composed of.

For what concerns the error metrics, we use the True Positive (TP) rate, the False Positive (FP) rate, the Precision, given by the proportion of the elements which truly have class x among all those which were classified as class x, and the F-Measure, which is the harmonic mean of precision and recall.

It can be observed that the quality of the results does not change significantly when adding information about the second month.

4.6.5 Correlations among neighbors

We conclude the analysis of the time series computed for various topological features by presenting an experiment that aims at detecting similarities in the temporal evolution of Web sites that are neighbors in the host graph. We investigate whether Web hosts and their neighbors are characterized by analogous trends in their respective feature time series.

We experiment with six features, which are the values assumed by Page-Rank, in-degree and estimated number of in-degree supporters within distance three in the month in which any site makes its first appearance in the data set, plus the \textit{growth} of each feature, expressed as the ratio between the value assumed in the last month and the value in the first month.

In each of these cases, we calculate a measure that aims at capturing the degree of similarity between the behavior of the nodes and the one exhibited by the nodes in their neighborhood. Our goal is to detect whether the Web hosts in the considered collection tend to share common evolution patterns

\textsuperscript{9}www.cs.waikato.ac.nz/ml/weka/
with their neighbors, or if nodes that are neighbors in the host graph show a 
significant diversity in the way they evolve over time.

In order to do so, given a feature $f_k$, we define a disassortativity index $D_k$ that we calculate on the whole graph by summing over every edge the absolute 
vale of the difference between the values assumed by feature $f_k$ at the two 
nodes representing the end-points of the edge.

Formally, the disassortativity index $D_k$ for feature $f_k$ is calculated as fol-

dows:

$$D_k = \sum_{(i,j) \in E} |p_k(i) - p_k(j)|,$$

where

$$p_k(h) = f_k(h,1) \text{ or } p_k(h) = \frac{f_k(h,T_h)}{f_k(h,1)}.$$

In principle, the result obtained for $D_k$ should be small when the values 
assumed for feature $f_k$ by nodes that are neighbors in the graph are close 
to each other. On the contrary, $D_k$ should take a large value if nodes that 
are neighbors in the graph are not experimenting a similar evolution for what 
concerns the property considered.

To evaluate the magnitude of the value assumed by the disassortativity 
index $D_k$ for a feature $f_k$, we compare the result obtained for each property 
against 200 results calculated by using a distinct random permutation of the 
nodes to exchange the values of feature $f_k$ among the vertices in the graph.

For every property taken into consideration, if the value assumed by the 
disassortativity index is considerably smaller than the ones computed after 
randomly permuting the values of the feature among the nodes, then we con-

clude that the Web hosts in the selected collection show an assortative 
behavior for what concerns that particular property, that is, hosts characterized by in-
ncreasing trends in the temporal evolution of the feature analyzed are mostly 
growing together with their neighbors.

Conversely, if the value assumed by the disassortativity index $D_k$ for a 
given feature $f_k$ is significantly larger than the ones computed using the ran-
don permutations, we hypothesize that hosts are not evolving together with 
their neighbors for what concerns the selected property: In other words, we 
conclude that the Web sites in our data collection are disassortative with re-
spect to feature $f_k$.

As an example, histograms in Figures 4.7 and 4.8 show the results obtained 
for in-degree and its growth over time. In the former case, the result obtained 
for the disassortativity index is about $476 \cdot 10^6$. This value is much higher 
than the ones computed using random permutations to exchange the in-degree
values among the nodes: this means that the hosts in our data collection are disassortative at the beginning of their existence, i.e., hosts with high in-degree are mostly connected to sites with low in-degree.

When the in-degree growth is taken into consideration, the value obtained for our index (419 024) is much lower than the ones calculated after exchanging the values of the same feature among the nodes according to a random permutation. This means that the growth of in-degree is assortative, i.e., nodes whose in-degree is growing over time are very likely to grow together.

We obtain similar findings for PageRank and in-degree supporters: also in the case of these properties, Web hosts exhibit a disassortative behavior in the first month of their existence, whereas they look assortative if we consider their temporal evolution with respect to the same features.

4.6.6 Macroscopic dynamics

Following previous works about the dynamics of the macroscopic structure of the Chilean Web and Wikipedia [26, 67], we characterize the connectivity
4.6. Host evolution over time

We notice that the relative size of the components keeps stable over the various snapshots: the size of the CORE is always more than 50% of the whole graph, while the OUT component is between 40% and 45% in every month. We also study the migration of nodes among the different components, computing, for every pair of consecutive snapshots, all the state change probabilities. These probabilities remain basically unaltered over different snapshots. The most important finding here is that CORE and OUT are very stable: the average probability of remaining in the same state is around 70% in both cases.

4.6.7 Birth and death rates

Inspired by previous work [173, 53], we compute several statistics about birth and death rates for Web sites and the link connections between them.

We observe that after one month 44% of the nodes and 50% of the links
4. CHARACTERIZATION OF THE UK WEB

are new. Starting from the third snapshot, the percentages of new nodes and edges are around 15% in each considered month. 50% of the sites in the last snapshot were existing in the first one, while 35% of its links were in the first graph. On average, 60% of the edges that newly appear in every snapshot come from new nodes, i.e., sites that did not exist in the previous month. The remaining 40% is coming from old nodes.
### 4.6. Host evolution over time

<table>
<thead>
<tr>
<th></th>
<th>odp sites</th>
<th>non-odp sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster 0</td>
<td>106 (33.33 %)</td>
<td>212 (66.67 %)</td>
</tr>
<tr>
<td>cluster 1</td>
<td>5 (15.63 %)</td>
<td>27 (84.38 %)</td>
</tr>
<tr>
<td>cluster 2</td>
<td>132 (28.03 %)</td>
<td>339 (71.97 %)</td>
</tr>
<tr>
<td>cluster 3</td>
<td>545 (45.15 %)</td>
<td>662 (54.85 %)</td>
</tr>
<tr>
<td>cluster 4</td>
<td>33 (27.97 %)</td>
<td>85 (72.03 %)</td>
</tr>
<tr>
<td>cluster 5</td>
<td>15 (9.09 %)</td>
<td>150 (90.91 %)</td>
</tr>
<tr>
<td>cluster 6</td>
<td>95 (42.22 %)</td>
<td>130 (57.78 %)</td>
</tr>
<tr>
<td>cluster 7</td>
<td>298 (38.16 %)</td>
<td>483 (61.84 %)</td>
</tr>
<tr>
<td>cluster 8</td>
<td>5 (62.50 %)</td>
<td>3 (37.50 %)</td>
</tr>
<tr>
<td>cluster 9</td>
<td>29 (15.76 %)</td>
<td>155 (84.24 %)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>odp sites</th>
<th>non-odp sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster 0</td>
<td>127 (39.20 %)</td>
<td>197 (60.80 %)</td>
</tr>
<tr>
<td>cluster 1</td>
<td>193 (19.75 %)</td>
<td>329 (33.67 %)</td>
</tr>
<tr>
<td>cluster 2</td>
<td>64 (37.87 %)</td>
<td>105 (62.13 %)</td>
</tr>
<tr>
<td>cluster 3</td>
<td>224 (37.02 %)</td>
<td>381 (62.98 %)</td>
</tr>
<tr>
<td>cluster 4</td>
<td>45 (44.12 %)</td>
<td>57 (55.88 %)</td>
</tr>
<tr>
<td>cluster 5</td>
<td>8 (34.78 %)</td>
<td>15 (65.22 %)</td>
</tr>
<tr>
<td>cluster 6</td>
<td>53 (41.73 %)</td>
<td>74 (58.27 %)</td>
</tr>
<tr>
<td>cluster 7</td>
<td>235 (35.71 %)</td>
<td>423 (64.29 %)</td>
</tr>
<tr>
<td>cluster 8</td>
<td>5 (25.00 %)</td>
<td>15 (75.00 %)</td>
</tr>
<tr>
<td>cluster 9</td>
<td>138 (27.38 %)</td>
<td>366 (72.62 %)</td>
</tr>
<tr>
<td>cluster 10</td>
<td>170 (37.44 %)</td>
<td>284 (62.56 %)</td>
</tr>
<tr>
<td>cluster 11</td>
<td>1 (100.00 %)</td>
<td>0 (0.00 %)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>odp sites</th>
<th>non-odp sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster 0</td>
<td>992 (35.98 %)</td>
<td>1765 (64.02 %)</td>
</tr>
<tr>
<td>cluster 1</td>
<td>21 (21.65 %)</td>
<td>76 (78.35 %)</td>
</tr>
<tr>
<td>cluster 2</td>
<td>11 (19.30 %)</td>
<td>46 (80.70 %)</td>
</tr>
<tr>
<td>cluster 3</td>
<td>239 (40.58 %)</td>
<td>350 (59.42 %)</td>
</tr>
</tbody>
</table>

Table 4.9: Percentage of ODP sites within each cluster
### Table 4.10: Lift coefficients for in-degree and PageRank clusters

<table>
<thead>
<tr>
<th></th>
<th>C_{I0}</th>
<th>C_{I1}</th>
<th>C_{I2}</th>
<th>C_{I3}</th>
<th>C_{I4}</th>
<th>C_{I5}</th>
<th>C_{I6}</th>
<th>C_{I7}</th>
<th>C_{I8}</th>
<th>C_{I9}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_{P0}</td>
<td>0.408</td>
<td>0.338</td>
<td>1.379</td>
<td>1.041</td>
<td>1.101</td>
<td>2.954</td>
<td>2.310</td>
<td>0.208</td>
<td>0</td>
<td>0.883</td>
</tr>
<tr>
<td>C_{P1}</td>
<td>1.265</td>
<td>1.235</td>
<td>0.999</td>
<td>0.937</td>
<td>0.517</td>
<td>0.827</td>
<td>0.750</td>
<td>1.136</td>
<td>0.449</td>
<td>1.132</td>
</tr>
<tr>
<td>C_{P2}</td>
<td>0.326</td>
<td>0.649</td>
<td>0.220</td>
<td>1.737</td>
<td>1.232</td>
<td>0</td>
<td>1.476</td>
<td>0.718</td>
<td>0</td>
<td>0.790</td>
</tr>
<tr>
<td>C_{P3}</td>
<td>1.240</td>
<td>0.906</td>
<td>1.244</td>
<td>0.807</td>
<td>0.541</td>
<td>1.020</td>
<td>0.361</td>
<td>1.365</td>
<td>0.725</td>
<td>0.756</td>
</tr>
<tr>
<td>C_{P4}</td>
<td>1.190</td>
<td>6.450</td>
<td>0.949</td>
<td>0.627</td>
<td>1.166</td>
<td>0</td>
<td>0.153</td>
<td>1.277</td>
<td>8.600</td>
<td>2.617</td>
</tr>
<tr>
<td>C_{P5}</td>
<td>1.919</td>
<td>14.303</td>
<td>0</td>
<td>0.126</td>
<td>0</td>
<td>0.924</td>
<td>0.678</td>
<td>0.391</td>
<td>76.283</td>
<td>5.804</td>
</tr>
<tr>
<td>C_{P6}</td>
<td>0.521</td>
<td>0.863</td>
<td>0.821</td>
<td>0.984</td>
<td>1.170</td>
<td>0.167</td>
<td>1.105</td>
<td>1.450</td>
<td>0</td>
<td>1.051</td>
</tr>
<tr>
<td>C_{P7}</td>
<td>0.671</td>
<td>0.333</td>
<td>0.532</td>
<td>1.467</td>
<td>0.361</td>
<td>0.582</td>
<td>1.066</td>
<td>1.086</td>
<td>0</td>
<td>2.03</td>
</tr>
<tr>
<td>C_{P8}</td>
<td>1.655</td>
<td>0</td>
<td>1.490</td>
<td>0.727</td>
<td>2.974</td>
<td>1.063</td>
<td>0</td>
<td>1.123</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C_{P9}</td>
<td>1.248</td>
<td>0.435</td>
<td>1.419</td>
<td>0.599</td>
<td>3.068</td>
<td>1.350</td>
<td>1.361</td>
<td>0.642</td>
<td>0</td>
<td>1.703</td>
</tr>
<tr>
<td>C_{P10}</td>
<td>1.896</td>
<td>1.932</td>
<td>0.935</td>
<td>0.685</td>
<td>0.262</td>
<td>0.281</td>
<td>0.172</td>
<td>1.445</td>
<td>0</td>
<td>1.806</td>
</tr>
<tr>
<td>C_{P11}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>438.625</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 4.11: Lift coefficients for in-degree and supporters clusters

<table>
<thead>
<tr>
<th>Property</th>
<th>Method</th>
<th>Cost</th>
<th>Validation</th>
<th>TP</th>
<th>FP</th>
<th>Prec.</th>
<th>F</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supp-3-1M</td>
<td>J48</td>
<td>6:1</td>
<td>perc-split</td>
<td>0.952</td>
<td>0.177</td>
<td>0.99</td>
<td>0.971</td>
<td>NORM HIGH</td>
</tr>
<tr>
<td>Supp-3-1M</td>
<td>J48</td>
<td>6:1</td>
<td>cross-val</td>
<td>0.954</td>
<td>0.227</td>
<td>0.989</td>
<td>0.971</td>
<td>NORM HIGH</td>
</tr>
<tr>
<td>Supp-3-1M</td>
<td>J48</td>
<td>6:1</td>
<td>bagg+cv</td>
<td>0.965</td>
<td>0.313</td>
<td>0.984</td>
<td>0.974</td>
<td>NORM HIGH</td>
</tr>
<tr>
<td>Supp-3-2M</td>
<td>J48</td>
<td>6:1</td>
<td>perc-split</td>
<td>0.958</td>
<td>0.194</td>
<td>0.989</td>
<td>0.973</td>
<td>NORM HIGH</td>
</tr>
<tr>
<td>Supp-3-2M</td>
<td>J48</td>
<td>6:1</td>
<td>cross-val</td>
<td>0.959</td>
<td>0.252</td>
<td>0.987</td>
<td>0.973</td>
<td>NORM HIGH</td>
</tr>
<tr>
<td>Supp-3-2M</td>
<td>J48</td>
<td>6:1</td>
<td>bagg+cv</td>
<td>0.968</td>
<td>0.307</td>
<td>0.985</td>
<td>0.976</td>
<td>NORM HIGH</td>
</tr>
</tbody>
</table>

### Table 4.12: Predicting high clusters using data from the first two months
4.7 Page and link dynamics

Inspired by previous work [173], we attempt to quantify the turnover rate of Web pages and links. This is an aspect of potential interest to search-engine designers, who have to deal with the highly dynamic nature of the Web in order to provide users with the most up-to-date results. We believe that a potential limit of the mentioned study is to be found in the fact that it was conducted on 154 Web sites collected by picking up the top-ranked pages from a subset of the topical categories of the Google Directory. Such a dataset cannot be considered a true sample of the Web itself, which is mostly composed by pages that are often not well maintained. Our dataset is characterized by a huge size (133 million pages and 5 billion links) and it has been collected by performing an extensive crawling of a real Web domain. Hence, it is a much more realistic sample of the Web.

In this section we present the results we obtained by computing the same statistics proposed by Ntoulas et al. [173] about birth and death of web pages and links.

4.7.1 Birth rate of Web pages

First of all, we examine how many new pages are created every month. Figure 4.9(a) shows, for each snapshot, the fraction of its pages that are not present in the previous crawls. We notice that the monthly birth rate of web pages varies within the range [17%, 35%]. The average rate is about 30%. Ntoulas et al. observed an average weekly birth rate equal to 8%. If we consider that a 8% of new pages per week corresponds to a 32% of new nodes after one month, we can conclude that our results are aligned with the findings presented by [173].

Ntoulas et al. used the outcome of this experiment to make conjectures about the size of the entire Web, motivating their claims with the fact that they collected exhaustive weekly downloads of 154 popular sites, by taking all the reachable pages in each site with a breadth first search strategy. We cannot make inferences on how fast the whole Web is growing because of our choice of collecting the snapshots using the stopping criterion of reaching 100M pages.

4.7.2 Birth, death and replacement of Web pages

We next quantify how many new pages are created and how many disappear over time. We also measure which fraction of pages is replaced with new pages after a given amount of time. Figure 4.9(b) shows the number of pages that

\[ \text{directory.google.com} \]
are captured in the first snapshot and still remain in the \( n \)-th one, and how many pages from the \( n \)-th crawl do not exist in the first one. The bars are normalized so that the number of pages in the first month is equal to 1. The red bars represent the pages from the first month that are still available in each given snapshot. The green bars represent the pages that exist in a given snapshot, but are not present in the first crawl.

We notice that after one month around 65% of pages existing in the first month are still available. Such a percentage becomes equal to 45% after six months, while 30% of first-month pages also appear in the last crawl. These results suggest that existing pages are replaced by new pages at a rapid rate. Figure 4.9(c) shows a normalized version of the previous plot: the numbers related to each month are now normalized to one. We notice that after six months about 45% of the pages are pages that also appeared in the first snapshot, while 55% are new, in the sense that they did not exist in the first month.

In the last snapshot, i.e., after one year, 35% of the pages that made their first appearance in the first crawl are still available, while 65% were not captured in the first month. These results are aligned to the ones obtained by Ntoulas et al. in [173], which were computed on a dataset composed by popular and well maintained sites. We believe that this similarity in the results is another validation for our data collection and we feel confident in the fact that the time-aware graph is accurate and can be used to study the temporal evolution of the Web.

### 4.7.3 Link structure evolution

The last experiment we present aims at studying how much the link structure changes over time. We analyze the birth and death rates of the hyperlinks. Figure 4.9(d) shows the fraction of edges that are newly created in each month.

We quantify how many new links appear in each month and how many links from the first snapshot are also present in the subsequent snapshots, comparing them against the links that are newly created. Figure 4.9(e) shows the number of links that make their first appearance in each month, normalized with respect to the number of links existing in the first month. The red portion of each bar shows the number of links from the first month that are present in each month, while the green and blue portions of the same bar represent the number of links existing in the given snapshot that are not present in the first month.

In particular, the green portion of the bar represents the new links coming from old pages, i.e., pages that made their first appearance in the first snapshot, while the blue portion corresponds to new links coming from new pages,
i.e., pages that did not exist in the first month. Figure 4.9(f) shows the same results applying a different normalization: the total number of links in each snapshot is normalized to one. Once again, the results we get are aligned to the ones reported in [173]: we observe that our data collection is characterized by a much more dynamic behavior in the evolution of the link structure rather than in the page dynamics. We notice that only 48\% of the links in the first snapshot do exist in the second one, and only 25\% of the first month links are present in the last crawl, that is, are still available after one year.

### 4.8 Content evolution over time

We now analyze how the content of Web pages evolves over time. We characterize content evolution by analyzing the frequency and the degree of change of the content of a page over a series of consecutive snapshots. The amount of change between two consecutive versions of the same page is measured in terms of Jaccard distance between set of shingles extracted from them. We restrict the analysis to the subset composed by the static, non duplicated pages that were successfully downloaded in every snapshot (i.e., they returned a 200 OK HTTP status code). The set of pages obtained is composed by 4,155,170 URLs altogether.

#### 4.8.1 Change frequency distribution

The first experiment we describe aims at quantifying the average change frequency of Web pages. In this context, we adopt the simplest possible definition of change: We just consider any alteration to a page as a change. For each page in the set considered, we determine the average change interval, which is measured as the average time interval between two consecutive occurrences of a change. For instance, if a page was modified twice during the twelve months, then the average change interval of the page is equal to $12/2 = 6$ months. Figure 4.10 shows the distribution of the average change interval. The average change interval is plotted on the horizontal axis, while the vertical axis shows the fraction of pages having average change interval $x$. The bar on the far right, marked $\infty$, represents the fraction of pages that have not changed at all. The large gap between the two rightmost bars in figure 4.10 encloses average change intervals that cannot arise in a 12-month experiment.
Figure 4.9: Turnover rate of Web pages and links: (a) Fraction of new pages between consecutive snapshots; (b) Fraction of pages from the first crawl still existing after \( n \) months (red bars) and new pages (green bars); (c) Normalized fraction of pages from the first crawl still existing after \( n \) months (red bars) and new pages (green bars); (d) Fraction of new edges between consecutive snapshots; (e) Fraction of edges from the first crawl still existing after \( n \) months (red bars), new edges from nodes existing in the first month (green bars), and new edges from new nodes (blue bars); (f) Normalized fraction of edges from the first crawl still existing after \( n \) months (red bars), new edges from nodes existing in the first month (green bars), and new edges from new nodes (blue bars).
Figure 4.10 provides evidence of the fact that most of the pages analyzed do change either very frequently or very rarely. Approximately half of the pages were not modified throughout the whole study. Conversely, 15% of pages were altered every month. These two extremes account for 65% of the pages; the remaining pages are characterized by an average change interval between these two extremes, distributed with a U-shaped structure (indicating that the majority of pages is close to the extremes). The distribution obtained by Ntoulas et al. exhibits a similar U-shaped structure.

### 4.8.2 Measuring approximate degree of change

The simple definition of change considered in the previous experiment can reveal the presence or absence of changes. It is easy to compute, but it ignores the extent of changes. For search-engine designers, quantifying the degree of change is even more important than detecting the presence of a change. Search engines have to carefully optimize the usage of their resources to maximize the accuracy of their local repositories, and, consequently, the quality of the service offered to users. Thus, they have to deal with the crucial issue of detecting the most significant changes, so that they can avoid an expensive and useless recrawling of the pages that were only affected by insignificant changes.
First, we associate each Web page with a MD5 fingerprint that is computed after stripping out HTML markups. Next, we introduce a measure of the degree of change of a Web page. Given $p_i$ and $p_{i+1}$, i.e., two versions of the same page $p$ downloaded in months $i$ and $i+1$ respectively, we first compare their fingerprints: if they are equal, then the content of the page has not changed during the considered time interval.

If this is not the case, we use shingling [61, 64] to derive a measure of the degree of change which the document has gone through. For this task, we represent each page as an ordered sequence of lower case words, which is obtained excluding all HTML tags. From every page in the data set we extract the set of 3-shingles, i.e., the set composed by all the subsequences of three adjacent words appearing in the page.

