UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Science and Technology
Department of Computer Science

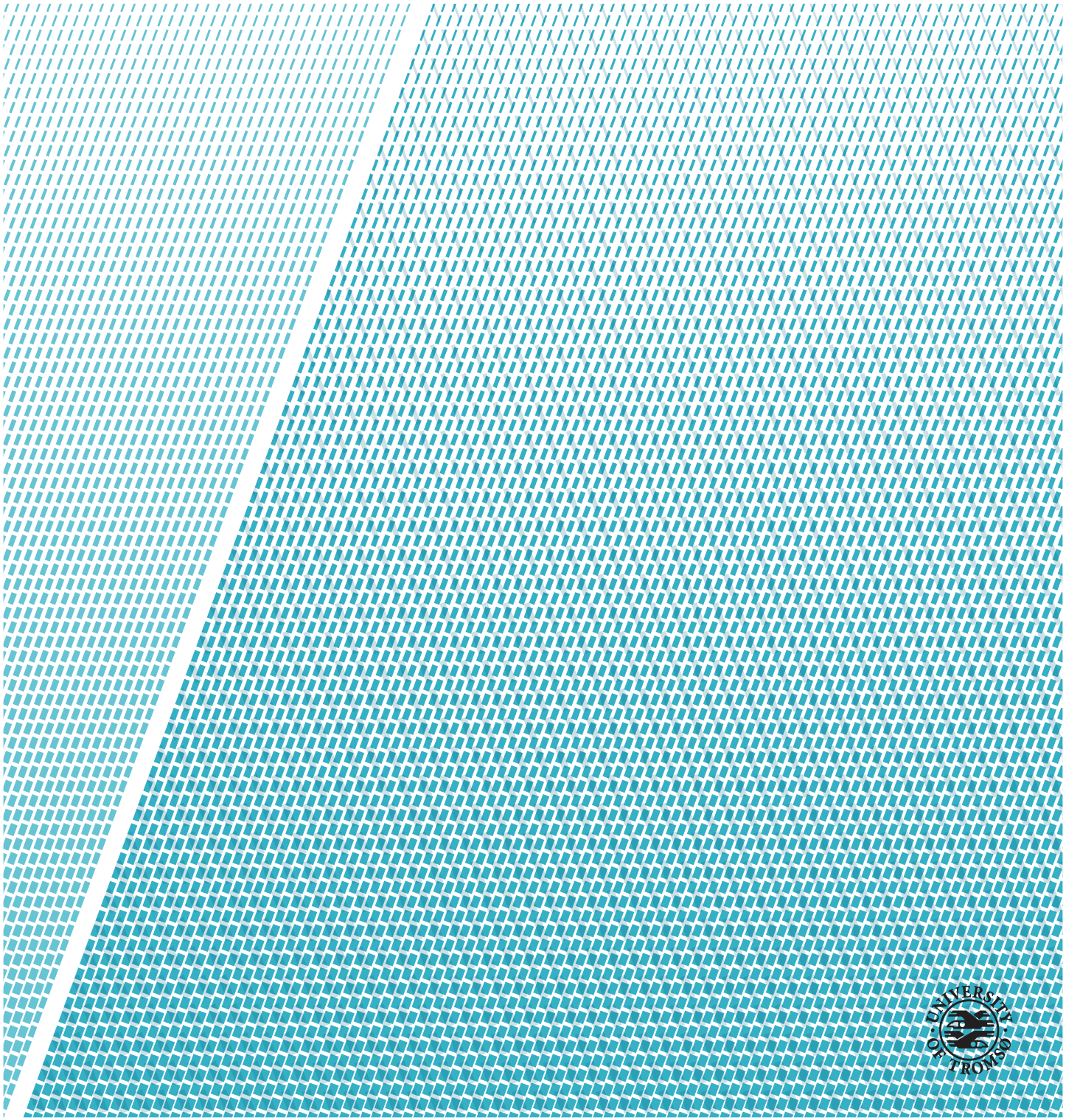
Incremental Information Retrieval

Finding new information by registering and ignoring already seen search results

—

Erlend Johannessen

INF-3990 Master's Thesis in Computer Science - June 2017



"I'll be more enthusiastic about encouraging thinking outside the box
when there's evidence of any thinking going on inside it."
–Terry Pratchett

"Wisdom comes from experience.
Experience is often a result of lack of wisdom."
–Terry Pratchett

"Real stupidity beats artificial intelligence every time."
–Terry Pratchett

"You can't kill me, because I've got a magic aaargh..."
–Terry Pratchett

Abstract

When searching the internet today we want immediate answers. We often search for a person, or a solution to a problem, or some topic we are interested in. The result quality off this kind of search is pretty good, most of the time we get the answers we need. The results, though, seems to be minor variations on the same results.

But what if the search for information is of a different nature, more like exploring. A typical case would be when a person has a hobby, and wants to search for information about it. Very soon all the quickly accessed information has already been seen, and is not that interesting in the context of new information.

It appears that no research has been done on search from this angle of approach. This thesis will look into this matter, to attempt to implement a system to give users the kind of search where the user may want to see new results on the same subject, maybe over a period of days, months, even years.

Acknowledgements

Working full time and doing a master's thesis at the same time is not really recommended. This thesis could not have been possible without the goodwill and/or support of several different agencies, most importantly family, friends, university and workplace.

Without discussion with and presence of class mates throughout the courses preceding the thesis, this would have been a much poorer experience. ☺

I am particularly grateful to my supervisor Randi Karlsen, who offered invaluable advice, structure and encouragement.

Heartfelt thanks also goes out to my fellow testers, André, Jørgen, Stephan and Easterine, who gave very good feedback on what was right - and wrong - with the application.

Easterine Kire, my girl, my proofreader, my work ethic inspiration, partner, friend and test subject for several parts of this thesis. Explaining inner workings of this to you made me distil and simplify my ramblings into something hopefully more coherent. Thank you! ♡

Author's note: This document is written in Queen's English - "the English language as written and spoken correctly by educated people in Britain". Any "spelling errors" would be those words where the reader would expect American English, which is an abhorrence. ☺

Contents

Abstract **iii**

Acknowledgements **v**

List of Figures **xiii**

List of Tables **xvii**

List of Listings **xix**

Glossary **xxi**

1 Introduction **1**
1.1 Motivation 2
1.2 Research question 3
1.3 Approach 3
1.3.1 Description 3
1.3.2 Selecting search method 4
1.4 Contributions 5
1.5 Limitations 5
1.6 Outline 6

2 Background **7**
2.1 How search engines work 7
2.1.1 Building and updating the index 7
2.1.2 Indexing 8
2.1.3 Querying 9
2.1.4 Ranking 10
2.2 Using traditional on-line search 10
2.3 Web scraping 10
2.4 Search engine APIs 11
2.4.1 Search engine API status 11
2.4.2 On-line search vs search APIs 12
2.5 Related work 13

3 Problem overview **15**
3.1 Query and QueryRun 15
3.2 Search example 16
3.2.1 Results graph 19
3.3 User search 20
3.4 Goal 21

4 Architecture and design **23**
4.1 Design goal 23
4.2 Architecture 24
4.3 How IIR search works 25
4.3.1 Search details 25
4.4 IIR features 26
4.4.1 Result status 28
4.4.2 Ranking 29
4.4.3 Result folders 30
4.5 Client design 31
4.6 System requirements 31
4.6.1 Web server hardware 31
4.6.2 Storage 32
4.7 Selecting search engine to work with 34
4.8 Hypothetical search progression 35

5 Data collection and analysis **39**
5.1 Why data collection 39
5.2 Queries 40
5.3 The anatomy of a query 41
5.4 Data collection environment 42
5.4.1 Hardware 42
5.4.2 Database 42
5.4.3 The batch job 44
5.4.4 Updating the database 45
5.5 Data collection problems and errors 47
5.5.1 Typing error 47
5.5.2 Premature termination 47
5.5.3 Exact search 48
5.5.4 Connection error 48
5.5.5 Parameter incorrect 49
5.5.6 Parse error 49
5.6 Batch result analysis 49
5.6.1 Data collection periods 49
5.6.2 Data selected for analysis and testing 50
5.6.3 Short analysis of new results 50
5.6.4 Result overlap between days 52

5.6.5 Duplicates . . . 53
5.7 Batch result patterns . . . 55
5.7.1 Many free results - many exact . . . 55
5.7.2 Many free results - no exact . . . 59
5.8 Simulation of usage . . . 60