We then measure the amount of change between two consecutive versions of a page $p$, denoted by $p_i$ and $p_{i+1}$, in terms of the Jaccard distance $J(p_i, p_{i+1})$ between the corresponding sets of shingles.

**Distribution of degree of change.**

For every page appearing in all the snapshots, we compute the degree of change that affected its content between every pair of consecutive crawls collected during the twelve-month taken into consideration. Figure 4.11 plots the distribution of the degree of change.

<table>
<thead>
<tr>
<th># Changes</th>
<th># URLs</th>
<th>%</th>
<th>Avg interval (months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2007697</td>
<td>48.318</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>487668</td>
<td>11.736</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>297991</td>
<td>7.171</td>
<td>5.5</td>
</tr>
<tr>
<td>3</td>
<td>186613</td>
<td>4.491</td>
<td>3.67</td>
</tr>
<tr>
<td>4</td>
<td>150807</td>
<td>3.629</td>
<td>2.75</td>
</tr>
<tr>
<td>5</td>
<td>114983</td>
<td>2.767</td>
<td>2.2</td>
</tr>
<tr>
<td>6</td>
<td>88849</td>
<td>2.138</td>
<td>1.83</td>
</tr>
<tr>
<td>7</td>
<td>87178</td>
<td>2.098</td>
<td>1.57</td>
</tr>
<tr>
<td>8</td>
<td>67568</td>
<td>1.626</td>
<td>1.38</td>
</tr>
<tr>
<td>9</td>
<td>62867</td>
<td>1.513</td>
<td>1.22</td>
</tr>
<tr>
<td>10</td>
<td>75597</td>
<td>1.819</td>
<td>1.1</td>
</tr>
<tr>
<td>11</td>
<td>527352</td>
<td>12.691</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.13: Distribution of the average change intervals of the pages.
4.8. Content evolution over time

Figure 4.11: Distribution of Jaccard distance for all pages.

The horizontal axis represents a given value of the degree of change, as measured in terms of the Jaccard distance $J(p_i, p_{i+1})$ between the sets of shingles associated with two versions $p_i$ and $p_{i+1}$ of a given page $p$. The values of the Jaccard distance on the horizontal axis are quantized in 20 bins. The vertical axis reports the fraction of total changes that assume a given value of our measure. The red bars represent the distribution of degree of changes, whereas the green bars report the cumulative distribution.

We observe that changes are mostly captured in the leftmost side of the plot, which means that they are actually very small. More than 80% of the changes drove to alterations that account for a value of the distance between the old and the new version of a page below 0.15. Moreover, 70% of the changes determine a distance that is less than 0.05: these cases usually correspond to very slight modifications, like the automatic update of a visit counter, a last-updated timestamp or an advertising space.

These results are similar to those obtained by Ntoulas and Cho [173], who measured the degree of change between two versions of a page by means of the cosine distance between their TF-IDF vector representations, and to the findings previously reported by Fetterly et al. [107], who also applied shingling based techniques to evaluate the extent of change in the content of a set of crawled pages.
It is interesting to observe a monotone descending trend in the fraction of changes characterized by higher values of our distance measure. This trend is only interrupted in the rightmost portion of the plot: the last bin, which captures changes that completely modified the content of a page, is bigger than the previous one. Fetters claimed that this phenomenon is due to spam pages, whose content is automatically generated and stuffed with porn keywords.

**Correlation between frequency of change and degree of change.** In the next experiment, we investigate the existence of a correlation between frequency and degree of change; if such correlation existed, it would be possible to estimate the extent of changes to the content of a Web page starting from its update frequency, which is clearly much easier to quantify. We group the Web pages by the average change frequency. The frequency of changes is computed by simply counting any alteration as a change, thus using fingerprints. For each group of documents characterized by a given average change frequency we then compute the average degree of change. The definitions and methods used here are those introduced and applied in the previous experiments.

Figure 4.12: Relation between frequency and degree of change, expressed in terms of Jaccard distance.
4.8. Content evolution over time

Results are shown in figure 4.12: the horizontal axis represents the number of times (between 1 and 11) that a page was modified during the 12 downloads, whereas the vertical axis shows the average degree of change of the pages in each group. The maximum value of the average degree of change is exhibited by those pages that are modified either frequently (far right of the plot) or seldom if ever (far left of the plot).

Instead, we found no discernible trend for the intermediate values of change frequency.

**Relationship between PageRank and degree of change**

We now examine the relationship between degree of change and PageRank of the Web pages. We consider the PageRank score as a measure of the quality of the page, and we test whether the pages that are characterized by the highest PageRank experiment relevant changes in their content. This would be significant information for developing adaptive crawling strategies. Crawling algorithms could decide to assign high priority to pages that were characterized by high values of PageRank in the previous crawls.

To conduct this analysis we take into account the pages with the highest degree of change. We extract from the collection of pages appearing in every snapshot the subset formed by the pages \( p \) such that, \( \forall i \in \{1, 11\}, J(p_i, p_{i+1}) > 0.9 \): these are the pages that were affected by the most significant modifications in every pair of consecutive months. We obtain 279 URLs in total.

![Figure 4.13: Time series of PageRank for the most dynamic pages.](image)

For each such page, we build \((f_1(p), \ldots, f_{12}(p))\) s.t. \( f_i(p) = \frac{PR_i(p)}{PR_1(p)}, \) i.e.,
the time series of the PageRank scores associated with the page in the twelve snapshots, normalized by the PageRank value computed in the first month. We consider normalized score because our goal is that of detecting increasing or decreasing trends in the evolution of PageRank, regardless of its absolute value.

Starting from these feature vectors, we apply EM [125] clustering to investigate the existence of similarities in the PageRank evolution of the documents collected.

<table>
<thead>
<tr>
<th># Cluster</th>
<th># URLs</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>83</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>196</td>
<td>70</td>
</tr>
</tbody>
</table>

Table 4.14: Size of the clusters built according to PageRank time series.

Figure 4.13 shows the results, reporting mean value and standard deviation of the considered property (i.e., the ratio between the PR measured in each month and the PR measured in the first month) over the twelve-month time window. Table 4.14 reports about the sizes of the two clusters. About 30% of the pages with high degree of change exhibit a considerable growth of PR, whose value has almost doubled after 12 months; instead, the remaining portion of pages (i.e., the most of them) is characterized by a less dynamic behavior of PR, which fluctuates around the value assumed in the first month.

<table>
<thead>
<tr>
<th># Changes</th>
<th># URLs</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>53</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>62</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 4.15: Distribution of the examined pages in each cluster extracted from the degree of change time series.

**Pages with the maximum growth of PageRank.** To complete the analysis of the correlation between PageRank and degree of change, we take into account the pages that experiment the most significant growth in the value of PageRank over the twelve months considered. This time we select the 200 documents with the maximum $\frac{PR_{t+2}(p)}{PR_{t}(p)}$. 

For each page $p$ in the sample, we build the time series $J(p_1, p_2), \ldots, J(p_{11}, p_{12})$ of the Jaccard distances between the sets of shingles computed for consecutive versions of the page. We apply clustering following the same approach described before. The goal is now that of detecting similar patterns in the content evolution of pages characterized by the highest growth in their PageRank value.

Figure 4.14: Times series of degree of change for the pages that experiment the maximum growth of PageRank.

Figure 4.14 reports about the clusters identified through the application of EM algorithm. The number of pages in each cluster is shown in table 4.15. About 65% of the pages is affected by constant modifications of small extent, corresponding to an average value of our degree of change measure between 0.1 and 0.2. Another group of pages (25% of the sample) is altered by constant changes of medium extent (i.e., about half of their content is replaced each month). Finally, there is an outlier (10% of the sample) that exhibits a very high degree of change ($0.8$). In short, each month about 90% of the URLs with a high growth of PR undergoes changes of either small extent (about 15% of their content has been replaced) or medium extent (about half of their content has been replaced).
4.8.3 Predictability of degree of change

In this section, we analyze the predictability of degree of change of Web pages. We first investigate whether past degree of change is a good indicator of future degree of change.

**Overall predictability** We define a pair of consecutive months \((i, i + 1)\) as a *gap*, and we test whether the degree of change that characterizes a document in a gap is a good estimator of the amount of change that affects the same page in the subsequent gap. The horizontal axis of figure 4.15 represents the degree of change in the current gap, while the vertical axis shows the degree of change in the following gap.

![Figure 4.15: Predictability of Jaccard distance over time.](image)

For each page \(p\) and for each gap \((i, i + 1)\), we draw the dot \((J(p_i, p_{i+1}), J(p_{i+1}, p_{i+2}))\) in the plot in figure 4.15. The points aligned across the diagonal (i.e., those points that satisfy the equation \(y = x\)) correspond to Web pages with high predictability of degree of change: they undergo changes of the same extent in two consecutive gaps.

We rank the change events according to their straight-line distance from the diagonal \((y = x)\), and we divide them into four groups:
4.8. Content evolution over time

1. **Group A**: Top 80% of pages in terms of proximity to the diagonal;
2. **Group B**: Pages that fall between the top 80% and the top 90%;
3. **Group C**: Pages that fall between the top 90% and the top 95%;
4. **Group D**: All the remaining pages (5%).

The plot uses one color for each group: black for group A, purple for group B, red for group C, and yellow for group D. The majority of changes occurring in a gap can actually be predicted by changes in the previous gap: e.g., the black and purple dots in figure 4.15, which correspond to pages in group A and B, lie in a band between \( y = x + 0.085 \) and \( y = x - 0.085 \); hence, for 90% of the pages in our dataset, given their degree of change in the current month, we can predict their degree of change in the following month with \( \pm 8.5\% \) error.

These results confirm previous findings [107, 173] about the high correlation between past and future extent of change in the content of Web pages.

**Predictability for individual sites.** In this experiment we study the predictability of individual Web sites. Starting from the domains that contain the highest number of pages in our dataset, we sample a small number of domains that respectively exhibit a predictability worse and better than the average:

1. Domains with predictability worse than the average:
   - www.cheapflights.co.uk (22 131 pages)
   - www.excite.co.uk (22 551 pages)
   - www.redhotchilli.co.uk (25 929 pages)
   - near.co.uk (35 054 pages)

2. Domains with predictability better than the average:
   - myweb.tiscali.co.uk (21 715 pages)
   - www.users.zetnet.co.uk (29 114 pages)

We divide the pages belonging to each domain into four groups according to the straight-line distance from the diagonal (as in the previous experiment). The domains that exhibit the highest predictability (myweb.tiscali.co.uk and www.users.zetnet.co.uk) belong to companies that provide free hosting services; the typical user of such sites publishes a small number of Web pages, and then he rarely updates them. Instead, the domains that exhibit the worst predictability provide products, services and news, which are probably updated by an editorial staff. These findings suggest that the predictability of the degree of change can vary significantly from site to site.
Figure 4.16: Predictability of degree of change for individual sites: (a) www.cheapflights.co.uk, (b) www.excite.co.uk, (c) www.redhotchilli.co.uk, (d) near.co.uk, (e) www.myweb.tiscali.co.uk, (f) www.users.zetnet.co.uk.
Predictability and in/out-degree We also investigate the existence of correlation between degree of change and local topological properties, such as in-degree and out-degree of pages in the Web graph. We apply the methodology described below:

1. For each gap \((i, i + 1)\) and for each URL \(u\) in the dataset, compute the average in-degree (out-degree) of \(u\) in month \(i\);
2. Group the dataset into three subsets of the same size:
   - Pages with low in-degree (out-degree);
   - Pages with medium in-degree (out-degree);
   - Pages with high in-degree (out-degree);
3. Plot predictability of degree of change for each subset of pages, dividing every group into four groups (A, B, C, D) according to their straight-line proximity to the diagonal.

The result is shown in figure 4.17: as in-degree and out-degree grow, predictability decreases. The Web pages with the best predictability are those with low out-degree.

4.8.4 Estimating degree of change using change history

Following an idea proposed in [140], we now use the whole preceeding change history of a URL to study the predictability of the future amount of change of its content. In other words, given the change history of a URL up to a certain point, we want to predict if a search engine will need to recrawl that URL in the next month.

Given a threshold \(t\) and the value of the Jaccard distance \(J(p_i, p_{i+1})\) between the sets of shingles extracted for a page \(p\) with respect to the gap \((i, i + 1)\), if \(J(p_i, p_{i+1}) > t\), then we define the corresponding rate of change as high; otherwise, the rate of change is low, meaning that we consider the modification made to the page as not substantial.

We empirically set the threshold \(t\) to 0.1, so that a low rate of change captures both the cases of complete absence of change or slight modifications (e.g. visitor counters and last modified timestamps), while a high rate of change corresponds to more significant changes of content, which might determine the necessity of recrawling.

We analyze all the pages that appear in every snapshot. We associate each page \(p\) with a vector \(J(p_i, p_{i+1}), i \in \{1, \ldots, 11\}\), which is formed by the
Figure 4.17: Predictability of Jaccard distance for the pages characterized by: (a) low in-degree, (b) low out-degree, (c) medium in-degree, (d) medium out-degree, (e) high in-degree, (f) high out-degree.
values of degree of change observed for $p$ in the eleven monthly gaps available in our data collection. The vector represents the change history of page $p$, as it summarizes all the available information about its content evolution over time.

As a first attempt, we measure the correlation between the rate of change of a page $p$ in a given gap $(i, i + 1)$ and its previous change history using the algorithm described below.

- Let $\hat{J}(p_i, p_{i+1}) = J(p_{i-1}, p_i)$;
- Compute the estimated rate of change:
  $$\hat{roc} = \begin{cases} \text{HIGH} & \text{if } \hat{J}(p_i, p_{i+1}) > 0.1 \\ \text{LOW} & \text{otherwise} \end{cases}$$
- Compute the actual rate of change:
  $$roc = \begin{cases} \text{HIGH} & \text{if } J(p_i, p_{i+1}) > 0.1 \\ \text{LOW} & \text{otherwise} \end{cases}$$
- If $\hat{roc} = roc$, then the estimation is correct; otherwise, the estimation is wrong.

| Number of correct estimates | 38,379,729 items (92.366%) |
| Number of wrong estimates   | 3,171,971 items (7.634%)   |

Table 4.16: Results of the naive estimation.

This simple method provides results of very good quality: the estimation is correct for 92.4% of the examined pages, as shown in table 4.16. We also investigated whether this result could be improved by using other sources of information. In the remaining of this section, we describe the various techniques applied.

### 4.8.5 Estimation using neighbors

In a recent work [174], Nunes et al. proposed to estimate the last modification date of a Web document by using information about the neighborhood of the page in the Web graph. Following this idea, we study the correlation between the degree of change experimented by a page $p$ in a given gap $(i, i + 1)$ and the amount of change that characterize its neighbors.

For any Web graph $G_t = (V_t, E_t)$ extracted from the snapshot crawled at time $t$, and for any node $u \in V_t$, we define:
• in-neighborhood: \( I_t(u) = \{ v \in V_t \setminus u | \exists (v, u) \in E_t \} \)

• out-neighborhood: \( O_t(u) = \{ v \in V_t \setminus u | \exists (u, v) \in E_t \} \)

• neighborhood: \( N_t(u) = I_t(u) \cup O_t(u) \)

Let \( i \) be the current month, and let \( p \) be a Web page, represented by node \( u \) in the graph \( G_t = (V_t, E_t) \); We estimate the extent of change characterizing page \( p \) in the time interval \((i, i+1)\) by computing the average degree of change of the (in/out)neighbors of \( p \) in the gap \((i-1, i)\).

Thus we have, depending on the set of neighbors taken into consideration for a given page \( p \):

1. 
\[
\hat{J}(p_i, p_{i+1}) = \frac{\sum_{q \in I_t(u)} J(q_{i-1}, q_i)}{|I_t(u)|}
\]

2. 
\[
\hat{J}(p_i, p_{i+1}) = \frac{\sum_{q \in O_t(u)} J(q_{i-1}, q_i)}{|O_t(u)|}
\]

3. 
\[
\hat{J}(p_i, p_{i+1}) = \frac{\sum_{q \in I_N(u)} J(q_{i-1}, q_i)}{|I_N(u)|}
\]

We now detail the algorithm used for evaluation. The description provided below is made with respect to the case in which the in-neighborhood is used. The plug-in of the other possible definitions is straightforward.

• Let
\[
\hat{J}(p_i, p_{i+1}) = \frac{\sum_{q \in I(p)} J(q_{i-1}, q_i)}{|I(p)|}
\]

• Compute the estimated rate of change:
\[
\hat{roc} = \begin{cases} 
\text{HIGH} & \text{if } \hat{J}(p_i, p_{i+1}) > 0.1 \\
\text{LOW} & \text{otherwise}
\end{cases}
\]

• Compute the actual rate of change:
\[
roc = \begin{cases} 
\text{HIGH} & \text{if } J(p_i, p_{i+1}) > 0.1 \\
\text{LOW} & \text{otherwise}
\end{cases}
\]

• If \( \hat{roc} = roc \) then the estimation is correct, otherwise it is wrong.

We obtain the best results considering only the in-neighborhood of a page; the fraction of correct estimations is 88.6%, as shown in table 4.17. Using information about the neighborhood of a document does not allow us to improve the method discussed in the previous section, which used the past degree of change to predict the future degree of change.
4.8. Content evolution over time

<table>
<thead>
<tr>
<th>Number of correct estimations</th>
<th>36,804,444 items (88.575%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of wrong estimations</td>
<td>4,747,256 items (11.425%)</td>
</tr>
</tbody>
</table>

Table 4.17: Results of the estimation using In-Neighborhood.

4.8.6 Estimation with Bayes’ theorem

The following experiment is inspired by the work by Kim and Lee [140], which proposed methods to estimate the frequency of successfully downloads of a Web page.

Given a page \( p \), we define its modification rate \( MR \) as follows:

\[
MR(p) = \frac{\# \text{ of gaps in which } p\text{'s rate of change has been HIGH}}{\#\text{gaps}}.
\]

For example, if \( p \) was successfully downloaded for 7 months and its rate of change was high only in the second gap (i.e., in every other gap it has been low), then \( MR(p) = 1/6 \). The MR represents how frequently the content of a Web page has undergone substantial modifications. We need to download a page at least twice to compute its modification rate.

Given a page \( p \), we use the history of its changes in the last \( N \) months to predict its rate of change between the current month and the following one. In the following experiment \( N \) is set to 7. The \( MR \) distribution is shown in table 4.18.

<table>
<thead>
<tr>
<th>( MR )</th>
<th>% pages</th>
<th>Change history</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>73.84%</td>
<td>6L</td>
</tr>
<tr>
<td>0.01 - 0.19</td>
<td>10.9%</td>
<td>5L1H</td>
</tr>
<tr>
<td>0.20 - 0.39</td>
<td>3.68%</td>
<td>4L2H</td>
</tr>
<tr>
<td>0.40 - 0.59</td>
<td>2.11%</td>
<td>3L3H</td>
</tr>
<tr>
<td>0.60 - 0.79</td>
<td>1.54%</td>
<td>2L4H</td>
</tr>
<tr>
<td>0.80 - 0.99</td>
<td>1.50%</td>
<td>1L5H</td>
</tr>
<tr>
<td>1</td>
<td>6.42%</td>
<td>6H</td>
</tr>
</tbody>
</table>

Table 4.18: Distribution of the modification rate \( MR \) in the first seven months of observation.

For example, if \( p \) was successfully downloaded for 7 months and its rate of change was high only in the second gap, then we can represent its change history as the sequence LHLLLL, where L stands for a low rate of change in a gap, and H stands for an high rate of change. To compute the distribution in
4. CHARACTERIZATION OF THE UK WEB

Table 4.18, we group the change histories into seven groups, according to the number of times that a page experimented high/low rate of changes, ignoring their order. The previous example belongs to the group with change history 5L1H, i.e. the group of pages that underwent a substantial modification in one out of six gaps of observation, no matter which.

Finally, we assume that the probabilities are distributed equally in each interval; for example, looking at table 4.18, \( P(MR = 0, 11) = 0.109/19 \). Suppose that \( p \) has the following change history: LLLLLH; therefore, \( MR(p) = 1/6 \).

Given a gap, we want to know if \( p \)'s rate of change in the following gap will be high or low, i.e. we want to know the value of \( P(H | 5L1H) \) and \( P(L | 5L1H) \).

Given a page \( p \) and its change history concerning the first seven months of the period taken into consideration, we estimate its rate of change in the subsequent gap applying Bayes’ theorem as follows:

From Bayes’ theorem we derive

\[
P(A | B) = \frac{P(B | A)P(A)}{P(B)} = \frac{P(B \cap A)}{P(B \cap A) + P(B \cap A^C)}
\]

From which we obtain

\[
P(H | 5L1H) = \frac{P(5L2H)}{P(6L1H) + P(5L2H)} = \frac{P(MR = 2/7)}{P(MR = 1/7) + P(MR = 2/7)} = 0.25
\]

and

\[
P(L | 5L1H) = \frac{P(6L1H)}{P(6L1H) + P(5L2H)} = \frac{P(MR = 1/7)}{P(MR = 1/7) + P(MR = 2/7)} = 0.75
\]

We apply the following procedure to estimate the future degree of change of a page \( p \).

- Compute \( P(H | p \text{'s change history}) \) and \( P(L | p \text{'s change history}) \);
- Compute the estimated rate of change:
  \[
r\hat{oc} = \begin{cases} \text{HIGH if } P(H | p \text{'s change history}) > P(L | p \text{'s change history}); \\
  \text{LOW otherwise} \end{cases}
\]
- Compute the actual rate of change:
  \[
  roc = \begin{cases} \text{HIGH if } J_p(7, 8) > 0.1; \\
  \text{LOW otherwise} \end{cases}
\]
4.8. Content evolution over time

<table>
<thead>
<tr>
<th>Number of correct estimations</th>
<th>3,894,337 (93.723%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of wrong estimations</td>
<td>260,833 (6.277%)</td>
</tr>
</tbody>
</table>

Table 4.19: Estimation based on Bayes’ theorem.

- If $\hat{roc} = roc$, then the estimation is correct; otherwise, it is wrong.

The estimation computed applying Bayes’ theorem is correct in 93.7% of cases, as shown in table 4.19.

**Estimation with Bayes’ theorem and page features**

Web server administrators often choose to disable the last modification date field of the document that are modified frequently\(^{11}\). Therefore, we assume that the existence of the Last-Modified field in the HTTP header is an indicator of a less dynamic evolution of the content of a Web page. We use this feature to partition our collection into two sets: the former, which we denote by $LM$, is made of those pages that have the last modification field in their HTTP header, while the latter, which we name $No LM$, comprise the documents that are not provided with such information. 58% of the URLs in our dataset do not have a Last-Modified field: our hypothesis is that the URLs in this group change more frequently than those belonging to the $LM$ subset.

<table>
<thead>
<tr>
<th>$MR$</th>
<th>$LM$</th>
<th>$No LM$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% pages</td>
<td>% pages</td>
</tr>
<tr>
<td>0</td>
<td>84.47</td>
<td>51.53</td>
</tr>
<tr>
<td>0.01 - 0.19</td>
<td>8.97</td>
<td>14.12</td>
</tr>
<tr>
<td>0.20 - 0.39</td>
<td>2.48</td>
<td>4.73</td>
</tr>
<tr>
<td>0.40 - 0.59</td>
<td>0.73</td>
<td>2.14</td>
</tr>
<tr>
<td>0.60 - 0.79</td>
<td>0.7</td>
<td>1.62</td>
</tr>
<tr>
<td>0.80 - 0.99</td>
<td>0.53</td>
<td>2.33</td>
</tr>
<tr>
<td>1</td>
<td>2.09</td>
<td>23.52</td>
</tr>
</tbody>
</table>

Table 4.20: Distribution of the modification rate $MR$ in the first seven months of observation, based on Last-Modified field.

To verify this intuition, we recompute the distribution of the modification rate $MR$ for the two partitions separately. Table 3.5 shows the two $MR$ distributions obtained for the groups $LM$ and $No LM$. Experimental results confirm

our assumption: the majority (84%) of the pages from the first group (LM) did not undergo any substantial modification during the first seven months, and only a small fraction (2%) was affected by significant changes in each crawl. Conversely, the pages from the second group (No LM) really have a more dynamic content: the fraction of pages that were not altered by significant changes during the first seven months is much lower (51%), and almost a quarter of the documents underwent substantial modifications in each gap.

In our last experiment we apply once more the formulae derived from Bayes’ theorem, choosing the MR distribution according to the presence/absence of Last-Modified field in the header of each page. The estimation is correct in 94.4% of the cases, as shown in table 4.21.

<table>
<thead>
<tr>
<th>Number of correct estimations</th>
<th>3922333(94.39%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of wrong estimations</td>
<td>232837(5.60%)</td>
</tr>
</tbody>
</table>

Table 4.21: Estimation based on Bayes’ theorem and Last Modified field.

We also performed a similar experiment, using the content size as the feature: we define large a page whose content is larger than 10KB (53% of the URLs in the dataset), otherwise we define it small (47% of the URLs); the results are slightly worse than those obtained using the Last-Modified field.

4.8.7 Summary

In this section, we have attempted to characterize the temporal evolution of the content of Web pages. We have conducted extensive experiments with the aim of measuring the frequency and the degree of change that affect the pages over consecutive snapshots. We applied several methods and tested ideas that were suggested in recent works about Web characterization [140, 173, 174].

Our results suggest that the majority of Web pages are modified either very rarely or very frequently. In fact, half of the pages in our dataset never changed during a year; instead, about 15% was modified often (at least once per month). The remaining fraction of pages fall within these two extremes. Moreover, those pages that were modified underwent slight modifications: more than 70% of the changes affected at most 5% of the content.

We have searched for a correlation between degree of change and PageRank. Our results indicate that the PR of 70% of pages with high degree of change did not change during one year. Instead, the remaining 30% experimented a remarkable growth of PageRank, that almost doubled in twelve months. Then, we have analyzed those pages that have undergone the largest growth of PageRank; the majority of them (90%) has undergone modifications of either slight or medium extent every month.
The results of the experiments about predictability of degree of change show that there is a significant correlation between the amount of past and future changes: given the degree of change in the current month, we were able to predict the degree of change in the following month with a ±8.5% error for 90% of the pages in our dataset.

Finally, we have tried to predict the future degree of change of a Web page using its change history; our best estimation correctly estimated 94.4% of the selected pages, using Bayes’ Theorem and Last-Modified information.
Chapter 5

Multiple-graph mining for query-log analysis

Query logs of search engines record a huge amount of data about the actions of the users who search for information on the Web. Hence, they contain a wealth of valuable knowledge about the users’ interests and preferences, as well as the implicit feedback that Web searchers provide when they click on the results obtained for their queries.

Extracting meaningful information from the massive data available in query logs is a task of critical importance for improving the services that search engines offer to Web users through the development of applications like query recommendation, user profiling and spam detection.

In this work we propose a general and completely unsupervised methodology for query-log analysis, which consists of aggregating different graph representations of a query log, each exploring a peculiar semantic dimension. The combination is carried out by applying simple graph-mining techniques.

We show that our approach achieves very good performance for two different applications, which are classifying query transitions and recognizing spam queries.
5. MULTIPLE-GRAPH MINING FOR QUERY-LOG ANALYSIS

5.1 Introduction

The Web-search experience is nowadays part of the life of a continuously growing number of people. According to a recent study released by ComScore\(^1\), the global Web-Search Market has increased by 41% in 2009, reaching more than 113 billion searches. Unique users and penetration\(^2\) have experimented an impressive growth in the last decade. The most up-to-date estimations report that Internet is currently used by more than one fourth of the whole world population. Moreover, the highest growth rates have been observed within those segments of the world market where penetration is still very low.

The results of these studies provide clear evidence of the fact that, despite the massive user adoption and the maturity of the services offered by the Web industry, further efforts are still needed to conquer those fractions of the market where the Web experience is newer and penetration less exhaustive.

Commercial search engines attempt to improve the appeal and the usability of their services by offering tools like query recommendation, user profiling and spam detection. They invest considerable amounts of resources in the development of instruments for gathering novel knowledge about the users’ interests and behavior.

In this scenario, query-log analysis is broadly applied to obtain useful insights about the way users refine their queries, and what kind of search strategies they are using to locate the information they need. Recent studies [42, 133, 181] have shown that the wealth of information stored in search logs can be successfully used to build a very accurate characterization of query-reformulation types. This is a key step towards improving the service provided by search engines and towards developing innovative web-search paradigms. However, the most common approaches adopted share a major limitation, which is to be found in the usage of supervised learning, be it as a sole learning mechanism, or combined with classifiers and exact look-up on dictionaries. These editorial resources are very costly, and thus difficult to obtain for specific languages or cultures.

In this work, we study the problem of developing effective approaches for query-log analysis. Namely, we want to develop methods that are simple and at the same time able to deal with huge data volumes. Hence, our approach is completely unsupervised and based on the map/reduce paradigm.

We choose to use graph structures to build compact and navigable representations of the information extracted from query logs. The idea of inferring graphs from query logs has been extensively explored by recent research [27, 39, 89, 206, 42, 97]. All these cited works focus on the analysis

\(^1\)http://www.comscore.com/

\(^2\)http://www.internetworldstats.com/stats.htm
of a single graph, which typically captures only some particular aspects of the interactions between users and search engines. None of these approaches is able to exploit exhaustively the huge amount of hidden information available in the log.

In this work, we propose to analyze query-log data through the joint mining of multiple graphs, as the combination of them is the ultimate wisdom-of-crowds approach. More specifically, our main contributions are listed below.

- We present **CONIUNGE ET IMPERA**, a general framework for query-log analysis, designed to provide an effective support for the execution of many tasks that are relevant from a search-engine point of view. The fundamental building blocks of our framework consist of (i) a collection of graph projections extracted from a query log according to different semantic criteria, and (ii) a set of operations to be used for both the mining and maintenance of the graphs.

- Concerning the choice of the graph representations, we build three graphs that relate queries according to various types of information: common words, common clicked results, session information. The definitions we use are introduced by Baeza-Yates [21].

- The toolbox built for graph analysis includes set operations and more complex graph operations, like extraction of subgraphs, connected components and articulation points. The choice of the operations was driven by the applications we had in mind, which are extensively described. Although interesting results can be obtained by applying operations as simple as set operations, we will show that more complex graph algorithms are needed to exploit all the wealth of information that is included in the graphs.

- The operations are also used for the maintenance of the framework: for example, the union of two graphs can be computed to merge the data extracted from subsequent snapshots of a query log. The extraction of subgraphs can be applied to restrict the analysis to a subset of queries satisfying some particular conditions.

- To demonstrate the practical applicability and usefulness of our method, we analyze a large log sample provided by a commercial search engine and we show how to customize our approach for two different applications, namely, classifying query transitions and recognizing spam queries.

We remark the fact that our work aims at providing a general instrument that can be useful for many different problems. For this reason, we impose no restrictions on the building blocks of our system, and we intentionally leave
room for extensions. For example, more complex graph definitions could be introduced to analyze the semantics behind queries at a finer granularity.

It is also worth to observe that the approach we follow is completely unsupervised: it does not require the usage of any knowledge bases or other expensive editorial resources. This characteristic makes the method very general ad applicable to different data, without any kind of linguistic and culture-specific issues.

The remaining of this chapter is organized as follows. Section 6.2 discusses previous work. In section 5.3, we present the definitions used to extract different graphs from raw logs. Section 5.4 provides some details about the procedures that were applied to build the graphs from the raw data: there were obtained by devising suitable MapReduce jobs that can be executed fast and efficiently in a parallel setting. Section 5.5 introduces the set of instruments that we included in our framework. Various examples of the practical usefulness of the chosen operations are provided. Section 5.6 describes the dataset we used.

Next, we show how to customize the Coniunge et Impera methodology for two practical applications, which are classifying query transitions (see Section 5.7) and detecting spam queries (5.8).

5.2 Related Work

Inferring graphs from query logs. The idea of inferring the graph structure from query-logs has been introduced only recently. The attention on the possibilities offered by using graphs to mine different semantic information implicitly contained in query-logs was captured by the recent work of Baeza-Yates [21]. Here many different types of such graphs, relying on different information, are studied. We report the definitions in Section 5.3.

Click graphs [39, 89, 206] have been extensively used for different purposes. A click graph is an undirected, weighted and labeled bipartite graph. The nodes are partitioned into two sets: the former contains the distinct queries occurring in the query log, while the latter is formed by a set of distinct documents. An edge between a query and a document denotes the fact that the considered query has led some user to click on the document. Every edge has associated a weight denoting the strength of the relation between the query and the document. Inspired by previous works on click graphs, Baeza-Yates and Tiberi [27] propose a novel way to represent queries in a vector space based on a graph derived from the query-click bipartite graph. They also present the first analysis of a graph extracted from a large query log, containing more than 20 million queries. They aim at exploiting the semantic relations that are implicitly captured in the actions of users that submit queries and click
answers. A limit of the work above is given by the fact that the query log concerned a short time interval, thus precluding the possibility of analyzing temporal aspects, as well as the number of users involved in different clicks.

A larger number of meaningful relations could be inferred through a further analysis of the graph structure, and taking into consideration other graphs as well.

Starting from the click graph, Castillo et al. [73] define two alternative graphs, the view graph and the anti-click graph. In the view graph the edge set of the click graph is replaced with the one containing edges that relate a query to the documents whose URL has been viewed in the answer list returned to the user, but not necessarily clicked. The view graph is a generalization of the click graph since each click is also a view. Moreover, a query could produce no click: in such a case, it would not exist in the click graph, but it would be present in the view graph.

The anti-click graph intends to capture the negative feedback that users implicitly give to a top ranked document when they ignore it by clicking on documents ranked below. This graph contains one edge between a query and any of not-clicked top-position documents before the one clicked by the user. In the work mentioned above, the authors use these two graphs to characterize the spamicity of queries and documents, proposing novel syntactic and semantic features for web spam detection.

Boldi et al. [42] introduce the query-flow graph, which is a graph modeling user behavioral patterns and query dependencies. The main focus here is on the phenomenon of the sequentiality of similar queries: intuitively, in this graph a directed edge from one query to another means that the two queries are likely to be part of the same search mission. Any path over the query flow graph may be seen as a searching behavior, whose likelihood is given by the strength of the edges along the path. Besides introducing the query-flow graph, Boldi et al. provide a methodology for constructing such a graph through the mining of query logs. They also show that the graph can be useful in two concrete applications, finding logical sessions and query recommendation.

Similarity or distance between queries. Projecting query-logs on graphs is not the only way to infer semantic relations from implicit user feedback. The most explored approach consists of defining a similarity function between queries. However, this approach has some clear drawbacks. First of all, using a function can make it harder to understand why two queries are similar, and, in some sense, it introduces artificial artifacts that are likely to add noise to data that is already noisy itself. Most of the work that has been done on query similarity is related to query expansion or query clustering.

Raghavan and Sever [182] propose to measure the similarity between two queries through the mining of the differences in the ordering of the documents
retrieved. Due to a matter of scalability, this technique is not of practical use when one has to deal with the whole Web.

Wen et al. [206] cluster similar queries to support recommendations for queries frequently submitted to search engines. They use several notions of query distance, based either on the keywords composing the query text or on the set of common clicked URLs.

Fonseca et al. [110] use association rules to discover related queries. Their approach views the log as a set of transactions, thus failing in discovering the most interesting related queries, which are the ones submitted by different users. It also encounters problems in determining successive query sessions that belong to the same search process.

Baeza-Yates et al. [24, 25] introduce a term-weight vector model to represent queries. In this model, a weight is assigned to each term occurring in the content of the Web pages clicked after a query, depending on the number of occurrences of the query within the log and the number of clicks of the documents which the term appears in.

5.3 Query-log model and definitions

We generate three query-log graphs using the definitions of Word Graph, Session Graph and Click Graph (called Url Cover Graph in the original paper) introduced in [21]. For the sake of completeness, we report such definitions below. In all cases, a node represents a query instance. Each graph represents a different projection of the data stored in the query log.

Word Graph: \((W)\) There exists an undirected edge between two nodes if and only if the sets of words of the corresponding queries have non-empty intersection. The weight assigned to each node is the number of occurrences of the query within the log. Each edge is weighted with the number of terms that the textual representations of the two queries have in common. Figure 5.1 shows a portion of this graph for the query audrey hepburn.

Session Graph: \((S)\) There is a directed edge from one node to another if and only if the two queries were submitted during the same user session and the first one happened before the second. Each node is weighted with the number of sessions which the query appears in, whereas every edge is weighted with the number of sessions that contain both the queries corresponding to its endpoints. Figure 5.2 shows a portion of this graph for the query audrey hepburn.

Click Graph: \((C)\) Two nodes are connected by an edge if and only if the set of clicked answers of the corresponding queries share at least one URL. Each node is weighted with the number of occurrences of the corresponding query.
5.3. Query-log model and definitions

The weight assigned to an edge is given by the cosine similarity of the URL vector representations of the two endpoint nodes (queries). Figure 5.3 shows a portion of this graph for the query *audrey hepburn*.

Baeza-Yates also introduces a second label for the edges in the Click graph. This label, called *type label*, captures information about the relation between the two sets of clicked URLs associated with the queries that represent the endpoints of a node. Specifically, the edges of the Click graph are now classified into three types:

**Type 1: Identical Cover** The two sets of answers are exactly equal. This is in fact an undirected edge, and it implies that the two queries are equivalent.

**Type 2: Strictly complete Cover** The set of clicked answers associated with the source query is strictly contained in the result set of the target query. This is a directed edge, which provides the information that the set of answers of the first query is strictly contained in the set of clicked results associated with the second query. Semantically, this should imply that the first query is more specific than the second one.

**Type 3: Partial Cover** The set of clicked results associated with the two queries have non-empty intersection, but they do not fulfill any of the previous conditions. This is the most frequent case, and it can happen for various reasons, from multi-topical URLs to truly related queries.
5. MULTIPLE-GRAPH MINING FOR QUERY-LOG ANALYSIS

We now introduce the notation used in the rest of the chapter.

A query log graph extracted from the log of a search engine is a graph $G = (V, E, w_V, w_E)$ such that:

- The set of nodes $V$ comprises all the distinct queries appearing in the (filtered) query log. Every query is identified by a set of words or a sentence.
- $E \subseteq V \times V$ is the set of edges. In some cases the edges are directed, while other query log graphs are defined as undirected graphs. An edge is inserted between two nodes to denote the existence of a relation between the corresponding queries, according to the particular criterion considered in the graph definition.
- $w_V : V \rightarrow \mathcal{N}$ is a weighting function that associates every node in a query log graph with a weight whose meaning changes depending on the definition taken into consideration and on the information used to generate the graph.
- $w_E : E \rightarrow \mathcal{R}$ is a weighting function that associates every edge in a query log graph with a weight representing the strength of the relation between the two queries that correspond to the endpoints of the edge.

Figure 5.2: Sample session graph for audrey hepburn.
5.4 Extracting query graphs from raw logs

The raw data we analyzed consisted of a set of annotated sessions; for each session, various kinds of information were reported, like user identifier, the queries submitted by the user with a timestamp, plus additional data about the actions of the user, like the results she clicked on, or the visualization of more result pages other than the first one.

We used Hadoop’s MapReduce to process the data and extract the graphs defined in Section 5.3.

Map Reduce. MapReduce [93] is a programming model introduced by Google that has recently gained great popularity for the processing of data sets of very large scale. This model allows users to write map/reduce components with functional type code. The user first specifies a map function that is used for processing the data; the results are then merged together using a reduce operation. The MapReduce run-time system schedules these components to distributed resources for processing, and at the same time it guarantees a transparent handling of many problems, like parallelization, network communication, and fault tolerance. Typical MapReduce computations are distributed sorting and information retrieval.

The map and reduce primitives are general and have a simple interface. Each receives a sequence of records, and it usually produces records in response. A record consists of a pair $<$ key, value $>$. Input records presented to the

---

3http://hadoop.apache.org/common/docs/current/mapred_tutorial.html
mapper by its caller have no guaranteed order or relationship to one another. The mapper has to create one or more (or perhaps none) records in response to any input record. The records presented to the reducer by its caller are grouped by key, so that all the records with a given key are presented as a package to the reducer. The reducer then examines the packages sequentially through an iterator.

Mappers and reducers are not used independently. A MapReduce job must specify a mapper and a reducer. A job operates on one or more input files distributed across bins and produces an output file that is also distributed across bins. The systems feeds input-file bins to the mapper instances and partitions the mapper instances’ output into bins according to their key. It then feeds each of these intermediate bins to a reducer instance. The reducers produce the output file of the job. A complete algorithm may cascade more jobs to implement a more complex process.

Although a user can employ MapReduce on a single machine (which we did to compute our graphs), MapReduce frameworks are designed to support operations on a cloud of computers. Distributing the computation on a large collection of commodity computers allows the technique to gain a great increase in terms of scalability, by having a large number of independent instantiations of the MapReduce job components on different partitions.

However, MapReduce processes are interesting beyond the cloud as well. Once a problem has been factored in terms of MapReduce primitives, the same primitives may be also useful for streaming computations or on a single computer equipped with a large disk.

In the remaining of this section we present a brief description of the fundamental steps that were necessary to build the graphs we chose to study in this work, i.e., the Word graph, the Click graph and the Session graph. Each step consists of a MapReduce job. The outcome of a job is typically used as the input of the subsequent job.

5.4.1 Word graph

The Word graph connects two queries if they have one or more common words in their text representation. Each query must be associated with a weight corresponding to its number of occurrences in the log, while every edge must be labeled with the number of words shared by the queries that represent the two endpoints of the edge.

Hence, the construction of this graph requires to collect the following information: the number of occurrences of each query in the log (Step 1), the adjacency list of each query (Steps 2,3), and the weight to be associated with every edge (Step4).
5.4. Extracting query graphs from raw logs

• **Step 1:**
  This job is designed to count and aggregate the occurrences of each query in the log. The mapper parses the annotated sessions that constitute our raw input, and, every time that an occurrence of a query \( q \) is encountered, it maps the pair \((q, 1)\). The reducer processes the tuples output by the mapper grouped by their key, which is the query in this specific case: given a set of pairs \((q, 1)\), it simply aggregates the values associated with these pairs, and it returns a pair \((q, \#occurrences)\). Tables 5.1(a) and 5.1(b) show how the mapper and reducer respectively work.

• **Step 2:**
  This job creates an inverted index that maintains, for each query term, the list of the queries having that word in their text representation. The mapper scans the list of unique queries built by the previous job and it splits the text of every query into words, mapping a list of pairs \((word, query-id)\); the reducer collects the pairs that have the same word as key and concatenates the values of such pairs, outputting a new tuple that associates the term considered with all the queries whose text contains that word. Examples are shown in Tables 5.2(a) and 5.2(b).

• **Step 3:**
  The next step uses the information maintained in the inverted index created by the second job to build the adjacency list of each query. The mapper scans the index and it iterates over the list of queries of any term, mapping tuples where the key is a query and the value is the whole list of queries currently considered. The reducer concatenates the partial lists of neighbors that were mapped for any given query and it builds the global adjacency list of the query, which is also cleaned from duplicates and self loops. Examples are provided in Tables 5.3(a) and 5.3(b).

• **Step 4:**
  The computation of the edge weights is very simple, as the text of the queries is the only information that is needed. This task does not require a MapReduce job; instead, it is simply executed by iterating over the edges in the graph. For each edge we compute the size of the intersection of the sets of unique words in the texts of the endpoint queries.