6 Implementation **65**
6.1 Technology . . . 65
6.1.1 MongoDB . . . 66
6.1.2 Go programming language . . . 66
6.1.3 Go packages . . . 66
6.1.4 Web client . . . 67
6.1.5 Server . . . 67
6.2 Database architecture . . . 68
6.3 Front-end . . . 71
6.4 Back-end . . . 73
6.5 Using Bing Search API . . . 76
6.5.1 Interfacing with the API . . . 76
6.5.2 Go wrapper for Bing Search API . . . 77
6.5.3 Calling Bing Search API . . . 77
6.5.4 Search during the test phase . . . 79
6.6 Development environment . . . 79
6.7 Code summary . . . 80

7 Testing **81**
7.1 User testing . . . 81
7.2 Manual for using IIR . . . 82
7.3 Instructions to the testers . . . 83
7.4 Questionnaire . . . 84
7.5 Questionnaire result . . . 85
7.5.1 Statement scores . . . 85
7.5.2 Replies in comment field 1 . . . 85
7.5.3 Replies in comment field 2 . . . 86
7.5.4 Replies in comment field 3 . . . 86
7.5.5 Questionnaire result summary . . . 87
7.5.6 Author testing . . . 88
7.6 Testing statistics . . . 88
7.7 Data analysis . . . 91
7.7.1 Test result plot details . . . 91
7.7.2 Few QueryRuns . . . 91
7.7.3 Energetic usage pattern . . . 93
7.7.4 Relaxed usage pattern . . . 95
7.7.5 Other patterns . . . 97

8 Discussion **101**
8.1 Findings . . . 101
8.2 The approach to search . . . 102
8.3 Implementing the prototype . . . 103
8.3.1 Storage . . . 104
8.3.2 Usability . . . 104
8.3.3 Reliability . . . 105
8.3.4 Scalability . . . 105
8.4 Data collection . . . 105
8.4.1 Ranking . . . 106
8.4.2 Query quality . . . 106
8.5 Test results . . . 107
8.5.1 Exact search . . . 108
8.5.2 Analysing new results . . . 108
8.5.3 Author's comments on testing IIR . . . 109
8.6 Problems, bugs and errors . . . 109
8.7 Is it commercially viable? . . . 110

9 Future work **111**
9.1 New features . . . 112
9.1.1 Content preview . . . 112
9.1.2 Rules or filters . . . 112
9.1.3 White- or black-listing . . . 113
9.1.4 Analysis of user interactions . . . 114
9.1.5 Handling many queries and results . . . 114
9.1.6 Miscellaneous features . . . 115
9.2 Refactoring . . . 115
9.2.1 Front-end . . . 115
9.2.2 Back-end . . . 116
9.2.3 Unit testing . . . 116
9.2.4 Error handling . . . 116
9.3 Change in type of application . . . 117
9.4 Search Engine . . . 117

10 Conclusion **121**

Bibliography **123**

A Data collection results **129**
A.1 Queries and the reasoning behind them . . . 129

A.2 Full data collection results 131
A.2.1 Full data collection results, totals 131
A.2.2 Full data collection results, details 133
A.3 Plots for results 154
A.3.1 Summary plots for all queries 154
A.3.2 Summary plots for all queries, with errors 155
A.3.3 Summary plots for each query 157

B IIR testing instructions 173

C IIR online manual 175

D Questionnaire 179

E Detailed test results 181

F IIR code and utilities 187
F.1 bingv2batch 187
F.2 bingv2analysis 188
F.3 bingv2analysispercent 188
F.4 bingv2convert 189
F.5 iirweb 189
F.6 iiranalysis 189
F.7 Summary 190

List of Figures

3.1 How Query and QueryRun are related. 16
3.2 First batch of results 17
3.3 Second batch of results 17
3.4 Third batch of results 18
3.5 Fourth batch of results 18
3.6 The expected curve for new results 19
3.7 User search results 1 20
3.8 User search results 2 20

4.1 IIR system at a glance. 24
4.2 User using IIR to search. IIR connects to the search engine API, and returns refined results to the user. 26
4.3 Main features available to the user from the UI of IIR. 28
4.4 The main user interface 31
4.5 Hypothetical search progression, search 1 36
4.6 Hypothetical search progression, search 2 37
4.7 Hypothetical search progression, search 3 38
4.8 Hypothetical search progression, search 4 38

5.1 Anatomy of a batch query 42
5.2 Data collection database entities, and corresponding database collections. 43
5.3 Distribution of days in the datacollection period 50
5.4 Plot of free searches from day 1 51
5.5 Screen capture of comparison of Query 17 53
5.6 Free query "Winds of winter" 56
5.7 Exact query "Winds of winter" 56
5.8 Day to day results of free query "Winds of winter" 57
5.9 Day to day results of exact query "Winds of winter" 58
5.10 Free query "mobile application health sensor data" 59
5.11 Exact query "mobile application health sensor data" 60
5.12 Progression for 25 results per day, for query 11 61
5.13 Progression for 100 results per day, for query 11 62
5.14 Progression for 250 results per day, for query 11 62

5.15 Progression for 500 results per day, for query 11 63

6.1 IIR database entities, and corresponding database collections. 68
6.2 The main user interface, with the first query loaded 71
6.3 IIR client architecture. 72
6.4 Search API call sequence diagram 77

7.1 Detailed test results for query 17 92
7.2 Detailed test results for query 29 93
7.3 Detailed test results for query 29, lower part 94
7.4 Detailed test results for query 35 95
7.5 Detailed test results for query 35, lower part 98
7.6 Detailed test results for query 36 98
7.7 Detailed test results for query 15 99

9.1 Top 3 search engines shown, out of 15 listed, courtesy of eBizMBA (see footnote). 118

A.1 Combined plots for all free searches 154
A.2 Combined plots for all exact searches 155
A.3 Full combined plots for free searches 156
A.4 Full combined plots for exact searches 157
A.5 Percentage-wise plots for queries 1 and 2 158
A.6 Data plots for queries 1 and 2 158
A.7 Percentage-wise plots for queries 3 and 4 159
A.8 Data plots for queries 3 and 4 159
A.9 Percentage-wise plots for queries 5 and 6 159
A.10 Data plots for queries 5 and 6 160
A.11 Percentage-wise plots for queries 7 and 8 160
A.12 Data plots for queries 7 and 8 160
A.13 Percentage-wise plots for queries 9 and 10 161
A.14 Data plots for queries 9 and 10 161
A.15 Percentage-wise plots for queries 11 and 12 161
A.16 Data plots for queries 11 and 12 162
A.17 Percentage-wise plots for queries 13 and 14 162
A.18 Data plots for queries 13 and 14 162
A.19 Percentage-wise plots for queries 15 and 16 163
A.20 Data plots for queries 15 and 16 163
A.21 Percentage-wise plots for queries 17 and 18 163
A.22 Data plots for queries 17 and 18 164
A.23 Percentage-wise plots for queries 19 and 20 164
A.24 Data plots for queries 19 and 20 164
A.25 Percentage-wise plots for queries 21 and 22 165
A.26 Data plots for queries 21 and 22 165

A.27 Percentage-wise plots for queries 23 and 24 165
A.28 Data plots for queries 23 and 24 166
A.29 Percentage-wise plots for queries 25 and 26 166
A.30 Data plots for queries 25 and 26 166
A.31 Percentage-wise plots for queries 27 and 28 167
A.32 Data plots for queries 27 and 28 167
A.33 Percentage-wise plots for queries 29 and 30 167
A.34 Data plots for queries 29 and 30 168
A.35 Percentage-wise plots for queries 31 and 32 168
A.36 Data plots for queries 31 and 32 168
A.37 Percentage-wise plots for queries 33 and 34 169
A.38 Data plots for queries 33 and 34 169
A.39 Percentage-wise plots for queries 35 and 36 169
A.40 Data plots for queries 35 and 36 170
A.41 Percentage-wise plots for queries 37 and 38 170
A.42 Data plots for queries 37 and 38 170
A.43 Percentage-wise plots for queries 39 and 40 171
A.44 Data plots for queries 39 and 40 171

B.1 Testing instructions 174

C.1 IIR on-line manual, page 1 176
C.2 IIR on-line manual, page 2 177

D.1 IIR questionnaire 180

E.1 Detailed test results for queries 1 and 2. 181
E.2 Detailed test results for query 3 182
E.3 Detailed test results for queries 5 and 6. 182
E.4 Detailed test results for query 7 182
E.5 Detailed test results for queries 9 and 10. 183
E.6 Detailed test results for queries 11 and 12. 183
E.7 Detailed test results for queries 13 and 14. 183
E.8 Detailed test results for queries 15 and 16. 184
E.9 Detailed test results for query 17 184
E.10 Detailed test results for query 29 184
E.11 Detailed test results for query 31 185
E.12 Detailed test results for query 33 185
E.13 Detailed test results for queries 35 and 36. 185
E.14 Detailed test results for queries 37 and 38. 186
E.15 Detailed test results for queries 39 and 40. 186

List of Tables

2.1 Discontinuing of web search APIs from the major search providers. 12

3.1 10 first days of results for free Query 11 19

4.1 IIR user actions available on a list showing only new results. 27
4.2 A result's possible statuses. 29
4.3 Weighting scores for IIR search word ranking. 30
4.4 Folders to show results with different statuses. 30

5.1 Summary of automated searches, run for 54days. 41
5.2 The "moving parts" of the data collection system. 41
5.3 Overlap in the first 100 results 52

6.1 IIR client components. 72
6.2 IIR server main controllers. 75
6.3 Parameters used in Bing Search API transactions 78

7.1 Statement score for the questionnaire 85
7.2 Replies in the first comment field 85
7.3 Replies in the second comment field 86
7.4 Replies in the third comment field 87
7.5 Explanation for columns in table 7.6 89
7.6 Test totals for query 1 - 40 90
7.7 Detailed test results for Query 17 92
7.8 Detailed test results for Query 29 95
7.9 Detailed test results for Query 35 97

A.1 Queries, query "owners" and a short reasoning behind the queries. 130
A.2 Explanation for columns in table A.3 131
A.3 Totals and averages for query 1 - 40 132
A.4 Error codes for the QueryRuns. 133
A.5 Query results for queries 1 - 2 134

A.6 Query results for queries 3 - 4 135
A.7 Query results for queries 5 - 6 136
A.8 Query results for queries 7 - 8 137
A.9 Query results for queries 9 - 10 138
A.10 Query results for queries 11 - 12 139
A.11 Query results for queries 13 - 14 140
A.12 Query results for queries 15 - 16 141
A.13 Query results for queries 17 - 18 142
A.14 Query results for queries 19 - 20 143
A.15 Query results for queries 21 - 22 144
A.16 Query results for queries 23 - 24 145
A.17 Query results for queries 25 - 26 146
A.18 Query results for queries 27 - 28 147
A.19 Query results for queries 29 - 30 148
A.20 Query results for queries 31 - 32 149
A.21 Query results for queries 33 - 34 150
A.22 Query results for queries 35 - 36 151
A.23 Query results for queries 37 - 38 152
A.24 Query results for queries 39 - 40 153

F.1 Summary of number of code lines 190

List of Listings

5.1 Go struct describing the data collection Query 43
5.2 Go struct describing the data collection QueryRun 43
5.3 Go struct describing the data collection Result 44
5.4 Running the queries in a command file 45
5.5 The starting configuration of the query, before any searches are run 45
5.6 The same query after 3 runs, with more information from the runs 45
5.7 The first out of the 823 results for query 27 on the first day . 46

6.1 Go struct describing the User 69
6.2 Go struct describing the Query 69
6.3 Go struct describing the QueryRun 70
6.4 Go struct describing the Result 70
6.5 Main program for IIR 73
6.6 Registering routes for the IIR web service 74
6.7 Main function for accessing the Bing Search API from Go . . 78
6.8 Summary of code used in IIR 80

F.1 Data collection application code summary 187
F.2 Data collection analysis code summary # 1 188
F.3 Data collection analysis code summary # 2 188
F.4 Data collection conversion code summary 189
F.5 IIR web service code summary 189
F.6 Test results analysis code summary 189

Glossary

API Application Programming Interface

AWS Amazon Web Services

CET Central European Time

CLI Command line (user) interface

CSS Cascading Style Sheets

DOM Document Object Model

GCS Google Custom Search

GUI Graphical User Interface

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IaaS Infrastructure as a Service

IIR Incremental Information Retrieval, this thesis

JSON JavaScript Object Notation

LAMP Linux, Apache, MySQL, and PHP/Python/Perl

OSE On-line Search Engine

P2P Peer-to-peer

PaaS Platform as a Service

PoC Proof of Concept

UI User Interface

URI Uniform Resource Identifier

URL Uniform Resource Locator

XML eXtensible Markup Language

1

Introduction

Information retrieval can be defined in very broad terms. Looking at your watch or reading the timetable on a bus stop is a form of information retrieval. However, as an academic field of study, information retrieval could be defined as dealing with the representation, storage, organization of, and access to information items[2].

The amount of digitally stored knowledge is growing at an exponential rate, and with more and more people creating content on-line[26], this will only continue to increase[20].

Information retrieval in the form of search engines is a crucial part of finding and using existing data on the Internet. It is increasingly important to be able to find relevant information across the Internet. At the same time it seems increasingly difficult to find relevant or precise information, as a direct consequence of the staggering amount of information that is becoming available.

When we are searching the Internet today we often search for a person, or a solution to a problem, or some topic we are interested in. The result quality is pretty good, but seems to have a tendency to give minor variations on the same results. The variation in results for the exact same search could stem from the user's profile[22], location[28], search history and search habits, among others.

Sometimes we want more information on a particular subject, a search that can present information that the user has not seen before. In this case today's

search engines have a tendency to repeatedly give the same or similar results. Using the major search engines, the user has to leaf through page after page of increasingly irrelevant answers to find relevant search result that is new to the user.

Experience tells us that searching for previously unseen information on the same subject in this way, can be more cumbersome and time-consuming, and may in the end make the user just give up searching, or the search may simply yield no new information on a subject.

An expected scenario in this kind of search is that the user has a hobby or a long-term special subject, *e.g.* a chronic illness, that the user is interested in getting more information on. The result items the user gets from the traditional on-line search engines may be precise, but the user has most likely seen them before. Finding previously unseen material, normally buried deeper down in the list of results, is more important. This is the kind of search where the user may want to see new results on the same subject over a period of days, months, even years.

An alternative route for the user can be to become a member of forums and communities that have the same interests. Many answers can be found this way as well. But this source of information is also limited to what other community members have found, or are able to contribute.

1.1 Motivation

Search engines are created to give immediate and relevant answers to search queries, with high precision. This situation often leads to the same or similar relevant answers for the same search query.

Sometimes users want to explore more information on a particular subject, but the nature of search engines becomes a hindrance more than a help for this kind of search.

The motivation for this thesis is thus the lack of new results, previously unseen information in both a short- and long-term situation. An incremental information retrieval (IIR) system could solve this.

1.2 Research question

This thesis builds on the motivation in section 1.1, and can be distilled into a single question.

> *How can a long term search service be created to discover previously unseen search results, regularly concealed in traditional on-line search?*

Long term search in this case refers to a web based search, performed several times over a period of weeks or months. Initially no search results have been seen before by the user, but as time goes by, the user has seen an increasing number of results. The results the user has seen is normally at the top of the list of search results from the on-line search engine.

This means that *traditional on-line search* is trying to help the user to find precise and relevant answers, but as time passes is obscuring other results that may be equally relevant for the user. The aim is to not show the seen results to the user, and only show *previously unseen search results*. Such a service will attempt to show results that are *regularly concealed* by the search engine's results.

1.3 Approach

In this section the thesis approach and search engine selection is presented.

1.3.1 Description

The goal of the thesis is to create an Incremental Information Retrieval (IIR) system, that can monitor the user's actions, and based on details of this usage hide previously seen results, and only show the user previously unseen results. The user can actively choose to save or discard results. Even if the user only browses through results, and does not use the system actively, IIR automatically records information on the results the user has seen. This way IIR will continue to show only those results that the user has not seen before.

To help users find new information, a prototype of IIR is created, a proof of concept[1] (PoC) type implementation, to search the web through the Bing Search API. This prototype is set up as a web based service that connects to

1. https://en.oxforddictionaries.com/definition/proof_of_concept

the search engine, and shows the results to the users.

The users have the possibility to save or discard results, and black-list or white-list domain names. Information about the results the user has interacted with is stored in a database, so they can be hidden from the users when they do subsequent searches.

In order to verify the usefulness of such a prototype, IIR was tested by users in a live situation. Users gave feedback in a questionnaire on the service created, and the testers' usage patterns were inspected, to evaluate their feedback. Data received from the search API were also analysed, to investigate staleness in the results returned over a period of time.