5.4.2 Click graph

The click graph captures a stronger notion of semantic similarity between queries, as it associates queries that have one or more clicked results.
• **Step 1.** We parse the annotated sessions to generate a list of triples (query,url,clicks). The mapper (see Table 5.4(a)) maps tuples where the key is a pair (query,clicked url) and the value a 1; the reducer just aggregates the clicks involving a given pair (query,url) to create the desired list (see Table 5.4(b)).
### 5.4. Extracting query graphs from raw logs

#### Table 5.2: Word Graph: Job 2 (a): Mapper; (b): Reducer.

<table>
<thead>
<tr>
<th>Key: Term</th>
<th>Value: Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>papageno</td>
<td>papageno covent garden</td>
</tr>
<tr>
<td>covent</td>
<td>papageno covent garden</td>
</tr>
<tr>
<td>pepperoni</td>
<td>pepperoni and mushroom pizza</td>
</tr>
<tr>
<td>mushroom</td>
<td>pepperoni and mushroom pizza</td>
</tr>
<tr>
<td>pizza</td>
<td>pepperoni and mushroom pizza</td>
</tr>
<tr>
<td>garden</td>
<td>papageno covent garden</td>
</tr>
<tr>
<td>papageno</td>
<td>papageno magic flute</td>
</tr>
<tr>
<td>magic</td>
<td>papageno magic flute</td>
</tr>
<tr>
<td>flute</td>
<td>papageno magic flute</td>
</tr>
<tr>
<td>meaning</td>
<td>meaning chaconne and passacalia</td>
</tr>
<tr>
<td>passacalia</td>
<td>meaning chaconne and passacalia</td>
</tr>
<tr>
<td>eating</td>
<td>eating pepperoni pregnant</td>
</tr>
<tr>
<td>pepperoni</td>
<td>eating pepperoni pregnant</td>
</tr>
<tr>
<td>pregnant</td>
<td>eating pepperoni pregnant</td>
</tr>
<tr>
<td>chaconne</td>
<td>chaconne violin solo</td>
</tr>
<tr>
<td>violin</td>
<td>chaconne violin solo</td>
</tr>
<tr>
<td>solo</td>
<td>chaconne violin solo</td>
</tr>
<tr>
<td>Key: Term</td>
<td>Value: Query List</td>
</tr>
<tr>
<td>chaconne</td>
<td>chaconne g minor</td>
</tr>
<tr>
<td></td>
<td>chaconne violin solo</td>
</tr>
<tr>
<td></td>
<td>meaning chaconne and passacalia</td>
</tr>
<tr>
<td></td>
<td>pachelbel chaconne f minor</td>
</tr>
<tr>
<td>kadavu</td>
<td>kadavu nagia</td>
</tr>
<tr>
<td></td>
<td>kadavu nagilia</td>
</tr>
<tr>
<td></td>
<td>kadavu papageno</td>
</tr>
<tr>
<td></td>
<td>kadavu resort calicut india</td>
</tr>
<tr>
<td>papagena</td>
<td>papagena burmese cats</td>
</tr>
<tr>
<td></td>
<td>papagena burmese kittens oxon</td>
</tr>
<tr>
<td></td>
<td>papagena fi</td>
</tr>
<tr>
<td></td>
<td>papagena papagena</td>
</tr>
<tr>
<td>papageno</td>
<td>kadavu papagena</td>
</tr>
<tr>
<td></td>
<td>papagena bar london</td>
</tr>
<tr>
<td></td>
<td>papageno covent garden</td>
</tr>
<tr>
<td></td>
<td>papageno english translation</td>
</tr>
<tr>
<td></td>
<td>papageno magic flute</td>
</tr>
<tr>
<td></td>
<td>papageno meciately translation</td>
</tr>
<tr>
<td></td>
<td>papageno papagena</td>
</tr>
<tr>
<td></td>
<td>papageno restaurant</td>
</tr>
<tr>
<td>pepperoni</td>
<td>eating pepperoni pregnant</td>
</tr>
<tr>
<td></td>
<td>fat pepperoni</td>
</tr>
<tr>
<td></td>
<td>pepperoni and mushroom pizza</td>
</tr>
<tr>
<td></td>
<td>pepperoni restaurant game</td>
</tr>
<tr>
<td></td>
<td>pepperoni schiedam</td>
</tr>
<tr>
<td>stradivari</td>
<td>antonio stradivari</td>
</tr>
<tr>
<td></td>
<td>antonio stradivari drawings</td>
</tr>
<tr>
<td></td>
<td>stradivari accident</td>
</tr>
<tr>
<td></td>
<td>stradivari and stradivari violins</td>
</tr>
</tbody>
</table>

- **Step 2.**

The output of the previous step is used to create a query-url inverted index, which associates each query with the list of URLs that have been clicked for that query. In this case the mapper is just an empty stub, which creates output records that are equal to the input records. This identity process is just used to satisfy architectural needs. The reducer (see Table 5.5) concatenates all the values in tuples that have a given query as their key, outputting the list of URLs that were clicked for the query.
MULTIPLE-GRAPH MINING FOR QUERY-LOG ANALYSIS

Table 5.3: Word Graph: Job 3 (a): Mapper; (b): Reducer.

- **Step 3:**
The third step actually does not depend from the output of the previous one. This means that the two jobs can be run in parallel. The map reads the same lists as before, that is, the list of click events, and maps the pairs (url, query). The reducer concatenates the values associated with a given key, and output tuples of the format (url, queryList), creating a url-query inverted index. See examples in Tables 5.6(a) and 5.6(b).

- **Step 4:**
The next job reads the url-query inverted index to create the list of successors of each query. All the queries that share a given url in their list of clicked answers must be mutually connected in the click graph: hence, the mapper iterates over this list of queries and for each query maps a tuple containing the whole list as its values. The reducer combines the partial lists associated with a given query to get the complete adjacency list of the query, which is also cleaned from duplicates and self loops. Examples are presented in Table 5.7(a) and Table 5.7(b).

- **Step 5. (Computing the weights:**) For each edge we compute the cosine similarity between the vectors of URLs of the two queries that correspond to the endpoints of the edge. We also compute the type label of every edge, following the definition reported in Section 5.3.
### 5.4 Extracting query graphs from raw logs

#### 5.4.3 Session graph

The format of the input data makes the computation of the Session graph quite easy to perform. An edge going from a source query to a target query represents the fact that the two queries were submitted consecutively to the search engine during one or more sessions. Every node (edge) is weighted with the number of distinct user sessions containing the query (transition).

Table 5.4: Click Graph: (a): Mapper; (b): Reducer.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th># Clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>adidas leather gazelle</td>
<td><a href="http://www.shoeclassics.com/page22.html">www.shoeclassics.com/page22.html</a></td>
<td>1</td>
</tr>
<tr>
<td>adopte un mec</td>
<td><a href="http://www.adopteunmec.com">www.adopteunmec.com</a></td>
<td>1</td>
</tr>
<tr>
<td>adidas leather gazelle</td>
<td><a href="http://www.trainerstation.com/adidas-gazelle-ibiza.html">www.trainerstation.com/adidas-gazelle-ibiza.html</a></td>
<td>1</td>
</tr>
<tr>
<td>adtrader</td>
<td><a href="http://www.adtrader.co.uk">www.adtrader.co.uk</a></td>
<td>1</td>
</tr>
<tr>
<td>admiral benbow skiathos</td>
<td>skiathosinfo.com/business/benbow.htm</td>
<td>1</td>
</tr>
<tr>
<td>adopte un mec</td>
<td><a href="http://www.adopteunmec.com">www.adopteunmec.com</a></td>
<td>1</td>
</tr>
<tr>
<td>adopting pet uk</td>
<td><a href="http://www.adog.co.uk/adopt-a-pet/adopt-a-pet-map.html">www.adog.co.uk/adopt-a-pet/adopt-a-pet-map.html</a></td>
<td>1</td>
</tr>
<tr>
<td>adtrader</td>
<td><a href="http://www.adtrader.co.uk">www.adtrader.co.uk</a></td>
<td>1</td>
</tr>
<tr>
<td>adrienne rich</td>
<td><a href="http://www.poets.org/poet.php/prmPID/49">www.poets.org/poet.php/prmPID/49</a></td>
<td>1</td>
</tr>
<tr>
<td>adtrader</td>
<td><a href="http://www.adtrader.co.uk">www.adtrader.co.uk</a></td>
<td>1</td>
</tr>
<tr>
<td>album music charte</td>
<td>uk.launch.yahoo.com/c/uk/album_charts.html</td>
<td>1</td>
</tr>
<tr>
<td>adtrader</td>
<td><a href="http://www.adtrader.co.uk">www.adtrader.co.uk</a></td>
<td>1</td>
</tr>
<tr>
<td>album music charte</td>
<td><a href="http://www.ukalbumchart.co.uk">www.ukalbumchart.co.uk</a></td>
<td>1</td>
</tr>
<tr>
<td>adtrader</td>
<td><a href="http://www.adtrader.co.uk">www.adtrader.co.uk</a></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th># Clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>adidas leather gazelle</td>
<td><a href="http://www.shoeclassics.com/page22.html">www.shoeclassics.com/page22.html</a></td>
<td>1</td>
</tr>
<tr>
<td>adidas leather gazelle</td>
<td><a href="http://www.trainerstation.com/adidas-gazelle-ibiza.html">www.trainerstation.com/adidas-gazelle-ibiza.html</a></td>
<td>1</td>
</tr>
<tr>
<td>admiral benbow skiathos</td>
<td>skiathosinfo.com/business/benbow.htm</td>
<td>1</td>
</tr>
<tr>
<td>adopte un mec</td>
<td><a href="http://www.adopteunmec.com">www.adopteunmec.com</a></td>
<td>2</td>
</tr>
<tr>
<td>adopting pet uk</td>
<td><a href="http://www.adog.co.uk/adopt-a-pet/adopt-a-pet-map.html">www.adog.co.uk/adopt-a-pet/adopt-a-pet-map.html</a></td>
<td>1</td>
</tr>
<tr>
<td>adrienne rich</td>
<td><a href="http://www.poets.org/poet.php/prmPID/49">www.poets.org/poet.php/prmPID/49</a></td>
<td>1</td>
</tr>
<tr>
<td>adtrader</td>
<td><a href="http://www.adtrader.co.uk">www.adtrader.co.uk</a></td>
<td>5</td>
</tr>
<tr>
<td>album music charte</td>
<td>uk.launch.yahoo.com/c/uk/album_charts.html</td>
<td>1</td>
</tr>
<tr>
<td>album music charte</td>
<td><a href="http://www.ukalbumchart.co.uk">www.ukalbumchart.co.uk</a></td>
<td>1</td>
</tr>
</tbody>
</table>
### 5. MULTIPLE-GRAPH MINING FOR QUERY-LOG ANALYSIS

#### Table 5.5: Click graph: Job 2.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>album music charte</td>
<td>uk.launch.yahoo.com/c/uk/album_charts.html 1</td>
</tr>
<tr>
<td></td>
<td><a href="http://www.ukalbumchart.co.uk">www.ukalbumchart.co.uk</a> 1</td>
</tr>
<tr>
<td>adidas leather gazelle</td>
<td><a href="http://www.shoeclassics.com/page22.html">www.shoeclassics.com/page22.html</a> 1</td>
</tr>
<tr>
<td></td>
<td><a href="http://www.trainerstation.com/adidas-gazelle-ibiza.html">www.trainerstation.com/adidas-gazelle-ibiza.html</a> 1</td>
</tr>
</tbody>
</table>

#### Table 5.6: Click graph, Job 3. (a): Mapper; (b): Reducer.

<table>
<thead>
<tr>
<th>Url</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.emigratenz.org">www.emigratenz.org</a></td>
<td>emigrate new zealand</td>
</tr>
<tr>
<td><a href="http://www.emigratenz.org">www.emigratenz.org</a></td>
<td>emigratenz org</td>
</tr>
<tr>
<td>en.wikipedia.org/wiki/Bullitt</td>
<td>bullitt cars</td>
</tr>
<tr>
<td><a href="http://www.emigratenz.org">www.emigratenz.org</a></td>
<td>new zealand immigration</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/stirling/stirling/index.html">www.undiscoveredscotland.co.uk/stirling/stirling/index.html</a></td>
<td>stirling bridge</td>
</tr>
<tr>
<td>en.wikipedia.org/wiki/Bullitt</td>
<td>bullitt</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/standrews/standrews/index.html">www.undiscoveredscotland.co.uk/standrews/standrews/index.html</a></td>
<td>st andrews</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/standrews/standrews/index.html">www.undiscoveredscotland.co.uk/standrews/standrews/index.html</a></td>
<td>stirling</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/standrews/standrews/index.html">www.undiscoveredscotland.co.uk/standrews/standrews/index.html</a></td>
<td>st andrews scotland</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/stirling/stirling/index.html">www.undiscoveredscotland.co.uk/stirling/stirling/index.html</a></td>
<td>street map stirling</td>
</tr>
<tr>
<td><a href="http://www.emigratenz.org">www.emigratenz.org</a></td>
<td>emigrating newzealand</td>
</tr>
<tr>
<td><a href="http://www.emigratenz.org">www.emigratenz.org</a></td>
<td>new zealand emigrate</td>
</tr>
<tr>
<td>en.wikipedia.org/wiki/Bullitt</td>
<td>bullitt</td>
</tr>
<tr>
<td><a href="http://www.emigratenz.org">www.emigratenz.org</a></td>
<td>emigrate new zealand</td>
</tr>
<tr>
<td><a href="http://www.emigratenz.org">www.emigratenz.org</a></td>
<td>emigrate nz</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/standrews/standrews/index.html">www.undiscoveredscotland.co.uk/standrews/standrews/index.html</a></td>
<td>new zealand immigration</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/standrews/standrews/index.html">www.undiscoveredscotland.co.uk/standrews/standrews/index.html</a></td>
<td>new zealand emigrate</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/stirling/stirling/index.html">www.undiscoveredscotland.co.uk/stirling/stirling/index.html</a></td>
<td>new zealand immigration</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/stirling/stirling/index.html">www.undiscoveredscotland.co.uk/stirling/stirling/index.html</a></td>
<td>st andrews scotland</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/stirling/stirling/index.html">www.undiscoveredscotland.co.uk/stirling/stirling/index.html</a></td>
<td>stirling bridge</td>
</tr>
<tr>
<td><a href="http://www.undiscoveredscotland.co.uk/stirling/stirling/index.html">www.undiscoveredscotland.co.uk/stirling/stirling/index.html</a></td>
<td>street map stirling</td>
</tr>
</tbody>
</table>
• **Step 1:**
The first step produces the node weights for the Session graph: it reads the list of annotated sessions and computes the number of sessions which every query appears in. This number can not be larger than the number of occurrences of the query in the log, but it can be smaller for those queries that appear more than once in the same session. The reducer consists of a simple counting process that sums the number of sessions which every query was found in. Tables 5.8(a) and 5.8(b) show a few examples.

• **Step 2:** The second job can be run independently from the previous one. It creates the edges of the Session graph, and it also computes the edge weights (the number of times that a given transition appeared in the log). The mapper maps each occurrence of an edge together with a value 1, and the reducer counts the occurrences of a given edge and output a list of triples \((source, dest, weight)\). See examples in Tables 5.9(a) and 5.9(b).
### Table 5.7: Click graph: Job 4. Creates the query-url inverted index. (a): Mapper; (b): Reducer.

<table>
<thead>
<tr>
<th>Key: Query</th>
<th>Value: Url List</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Key: Query</th>
<th>Value: Url</th>
</tr>
</thead>
</table>
### 5.4. Extracting query graphs from raw logs

#### Table 5.8: Session Graph: Job 1. Creates the node weights. (a): Mapper; (b): Reducer.

<table>
<thead>
<tr>
<th>Key: Query</th>
<th>Value: # Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbey</td>
<td>1</td>
</tr>
<tr>
<td>abby bank</td>
<td>1</td>
</tr>
<tr>
<td>abbey</td>
<td>1</td>
</tr>
<tr>
<td>abbey business banking</td>
<td>1</td>
</tr>
<tr>
<td>abby bank</td>
<td>1</td>
</tr>
<tr>
<td>aberdeen industrial doctors</td>
<td>1</td>
</tr>
<tr>
<td>abby bank</td>
<td>1</td>
</tr>
<tr>
<td>abc trust</td>
<td>1</td>
</tr>
<tr>
<td>abby bank</td>
<td>1</td>
</tr>
<tr>
<td>abby</td>
<td>1</td>
</tr>
<tr>
<td>abby bank</td>
<td>1</td>
</tr>
<tr>
<td>abs cbn interactive</td>
<td>1</td>
</tr>
<tr>
<td>abs cbn interactive</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Table 5.9: Session graph: Job 2. Creates the edges. (a): Mapper; (b): Reducer.

<table>
<thead>
<tr>
<th>Source Query</th>
<th>Target Query</th>
<th># Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>alliance and leicester</td>
<td>compare mortgages</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>compare savings</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>connells estate agents</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>compare travel insurance</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>compare mortgages</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>computer active</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>confetti</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>connells estate agents</td>
<td>1</td>
</tr>
<tr>
<td>accurate time</td>
<td>greenwich mean time</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>connells estate agents</td>
<td>1</td>
</tr>
<tr>
<td>accurate armour</td>
<td>hannants</td>
<td>1</td>
</tr>
<tr>
<td>accurate armour</td>
<td>historex</td>
<td>1</td>
</tr>
<tr>
<td>accurate armour</td>
<td>roman armour</td>
<td>1</td>
</tr>
<tr>
<td>accurate armour</td>
<td>time uk</td>
<td>1</td>
</tr>
<tr>
<td>accurate armour</td>
<td>google</td>
<td>1</td>
</tr>
<tr>
<td>accurate time</td>
<td>time</td>
<td>1</td>
</tr>
<tr>
<td>accurate time</td>
<td>ntl webmail</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>connells estate agents</td>
<td>1</td>
</tr>
<tr>
<td>accurate time</td>
<td>time</td>
<td>1</td>
</tr>
</tbody>
</table>

---

5.4. Extracting query graphs from raw logs

#### Table 5.8: Session Graph: Job 1. Creates the node weights. (a): Mapper; (b): Reducer.

<table>
<thead>
<tr>
<th>Key: Query</th>
<th>Value: # Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbey</td>
<td>3</td>
</tr>
<tr>
<td>abbey business banking</td>
<td>1</td>
</tr>
<tr>
<td>abby bank</td>
<td>5</td>
</tr>
<tr>
<td>abc trust</td>
<td>1</td>
</tr>
<tr>
<td>aberdeen industrial doctors</td>
<td>1</td>
</tr>
<tr>
<td>abs cbn interactive</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Table 5.9: Session graph: Job 2. Creates the edges. (a): Mapper; (b): Reducer.

<table>
<thead>
<tr>
<th>Source Query</th>
<th>Target Query</th>
<th># Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>alliance and leicester</td>
<td>compare mortgages</td>
<td>2</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>compare savings</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>compare travel insurance</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>computer active</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>confetti</td>
<td>1</td>
</tr>
<tr>
<td>alliance and leicester</td>
<td>connells estate agents</td>
<td>4</td>
</tr>
<tr>
<td>accurate armour</td>
<td>hannants</td>
<td>1</td>
</tr>
<tr>
<td>accurate armour</td>
<td>historex</td>
<td>1</td>
</tr>
<tr>
<td>accurate armour</td>
<td>roman armour</td>
<td>3</td>
</tr>
<tr>
<td>accurate time</td>
<td>google</td>
<td>1</td>
</tr>
<tr>
<td>accurate time</td>
<td>greenwich mean time</td>
<td>1</td>
</tr>
<tr>
<td>accurate time</td>
<td>ntl webmail</td>
<td>1</td>
</tr>
<tr>
<td>accurate time</td>
<td>time</td>
<td>1</td>
</tr>
</tbody>
</table>

---


5.5 **Software framework**

In this section, we describe the algorithms that we chose to include in our framework. We adopted an application-driven approach, and we conducted a careful study to identify the operations that were necessary for the purposes of

- Data mining, which includes analyzing and aggregating the available data to derive meaningful information for relevant applications;

- Maintenance of the framework, which includes extending and/or updating our framework, for example, by aggregating the graphs extracted from different snapshots of the query log.

We identified a collection of crucial tools, which includes set operations and graph algorithms, like graph traversal, connected components and articulation points.

A recent work [127], proposes a formal algebra that manipulates graphs as basic units of information, maintaining the same basic characteristics and the same expressive power as the relational algebra. We chose not to follow a similar approach because our needs require to handle complex graph problems in their full generality, and these problems can not be efficiently represented and studied by means of relational algebra.

In the following, we describe the operations that were selected and implemented. For each operation, we will show examples of its practical relevance.

The presentation we give here is intentionally informal and descriptive. The list should by no means be considered exhaustive. Different operations might be needed for different applications.

### 5.5.1 Operations

We use both *binary* operations and *unary* operations.

**Binary operations.** The binary operations are the fundamental set operations that can be applied on graphs: union, intersection, difference. The definitions of these operations are immediate.

- **Union.** Given two query-log graphs $G$ and $H$, their union is represented by a graph $F = G \cup H$ such that $V(F) = V(G) \cup V(H)$ and $E(F) = E(G) \cup E(H)$.

**Example.** The union of two graphs is a critical operation for the maintenance of our framework: it is necessary to merge data extracted from
different snapshots of the log. Updates might be needed from time to
time to regenerate to graphs, given that the knowledge extracted from
query logs suffers a slow aging effect, as demonstrated in ??.

• **Difference.** The set difference of two graphs \( G \) and \( H \) is a graph
\( F = G \setminus H \) such that \( V(F) = V(G) \setminus V(H) \) and \( E(F) = E(G) \setminus E(H) \).

**Example: Similar queries.** Similar queries \([27, 42]\) are queries that
express closely related information needs, even if they don’t look sim-
ilar from a syntactic point of view, for example because their textual
representations contain different words. Identifying such queries is very
important for the task of generating query recommendations \([25]\).

In the *Coniunge et Impera* framework, a very simple solution for the
problem of detecting similar queries consists of computing the set differ-
ence between the Click Graph and the Word Graph: the edges belonging
to the result set connect queries that have common clicked results, but
no common words in their textual representations. Furthermore, a fil-
ter can be applied on the edge weights, to select the most semantically
significant relations.