1.3.2 Selecting search method

There are a lot of different types of search engines, *e.g.* traditional on-line search, meta-search, geographically limited scope, semantic, enterprise, legal, medical, and more. However, a traditional on-line search engine should be used, to be able to search across a broad range of topics. So the focus is the major existing search engine APIs, and what they can offer. These are Google, Bing, and Yahoo.

One technique for accessing search results is "web scraping"[2], parsing results directly from the HTML returned from an on-line search result. This is possible, but it violates the terms of service for search engines, and are (by some) considered illegal[3].

So a better way is using an existing search engine API, *i.e.*, services that have been created for just this kind of purpose.

In this thesis, Bing Search API has been selected as the data provider for the solution.

2. http://wiki.c2.com/?WebScraping
3. http://blog.icreon.us/advise/web-scraping-legality

1.4 Contributions

This thesis makes the following contributions:

1. A proof of concept implementation of the IIR system, a prototype application that monitors the user's activity, and applies this user context to the system, to hide results from the user that the user has seen before.

2. By collecting and analysing data from Bing search engine through Bing Web Search API, showing that search engines yield the same or similar search results for the same search query, when run repeatedly.

In addition to these two contributions, a Go language[41] wrapper for the Bing Web Search API v2 was implemented.

1.5 Limitations

This project has no financing attached to it, so this puts a limitation on what kind of environment can be created to support the thesis. As a consequence, the cost of doing the thesis must be as close to zero as possible.

Another matter is the sheer number of search engines that theoretically could be used to implement this kind of search. There are different technologies used (like P2P) and different search engine types, like mash-up or aggregate. There are also dedicated search engines; fashion, genealogy, jobs, legal, medical etc., in addition to specific services search, like Twitter[4] or Flickr[5] search APIs.

In theory, several search engines could be used as a test search engine and data collection tool for this thesis. In practice - to reduce the complexity of the thesis - only one of the major search engines is used. Google, Bing and Yahoo! are general purpose search engines, the top three most used search engines[15] on many lists, among them Alexa[6]. Out of these three, Bing has been chosen as search engine.

4. https://dev.twitter.com/rest/public/search
5. https://www.flickr.com/services/api/flickr.photos.search.html
6. http://www.alexa.com/topsites/category/Computers/Internet/Searching/Search_Engines

1.6 Outline

The outline for the rest of this thesis is as follows.

Chapter 2	Describes the background and related work.
Chapter 3	Gives a problem description for this thesis.
Chapter 4	Prototype architecture and design is presented.
Chapter 5	Data collection is described and analysed.
Chapter 6	Implementation of the chosen design is detailed.
Chapter 7	User-testing of the solution is presented and evaluated.
Chapter 8	Findings and implications are discussed.
Chapter 9	Future work is presented.
Chapter 10	A summary and concluding remarks.

2 Background

Information retrieval is a wide subject. For the purpose of this project, the discussion will be limited to information indexers and search engines for the World Wide Web.

In this project, a service that uses search engines' data as data source, is created. So in order to know how the solution is influenced by the search engine, the search engine's inner workings need to be examined more closely.

2.1 How search engines work

A search engine is in principle no more than a combination of applications that index the web and provide the index to us, the users, so we can search the index for subjects we are interested in[23] [42] [5] [6]. In practice this can be quite difficult, with the vast amount of information available, and the large increase in information creation.

2.1.1 Building and updating the index

The main mechanism for updating the search engine index with new or changed information is an application that can browse the web in an automated manner. This type of application, called a web crawler, saves a copy of all the browsed

pages for later processing by the search engine[11] [8] [16].

2.1.2 Indexing

Different search engines may have their own ways of analysing the output from the web crawler, but some of techniques used are common to most of them. Many different types of sources are indexed - natural language documents, but also media like audio, video and images[14].

Inverted indexes

When documents are added to the search engine storage, data structures are created so that the documents could be found quickly through text search. This type of indexes are called *inverted indexes*[12], in that they are mapping words or numbers to the relevant documents.

Tokens and terms

During the index creation phase, sequences of characters, numbers and other elements are analysed and processed to optimise search. There are many issues to consider, like punctuation, capitalisation, stemming and stop words[14].

Punctuation like ".", "-", ",", "#" or "$" is also normally not indexed, something that is problematic *e.g.* when searching for "C#".

The search engine also needs to decide what to do with *capitalisation*. There is a difference between "WHO", the abbreviation of World Health Organization, and the word "who" (which incidentally may also be a stop word, see *stop words* below). There are also other capitalisation variants, like "Cat", cAt", "CAT" or "caT" which are different words but may refer to the same thing.

Stemming is when different forms of a word are reduced to their common base[19]. An example can be "stems", "stemmer", "stemming", "stemmed" which all have "stem" as their base. Storing the base of the word in the index makes the index smaller and faster to search, but then searching for the original word like "stemming" may give additional results that may not be interesting for the user.

Stop words[42] like "the", "a", "in", or "which" are not considered important and is a candidate for removal, something which can make a search for *e.g.* the band "The the" more difficult.

When the search engine index is maintained, all this needs to be considered and managed appropriately.

Duplicates

It can be difficult for the search engine to distinguish between duplicates within the same site, but near-duplicates[25] can also be a problem. Near-duplicates stem mostly from different sites or addresses that show the same content, but differ in session id, time stamp, visitor count etc. in addition to URLs[4].

One source of duplicates can be home pages for *e.g.* newspapers, like `http://www.nytimes.com/`, *i.e.*, at the root of their domain name. These would normally have different content even when the URL, Title and Description is the same, since news content changes from day to day. Indexing of root domains can also be adjusted by the respective websites' *robot.txt*[1], which suggests to the search engine's web crawler (web indexer) which paths are available on their web site.

2.1.3 Querying

When searching, the user enters a query into the search engine and gets results according to the terms and words entered. But there is more happening to this query under the bonnet[2].

When the search engine receives a query, it often needs to rewrite it in order to help the user obtain better precision in the result[24] [31] [1] [7].

Using a search for *angora cats* as an example, the search engine may need to rewrite "cats" to "cat" to possibly get better results. Or what if the user misspelled cats as "ctas". The search engine needs to handle this gracefully[44].

A normal way of forcing the search engine to accept an exact query is to put the query in quotes. All the three major search engines offer this mode of searching. Searching for an exact phrase may yield no results from the search engine. When this happens, search engines have different strategies for what to show the user. Google tells you that no results were found, and gives the results for the non-exact (free) version of the phrase given. Bing shows no results, but suggests alternative searches that may give relevant answers. Yahoo in the same way as Google shows the non-exact version of answers, without any warning that the exact phrase was not found.

1. `http://www.robotstxt.org`
2. "The hinged metal canopy covering the engine of a motor vehicle."

2.1.4 Ranking

In order to present relevant information to someone using a search engine, some ranking mechanism is necessary. Results that appear at the top of the result list would be considered of a higher rank than results further down on the list. This is achieved by organising the results through algorithms that weighs the importance of each result[36] [13].

Many factors contribute to the result's importance. These include - but are not limited to - user behaviour, user location, general popularity, time frame, user's connections with other users, user's search history and particular interests. Other more technical factors apply as well, *e.g.* mobile friendly web sites may be ranked higher in mobile searches.

The major search engines also allow parties to pay for a higher ranking in the result[29] [32].

The outcome is that results may vary from one search to another, even if the same search is done by the same user over a short time span.

2.2 Using traditional on-line search

Search engines uses algorithms for ranking search results to produce a useful result for the user. Search engines also display advertisements as a part of the search result, called "paid listings", "pay per click"listings or "sponsored links"[29] [32].

Users normally do not page around in the search results much. A study done in 2013[10] by the on-line ad network Chitika[3] suggests that 91.5% of all registered search engine traffic was on the first results page, 4.8% of users read the second page, and only 3.7% went further than the second page. This suggests that users generally either re-query with a different wording, give up, or go elsewhere (other search engine or other information retrieval system).

2.3 Web scraping

One technique for accessing search results is "web scraping"[4], parsing results directly from the HTML returned from a traditional on-line search

3. `http://chitika.com`
4. `http://wiki.c2.com/?WebScraping`

engine[35].

This is technically possible to use as an information retrieval source, but it is by many considered illegal[5]. Web scraping violates the Terms of Service (TOS) for search engines, see Google TOS[6].

> "*Do not misuse our Services, for example, do not interfere with Services or try to access them using a method other than the interface and the instructions that we provide.*"

2.4 Search engine APIs

An Application Programming Interface (API)[7], is a way of allowing different programs to communicate with each other. Any piece of software can talk to the API as long as the rules of the API are followed, like authentication and method/procedure signatures.

The search engine API lets the consumer of the API connect to the provider's search engine, and use it for searching the provider's database of web results.

The search engines' APIs also differ in what features they offer to their consumers. Differences include what parameters the APIs accept, or the data format they return their results in.

2.4.1 Search engine API status

All of the three major search engines; Google, Bing, and Yahoo, offers or have offered connections to their search engines through APIs.

Google's remaining search API service is called Google Custom Search, and is a free service that can be used to search specific websites, typically blogs or small-scale home pages. Ads and Google branding are required with GCS[8].

Microsoft Bing Web Search API v2 is replaced by v5, which is a part of Microsoft Cognitive Services[9]. Version 5 offers a web search API for a fee, where the entire Bing search engine database can be searched. The former version (v2)

5. http://blog.icreon.us/advise/web-scraping-legality
6. http://www.google.com/policies/terms
7. http://wiki.c2.com/?ApplicationProgrammingInterface
8. http://searchengineland.com/google-site-search-way-now-271366
9. https://www.microsoft.com/cognitive-services

had 5000 free searches per month.

Yahoo BOSS API was Yahoo's search API offering. It was discontinued and replaced by Yahoo Partner Ads (YPA), a system created to "Monetize your website across desktop, tablet and mobile" [10].

Shutting down search engine APIs seems to be a trend. Other similar search services, like entireweb Search API[11], has also been discontinued.

Search API	Type	Deprecated	Discontinued
Google Web Search API	Search API	Nov. 1, 2010	Sept. 29, 2014
Google Custom Search[12] (GCS)	Site search		
Google Site Search[13] (GSS)	Site search	March 31, 2017	March 31, 2018
Microsoft Bing Web Search API v2[14]	Search API	Dec. 15, 2016	March 31, 2017
Microsoft Bing Web Search API v5[15]	Search API		
Yahoo BOSS API[16]	Search API		March 31, 2016

Table 2.1: Discontinuing of web search APIs from the major search providers.

2.4.2 On-line search vs search APIs

APIs have been created for computers to be able to retrieve results from search engines in a machine readable format. APIs can generally be more precisely controlled, give results in a machine-readable format like JavaScript Object Notation (JSON), and also contain meta-data about the results.

There are limitations with using APIs vs the on-line search engines. One major problem is that the results vary between the on-line search and the API search for the same search engine, doing the same query.

An article by Kumar *et al*. [30] discusses the differences between the online and API version of the major search engines. And in an article comparing Google on-line search with using their API, Mayr and Tosques concludes that "... it has to be clear that querying the Google APIs does not deliver the same result data as the highly optimized Google Standard interface"[34].

Even though these are older articles, there is no reason to doubt that there

10. https://developer.yahoo.com/ypa
11. http://www.entireweb.com/services
12. https://developers.google.com/custom-search/docs/overview
13. https://enterprise.google.com/search/products/gss.html
14. https://datamarket.azure.com/dataset/bing/search
15. https://www.microsoft.com/cognitive-services/en-us/bing-web-search-api
16. https://developer.yahoo.com/boss/search/

still are differences between on-line search and API based search. Reasons that results differ in this way include, among others, features such as real-time results, social features, ranking, or personalized results. APIs also have different uses and other market targets than on-line search.

In addition, web search APIs have limits on how many results can be returned from the search engine.

2.5 Related work

A close to exhaustive search for directly relatable work has been fruitless. There seems to be no one that has done similar research or has made a comparable solution.

However, several articles have been found on personalisation and user context based work. These discuss how to use the user's context and preferences to adjust the results of queries to fit the user.

Some interesting ones are "Contextual search: Issues and challenges" by Gabriella Pasi[37], and "Personalised Information Retrieval: survey and classification", by Ghorab, M Rami *et al*. [21]. But none of them directly attempts the approach of this thesis.

Google has made an extension for their web browser Chrome called *Personal Blocklist*[17], which blocks domains/hosts from appearing in your Google search results. The description is as follows: "*The personal blocklist extension will transmit to Google the patterns that you choose to block. When you choose to block or unblock a pattern, the extension will also transmit to Google the URL of the web page on which the blocked or unblocked search results are displayed.*" The extension can be installed by anyone and can be used to permanently keep less relevant results away from Google on-line search, as long as Chrome is used as browser. The IIR prototype offers domain name black-listing, but also much more.

In a study that has some similarities, Karlsen *et al*. examines "*ranking of diabetes health videos on YouTube*[18] *over a time period, to learn whether videos from credible sources are ranked sufficiently high to be reachable to users*"[27]. Findings in this study indicate that many relevant videos (over time) consistently were given a low ranking, and thus less available to the user, even when querying

17. `https://chrome.google.com/webstore/detail/personal-blocklist-by-goo/nolijncfnkgaikbjbdaogikpmpbdcdef`
18. `http://youtube.com`

multiple times. Their conclusion was that new tools are needed for finding relevant and trustable videos.

Pocket[40] is a commercial system for saving video, images, text and other content, to read or watch later. Their motto is "Save for later, view when ready". Over 1500 applications support Pocket, which has more than 22 million users, saving more than 2 billion items. Their website sums it up. "*Save directly from your browser or from apps like Twitter, Flipboard, Pulse and Zite. If it's in Pocket, it's on your phone, tablet or computer. You don't even need an Internet connection.*" IIR could easily implement saving to Pocket in addition to, or instead of, saving search results in IIR.

Another interesting aspect of search is the invisible web[19] [20]. This is marginally related to IIR, in that information searched for in a "deep web" type search is not normally available through traditional on-line search.

19. https://www.lifewire.com/search-the-invisible-web-20-resources-3482497
20. http://deep-web.org/how-to-research/deep-web-search-engines/

3

Problem overview

When using traditional on-line search, the search engines will try to help the user find the most precise results according to what the user is searching for.

This means that for the same search, the results may be variations of the same list. Using the same search query will make it more difficult to find something new.

3.1 Query and QueryRun

In the Incremental Information Retrieval (IIR) solution, search will be carried out differently. The user sets up a search query, to be run by IIR several times. Each time the query is run, the results from the run are compared to results that were seen by the user in previous runs, to see if the user had been presented with those results before.

To this end, two terms have been devised, to be able to discuss these mechanisms more precisely.

Query To be able to compare results from one search with results from previous searches, every main search topic needs to be organised into a group, a Query. This represents a container for the textual query sent to the search engine.

QueryRun This represents a single search, a one time run of the textual query contained in the Query, where results are returned to the user.

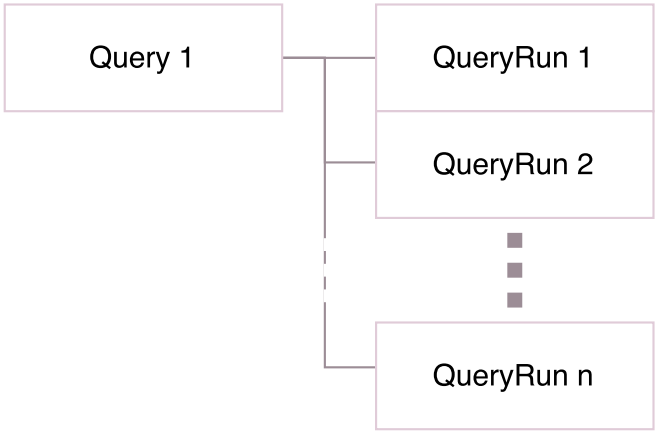


Figure 3.1: How Query and QueryRun are related.

3.2 Search example

In the course of working with this thesis, some characteristics of the results emerged. As further detailed in chapter 5, queries were executed every day to collect data through the Bing Search API. The collected result items showed a clear overlap from day to day.

The following illustrates a sequence of results from an imagined generic query. The concrete search text is not important here, only how the results appear over time. The query is imagined run once per day over four consecutive days. The results will vary, but not a great deal.

On the first day, shown in figure 3.2, all results are new, they have not been seen before.

On the second day, shown in figure 3.3, many results were already in search results from the first day, so only the new results here are interesting.

This accumulates to the third day, shown in figure 3.4, where the situation is similar, most search results were seen on the first and second day.

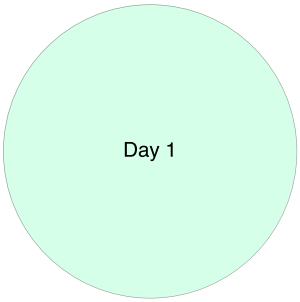


Figure 3.2: First batch of results

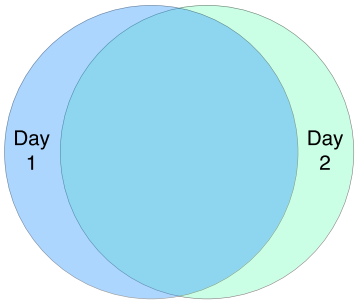


Figure 3.3: Second batch of results, where most of the search results the second day also appeared the first day.