<table>
<thead>
<tr>
<th>First Query</th>
<th>Second Query</th>
<th>Type of relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>gladular fever</td>
<td>glandular feaver</td>
<td>Error correction</td>
</tr>
<tr>
<td>keeley hazel</td>
<td>keely hazell</td>
<td>Error correction</td>
</tr>
<tr>
<td>keira knightly</td>
<td>kiera knightley</td>
<td>Error correction</td>
</tr>
<tr>
<td>nursery ryhmes</td>
<td>nursery rhymes</td>
<td>Error correction</td>
</tr>
<tr>
<td>patrick swayzee</td>
<td>patrick swayze</td>
<td>Error correction</td>
</tr>
<tr>
<td>duty free allowances</td>
<td>notice 700 vat guide</td>
<td>Rephrasing</td>
</tr>
<tr>
<td>notice 700 vat guide</td>
<td>uk customs charges</td>
<td>Rephrasing</td>
</tr>
<tr>
<td>government gateway  gov uk</td>
<td>notice 700 vat guide</td>
<td>Rephrasing</td>
</tr>
<tr>
<td>excel reader</td>
<td>xls file</td>
<td>Rephrasing</td>
</tr>
</tbody>
</table>

Table 5.10: Examples of queries representing similar information needs.

Table 5.10 presents a sample of the outcomes obtained with the method
described above. The method detects query reformulations that are
basically of two types: error correction or rephrasing.
**Intersection.** The intersection of two graphs $G$ and $H$ is a graph $F = G \cap H$ such that $V(F) = V(G) \cap V(H)$ and $E(F) = E(G) \cap E(H)$.

**Example: Logical session finding.** Another interesting task is that of identifying changes in the *search mission*, that is, in the information need [42, 133] expressed by a user within a given session. Automatic task segmentation is a critical issue for modeling user behavior and predicting user satisfaction. To detect different search missions within a given user session, we compute the set difference between the Session graph and the intersection between the Click Graph and the Word Graph. In such a way, we select queries that have been submitted to the search engine within the same user session, but have no clicked results in common and no common words in the respective textual representations.

<table>
<thead>
<tr>
<th>First Query</th>
<th>Second Query</th>
<th>Type of transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>02 arena</td>
<td>anti spyware</td>
<td>Different mission</td>
</tr>
<tr>
<td>02 arena</td>
<td>art attack</td>
<td>Different mission</td>
</tr>
<tr>
<td>audi a3</td>
<td>gucci watches</td>
<td>Different mission</td>
</tr>
<tr>
<td>audi a3</td>
<td>heathrow terminal 5 news</td>
<td>Different mission</td>
</tr>
<tr>
<td>air hostess</td>
<td>free horoscopes</td>
<td>Different mission</td>
</tr>
<tr>
<td>air hostess</td>
<td>daily newspapers</td>
<td>Different mission</td>
</tr>
</tbody>
</table>

Table 5.11: Examples of transitions within a user session.

Some examples are reported in Table 5.11.

**Note.** The set operations can to be applied on graphs that have weights both on the nodes and on the edges. We impose no restrictions on the scheme to be used for combining the weights, because the best choice clearly depends on the particular task that one has in mind.

**Unary operations.** We consider the following unary operations:

- **Node filter.** Given a query-log graph $G$ and a mathematical condition $c$, this operation returns a query-graph $H$ corresponding to the induced subgraph of $G$ whose vertex set is formed by all the vertices in $V(G)$ such that the weight associated with the node satisfies $c$.

- **Edge filter.** Given a query-log graph $G$ and an mathematical condition $c$, this operation returns the (non induced) subgraph of $G$ whose vertex set includes all the vertices in $V(G)$, while the set of edges is formed by
the edges in $E(G)$ associated with a weight that satisfies the specified condition.

**Example.** We used the filter operations in the preliminary phases of our study, when we extensively analyzed the structural properties of the graphs extracted from the log. We tested various thresholds on the weights of nodes and edges, to study the differences between the resulting denser or sparser versions of the same graph.

In general, the filter operations can be widely used to select portions of the graphs that are relevant for a particular application. For example, consider the application of a filter on the edges of the Click graph: when looking for similar queries, we retain the edges that have a large weight, which indicates a considerable similarity between the result sets associated with the two queries. For example, see the methods designed for detecting error corrections in Section 5.7.

Next, consider the task of identifying *spam queries*, i.e., queries that attract a large fraction of spam pages in their result set. The heuristic that we developed for this application, which is presented in Section 5.8, uses as a starting point an opposite edge filter on the Click graph. The idea here is that a low weight is a sign of a relation of very poor quality between the queries connected by the edge. Thus, the clicked results shared by such queries are usually spam or multi-topical URLs. The queries containing these URLs in their set of answers are included into a base set of spam candidates. See Section 5.8 for more details.

- **Connected components.** This operation takes as input an undirected query-log graph $G$ and returns a set $S = \{G_1, G_2, \ldots, G_k\}$ of query-log graphs of the same type, such that, for each $i = 1, 2, \ldots, k$ $G_i$ is isomorphic to a maximal connected subgraph of $G$.

**Example: Clustering queries.** The computation of the connected components in the Session graph and/or in the Click graph is required for the task of clustering queries, which is useful for many purposes, such as reranking, query recommendation, logical session finding.

- **Biconnected components.** This operation takes as input an undirected query-log graph $G$ and returns a set $S = \{G_1, G_2, \ldots, G_k\}$ of query-log graphs such that, for each $i = 1, 2, \ldots, k$ $G$ is isomorphic to a maximal biconnected subgraph of $G$.

- **Articulation points.** Given an undirected query-log graph $G$, this operation returns a set of query instances
$S = \{q_1, q_2, \ldots, q_k\}$ such that $S \subseteq V(G)$ and, for $i = 1, 2, \ldots, k$ $q_i$ is an articulation point in $G$. An articulation point may belong to one or more biconnected components.

**Example: Polysemic queries.** Polysemic queries [189, 180] are related by a syntactic point of view (for instance, they do share one or more common words in their text representations), but they cover different, unrelated semantic aspects. Articulation points in the Click graph and/or in the Word graph are natural candidates for polysemy.

### 5.5.2 Graph representation

We generated the graphs running the MapReduce routines described in Section 5.4 on a server with 3G RAM. We needed 2 hours to generate the Session graph and 4 hours to create the Word graph and the Click graph. The time required for the Session graph was lower because the particular format of the input data made the extraction of the information required for this graph immediate. Altogether, the time used for generating all the graphs is less than half a day for our sample log.

The operations included in our framework are complex and procedural by nature; they require global computations that involve all the nodes in a graph. This consideration oriented us towards the choice of compressed graph representations to store and manipulate our data.

Compressed graph representation has become an attractive research topic because of its applications in the manipulation of graphs in main memory. Many modern applications have storage requirements that exceed the capacity of the faster memories. The development and usage of compressed data structures allows to overcome this limitation.

In the area of Web search, compressed data structures for graphs have gained a great interest because a graph is a natural model for the Web structure. Several algorithms that are used by search engines to rank pages, discover communities and so on are run on these Web graphs. The graphs are typically huge and maintaining them in memory for navigation purposes may be challenging.

For graph representation, we use Webgraph [49], a framework that obtains state-of-the-art results in terms of compression through the combination of several mechanisms, such as node reordering, differential encoding, compact interval representations, and references to similar adjacency lists.

Webgraph provides algorithms for accessing a graph without actually decompressing it, using lazy techniques that delay the decompression until it is really necessary. The package also contains several tools that allow to modify, symmetrize, transpose and recompress a graph.
5.5. Software framework

5.5.3 Implementation

The choice of WebGraph for graph representation allows us to represent the data extracted from query logs by means of very compact graphs, which can be efficiently kept in main memory for navigation purposes. Exploiting this characteristic, we developed main-memory implementations of our algorithms, which we make freely available \(^4\).

In the following, we sketch the implementation of the most important operations included in our framework. Our realizations are all based on standard algorithms. All the operations require time linear in the size of the input graphs. In our experiments, we reported running times in the order of minutes for all the algorithms implemented.

**Set operations.** Implementing the set operations requires to identify the nodes that will be part of the result set. The graphs that we analyze in this work were built restricting to the same set of nodes, but our general solution also works for unaligned graphs. To handle the problem of tracking nodes that correspond to the same queries across different graphs, we use efficient implementations \(^5\) of minimal perfect hash functions to map nodes to queries and queries to nodes, and to perform membership searches in constant time. The computation of the edge set of the resulting graph can then be performed in time linear in the size of the two graphs.

**Filter Operations** The filter operations can be easily performed by visiting the graph and selecting the nodes and or the edges that satisfy the specified constraint. The time complexity of these operations is linear in the size of the input graph.

**Connected Components** For the computation of connected components, we implement a single algorithm based on BFS, which takes time linear in the size of the graph.

**Biconnected Components and Articulation Points** The algorithm we implement for the computation of biconnected components and articulation points is based on DFS. In a DFS tree of an undirected graph, a node \(u\) is an articulation point if, for every child \(v\) of \(u\), there is no back edge from \(v\) to a node higher than \(u\) within the DFS tree. This means that every node in the tree rooted at \(u\) has no way to visit other nodes without passing through \(u\).

The algorithms needs to maintain a stack to trace back the recursive calls. When we process an edge \((u, x)\), either by a recursive call on \(x\) from \(u\), or because \((u, x)\) is a back edge, we push that edge onto the stack. If we later identify \(u\) as an articulation point, then all the edges from the top of the stack

\(^4\)http://www.dis.uniroma1.it/~bordino/coniunge-et-impera

\(^5\)http://dsiutils.dsi.unimi.it/
down to \((u, x)\) are the edges of one biconnected component. Hence, we pop edges out of the stack until the top of the stack is \((u, x)\).

The worst-case time complexity of this algorithm is \(O(n + m)\), if \(n\) is the number of nodes in the input graph, and \(m\) is the number of edges. The algorithm needs to keep in main memory not only the whole graph, but also the stack that is used to perform the recursive calls on the edges.

**MapReduce implementations.** Besides main-memory implementations, we explored other possible realizations for our algorithms. In particular, given the fact that cloud computing has recently gained great popularity for the processing of huge volumes of data, we developed MapReduce implementations of the operations included in our framework. We decomposed each graph operation into a series of MapReduce processes, adopting an approach similar to the one applied in [81].

The considered operations would be easy and natural to implement if one could afford graph traversal, but graph traversing in MapReduce is very inefficient, because a mapper only reads a random record for each map operation, thus causing the need for a large number of iterations. The common pattern used by our algorithms consists of reading and mapping the nodes together with their adjacency lists, plus some information about nodes and/or edges. The reducer combines the records associated with a node computing the necessary updates.

The obtained implementations are still inefficient, because they require many iterations, not to mention the fact that the initialization of every job is very expensive. To make graph algorithms really feasible, the MapReduce model must be enriched with easier support for iterative computations and graph traversing. J.Ekanayake [5] has very recently developed \(i\)-MapReduce, a streaming-based MapReduce runtime environment that supports iterative MapReduce computations efficiently. We plan to explore the usage of this instrument in future work.

### 5.6 Dataset

We studied a sample log of early 2008 from the Yahoo! UK search engine. The raw data consisted of a sequence of annotated sessions, containing more than 20 million distinct queries.

#### 5.6.1 Word graph, Click graph, Session graph

In principle, the three graphs that we take into consideration rely on definitions that impose different constraints and lead to selecting non homogeneous sets
5.6. Dataset

<table>
<thead>
<tr>
<th>Graph</th>
<th># Nodes</th>
<th># Arcs</th>
<th># Isolated nodes</th>
<th># CC</th>
<th>Size of max CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word</td>
<td>801 876</td>
<td>565 095 123</td>
<td>121 423</td>
<td>135 942</td>
<td>649 298</td>
</tr>
<tr>
<td>Click</td>
<td>801 876</td>
<td>21 971 751</td>
<td>161 344</td>
<td>195 379</td>
<td>549 217</td>
</tr>
<tr>
<td>Session</td>
<td>801 876</td>
<td>6,654 614</td>
<td>25 657</td>
<td>26 199</td>
<td>775 096</td>
</tr>
</tbody>
</table>

Table 5.12: Basic statistics.

of queries. For uniformity we focused on a common subset of queries in the construction of all graphs.

Following [21] and [42] we applied common filters on both nodes and edges of the graphs, with the aim of reducing noise: for example, we retained only the queries whose text representation consisted of words formed with characters of the English alphabet. We also filtered out the queries with less than 5 occurrences in the log. The final set of nodes selected contains 801 876 queries. Table 5.12 reports statistics about the three graphs we built.

**Basic Characterization.**

For each graph built we analyzed several properties, including degree distributions and connected components. The exploited features exhibit a clear power law behavior.

Figure 5.4(d) shows the results concerning connected components. The largest connected component always collects a major fraction of the nodes in the graph. This huge component comes from multi-topical Web sites [27].

For the Click Graph, we also studied the distribution of the number of clicks per query and the number of clicks per URL.

Figures 5.4(e) and 5.4(f) plot such distributions.

The total number of clicks associated with a query is computed as the sum of the number of times that each URL in the related set of answers has been clicked for the query. The total number of clicks for a given URL is obtained by aggregating the clicks done by the users on the selected URL for all the queries having that result in their clicked set. Once again, both distributions follow a power law.
Figure 5.4: Basic characterization of the three graphs: (a): Query frequency distribution; (b): Degree distribution for the Word graph and the Click graph (undirected); (c): In/Out-degree distribution for the Session Graph; (d): Distributions of the size of connected components; (e): Click-Query distribution; (f): Click-Url distribution.
5.7 Classifying query transitions

We now show how to apply the unsupervised mining of different query-log graphs to the task of classifying query transitions. By transition we mean a pair of queries that a user submitted consecutively to a search engine. The analysis of these connections between queries is extremely important for the purpose of understanding how the users refine their queries to better locate the desired information.

It is well known that the retrieval of information from the Web is a process that requires an iterative interaction between the user and the search engine. Only in half of the cases an information need is satisfied with just a single query [184]. Conversely, a user may have to provide a refined query, because her previous question could not be associated to relevant documents, or it contained errors, or it was either too general or too specific for the purpose that she had in mind. In this case, the queries that the user is entering subsequently are still part of the same search mission: this type of connection is usually referred to as a query reformulation.

Characterizing query reformulation patterns is a task of critical importance for web search engines, as it can provide them with useful insights about users’ interests and behavior. Such a knowledge can be applied to improve the relevance of web search results, or to refine tools for query recommendation.

For what concerns the types of query reformulations that we want to distinguish, inspired by the taxonomy introduced and used in [42, 44], we focus on the task of recognizing the following types of transitions:

- **Error correction**: the user is trying a different spelling or capitalization of a query. E.g., *audry hepburn, audrey hepburn*.

- **Generalization**: the second query is more general than the first one. E.g., *audrey hepburn quotes and audrey hepburn*.

- **Specialization**: the first query is more specific than the second one. E.g., *audrey hepburn films and audrey hepburn breakfast tiffany s*.

- **Parallel move**: the user is modifying her query to search for something related but not equivalent. E.g., *audrey hepburn and sophia loren*.

- **Different mission**: the user is trying to satisfy a completely different information need. E.g., *audrey hepburn and runners world*.
The last two types of query transitions, *Parallel move* and *Different mission*, can in fact be considered as belonging to a broader category, which comprises the transitions that represent a change in the information need expressed by the user. In the work of Jones et al. [133], these two transition types are both labeled as *Different chain*. We adopt the same approach, aggregating Parallel move and Different mission into a more general category, which we call *Different goal*.

In the remainder of this section we describe how to extract and aggregate information from the Word Graph, the Click Graph and the Session Graph to classify query transitions.

### 5.7.1 Unsupervised approach to query-transition classification

Our methodology essentially combines three independent heuristics, which are tailored to labelling non overlapping sets of transitions. This is an extremely advantageous characteristic, which implies that the three building blocks of our method can be either applied in parallel, for example on a grid, or they can be executed consecutively in a sequential setting, from the most to the least aggressive approach, so that the amount of data to be analyzed is drastically reduced at each step. Table 5.13 summarizes the notation used throughout the presentation of the various heuristics.

**Identifying different goals.**

The first method we propose aims at identifying changes in the goal of the *search mission*. A search mission is an information need expressed by the user within a given session: it consists of one or more *search goals*. A search goal is an atomic information need, resulting into one or more queries.

Queries that are expression of the same search goal are very likely to share common words in their text representations and/or common clicked results in the respective sets of answers.

In the Session graph, every (directed) edge represents a transition between two queries that have been submitted subsequently to a search engine. Our method for detecting transitions that identify a change in the search mission of the user is based on computing the set difference between the Session graph and the union of the Click graph and the Word graph. Applying this operation corresponds to filtering the set of edges of the Session graph: all the edges connecting two queries that have common clicked results or common words in their text representations are not included in the result.

Figure 5.5 presents a pseudo-code description of our algorithm for detecting *DifferentGoal* transitions.
### 5.7. Classifying query transitions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = (V_S, E_S)$</td>
<td>Session graph</td>
</tr>
<tr>
<td>$w_{SV}(q)$</td>
<td>Weight of the node $q$ in the Session graph</td>
</tr>
<tr>
<td>$w_{SE}(q_i, q_j)$</td>
<td>Weight of the edge $(q_i, q_j)$ in the Session graph</td>
</tr>
<tr>
<td>indeg$_S(q_i)$</td>
<td>Indegree of node $(q_i)$ in the Session graph</td>
</tr>
<tr>
<td>outdeg$_S(q_i)$</td>
<td>Outdegree of node $(q_i)$ in the Session graph</td>
</tr>
<tr>
<td>$C = (V_C, E_C)$</td>
<td>Click graph</td>
</tr>
<tr>
<td>$w_{CV}(q)$</td>
<td>Weight of the node $q$ in the Click graph</td>
</tr>
<tr>
<td>$w_{CE}(q_i, q_j)$</td>
<td>Weight of the edge $(q_i, q_j)$ in the Click graph</td>
</tr>
<tr>
<td>$t_{CE}(q_i, q_j)$</td>
<td>Type label of the edge $(q_i, q_j)$ in the Click graph</td>
</tr>
<tr>
<td>deg$_C(q_i)$</td>
<td>Degree of node $(q_i)$ in the Click graph</td>
</tr>
<tr>
<td>QC$(q_i)$</td>
<td>Set of clicked results associated with query $q_i$</td>
</tr>
<tr>
<td>$U$</td>
<td>Set of URLs clicked for queries in $V_C$</td>
</tr>
<tr>
<td>$W = (V_W, E_W)$</td>
<td>Word graph</td>
</tr>
<tr>
<td>$w_{WV}(q)$</td>
<td>Weight of the node $q$ in the Word graph</td>
</tr>
<tr>
<td>$w_{WE}(q_i, q_j)$</td>
<td>Weight of the edge $(q_i, q_j)$ in the Word graph</td>
</tr>
<tr>
<td>deg$_W(q_i)$</td>
<td>Degree of node $(q_i)$ in the Word graph</td>
</tr>
<tr>
<td>text$(q)$</td>
<td>Text representation of query $q$</td>
</tr>
<tr>
<td>edit$(q_i, q_j)$</td>
<td>Normalized Levenshtein distance between query $q_i$ and query $q_j$</td>
</tr>
<tr>
<td>lab$(q_i, q_j)$</td>
<td>Label assigned to transition edge $(q_i, q_j)$ by our classification method.</td>
</tr>
</tbody>
</table>

Table 5.13: Notation used throughout the presentation of the heuristics.

#### Error corrections with no common words.

A query transition that does not capture a change in the information need of the user is usually called a *query reformulation* [44]. A common type of query reformulation is the one that represents an *Error correction*: the user submitted two queries subsequently with the clear aim of fixing a typo, usually trying a different spelling or capitalization of a query. The queries involved in a transition labeled as *Error correction* may contain common words, or they may not. The second heuristic we present focuses on this latter case.

The method consists of the following sequence of operations:

**Step 1: Edge filter on the Click Graph.** We start from the Click graph and we filter out the edges whose weight is less than 0.3. In this way, we only retain the top 10% strongest semantic relations, which is useful because in the case of an error correction the search goal remains the same.

**Step 2: Intersection with the Session graph.** We intersect the subset of the Click graph that was computed in the previous step with the Session
DIFFERENT GOAL
Input: $S = (V_S, E_S), C = (V_C, E_C), W = (V_W, E_W)$
Output: Set of labeled transitions $J = \{(q_i, q_j) \in E_S : \text{lab}(q_i, q_j) = \text{DifferentGoal}\}$

1: Compute $I = S \setminus (C \cup W)$
2: Return $I$

Figure 5.5: Algorithm for detecting DifferentGoal transitions.

graph: This is needed to identify query transitions.

**Step 3: Set difference with the Word graph.** We compute the set difference between the subgraph obtained at the previous step and the Word graph: this is required to remove transitions with common words. (These transitions are considered by our third algorithm).

**Step 4: Filter by Click labels.** We refine the partial result obtained in the previous step by selecting the edges whose type label in the Click graph is equal to 2, meaning that the set of results that were clicked for the first query is strictly contained in the set of answers associated with the second query. Upon detecting errors in the text of the query submitted by a user, search engines use to suggest the proper spelling, presenting the top-ranked answers of the correct formulation in the first positions of the result list. This practice makes the above condition on click labels very likely to be satisfied by error-correction transitions.

**Step 4: Filter by edit distance.** Finally, we use Levenshtein distance to isolate error corrections from other cases. We retain only the transitions where the edit distance between the two queries is no greater than 0.2. The threshold was chosen after experimenting with different values (0.2, 0.3, 0.4): As expected, the lower the threshold, the more we are effective in isolating error corrections from other reformulations.

<table>
<thead>
<tr>
<th>Transition type</th>
<th>Edit distance</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Correction</td>
<td>0.2</td>
<td>60.260</td>
<td>2978.532</td>
<td>21.679</td>
<td>87.427</td>
<td>0.735</td>
<td>0.408</td>
<td>0.965</td>
</tr>
<tr>
<td>Error Correction</td>
<td>0.3</td>
<td>76.216</td>
<td>2975.854</td>
<td>47.939</td>
<td>78.145</td>
<td>0.614</td>
<td>0.494</td>
<td>0.860</td>
</tr>
<tr>
<td>Error Correction</td>
<td>0.4</td>
<td>87.597</td>
<td>2969.098</td>
<td>108.475</td>
<td>72.030</td>
<td>0.4367</td>
<td>0.548</td>
<td>0.944</td>
</tr>
</tbody>
</table>

Table 5.14: Tuning edit distance for error corrections.
Figure 5.6 presents our algorithm for detecting error corrections with no common words.