On the fourth day, shown in figure 3.5, only a small amount of new results are present. Most of the results from day 4 have already been seen the previous days.

Each search for the same text query in the days following day four, will show the same pattern. The number of new results *will* vary, though, *e.g.* based on new information collected by the search engine's web crawler, see section 2.1.1. See also table 3.1 which shows variation in results after day one.

The pattern shown in figure 3.2 to 3.5 demonstrates that the search API show same or similar results each time the same search is executed.

This sequence of results is backed up by collected data. Table 3.1 shows the ten first days of data collected for Query 11, "winds of winter". This shows 96.4% new results the first day, and a dramatic drop in new results for the next days; 8.5%, 3.5%, 5.9%, etcetera. See also section 5.6.3 *Short analysis of new results*, and tables for this and other Queries in section A.2.2 in appendix A.

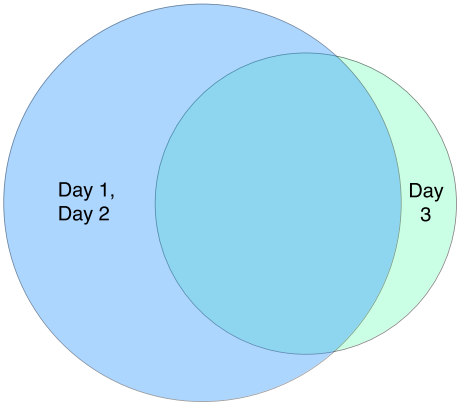


Figure 3.4: Third batch of results, where most search results from the third day still appeared the first and second day.

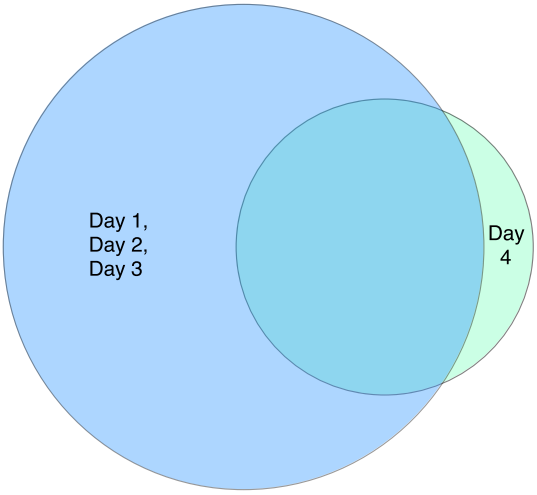


Figure 3.5: Fourth batch of results, where almost all of the search results have been seen the previous days.

Date	Day in period	Results	New	New %
2016.11.18	1	949	915	96.4
2016.11.19	2	999	85	8.5
2016.11.20	3	999	35	3.5
2016.11.21	4	999	59	5.9
2016.11.22	5	CT 948	62	6.5
2016.11.23	6	998	4	0.4
2016.11.24	7	997	0	0.0
2016.11.25	8	999	52	5.2
2016.11.26	9	1000	43	4.3
2016.11.27	10	999	3	0.3

Table 3.1: 10 first days of results for free Query 11, "winds of winter", showing a steep drop from the first to the second day of data collection. This is an excerpt from table A.10 in appendix A.

3.2.1 Results graph

Given the Venn diagram based description above, the expected curve for unique results is a rapidly descending curve, as shown in figure 3.6. The more times a search is run, the more results have already been found in previous runs.

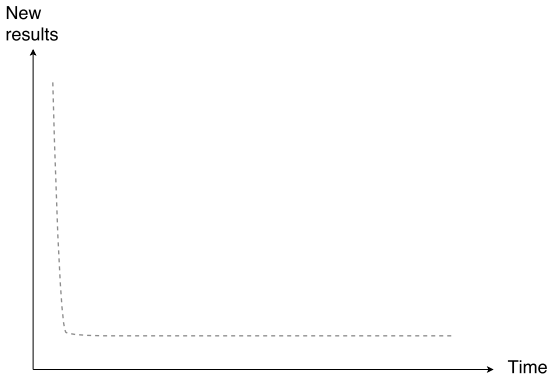


Figure 3.6: The expected curve for new results

The largest difference is between the first and second QueryRun. In the first QueryRun all results would be new, while in the second QueryRun a lot of the results found would also have been found in the first QueryRun, as indicated in figure 3.3.

3.3 User search

This section looks at new results from the viewpoint of the user, when the user searches, either through a traditional on-line search engine, or through IIR. In this context, *new* results means results that were previously unseen by the *user*. In figures 3.7 and 3.8, the green area represents new results coming from the search engine, that the *user* has not seen before. Note that new results in figure 3.7 (green area) may well also be found in new results in figure 3.8 (green area), as opposed to figures 3.2 - 3.5, where the green area represents new results that are not seen in any of the previous searches.

An example of the first search the user does is shown in figure 3.7. It contains many results, of which the user usually browses through the first page and maybe the second or third[10], and therefore sees only a few results.

Figure 3.7: The first time a user searches, (s)he only sees a small part of the result.

When the user does the exact same search a second time, the user may see some more results, and likely some of the same results as the first day, see figure 3.8.

Figure 3.8: The second time a user searches, (s)he still sees only a small part of the result, but has in total seen more than in the first search.

Even though many of the results in the second search were same as the first search, as shown in figures 3.2 - 3.5, the *user* doing the search has not seen all of them. The total number of new results (previously unseen by user) stays high, but the normal user pattern shown in the Chitika study[10] shows that the user rarely bothers to browse beyond the first page of results.

The problem originates from the search engine's ranking mechanism. As seen in the study by Karlsen *et al*. [27], ranking of results are consistent. Their study was conducted with YouTube as data source, but it is plausible that this applies to search engines as well. See also section 5.6.4, which has some insights into this for some of the Queries run through the Bing Search API v2. Rankings are consistent because search engines want to give the most relevant results for the user. When the user gets a search result from a search engine, the set of results returned often consists of most of the same results, ranked mostly the same way. This masks other possibly relevant results by pushing them to page two, three or further back in the list of results coming from the search engine. As a consequence, the user gets few previously unseen results among the top-ranked results.

If the user used a traditional on-line search engine to do the search, the number of previously unseen results are practically unlimited. Even if the number of results is finite, *e.g*. 2 million results, the user is not expected to page through all of them. So for all intents and purposes, the number of previously unseen results are infinite.

When searching through the search APIs by the major search engines, however, there is a much lower limit to how many results are returned. For Bing Search API v2 this was 1000 per search, the other search engines had similar limits.

3.4 Goal

This thesis will explore the mechanism shown in section 3.3 *User search*, and find a way to show only the previously unseen results, the "green bits" of the Venn diagrams, and hide results the user has already seen. This is accomplished by implementing a software solution to discard the results seen before.

4

Architecture and design

This chapter will outline the architecture and design for the Incremental Information Retrieval (IIR) system.

4.1 Design goal

The goal of the IIR system is to help the user to find information that normally is hidden several pages down in an on-line search, using one of the traditional on-line search engines, as described in section 2.2.

The design goal of the IIR system is to give the user opportunity to save and discard results, to white-list domain names they find that always contain interesting information, and black-list domain names that never contain interesting results.

It is important to note that this IIR system is not intended to replace ordinary on-line search. The IIR system can complement the ordinary search, and can be used when a deeper probe is appropriate.

Note that when referring to the result status *Seen*, it is shown in *italics*, see table 4.2.

4.2 Architecture

The IIR system is implemented as a web service, that will connect to a search engine API, extract search results from it based on a user's query, and save data, including some meta data in the web service database. This will enable the web service to discard incoming results for the same query, if needed.

Looking at figure 4.1, the left side of the figure represents the on-line search engine, and the right side represents the IIR system.

The user A is using a traditional on-line search engine via its on-line user interface. This type of use will get the ordinary immediate answers from the search engine, and will have the normal search experience that the on-line search engine offers.

User B is using the IIR search service. The IIR service differs in that it has its own separate user interface, and interacts with the user in a slightly different way, compared to traditional on-line search.

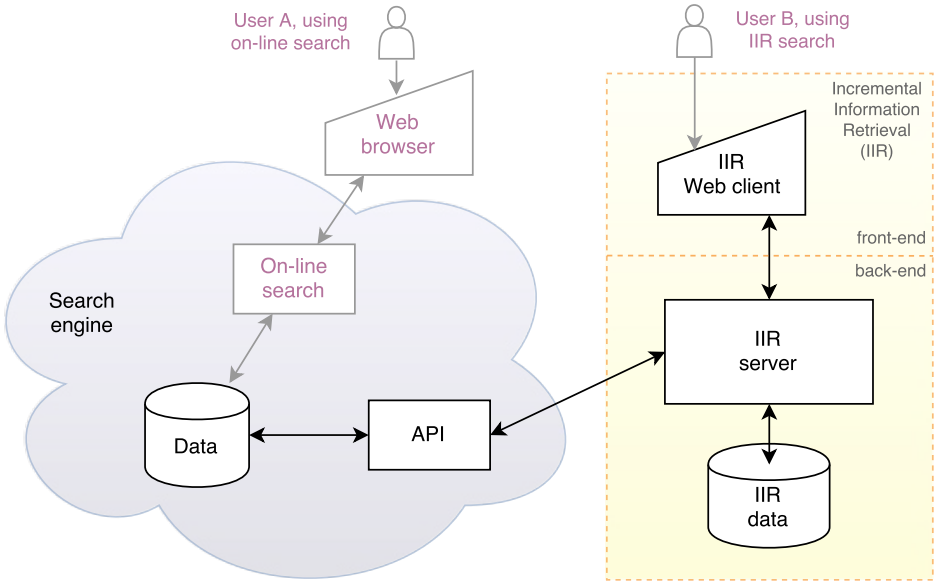


Figure 4.1: IIR system at a glance.

When using the IIR service, the user will interact with the results by saving interesting results or discarding uninteresting results. In addition to user actions, the system could analyse the user's interactions with the system, and

manipulate the results based on these interactions. The goal of the IIR system is to help the user find new information without the user having to relate to results they have already seen and found uninteresting.

The intent of the IIR user interface is to make it as easy as possible for the user to use the system. This means doing smart analysis of user actions, and giving the user some simple but smart functionality for marking and archiving results after a query.

4.3 How IIR search works

The user will create a Query, with a text to search for. IIR will do calls to the search engine API, retrieve data for the user, and present the results in the IIR user interface. The user can browse through the results, save or discard results, or white- or black-list results. When the user uses IIR to do subsequent searches for the same Query through the search engine API, these results are compared with the previous results for this Query. The purpose is to make sure that the user only sees new results, *i.e.*, results that the user has not seen before. To be able to do this, all results returned for a search is stored in a database for comparison and reference.

As described in section 3.1, every main search topic needs to be organised into a Query. When doing further searches on a topic, the topic's group must be selected or referenced, so IIR knows what previous results to compare the incoming results to.

The reason for grouping results into a Query, is that a hypothetical topic *A* and topic *B*, theoretically could contain the same result item *R*. This result item could be uninteresting for topic *A*, but interesting for topic *B*.

If topics A and B were not in separate groups, the result *R* would not appear for topic *B* if it had been discarded while searching for topic *A*.

4.3.1 Search details

When a search is initiated by the user by using the IIR web client, the request goes to the IIR server, which consists of several mechanisms, working together to handle the requests, see figure 4.2.

The data retrieval component is responsible for connecting the search engine API and retrieving results from the search engine. The results are then compared to already existing results for the same Query in the filtering process.

This is where already seen results are removed from the API results. After filtering, a check is done to see if these API results have domain names that are white-listed or black-listed. If they are black-listed, they are ignored, if they are white-listed, they are saved as white-listed. Then a proprietary IIR ranking (see section 4.4.2) is applied, after which the remaining results are sent to the IIR client for display.

Note that figure 4.2 is more a conceptual view of the mechanisms, in the actual implementation of the search, these mechanisms are grouped together. More on this in chapter 6.

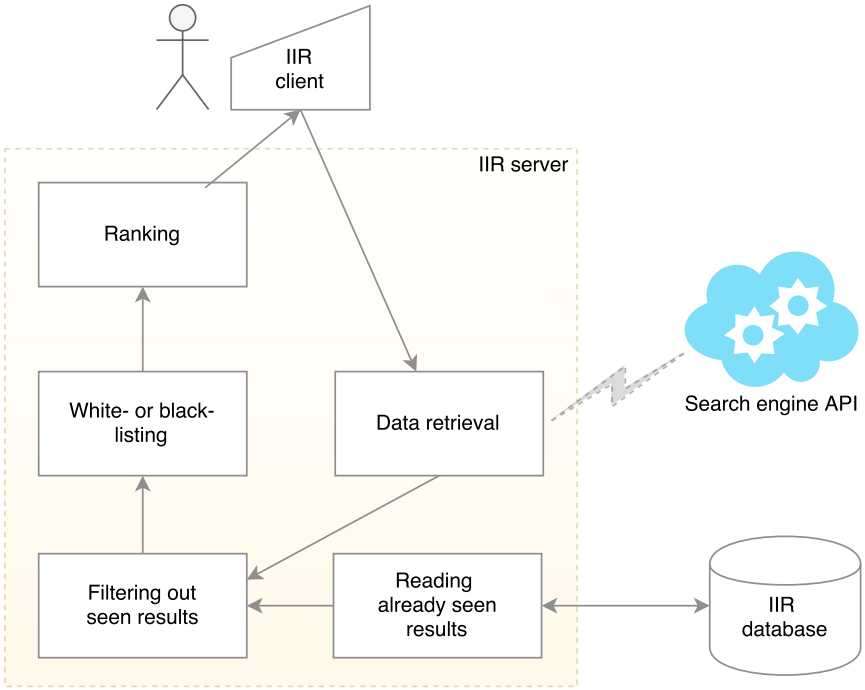


Figure 4.2: User using IIR to search. IIR connects to the search engine API, and returns refined results to the user.

4.4 IIR features

What an IIR user actually is doing, is categorising the results retrieved from the search engine. So the main features of IIR are designed to help the user save, discard, white- or black-list retrieved results, see table 4.1 for user actions that the user can apply to new results.

Feature	Description
Save single result	Inside every shown result boundary is a button for saving this result.
Discard single result	Inside every shown result boundary is a button for discarding this result.
Filter results by text	A text box gives the user a chance to narrow the results by searching.
Save filtered results	This is a button that saves all the filtered results. If no filter text is entered, all results are saved.
Discard filtered results	This is a button that discards all the filtered results. If no filter text is entered, all results are discarded.
Filter by Query search text	When the query is loaded, the Query's search text is shown as a clickable text. When clicked, the Query search text is entered into the filter text box, and the result list is automatically filtered accordingly.
Clear filter	Removes the search text from the text box, and refreshes the list of results.
Filter by domain name	By clicking on the shown domain name, the domain name is entered into the text box, and the result list is filtered automatically.
White-list domain name	A small button with an up-arrow symbol, shown behind the domain name, lets the user white-list this domain name.
Black-list domain name	A small button with an down-arrow symbol, shown behind the domain name, lets the user black-list this domain name.
Paging	IIR contains a paging bar, with page numbers, that lets the user leaf to *first*, *previous*, *next*, and *last* page. The user can also choose to go to page *n*. Page size is 15 results per page.
Change sort order	A button is available for changing the sort order of the results. Options are a) IIR sort order and b) original search engine sort order.

Table 4.1: IIR user actions available on a list showing only new results.

Number of results shown per page is 15. The page length of shown results should not be too large, and not too small. If it is too large, the user may lose overview of the results, if it is too small, the user needs to work through more pages.

The user can choose to save or discard a result, and a status of this is written back to the database. If the user just pages through the result, the results that are paged past are saved as *Seen*.

The user can then watch the saved result in a special saved results list, or the discarded result in a special discarded results list. The latter can be useful if the user *e.g.* makes an error and discards a result instead of saving it.

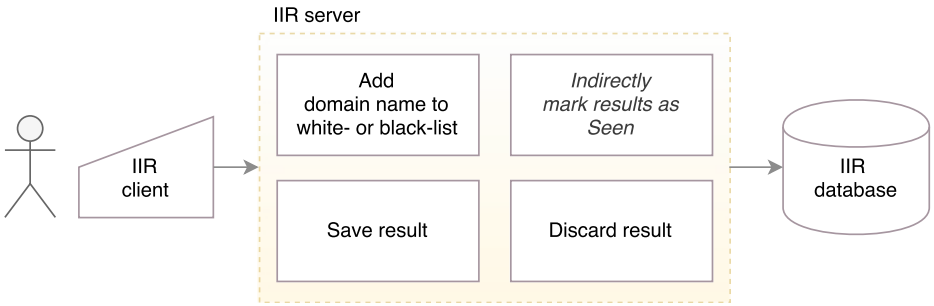


Figure 4.3: Main features available to the user from the UI of IIR.