<table>
<thead>
<tr>
<th>ERROR CORRECTIONS WITH NO COMMON WORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> ( S = (V_S, E_S), C = (V_C, E_C), W = (V_W, E_W) )</td>
</tr>
<tr>
<td><strong>Output:</strong> Set of labeled transitions ( J = { (q_i, q_j) \in E_S : \text{lab}(q_i, q_j) = \text{ErrorCorrection} } )</td>
</tr>
</tbody>
</table>

1: Compute \( F = (V_C, E_F) : E_F = \{ (u, w) \in E_C : w(u, w) > 0.3 \} \)

2: Compute \( I = S \land F \)

3: Compute \( H = I \setminus W \)

4: For each \( (q_i, q_j) \in H \)
   - if \( t_{CE}(q_i, q_j) = 2 \) and \( (q_i, q_j) \in E_S \) and \( \text{edit}(q_i, q_j) < 0.2 \)
   - \( \text{lab}(q_i, q_j) = \text{ErrorCorrection} \)
   - insert \( (q_i, q_j) \) into \( J \)

Return \( J \)

Figure 5.6: Algorithm for error corrections with no common words.

**Transitions involving queries with common words.**

The last heuristic we describe aims at analyzing the transitions that involve queries with common words in their text representations.

**Step 1:** Intersection between the Session graph and the Word graph.

This is needed to isolate the set of transitions we want to focus on. Many types of reformulations are characterized by overlapping query texts. The following steps apply some tests to distinguish the various cases.

**Step 2.** The text of the first query is a subset of the text of the second query: then we label as Specialization.

**Step 3.** The text of the first query is a superset of the text of the second query: then we label the transition as Generalization.

**Step 4.** We use edit distance to isolate error corrections.

Figure 5.7 presents the algorithm for classifying transitions that involve queries with common words.
5. MULTIPLE-GRAPH MINING FOR QUERY-LOG ANALYSIS

### TRANSLATIONS INVOLVING QUERIES WITH COMMON WORDS

**Input:** \( S = (V_S, E_S), C = (V_C, E_C), W = (V_W, E_W) \)

**Output:** Set of labeled transitions \( J = \{ (q_i, q_j) \in E_S \} \)

1: Compute \( I = S \land W \)

2: For each \( (q_i, q_j) \in I \)
   - if \( \text{text}(q_i) \in \text{text}(q_j) \) and \( \deg_C(q_j) < \deg_C(q_i) \)
     - then \( \text{lab}(q_i, q_j) = \text{Specialization} \)
   - else if \( \text{text}(q_j) \in \text{text}(q_i) \) and \( \deg_C(q_j) > \deg_C(q_i) \)
     - then \( \text{lab}(q_i, q_j) = \text{Generalization} \)
   - else if \( \text{edit}(q_i, q_j) < 0.2 \)
     - then \( \text{lab}(q_i, q_j) = \text{ErrorCorrection} \)
   - insert \( (q_i, q_j) \) into \( J \)

Return \( J \)

Figure 5.7: Algorithm for transitions between queries with common words.

Table 5.15 shows examples of the results obtained. For each transition, we report the labels assigned by the query-flow graph (QFG) and by our algorithm (CEI) respectively.

#### 5.7.2 Evaluation

We use an annotated query-flow graph [42, 44] extracted from the same data as the ground truth for evaluating the quality of results. This graph contains \( \sim 21M \) nodes and \( \sim 43M \) edges. Restricting to the nodes included in our query graphs, we extract a subgraph that contains \( \sim 3.5M \) edges. The model introduced by Boldi et al. [44] and used to annotate the edges of the query-flow graph is able to produce an automatic classification of query reformulation types with an accuracy as high as 92%. This method represents the state of the art for the task of classifying query transitions. However, we remark the fact that it consists of a supervised approach, based on machine learning from a human-labeled log sample.
5.7. Classifying query transitions

<table>
<thead>
<tr>
<th>Transition</th>
<th>QFG Label</th>
<th>CEI Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Query</td>
<td>Target Query</td>
<td></td>
</tr>
<tr>
<td>brittany ferries</td>
<td>virgin records</td>
<td>Different Chain</td>
</tr>
<tr>
<td>brittany ferries</td>
<td>java download</td>
<td>Different Chain</td>
</tr>
<tr>
<td>tv listings</td>
<td>youtube video</td>
<td>Parallel Move</td>
</tr>
<tr>
<td>norwich airport</td>
<td>monarch flights</td>
<td>Parallel Move</td>
</tr>
<tr>
<td>hyde park</td>
<td>hyde park concerts 2008</td>
<td>Specialization</td>
</tr>
<tr>
<td>football shirts</td>
<td>retro football shirts</td>
<td>Specialization</td>
</tr>
<tr>
<td>bruchetta</td>
<td>bruchetta</td>
<td>Error Correction</td>
</tr>
<tr>
<td>google schollar</td>
<td>google scholar</td>
<td>Error Correction</td>
</tr>
<tr>
<td>led zeppelin</td>
<td>led zeppelin</td>
<td>Error Correction</td>
</tr>
<tr>
<td>home made soups</td>
<td>soups</td>
<td>Generalization</td>
</tr>
<tr>
<td>pakistan cricket news</td>
<td>pakistan news</td>
<td>Generalization</td>
</tr>
</tbody>
</table>

Table 5.15: Example of transitions classified with our method.

<table>
<thead>
<tr>
<th>Transition type</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different Goal</td>
<td>2594560</td>
<td>318125</td>
<td>516501</td>
<td>4970</td>
<td>0.834</td>
<td>0.998</td>
<td>0.848</td>
<td>0.909</td>
</tr>
<tr>
<td>Generalization</td>
<td>58627</td>
<td>3366295</td>
<td>2660</td>
<td>6574</td>
<td>0.956</td>
<td>0.899</td>
<td>0.997</td>
<td>0.927</td>
</tr>
<tr>
<td>Specialization</td>
<td>179774</td>
<td>2827724</td>
<td>19</td>
<td>42639</td>
<td>0.999</td>
<td>0.296</td>
<td>0.878</td>
<td>0.457</td>
</tr>
<tr>
<td>Error Correction</td>
<td>60330</td>
<td>3249459</td>
<td>21685</td>
<td>102682</td>
<td>0.736</td>
<td>0.370</td>
<td>0.964</td>
<td>0.492</td>
</tr>
</tbody>
</table>

Table 5.16: Classification of query transitions: quality of results.

To evaluate the quality of our method, we compare the classification generated by our algorithms with the labels associated to the same transitions in the query-flow graph. We compute True Positives, True Negatives, False Positives, False Negatives, Precision, Recall, Accuracy, and F-Measure, which is the harmonic mean of Precision and Recall, given in Table 5.16.

We report the global evaluation of our methodology in Table 5.16. We remark two main results. The first one is that every part of our methodology relies on merging the information coming from at least two different query graphs. The second one is that the approach is totally unsupervised, and uses operations which are linear in the size of the graphs involved. Moreover, since the subsets of transitions labeled by each heuristic are not overlapping, the number of edges to consider decreases at each step.

Altogether, the labels obtained with our method agreed with the one assigned to the same transitions in the query-flow graph in 84% of the cases.
More specifically, we obtained very good results in the cases of generalization and different goal, for which we measured an accuracy respectively of 99 and 85%. On the other end, we noticed a very low recall in the cases of specialization and error correction. Since the query-flow graph was labeled using a model, we wondered whether this poor accuracy could be induced by errors in the query-flow graph instead that in our methodology.

To investigate this hypothesis, we manually evaluated a uniformly random sample of the transitions for which our method and the one based on the query-flow graph did not agree. We selected 1K transitions labeled as specialization in the query-flow graph, and 1K transitions labeled as error correction. To build the sample, we considered the transitions divided into buckets according to their frequency of occurrence in the log. Three human assessors were asked to evaluate the transitions, assigning them one of the reformulation types considered in this study.

We assumed the manual assessment to be the golden truth for the sample, and we measured how well the two automatic classification methods (ours, and the one using the query-flow graph) agreed with the golden truth. The evaluation gave the following results: the labels produced by our method agreed with the ones created by human assessors in 80.4% of cases, while the query-flow-graph labels resulted equal to the ground truth in 12.8% of cases.

To evaluate the statistical significance of this result, we associated each automatic method with a vector that has a cell for each sampled transition, containing a 1 if the label assigned by the method agreed with the ground truth, and a 0 otherwise. We then performed a Wilcoxon signed-rank test, comparing the positive differences between the judgements obtained for the two methods. The test determined a statistical significance at $p \ll 1\%$ for the difference between our methodology and the query-flow graph.
5.8. A HEURISTIC FOR SPAM QUERIES

### Table 5.18: Evaluating error about specializations: sample transitions.

<table>
<thead>
<tr>
<th>Source Query</th>
<th>Target Query</th>
<th>QFG Label</th>
<th>CEI label</th>
</tr>
</thead>
<tbody>
<tr>
<td>travel</td>
<td>supermarket</td>
<td>Specialization</td>
<td>Different Goal</td>
</tr>
<tr>
<td>thomson</td>
<td>teletext</td>
<td>Specialization</td>
<td>Different Goal</td>
</tr>
<tr>
<td>thomas</td>
<td>cook</td>
<td>Specialization</td>
<td>Different Goal</td>
</tr>
<tr>
<td>marks</td>
<td>spencer</td>
<td>Specialization</td>
<td>Different Goal</td>
</tr>
<tr>
<td>oasis</td>
<td></td>
<td>Specialization</td>
<td>Different Goal</td>
</tr>
<tr>
<td>car hire</td>
<td>edinburgh</td>
<td>Specialization</td>
<td>Generalization</td>
</tr>
<tr>
<td>environment</td>
<td>jobs</td>
<td>Specialization</td>
<td>Generalization</td>
</tr>
<tr>
<td>neighbours</td>
<td>time</td>
<td>Specialization</td>
<td>Generalization</td>
</tr>
<tr>
<td>consumer</td>
<td>protection</td>
<td>Specialization</td>
<td>Generalization</td>
</tr>
<tr>
<td>confused car</td>
<td></td>
<td>Specialization</td>
<td>Generalization</td>
</tr>
<tr>
<td>radioblogclub</td>
<td>radio blog club</td>
<td>Specialization</td>
<td>Error Correction</td>
</tr>
<tr>
<td>mcdonalds</td>
<td>mcdonald s</td>
<td>Specialization</td>
<td>Error Correction</td>
</tr>
<tr>
<td>facebooks</td>
<td>facebook</td>
<td>Specialization</td>
<td>Error Correction</td>
</tr>
<tr>
<td>futur tv</td>
<td>future tv</td>
<td>Specialization</td>
<td>Error Correction</td>
</tr>
<tr>
<td>steve earl</td>
<td>steve earle</td>
<td>Specialization</td>
<td>Error Correction</td>
</tr>
</tbody>
</table>

Hence, we believe that the actual precision of our approach is higher than the one obtained with this experiment, and that the disagreement with the query-flow graph is due to errors in the model used by the latter method to train the classifier.

### 5.8 A heuristic for spam queries

*Spam queries* [73] are queries that generate a high number of spam pages in the top positions of their answers lists. Identifying these queries can uncover meaningful information for the task of designing more robust spam-detection algorithms. We now focus on detecting queries that have collected many spam results. Spam pages typically include many unrelated keywords or links, advertising and machine-generated content. These characteristics make spam pages likely to be selected as answers for very different queries, which may express unrelated or poorly related information needs.

We first use the Click graph to identify candidate spam pages. Given that spam pages often cover a large number of unrelated topics, we follow the approach that was suggested in [27] to identify multitopical URLs. In
the Click graph, edges connecting queries whose sets of clicked results have a low cosine similarity indicate a poor-quality relation between the involved queries. In our experiment, we consider all the click-graph edges whose weight is $< 0.005$; in this way, we isolate the least significant 25% edges in the graph.

We count each edge as a spamicity vote for all the URLs in the intersection of the result sets associated with the two queries. We consider the top 200 URLs that obtain the highest number of votes. All the queries that have such URL in their result set form a clique in the Click graph. We retain the groups of queries that are also mutually connected in the other graphs.

As a result, we obtain a list of 9140 queries. We filter out the queries that have more than 1K occurrences to get rid of navigational queries. We then sort the queries by decreasing degree in the Click graph and we select the top 10% of the queries with highest degree: the intuition is that queries that form small dense subgraphs in the three graphs, while maintaining many connections to other nodes in the graph, are good spam candidates. In the end, we extract a set of 928 queries.

<table>
<thead>
<tr>
<th>Min</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>1,509,071</td>
</tr>
<tr>
<td>Avg</td>
<td>1,249</td>
</tr>
<tr>
<td>Median</td>
<td>60</td>
</tr>
<tr>
<td>First quartile</td>
<td>15</td>
</tr>
<tr>
<td>Third quartile</td>
<td>276</td>
</tr>
<tr>
<td>90 percentile</td>
<td>1,051</td>
</tr>
</tbody>
</table>

Table 5.19: Statistics about the frequency of queries in the log.

<table>
<thead>
<tr>
<th>Min</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>6,742</td>
</tr>
<tr>
<td>Avg</td>
<td>120</td>
</tr>
<tr>
<td>Median</td>
<td>15</td>
</tr>
<tr>
<td>First quartile</td>
<td>4</td>
</tr>
<tr>
<td>Third quartile</td>
<td>56</td>
</tr>
<tr>
<td>90 percentile</td>
<td>213</td>
</tr>
</tbody>
</table>

Table 5.20: Statistics about node degree in the Click graph.
5.8. A heuristic for spam queries

5.8.1 Experimental evaluation

The above method led us to isolate 928 queries. We test the effectiveness of the developed strategy by conducting an experimental evaluation that aims at assessing the spamicity of the selected queries. For each query, we manually evaluate its ability of attracting spam results by resubmitting the query to the Yahoo! search engine and counting how many spam pages it obtains within the top ten results. We mark as spam every query containing at least one spam result in the list of the top answers.

Given that no agreement on a univocal and clear definition of spam page has been reached so far, we choose to enforce the strength of our judgements following the spam labeling guidelines that have been used for the construction of the WEbspam-UK2007\(^6\) collection, which is a Web spam dataset built through the collaborative effort of a team of volunteers to advance research on Web spam detection.

We also match the top results obtained for each query in the sample against the Yahoo! Directory. Table 5.21 shows the most frequent categories associated with the results returned for the queries. Only the top levels are considered: Level 1 is the top category, level 2 the second highest category and so on. The main categories obtained include Shopping, Business, Movies, Sex and Adult galleries.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regional</td>
<td>Shopping and Services</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>Business and Economy</td>
<td>US States</td>
<td>California</td>
</tr>
<tr>
<td>Entertainment</td>
<td>Countries</td>
<td>Apparel</td>
</tr>
<tr>
<td>Arts</td>
<td>Business to Business</td>
<td>Sex</td>
</tr>
<tr>
<td>Society and Culture</td>
<td>Music</td>
<td>Artists</td>
</tr>
<tr>
<td>Recreation</td>
<td>Movies and Film</td>
<td>Titles</td>
</tr>
<tr>
<td>Computers and Internet</td>
<td>Television Shows</td>
<td>Literature</td>
</tr>
<tr>
<td>News and Media</td>
<td>Humanities</td>
<td>Telephones</td>
</tr>
<tr>
<td>Government</td>
<td>Actors</td>
<td>Australia</td>
</tr>
<tr>
<td>Health</td>
<td>Games</td>
<td>Animation</td>
</tr>
</tbody>
</table>

Table 5.21: Top categories obtained from Yahoo! Directory.

\(^6\)http://www.yr-bcn.es/webspam/datasets/uk2007/guidelines/
The results obtained in the assessment phase are really encouraging: two thirds of the evaluated queries collect at least one spam result and are consequently marked as spam queries. Some examples are shown in Table 5.22, which also reports, for each query, the number of spam results, the main category assigned by Yahoo! Directory and the number of different categories associated.

The inspection of the categories obtained for our sample set makes us notice that most of the queries analyzed, even those that are not classified as spam, are associated with (top) results related to adult, celebrities, video or shopping. We believe this is a confirmation of the effectiveness of our strategy in selecting queries that are characterized by a significant degree of spamicity, or, if not spam, by the covering a broad set of topics.

The peculiar nature of the queries discovered heuristic suggests an immediate application of our tools to developing parental filters.
Chapter 6

Query Similarity by
Projecting the Query-Flow Graph

Defining a measure of similarity between queries is an interesting and difficult problem. A reliable query-similarity measure can be used in a variety of applications such as query recommendation, query expansion, and advertising.

In this work [54], we exploit the information present in query logs in order to develop a measure of semantic similarity between queries. Our approach relies on the concept of the query-flow graph. The query-flow graph aggregates query reformulations from many users: nodes in the graph represent queries, and two queries are connected if they are likely to appear as part of the same search goal. Our query-similarity measure is obtained by projecting the graph (or appropriate subgraphs of it) on a low-dimensional Euclidean space. Our experiments show that the measure we obtain captures a notion of semantic similarity between queries and it is useful for diversifying query recommendations.
6. QUERY SIMILARITY BY GRAPH PROJECTIONS

6.1 Introduction

Finding a measure of similarity between queries can be very useful to improve the services provided by search engines. First, the ability to identify similar queries is in the core of any query-recommendation system. Second, query similarity can be used for performing query expansion and enhancing search results. Additionally, a reliable notion of query similarity can be used for broad matching of advertisements to queries, or even for suggesting keywords to advertisers. However, defining a query-similarity measure is not an easy task as it strongly depends on user intent: syntactically similar queries may originate from completely different intents. Issues such as polysemy, synonymy, high levels of noise, and the small amount of available information make the problem challenging.

In such a complex setting, information extracted from query logs has shown to be effective. The information on how users interact with search engines has often been used to improve the user search experience. In particular, query-log analysis is used to provide insights on how users refine their queries, and what kind of search strategies they are using to locate the information they need.

In this work we describe a method of obtaining a query-similarity measure, based on query-log analysis. Our method relies on an aggregated representation of a query log by the means of a reformulation graph, which is known as the query-flow graph [42]. In this graph, nodes represent queries and two queries are connected if they are likely to appear as part of the same search goal [133].

Our main intuition is that related queries will tend to cluster in local neighborhoods of the graph. Graph-projection methods are known to map graph nodes into geometric spaces so that the distance distortion is minimized. Thus, we suggest projecting the query-flow graph (or appropriately defined subgraphs of it) and then measure query similarity on the resulting geometric space. The technique is general and it can be applied to other graphs obtained from query-logs, for example, the click graph [39, 88, 163].

We use the resulting query-similarity measure for diversifying query recommendations. Query-recommendation systems are provided by all major search engines and they aim at helping users to find more easily what they are searching for. The idea is that a diversification algorithm requires a notion of query similarity, for which we use the proposed measure. We show that the resulting system provides diverse yet relevant recommendations.

Our main contributions are summarized as follows:

- we describe a method for measuring similarity between queries by projecting a query reformulation graph;
• we show that our similarity measure captures the human notion of related queries better than other measures on the original graph;

• we apply this method to the task of producing diverse and useful recommendations;

• we show how to improve its efficiency further, by projecting only the neighborhood of the input query.

The rest of this chapter is organized as follows: Section 6.2 introduces related work about this topic, Sections 6.3 and 6.4 describe the framework we use to define our measures of query similarity, and Section 6.5 explores different variants for optimizing our measures. Finally, Section 6.6 describes our application for diversifying query recommendations.

6.2 Related work

Usage mining is a topic that has attracted considerable attention in the last few years; our review should by no means be considered complete.

Query graphs. Graphs might be used to provide a navigable, compact representation of the query-related information extracted by query-logs. Query graphs of different types have been extensively studied in literature [21, 39, 27, 163, 42, 44]. In most of these studies query-logs are represented by query-query graphs, where queries form the set of vertex and edges among queries capture various types of query-information. Baeza-Yates [21] articulates a first query-graph taxonomy, introducing five different types of graphs: word graph, session graph, URL cover graph, and URL link graph. In these graphs, a link is introduced between two queries if they contain the same word(s), they belong to the same session, they share common (viewed or clicked) URLs in the list of their results, there is a link between the two clicked URLs, or there are common terms in the content of the two URLs.

Query-document graphs, also known as click graphs, have also been extensively studied [39]. They are bipartite graphs $G = \{Q, D, E\}$ where $Q$ is the set of queries and $D$ is the set of documents. An edge $e \in E$ appears between a query $q \in Q$ and a document $d \in D$ if the user that issued the query clicked on the URL corresponding to $d$ in the list of results. From a click-graph it is always possible to derive a corresponding query-query graph representation [163].

Tiberi et al. [27] studied several aspects of the cover-graph and the click-graph to infer semantic relations among queries and to detect multi-topical
Another example of query-query graph, also known as Query-Flow Graph (query-flow graph), is introduced in [42]. The query-flow graph differs from the previous graphs as it aggregates different sources of information in order to capture the latent query behavior of users. Each link is labeled with a probability that indicates if the two endpoints of the correspondent link are related. Link labels could also be enriched with query reformulation types (generalization, specialization, error correction, and parallel move) [44]. Query-flow graphs are presented more in detail in Section 6.3.1.

A concept similar to our query flow graph, but in the context of web browsing behavior, is introduced by Levene and Loizou [154]: "Hypertext Probabilistic Automata" are automata where the arcs of the reachability relations are labeled with probabilities that are computed from statistical information related to the frequency that users choose to navigate through two states. However, the work is focused on browsing behavior inside a Web site and not on querying behavior. Borges and Levene later introduce an improved method for measuring the ability of a variable-length Markov model to summarize user Web navigation sessions up to a given length [58].

**Query recommendations.** Most query recommendations methods are based on query similarity measures obtained by mining (i) the query strings themselves, (ii) the clicked documents for the queries, and/or (iii) the user sessions containing the queries. Typically methods use a combination of these factors.

**Query recommendation based on clicked documents.** Baeza-Yates et al. [28] find, given a query, related queries issued by other users and build query expansion methods to construct artificial queries. Their technique is based on a term-weight vector representation of queries, obtained from the aggregation of the term-weight vectors of the URLs clicked after the query. Wen et al. [206] also present a clustering method for query recommendation that is centered around four notions of query distance: the first notion is based on keywords or phrases of the query; the second on string matching of keywords; the third on common clicked URLs; and the fourth on the distance of the clicked documents in some pre-defined hierarchy.

Craswell and Szummer [88] describe a method based on random walks on the query-click graph [39], the graph connecting queries to the documents clicked on the search engine result pages for those queries. They test their method in an application of image search. Fuxman et al. [112] use the query-click graph to find related keywords for advertising. Antonellis et al. [14] also use the query-click graph, and exploit the idea of co-citation through its generalization known as SimRank [131]. Mei, Zhou and Church [163] instead use a computation of hitting time for ranking related queries.
Query reformulation based on query reformulations. Many effective approaches focus on the analysis of user query sessions [110, 213, 134]. Fonseca et al. [110] propose a query recommendation system based on association rules applied to query logs. In their framework a user session (or transition) is defined as the set of queries issued by a single user within an interval $t$. An other attempt of extracting information from the user query behavior is due to Zhang and Nasraoui [213]. In their work, each user session is represented by a complete graph where consecutive queries are connected with an edge of a predefined weight $d$. Not consecutive queries are connected by an edge weighted with the product of the weights on the walk connecting them. Recent works have shown that not only the previous query, but also the long-term interests of users, are important for understanding their information needs [157, 183].

A different kind of application is presented in [134], where the notion of query substitution is introduced: for each query, a set of similar queries is obtained by replacing the whole query or only its sub-phrases. This set of candidates is ranked using a rewriting probability model.

White et al. [208, 209] use the query rewrites observed in a query log to generate query recommendations. Given an input query, they generate a set of candidates containing (a) the top 100 queries that contain the original query as a sub-string, and (b) the top 100 queries which followed the input query. Each candidate query is then scored by multiplying its smoothed overall frequency of following the target query in the past sessions, using Laplacian smoothing.

Sadikov et al. [185] have recently proposed to cluster the refinements of a user query by performing a random walk on a query-document graph that incorporates both session and click information.

6.3 Preliminaries

6.3.1 The query-flow graph

In this section we recall the definition of the query-flow graph. A query-flow graph is a directed graph $G = (V, E, w)$ where:

- $V = Q \cup \{s, t\}$ is the set of distinct queries $Q$ submitted to the search engine plus two special nodes $s$ and $t$, representing a starting state and a terminal state of any user search task;
- $E \subseteq V \times V$ is the set of directed edges;
- $w : E \to (0..1]$ is a weighting function that assigns to every pair of queries $(q, q') \in E$ a weight $w(q, q')$.

In the query-flow graph, two queries $q$ and $q'$ are connected by an edge if there is at least one session of the query log in which $q'$ follows $q$. The weight
$w$ may depend on the application; in the following we simply consider the
weight to be the frequency of the transition in the query log.

The edge probabilities along with other data associated to each transition,
are used to segment physical sessions into chains. Here, physical sessions are
defined as sequences of the activities of a single user before a timeout of 30
minutes, while chains are defined as sequences of activities that are topically
related. This step is important for applications aimed at improving the user
search experience.

6.3.2 Spectral projection

The query-flow graph captures implicit similarity between queries: queries
connected with a high-frequency edge are similar in the sense that they are
motivated by the same user information need. Query logs collected over a
few months are rich in information, but they also contain a lot of noise. Our
approach is motivated by the idea of defining a similarity measure between
queries that takes into account the global structure of the query-flow graph,
instead of taking into account only pairs of queries.

Measuring distances of nodes on large graphs is a well-studied problem,
and many approaches have been proposed, including the shortest-path distance
and the commute time. A drawback of those measures is that they are very
inefficient measures to compute, as their complexity is at least linear to the
number of the nodes in the graph, while one would like to have measures
whose complexity depends only on the nodes under consideration (and possibly
features of the nodes) but not on the whole dataset.

One of the key methods for measuring distances in a graph, which has been
used extensively for visualizing graph data, is the idea of graph projection. The
idea is to project the original graph into a low-dimensional Euclidean space
and then measure distances between graph nodes by considering the distances
of the corresponding projected points. There are many techniques that can
be used to obtain such projections, including multidimensional scaling [148],
spectral projections [79, 146, 172], IsoMap [195], maximum-variance unfold-
ing [193], and many more. In this work we use the spectral projection, which
we briefly describe below:

1. Given a graph $G$ with adjacency matrix $A$, the Laplacian matrix $L_G = D - A$ is computed using the diagonal matrix $D$, whose entry $d_{ii}$ equals
to the degree of the $i$-th node of $G$.

2. An embedding $\phi : V \rightarrow \mathbb{R}^m$ is computed by finding the $(m + 1)$
eigenvectors of $L_G$ that correspond to the smallest eigenvalues. Only $m$ of
these are used as the one corresponding to the smallest eigenvalue is the
vector of all 1's.
This spectral embedding, known as *Fiedler embedding*, has the property of preserving the distances in the projected space. For “near-by” nodes $u$ and $v$ in the graph $G$, the Euclidean distance between the vectors $\phi(u)$ and $\phi(v)$ is small. Details on the properties of spectral embeddings of graphs and spectral algorithms can be found in [79, 146, 172].

For spectral projection, the notion of “near-by” nodes in the graph is related to the expected time of coming across node $v$ in a random walk starting from node $u$. Hence, we expect that queries that are similar are projected to points that are relatively close in the embedding. Regarding the property of maintaining distances, the spectral projection optimizes a global objective function that can be interpreted as the overall distortion of the graph projection.

On the projected space, various distance metrics can be used, for example the Euclidean distance or the cosine distance. In our experiments, we observed that cosine similarity outperforms Euclidean distance, hence, in the rest of the work we focus on cosine similarity. Note, that since the projected vectors may contain negative numbers, the cosine between two vectors can be negative. In order to obtain a measure in $[0, 1]$ we rescale the cosine as follows:

$$\text{Sim}(q, q') = \frac{1 + \cos(q, q')}{2}.$$  

Recently the notion of directed Laplacian was introduced and analyzed by Fan Chung [78]. Since the query-flow graph is a directed graph, using the directed Laplacian is more appropriate for our application scenarios. However, our evaluation showed that the projection based on the directed Laplacian does not yield any improvement.

In the next two sections we describe our experiments with projections of the query-flow graph. Our objective is to explore systematically the space of possible parameters and alternatives for defining appropriate subgraphs to project, in order to optimize the quality of query similarities.

We note that we have not tried to explore the possibility of improving our empirical results using different graph embedding algorithms. Any graph embedding algorithm can be used as a black-box in our method, instead, our main focus has been to leverage the idea that graph projections can yield meaningful notions of query similarity, and we have also experimented extensively with finding the best subgraphs to project.
6.4 Framework for studying query similarity

6.4.1 Dataset

The query-flow graph was built using a set of sessions extracted from a query log of the Yahoo! search engine.

We improve the graph by estimating the probability that both the queries $q$ and $q'$ in a transition belong to the same “search mission” [133] (also known as “query chain” [181]). This modification prunes out transitions to frequent navigational queries such as popular web portals.

Motivated by the results of some preliminary assessments, we apply a number of filters on the graph, such as removing the transitions that have frequency less than 5. We also remove $s$ and $t$ and prune all the nodes that remain isolated: these correspond to sessions composed by singleton queries. Altogether, starting from a graph with 58,312,610 nodes and 131,836,560 edges, we obtain a graph with 4,152,773 nodes and 7,788,232 edges. This is the graph $G$ that we consider in the rest of this work.

6.4.2 Evaluation method

We select 140 queries from the log, sampling queries that are neither too frequent nor too infrequent (torso queries), as head queries tend to be of a particular type (e.g., navigational), while tail queries give information that is too sparse. We focus on single-term queries that are likely to have more than one interpretation. For each of these queries, we build a test set of related queries by selecting a small set of their most frequent successors in the query-flow graph. Each test set is then clustered into 2 to 5 clusters by human editors; the editors decided the number of clusters for each set.\footnote{The clustering of the query test sets was done by four editors. We tested whether our results depend on the subjective perception of the editors about clustering, by repeating our analysis for the queries labeled by each editor separately. We found that in all cases we obtain similar results.} The clustering of a test set represents the golden truth for that set. Table 6.1 shows examples of clustered test sets. We then apply our graph projection method, and we obtain a similarity measure for queries, for which we evaluate its agreement with the human-defined clustering. We compare different variants of graph projections by testing which variants yields similarities that agree better with the golden truth.

Note that to apply our methodology we need to measure agreement between a clustering and a similarity function. We use the following measure:
6.4. Framework for studying query similarity

<table>
<thead>
<tr>
<th>Query</th>
<th>Clusters</th>
</tr>
</thead>
</table>
| **sun** | 1. the sun newspaper, mirror, times  
2. earth, mars, mercury  
3. sun java, sun microsystems |
| **stone** | 1. stone weight, stone measurement  
2. rock, granite  
3. stone brick, stone masonry |
| **spoiler** | 1. movie spoiler, tv show spoiler  
2. car spoiler, custom car spoiler |

Let $V$ be a set and $C = \{C_1, \ldots, C_k\}$, with $V = \bigcup_i C_i$, be a clustering of $V$. For a similarity function $\text{Sim}(q, q')$, we introduce two scores:

**intra-cluster similarity of cluster $C_i$:**

$$\text{InSim}(C_i) = \sum_{q_h, q_j \in C_i, h \neq j} \frac{2 \cdot \text{Sim}(q_h, q_j)}{|C_i| |C_i - 1|}$$

**inter-cluster similarity of cluster $C_i$:**

$$\text{OutSim}(C_i) = \sum_{l=1 \ldots k, l \neq i} \left[ \sum_{q_h \in C_i} \sum_{q_j \in C_l} \frac{2 \cdot \text{Sim}(q_h, q_j)}{|C_i| |C_l|} \right]$$

The intuition is that a similarity measure agrees with clustering $C$ if the InSim score is large compared to the OutSim score. Thus, we capture the quality of a similarity measure, with respect to the clustering $C$, using the ratio

$$M_C(\text{Sim}) = \frac{\mathbb{E}[\text{InSim}(C)]_{C \in \hat{C}}}{\mathbb{E}[\text{OutSim}(C)]_{C \in \hat{C}}}.$$

Given two different similarity measures, the best one is the one that maximizes the measure $M_C(\text{Sim})$.

6.4.3 Sub-graph construction method

We experiment with two alternatives for extracting subgraphs from the query-flow graph: query-dependent and query-independent methods.
**Query-dependent subgraphs.** Given a query $q$ we extract a subgraph around $q$ by a breadth-first search from $q$. For a graph $G = (V, E)$ and two nodes $q, q'$, let $d(q, q')$ be the length of the shortest path from $q$ to $q'$ following directed edges in $E$. Let $V_d(q) = \{q' \in V \text{ such that } d(q, q') \leq d \text{ or } d(q', q) \leq d\}$. For instance, $V_0(q) = \{q\}$ and $V_1(q)$ contains the in-neighbors and out-neighbors of $q$. We define the sets:

- $E_d(q) = \{(q_1, q_2) \in E : q_1 \in V_d(q) \land q_2 \in V_d(q)\}$
- $O_d(q) = \{(q_1, q_2) \in E : q_1 \in V_{d-1}(q) \land q_2 \in V_d(q)\}$
- $I_d(q) = \{(q_1, q_2) \in E : q_1 \in V_d(q) \land q_2 \in V_{d-1}(q)\}$

We experimented with the following subgraphs of $q$:

- $F_d(q) = (V_d(q), E_d(q))$
- $S_d(q) = (V_d(q), E_{d-1}(q) \cup O_d(q) \cup I_d(q))$

Figure 6.1: Example $F_1(q)$ and $S_1(q)$.

Figure 6.1 shows an example. The sizes of the obtained subgraphs vary widely, with $|V_{2(d)}|$ ranging from 31 to 20702 queries (median: 2320 queries).

**Query-independent subgraphs by partitioning.** Query-dependent subgraphs may be expensive to compute at query time, so we also experimented with query-independent subgraphs obtained by partitioning the graph. The partitioning was done using METIS [164] and varying the number of clusters that METIS takes as a parameter.

### 6.5 Optimizing the similarity measure

The objective of our study is twofold: (i) to demonstrate the effectiveness of the query-flow graph projections in order to define a measure able to capture
the human notion of similarity between queries, and \( (ii) \) to optimize such a query-similarity measure.

To address the first objective we define a baseline measure that relies on the query-flow graph but does not use projections. To obtain more refined graphs, and thus similarities of better quality, we apply the projection method locally, on the neighborhood of a given query. This approach is based on constructing query-dependent subgraphs as we discuss in detail in Section 6.4. Applying projections locally yields a better similarity measure, but unfortunately, the method is computationally more expensive since it requires to build a different subgraph for each query. Thus we propose a “hybrid” approach, which builds query-independent subgraphs by using partitions of the query-flow graph obtained with the metis graph-clustering algorithm.

Overall, we perform a large number of experiments to assess the following parameters: \( (i) \) the number of dimensions of the spectral embedding, \( (ii) \) the choice of the weighting scheme, \( (iii) \) the method to be used in the construction of the query-dependent subgraphs, and \( (iv) \) the number of partitions to be considered in the computation of query-independent subgraphs.

### 6.5.1 Similarity without graph projections

Our first similarity measure is purely based on the query-flow graph: the similarity between two queries is given by the cosine similarity of their vectors of neighbors. Given a query \( q \), we denote by \( N(q) \) its vector of neighbors in the query-flow graph.

For our test sets, the average value of \( M_\mathcal{C} \) for this baseline similarity measure is 1.03 with a variance of 0.03. Figure 6.4 reports the results obtained for the queries \( \text{watch} \) and \( \text{time} \); we see little separation between queries in different clusters. These results suggest that the metric based on the cosine similarity of the neighborhoods of two queries does not capture well the similarity of queries. We stress here the fact that methods based on random walks [88] or hitting time [163] are not able to capture the notion of semantic similarity we aim to. In the example of Figure 6.4, the probability to end up in frequent queries like “\textit{rolex watch}” or “\textit{watch free movies online}”, starting from the query “\textit{watch}”, might be quite similar even if such queries are not semantically related.

### 6.5.2 Similarity with graph projections

A quite natural objection to the use of the Fiedler embedding of the query-flow graph is that this projection is intended to be applied on undirected graphs. Surprisingly, preliminary experiments showed that using the directed
Laplacian does not yield any improvement. Hence, we retained the standard method. We assume that similar queries are projected onto points that are close in the embedded Euclidean space. We then measure the distance between two queries in terms of cosine distance between the two corresponding Euclidean vectors. The extensive set of experiments we performed confirms that the distance measure defined by means of the spectral projection of the query-flow graph into a lower-dimensional space captures the notion of similarity among queries better than the one relying on the original graph.

**Choice of dimensions.** In order to investigate how varying the number of dimensions of the spectral projection affects performance, we build three embeddings of $G$, which respectively have $m = 3$, $m = 5$ and $m = 7$ dimensions. To get a more complete picture, we do the same for two types of query dependent subgraphs, $S_2$ and $S_3$. For each of these cases, we use the projections to compute the similarity between all the pairs of queries in every test set collected, and then we compute the measure $M_\hat{C}$ to evaluate agreement with the clusters created by hand. We also perform $t$-tests to figure out whether the differences among the various cases are statistically significant or not.

Our results, reported in Table 6.2, show that increasing the number of dimensions does not determine a considerable gain in terms of quality. A slight improvement in terms of less variance is observed, but the differences between the various cases are not statistically significant. For this reason we fix $m = 5$ in the remainder of our experimentation.

**Choice of weighting scheme.** In the query-flow graph, every edge connecting two queries is weighted with the frequency of the transition in the original query log, and the transition probabilities of the edges can be used to identify queries that represent similar information needs. We perform a number of tests to investigate whether taking the edge weights into account during the computation of the spectral embeddings improves our method.

We experiment with three different weighting functions:
6.5. Optimizing the similarity measure

Table 6.3: Average $\mathcal{M}_C$ obtained for $S_2(q)$, $S_3(q)$ using different weighting schemes.

<table>
<thead>
<tr>
<th>Weighting</th>
<th>Significant differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin.</td>
<td>log</td>
</tr>
<tr>
<td>$S_2$</td>
<td>1.9 1.8 1.3</td>
</tr>
<tr>
<td>$S_3$</td>
<td>2.2 1.9 1.7</td>
</tr>
</tbody>
</table>

- **binary**: $w : E \rightarrow \{0, 1\}$ s.t. $w(q, q') = 1$. This case corresponds to the baseline (unweighted graph).
- **raw-count**: $w : E \rightarrow \mathbb{N}$ s.t. $w(q, q') = c(q, q')$ (c(q, q') is the occurrence count of the transition in the log).
- **log(count)**: $w : E \rightarrow \mathbb{R}$ s.t. $w(q, q') = \log(c(q, q'))$.

When an edge between two queries exists in both directions, we symmetrize the weights choosing the maximum of the two. This step is needed for the computation of the projection. We also tried with minimum, average or sum of the two weights, but we observed no substantial changes.

We compare the above weighting schemes for two types of query-dependent subgraphs: $S_2(q)$ and $S_3(q)$. Table 6.3 shows the results. Binary and log weights yield the best results, whereas the usage of raw counts hurts performance, and the difference between this weighting scheme and the other two is statistically significant. For this reason we discard the raw counts, and we focus on the simplest of the two schemes that exhibit the best behavior, which is the one using binary weights. We consider projections of unweighted graphs in the remainder of this section.

6.5.3 Query-dependent subgraphs

We now study if projecting the neighborhood of a given query improves performance. We consider three types of query-dependent subgraphs: $F_1(q)$, $S_2(q)$, $S_3(q)$. The first is the subgraph induced by the neighbors of the input query, while the other two graphs are obtained by applying two or three steps of a breadth-first search on the query-flow graph, starting from the input query.

We compare the embeddings of the query-dependent subgraphs against the methods discussed before, i.e., projecting the whole graph or using the query-flow graph directly. The results of this study are presented in Figure 6.2 and Table 6.4: $N$ is the baseline, while the other cases represent the methods that project either the full graph $G$ or the corresponding query-dependent
6. QUERY SIMILARITY BY GRAPH PROJECTIONS

Table 6.4: $M_\hat{a}$ for query-dependent subgraphs.

<table>
<thead>
<tr>
<th>Method</th>
<th>Average $\pm$ st.dev</th>
<th>Significant differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>1.02 $\pm$ 0.17</td>
<td>$N$ ***</td>
</tr>
<tr>
<td>$G$</td>
<td>1.22 $\pm$ 0.74</td>
<td>$G$ ***</td>
</tr>
<tr>
<td>$F_1$</td>
<td>1.49 $\pm$ 2.07</td>
<td>$N$ **</td>
</tr>
<tr>
<td>$S_2$</td>
<td>1.63 $\pm$ 0.88</td>
<td>$N$ ***</td>
</tr>
<tr>
<td>$S_3$</td>
<td>1.68 $\pm$ 1.45</td>
<td>$G$ ***</td>
</tr>
</tbody>
</table>

subgraphs. Local projections introduce a relative improvement in the average $M_\hat{a}$ in the range of the 16% – 39%. Even if the variances of local projections are also increasing we can observe that the lower bound for both $S_2$ and $S_3$ match the upper bound of $N$. Also, the differences between the query-dependent subgraphs and the other two approaches are statistically significant (see Table 6.4).

Figure 6.4 reports a qualitative comparison of the above methods for the queries *watch* and *time*.

6.5.4 Query-independent sub-graphs

Using local projections of small query-dependent subgraphs allows to define a metric that captures better the similarity between queries. However, this method has the clear drawback of requiring query time computational processing, which may be expensive. For this reason we investigate whether projecting larger, query-independent subgraphs allows to trade-off performance and computational costs. We generate sets of query-independent subgraphs using the METIS algorithm, which partitions the nodes of a graph into balanced clusters minimizing the number of edges in the cut. The number of clusters to be created is chosen by the user. In the following, we briefly describe how we use a partition of the nodes of the query-flow graph created with METIS to derive query-independent subgraphs. We then compute the spectral embeddings of these subgraphs.

Cluster expansion. We first partition the graph into 100 or 200 clusters. We choose these values because they yield cluster sizes comparable (on average) to the size of query-dependent subgraphs for which our method obtains the best performance. As first attempt, we directly project the partition created by
6.5. Optimizing the similarity measure

Figure 6.2: Comparison among local projections, projection of the full graph and direct use of the query-flow graph

**Figure 6.2**: Comparison among local projections, projection of the full graph and direct use of the query-flow graph

**Metis.** This solution performs very poorly (results are omitted), because the raw clusters do not typically include a significant fraction of the neighborhood of each node. To overcome this limitation, we study how to make the partitions include (a significant fraction of) the neighborhood of each node assigned to them. We test two methods. The former strategy consists of enlarging each partition with the in/out-neighbors of every node originally included in it. This method leads to the construction of clusters that may overlap: in the experiments, the test for a query is performed on the embedding of the cluster obtained from the partition which the query was originally assigned to by Metis. The application of this method does not improve performance.

We then experiment with a more expensive strategy, which consists of adding to each partition the two-step neighborhood of every node originally assigned to it. This solution creates larger clusters, and experimental evaluation shows that it provides results comparable to those obtained by projecting the full graph (see Table 6.5). Hence, we retain this approach as our cluster expansion method. **Choice of number of clusters.** The expansion step is necessary to make the clusters include many neighbors of a given node, i.e., queries that are likely to be related to the input query. However, this operation creates clusters of very large size: we believe that this can be a reason for not having a significant improvement in performance. Hence, we perform more extensive experiments, using Metis to divide the query-flow graph into a larger number of partitions and applying the expansion step to the sets of nodes ob-
Table 6.5: $\mathcal{M}_c$ for varying numbers of clusters; none of the pair-wise differences is statistically significant

<table>
<thead>
<tr>
<th>Method</th>
<th>Expansion distance</th>
<th>Number of clusters</th>
<th>Average ± st. dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full graph</td>
<td>-</td>
<td>-</td>
<td>1.2 ± 0.2</td>
</tr>
<tr>
<td>Clustering</td>
<td>1</td>
<td>100</td>
<td>1.0 ± 0.1</td>
</tr>
<tr>
<td>Clustering</td>
<td>2</td>
<td>100</td>
<td>1.2 ± 0.5</td>
</tr>
<tr>
<td>Clustering</td>
<td>2</td>
<td>200</td>
<td>2.0 ± 10.3</td>
</tr>
<tr>
<td>Clustering</td>
<td>2</td>
<td>1000</td>
<td>4.2 ± 25.8</td>
</tr>
<tr>
<td>Clustering</td>
<td>2</td>
<td>5000</td>
<td>3.9 ± 27.0</td>
</tr>
<tr>
<td>Clustering</td>
<td>2</td>
<td>20000</td>
<td>6.3 ± 46.7</td>
</tr>
</tbody>
</table>

tained. Table 6.5 shows how the method behaves with 1,000, 5,000 and 20,000 clusters. Starting from 5,000 we get a little improvement in performance, but the differences are not significant at $p < 0.1$. Although the method performs at least as well as the projection of the full graph, our intuition is that it would be worth exploring other approaches to extract query-independent subgraphs from the query-flow graph.

6.5.5 Summary

Figure 6.3 summarizes the tested methods, which can be divided into three groups:

- Using query-flow graph without projection ($N$): this performs close to random;
- Projecting the whole graph ($G$) or query independent clusters ($M_K$): good performance, with a possible advantage to systems based on clustering;
- Projecting query-dependent sub-graphs ($F_d$ or $S_d$): performs the best, with a small advantage for the systems that expand the neighborhood at 2 or 3 steps over the subgraph induced by the direct neighbors.

With respect to effectiveness, for all our methods, in at least 75% of the cases the system gave a larger similarity to queries in the same user-defined clusters than to queries in different user-defined clusters, meaning that they capture a notion of semantic similarity between queries.
6.6 Application to query recommendations

In this section we describe how we apply the proposed method for producing diverse query recommendations. Diversification of search results or query recommendations is a strategy adopted by search engines to improve the user experience and minimize the risk that the information need of the user will not be satisfied.

For our experiment we use a random sample of 100 queries. For each query, we generated a set of baseline query recommendations using a method suggested in [43]. This method (QUERYFLOW-SP) associates a query \( q \) with a set \( Q \) of recommendations obtained by selecting the most frequent reformulations from \( q \). Each query \( q' \in Q \) is assigned a ranking score given by the frequency of the transition \((q,q')\). This method was shown to perform as well as more sophisticated recommendation algorithms. In the following, we refer to it as the baseline.

We now outline a method for diversifying the set of recommendations obtained by this baseline. We use the diversification method described by Manca and Pintus [160]. This algorithm is similar, on a high level, to other diversity algorithms, e.g., [71]. The idea is to apply a greedy search that maximizes diversity while maintaining high relevance. The algorithm takes as input the set \( Q \) and builds a diverse set \( \mathcal{A} \) of queries. First, the query \( q_0 \in Q \) with the highest relevance score is selected, removed from \( Q \), and inserted into \( \mathcal{A} \). Observe that this ensures that the most popular query related to the input
6. QUERY SIMILARITY BY GRAPH PROJECTIONS

<table>
<thead>
<tr>
<th>Query: watch</th>
<th>rolex watch</th>
<th>citizen watch</th>
<th>seiko watch</th>
<th>watch free movie online</th>
<th>watch movies.net</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolex watch</td>
<td>-</td>
<td>5.2</td>
<td>4.4</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>citizen watch</td>
<td>-</td>
<td>-</td>
<td>8.7</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>seiko watch</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.4</td>
<td>0.1</td>
</tr>
<tr>
<td>watch free movie online</td>
<td>0.3</td>
<td>0.4</td>
<td>0.4</td>
<td>-</td>
<td>4.0</td>
</tr>
<tr>
<td>watch movies.net</td>
<td>0.2</td>
<td>0.2</td>
<td>0.3</td>
<td>4.0</td>
<td>-</td>
</tr>
</tbody>
</table>

(a) Method: cosine similarity between vectors of neighbors in the QFG ($\times 100$)

<table>
<thead>
<tr>
<th>Query: watch</th>
<th>rolex watch</th>
<th>citizen watch</th>
<th>seiko watch</th>
<th>watch free movie online</th>
<th>watch movies.net</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolex watch</td>
<td>-</td>
<td>0.99</td>
<td>0.99</td>
<td>0.12</td>
<td>0.12</td>
</tr>
<tr>
<td>citizen watch</td>
<td>-</td>
<td>1.00</td>
<td>-</td>
<td>0.09</td>
<td>0.08</td>
</tr>
<tr>
<td>seiko watch</td>
<td>-</td>
<td>1.00</td>
<td>-</td>
<td>-</td>
<td>0.11</td>
</tr>
<tr>
<td>watch free movie online</td>
<td>0.12</td>
<td>0.09</td>
<td>0.12</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>watch movies.net</td>
<td>0.12</td>
<td>0.08</td>
<td>0.11</td>
<td>1.00</td>
<td>-</td>
</tr>
</tbody>
</table>

(b) Method: spectral embedding of $G$

<table>
<thead>
<tr>
<th>Query: time</th>
<th>time magazine</th>
<th>new york times</th>
<th>time zone</th>
<th>world time</th>
<th>what time is it</th>
<th>time warner</th>
<th>time warner cable</th>
</tr>
</thead>
<tbody>
<tr>
<td>time magazine</td>
<td>0.8</td>
<td>0.4</td>
<td>0.7</td>
<td>0.4</td>
<td>0.2</td>
<td>0.62</td>
<td>0.88</td>
</tr>
<tr>
<td>new york times</td>
<td>0.4</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.88</td>
<td>0.95</td>
</tr>
<tr>
<td>time zone</td>
<td>0.3</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>world time</td>
<td>0.4</td>
<td>0.2</td>
<td>0.5</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
<td>0.1</td>
</tr>
<tr>
<td>what time is it</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>time warner</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>time warner cable</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
</tbody>
</table>

(c) Method: spectral embedding of $S_2(q)$

<table>
<thead>
<tr>
<th>Query: time</th>
<th>time magazine</th>
<th>new york times</th>
<th>time zone</th>
<th>world time</th>
<th>what time is it</th>
<th>time warner</th>
<th>time warner cable</th>
</tr>
</thead>
<tbody>
<tr>
<td>time magazine</td>
<td>0.89</td>
<td>0.62</td>
<td>0.52</td>
<td>0.68</td>
<td>0.98</td>
<td>0.95</td>
<td>0.96</td>
</tr>
<tr>
<td>new york times</td>
<td>0.62</td>
<td>0.57</td>
<td>0.57</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>time zone</td>
<td>0.62</td>
<td>0.87</td>
<td>0.87</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>world time</td>
<td>0.52</td>
<td>0.85</td>
<td>0.85</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>what time is it</td>
<td>0.68</td>
<td>0.93</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>time warner</td>
<td>0.86</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>time warner cable</td>
<td>0.86</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
</tbody>
</table>

(d) Method: cosine similarity between vectors of neighbors in the QFG

<table>
<thead>
<tr>
<th>Query: time</th>
<th>time magazine</th>
<th>new york times</th>
<th>time zone</th>
<th>world time</th>
<th>what time is it</th>
<th>time warner</th>
<th>time warner cable</th>
</tr>
</thead>
<tbody>
<tr>
<td>time magazine</td>
<td>0.88</td>
<td>0.62</td>
<td>0.52</td>
<td>0.68</td>
<td>0.98</td>
<td>0.95</td>
<td>0.96</td>
</tr>
<tr>
<td>new york times</td>
<td>0.62</td>
<td>0.57</td>
<td>0.57</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>time zone</td>
<td>0.62</td>
<td>0.87</td>
<td>0.87</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>world time</td>
<td>0.52</td>
<td>0.85</td>
<td>0.85</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>what time is it</td>
<td>0.68</td>
<td>0.93</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>time warner</td>
<td>0.86</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>time warner cable</td>
<td>0.86</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
</tbody>
</table>

(e) Method: spectral embedding of $G$

<table>
<thead>
<tr>
<th>Query: time</th>
<th>time magazine</th>
<th>new york times</th>
<th>time zone</th>
<th>world time</th>
<th>what time is it</th>
<th>time warner</th>
<th>time warner cable</th>
</tr>
</thead>
<tbody>
<tr>
<td>time magazine</td>
<td>0.88</td>
<td>0.62</td>
<td>0.52</td>
<td>0.68</td>
<td>0.98</td>
<td>0.95</td>
<td>0.96</td>
</tr>
<tr>
<td>new york times</td>
<td>0.62</td>
<td>0.57</td>
<td>0.57</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>time zone</td>
<td>0.62</td>
<td>0.87</td>
<td>0.87</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>world time</td>
<td>0.52</td>
<td>0.85</td>
<td>0.85</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>what time is it</td>
<td>0.68</td>
<td>0.93</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>time warner</td>
<td>0.86</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>time warner cable</td>
<td>0.86</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
</tbody>
</table>

(f) Method: spectral embedding of $S_2(q)$

Figure 6.4: Example query similarities using cosine similarity of neighbors (no projection), projection of the full graph, and projection of the sub-graph of $S_2(q)$. Top: query “watch”. Bottom: query “time”. Double lines separate manually-assigned clusters for these queries.
query is always selected for recommendation. Next, the algorithm starts an iterative phase: at each step the query \( q \in Q \) with maximum score \( s(q) \) is removed from \( Q \) and inserted into \( A \). The score \( s(q) \) is a combination function of the relevance of \( q \) with the distance \( d(q,A) \) of query \( q \) from the set \( A \) of queries that have already been selected. The algorithm balances diversity with relevance. Given that the two measures, distance from other queries and relevance, are not comparable, the algorithm tries to maximize the product of the two, picking up queries that have a high ranking score while being not too similar to the queries that have already been selected.

We derive the distance metric that we use for diversification from our projection method. We experiment with three schemes: \( N \), \( G \) and \( S_2(q) \). These methods measure similarity between queries in terms of cosine similarity between (a) their vectors of neighbors in the query-flow graph; (b) the vectors associated with the queries in the projection of the full graph; (c) the vectors obtained projecting the subgraph \( S_2(q) \). In the case of \( N \), we define the distance of a query \( q \) from the set \( A \) as the minimum distance between \( q \) and a query in \( A \):

\[
d(q,A) = \min_{t \in A} \{d(v(q), v(t))\}.
\]

In the case of \( G \) and \( S_2(q) \) the distance of \( q \) from \( A \) is defined as the distance between \( q \) and the centroid \( c(A) \) of the set \( A \):

\[
d(q,A) = d(v(q), c(A)).
\]

**Perceived diversity.** The task is highly subjective, and when measuring the agreement of the assessors on a subset of questions in which they overlap, we observe a moderate level of agreement (Cohen’s \( \kappa = 0.49 \)).

In Table 6.6 we show the results, which are expected given the outcomes from previous sections, as \( S_2(q) \) is the best method (in 51% of the cases it
is perceived as more diverse than the baseline, in 14% of the cases as less diverse), followed by $G$ (projecting the full graph), followed by $N$ (using the graph without applying projection).

**Perceived relevance.** Next we examine 100 queries and take the union of the top-3 recommendations from all the systems that are compared. This yields 460 distinct query pairs. The assessment this time is to measure if the recommendation is *relevant* to the original query. In this case we measure an inter-assessor agreement of $\kappa = 0.53$ on a subset of overlapping query pairs. We observed that on average 97% of the queries recommended by the baseline were relevant to the original query. When using the projection on the full graph, instead, this figure drops to 90%, and to 92% when using the $S_2$ method. Instead, when using $N$ there was basically no drop in relevance, measuring a 97% of recommendations relevant to the original query.

These results suggest that (i) $N$ does only a small change in the recommendations, and (ii) $S_2$ and the method that projects the full graph change the recommendations but still keep the fraction of recommended queries that are relevant to the original query at 90% or more.
Chapter 7

Conclusions and Future Work

This thesis has studied several topics of uttermost importance in the fields of graph mining and Web search.

We have focused on a few distinct problems, which were selected through a careful investigation of the critical issues arising in the various graph applications that have emerged in the context of Web and social media.

A first crucial problem is given by the huge size of the data collections. When the data to be processed is huge, one can often only afford to store them in secondary memory and process them with a small number of passes. The analysis of such data requires suitable models and algorithms.

We have built a framework of random-sampling streaming algorithms that approximate the number of occurrences of all the subgraphs of three and four nodes, in both directed and undirected networks. Prior to our work, existing algorithms and tools for subgraph counting had mostly focused on the analysis of small networks and could not handle massive data.

Our algorithms are able to provide high-precision estimates of the number of subgraphs of three or four nodes in networks of very large scale. We performed tests on graphs from more than ten different application domains, with size up to one billion edges. The algorithms read the input graph three times in a streaming fashion, and they use limited storage and per-unit processing time. The number of samples only depends on the required precision of the approximation and on the structure of the network.

We are able to provide evidence of the fact that the frequencies of these subgraphs can be used as meaningful features to characterize different families of networks.

In the future, we plan to use such subgraph-counting features to improve
tools for assessing the quality of Web sites, as well as detecting mirrors, graph anomalies and malicious activity.

We also plan to extend our algorithms in order to gain the ability of counting subgraphs with only one pass over the input graph. Moreover, we will make further investigations on the significance of the structure of small subgraphs and we will afford the issue of formalizing a more accurate statistical test for the patterns of 4 or more nodes.

Another fundamental problem is posed by the distributed nature of several informative sources. In many real-world scenarios, like for example traffic-monitoring applications, sensor networks and peer-to-peer environments, data is collected over a broad area, and it typically streams into remote sites at a very rapid rate. The volume of the collected data is usually very high, while the capacity for communication is very low. Thus, the data cannot be stored in one place, and a distributed processing is necessary. In this scenario, a basic and challenging task is to provide every node with primitives for monitoring its neighborhood by issuing continuous aggregate queries over the streams of events observed in its vicinity.

The work done for this problem has resulted in the development of an algorithmic toolbox for computing high-quality approximations of the frequency moments on the data observed within a neighborhood of limited size. The algorithms operate in a fully decentralized setting, and they are able to answers locally continuous aggregate queries using sub-linear space at every node in the network.

We have provided theoretical analysis of scenarios not considered in previous work and we have presented experimental evidence of the efficiency and accuracy of our strategies on realistic simulated scenarios. We have analyzed simple strategies that minimize the amount of communication needed to update the aggregate estimations when new events are observed.

In future work, we plan to demonstrate the usefulness of our algorithms by using them to develop tools for practical applications, like, for instance, distributed consensus in decentralized systems or measurement of statistics on the observations collected in a sensor network.

We remark the fact that the distributed streaming model is a natural extension of the semi-streaming model [105] which has been used, for example, for computing PageRank [91], or for approximating the local number of triangles which every node belonging to a network is involved in [35]. Hence, we believe that our results can be extended to assess the quality of single components in networks of large scale.

From a theoretical point of view, we plan to complete this work by affording a deeper analysis, aimed at proving a $\Omega(\sqrt{n})$ lower bound for the amount of memory needed to estimate the second frequency moment in distributed
settings that have to cope with data duplication.

In the field of Web Mining, graph structures have a great usefulness in the analysis of the extremely dynamic and rich information about Web data, as well as about users and their preferences and actions.

We have studied one year of data about the .uk Web, made of a series of twelve snapshots that were gathered at a monthly rate. The Web graphs extracted from the various snapshots have been merged into a global time-aware graph, provided with labels that allow constant-time access to temporal information for both nodes and edges. We have extensively analyzed the topology of this graph, both at the level of interconnection of hosts and of Web pages. We have studied how the data evolves over time. We have observed an extremely dynamic behavior in the turn-over rate of both pages and hyperlinks, whereas the content of Web pages has appeared to be affected by a significantly smaller degree of change. In most of cases, either pages change very rarely, or they are only affected by minor changes during the time interval taken into consideration. The amount of change experimented in the past is a good predictor of future change: our best results were obtained when considering the whole history of past changes. We believe that these findings may have practical impact in the design of search engines: for example, they can be used to develop more efficient crawling strategies, or to refine ranking algorithms based on temporal aspects.

Future work will focus on how to deal with dynamic pages, which make particularly challenging the problem of identifying URLs in different snapshots corresponding to the same Web page. Alignment is a non-trivial issue because if a URL is not static it might contain session-generated data (e.g., a session ID) that makes de facto identical URLs appear as syntactically distinct. We will develop a reasonable alignment technique for dynamic URLs. Moreover, we will use the extracted knowledge to predict the evolution of Web pages and to define a new measure of page freshness for improving ranking models.

In the last few years, Web 2.0 has turned the World Wide Web into a virtual society where users, besides the chance of accessing to enormous volumes of information, have also gained the possibility of freely publishing their own content and interacting with other people to share knowledge and opinions through online communities, question/answering portals, blogs, newsgroups, newsletters, forums, etc.

The impressive heterogeneity of the data generated by this massive user contribution adds more and more difficulties to the problem of extracting relevant content to satisfy the information needs of users who interact with search engines and information systems.

A very important and ubiquitous characteristic of the applications that have emerged in this scenario consists of the fact that the activities and rela-
relationships between users can be naturally modeled through a graph structure. Thus, many interesting graph data have appeared in Web and social networks.

We have chosen to devote particular attention to the graphs extracted from the query-log data that summarize the interactions of users with a search engine. In this thesis, we have studied several graphs extracted from search-engine logs according to different definitions.

In one work, we have built and analyzed three graphs defined by Baeza-Yates in [21]: the Word graph, the Click graph and the Session graph. The raw data was provided by Yahoo!.

We have shown that different graph representations of query logs coming from different relevance signals, are able to tackle different problems related to queries. To the best of our knowledge, the idea of combining and aggregating different query-log graphs has not been explored by previous work.

The combination of the graphs is done with very simple graph operations. In spite of the simplicity of the approach we show that we can obtain very good results (e.g. precision over 80%) with few queries (less than a day) for two different applications, which are detecting the type of a query transition and recognizing query spam. The former can be used to adapt the ranking of new queries, whereas the results obtained in the latter case could be used directly in the design of parental filters.

Our approach is completely unsupervised: this is another crucial characteristic, which makes the method useful especially in cases where we do not have enough labeled data, such as in non-popular languages. Thus, we obtain an improvement with respect to state-of-the-art tools for the analysis of query reformulations. These tools use semi-supervised methods. Hence, they require expensive editorial efforts.

Further work includes scalability issues of the method proposed, and implementation in the new iterative-MapReduce scenario. We also plan to apply these ideas to other applications, like identifying polysemic queries, recognizing logical sessions (missions) or recommending related queries.

We believe that our framework for query-log analysis, based on the joint mining of multiple graphs, will be a valid support for the analysis of the huge amount of data stored in the logs of search engines, providing us with efficient methods for combining different aspects of the same multi-faceted scenario and acquiring novel knowledge that might not be obtained through other methods, in particular if we have the restriction of having to use unsupervised methods.

The last contribution of our thesis has consisted of projecting the query flow graph [42] onto lower-dimensional spaces to define a similarity measure between queries.

We have shown that projecting reformulation graphs into a low-dimensional space allows us to describe a measure that effectively captures what users con-
sider as related queries. To the best of our knowledge, this is the first attempt to apply spectral methods to query-reformulation analysis.

After methodically exploring several design choices, we found two methods that perform well: one for off-line processing and one for on-line processing. The method for off-line processing basically works by storing 3-5 spectral co-ordinates per query and then using them at query time at constant cost. The method for on-line processing requires computing a small sub-graph at query time and then projecting this graph to obtain the coordinates. Our experimental results suggest that the on-line method is significantly more effective as a similarity measure, but of course it has a higher computational cost.

To demonstrate the practical impact of this research, we tested our measure as a component of a system for producing diverse query recommendations. Our experiments show that, incurring in a small risk of showing non-relevant recommendations, our method can be used to produce diverse query recommendations for users’ queries.

As future work, we would like to seek more effective off-line methods than projecting the full graph. The positive results with query-dependent sub-graphs suggest that the use of query-independent sub-graphs is a promising direction, perhaps through the use of graph-clustering techniques beyond the ones tested so far.

In short, graphs extracted from query logs have emerged as a very effective tool that can be used to gain a better understanding of how the users interact with each other and with search engines. This idea has allowed significant advances in Web research in the era of social networks, when the users have become extremely active in dealing with information systems.

In the future, it will be crucial to understand how graph-mining techniques can be applied to cope with the new challenges emerging with the advent of Web 3.0. The Web is going towards the direction of a greater and greater focus on individualization, with the goal of replacing social networking with people search, making everyone’s online presence searchable, taggable and sortable by community validation and by relevance.
Appendix: Subgraph
dictionary for directed graphs

Figure 1: 3-node directed subgraphs
Subgraph dictionary

$m_{13} \quad m_{14} \quad m_{15} \quad m_{16} \quad m_{17}$

$m_{18} \quad m_{19} \quad m_{20} \quad m_{21} \quad m_{22}$

$m_{23} \quad m_{24} \quad m_{25} \quad m_{26} \quad m_{27}$

$m_{28} \quad m_{29} \quad m_{30} \quad m_{31} \quad m_{32}$

$m_{33} \quad m_{34} \quad m_{35} \quad m_{36} \quad m_{37}$

$m_{38} \quad m_{39} \quad m_{40} \quad m_{41} \quad m_{42}$

$m_{43} \quad m_{44} \quad m_{45} \quad m_{46} \quad m_{47}$
Figure 2: 4-node directed subgraphs
Bibliography


[95] M. Deshpande, M. Kuramochi, and N. Wale. Frequent substructure-based approaches for classifying chemical compounds. TKDE, 17(8), 2005.