There are also several mechanisms to filter the results. Domain names in the results are registered, and if the user adds the domain name to a black-list, the results with this domain name is not shown any more. Similarly, the user can choose to approve a particular domain name. The results with this domain name is white-listed, and is listed separately.

If the user has seen a result without doing anything with it, it is automatically marked as *Seen*. This is an indirect mechanism, for the situation where the user does not really know if the results showing are important or not. When IIR is showing result page p and the user goes to page $p+1$ or page $p+x$, results on page p, the page the user is leaving, are updated with the result status *Seen* by the IIR web client.

The user can choose to show results ordered by the proprietary IIR ranking, or in the original sort order the results had when they were returned from the search engine.

4.4.1 Result status

Each result can have one of several possible statuses, based on the mechanisms described in sections 4.3 and 4.4.

Status	Description
New	When the result is returned from the search engine, ranked and presented to the user, it has the new status. This is the first time the user sees this result.
Seen	A result gets the seen status when the user has seen the result, but not acted upon it. Note that when referring to the result status *Seen*, it is shown in *italics*. *Seen* results are always updated automatically, results cannot be set as *Seen* directly by the user.
Saved	If a user finds a result relevant, the user can save it for further inspection. This must be done specifically via the user interface.
Discarded	If a result is not found interesting, it can be filed as discarded. This must be done manually by the user, via the user interface.

White-listed	If a user white-lists a domain name, the results with this domain name will get the status *White-listed*. This will happen when the user white-lists a domain name by using the UI, but also automatically filtered and saved as white-listed when reading results from the API, see figure 4.2.
Black-listed	If a user black-lists a domain name, the results with this domain name will get the status *Black-listed*. This will happen when the user black-lists a domain name by using the UI, but also automatically filtered and saved as black-listed when reading results from the API, see figure 4.2. This status is mainly for testing the prototype, in a fully implemented system, the result would just be removed.
Automatically discarded	If a result in the current batch of results returned from the search engine has been seen before, it gets auto-discarded. This will also happen if the filtering component shown in 4.2 finds duplicates in the result returned from the search engine. This status is mainly for testing the prototype, in a fully implemented system, the result would just be removed.

Table 4.2: A result's possible statuses.

4.4.2 Ranking

Search engines are ranking results, to make sure the most relevant results are shown first. This takes into account the user's spelling errors and other issues, using mechanisms described in section 2.1, and specifically 2.1.4.

Personal experience shows that sometimes this ranking does not show relevant results according to the text being searched for. So as an experimental feature, a simple IIR ranking of results is introduced. It is optional and can be turned off. When turned off, IIR shows the filtered results in the order of the original search engine ranking.

A numeric word rank is introduced for each result, where different ranking scores are given according to how many of the search words are found in the result.

The weighting of a result is based on criteria described in the following list. The sum of the relevant numbers gives a total score for the result.

Weighting	Description
50	The title of the result contains the exact phrase searched for.
30	The description of the result contains the exact phrase searched for.
15	If the exact phrase is *not* found in the title, the title is checked for all words appearing, though not in the exact order.
10	If the exact phrase is *not* found in the description, the description is checked for all words appearing, though not in the exact order.
1	If *none* of the above, each search word actually appearing in title or description is given one point.

Table 4.3: Weighting scores for IIR search word ranking.

4.4.3 Result folders

The results will have different statuses, as described in table 4.2. A good way to separate these different categories of results, is to place them in folders, as shown in table 4.4.

Folder	Description
New results	This is the main folder, which shows results for the latest QueryRun only. New results from the search engine arrive in this folder, after filtering and ranking shown in figure 4.2. This is also where user actions shown in 4.3 are available, as well.
Seen results	This folder contains the results with the *Seen* status. Refer to page 28 for an explanation of the *Seen* status.
Saved results	When the user saves results, this folder is where the *Saved* results can be found.
White-listed results	If the user white-lists domain names, this folder is where results with the white-listed domain names are located.
Discarded results	Manually discarded results end up here. In the test phase, all types of discarded results ended up in this folder; Manually discarded results, Auto-discarded results (duplicates and already seen results), and Black-listed results.

Table 4.4: Folders to show results with different statuses.

4.5 Client design

The design approach for the user interface is a straightforward four part layout. Figure 4.4 shows a top area and a bottom area, with a two part main area in the middle where most of the functionality is located.

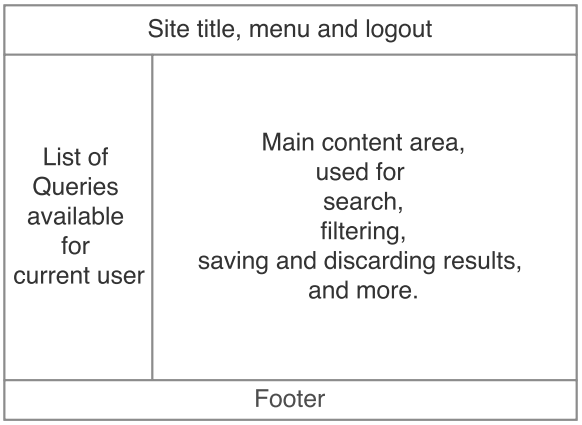


Figure 4.4: The main user interface

Header and footer of the page is mostly left alone after login, the operational part of the system is the middle left and right area.

The middle left area contains a list of the user's registered Queries, the middle right area contains all results and manipulation features for the results, as described in section 4.4.

4.6 System requirements

Armed with more knowledge about what IIR is going to solve, some requirements for the system can be defined.

The usage and features of the IIR prototype is quite limited, so at this stage the IIR server will not need much in terms of computing infrastructure. A single server will do, there will not be many users using the IIR prototype.

4.6.1 Web server hardware

The IIR prototype does not require much in terms of hardware, and should even be able to run *e.g.* on a laptop, enabling IIR prototyping and development on available laptops and desktop computers.

But deploying the solution for wider use will make more demands when it comes to infrastructure. The best way of setting up the infrastructure might be renting it in the cloud, either Infrastructure as a service (IaaS), or Platform as

a service (PaaS), depending on the finished implementation[9]. By choosing a cloud infrastructure, the finished IIR solution has room to grow.

Amazon Web Services[1] (AWS), Google Cloud Platform[2], or Microsoft Azure[3] are all good platforms. This will ensure safe storage and scaling of the solution. Microsoft Azure has for practical reasons been chosen as the platform for the testing phase of this project.

4.6.2 Storage

The IIR system needs to store the user's results, so it needs a database. What needs to be stored is information about the user, the user's Queries, the information about all the user's QueryRuns, and all the results the user interacts with, as described in section 4.4.

Trying to speculate on how much data a user will accumulate can be difficult. It depends among other things on how active the user is, *i.e.*, how many Queries the user has in the system, how many times each Query has been run against the search engine, how many results are seen, saved, white-listed, or discarded.

Result storage

The results would require the largest share of the storage, mostly because they are so many, so calculating their storage is key.

From available test data, a quick preliminary calculation shows how much storage could be needed for each result. A set of 302, 649 results uses a storage of 282, 148, 976 bytes, which means that average size per result is 282, 148, 976/302, 649 = 932.3 bytes.

The result size of 932.3 bytes is used for calculations in sections *IIR storage* and *IIR testing scenario storage* below.

IIR storage

Storage is a direct consequence of usage, so usage pattern is key here.

1. `https://aws.amazon.com/`
2. `https://cloud.google.com/`
3. `https://azure.microsoft.com/`

As an example, take one user's single Query, and examine its possible usage. Let the Query run daily for a year), creating a QueryRun daily, and let every QueryRun *e.g.* return on average a thousand results. Even though IIR is not a traditional search engine, let us assume that the user follows the usage patterns of the Chitika study [10], and only looks at the first and maybe the second page of results. When a QueryRun returns its results, the user saves 10 results and discard 20 results, and white-lists 2 domain names and black-lists 4 domain names. Then the user goes to page two, and finds nothing in page two. Then the user stops using IIR (closes the browser).

This would result in

1. 5 saved results
2. 10 discarded results
3. Let us for simplicity assume that there were 5 results from the white-listing action, though the number would depend on results and what domain names actually were white-listed.
4. No black-listed results saved. These results would be deleted, and the domain names saved to the black-list configuration for the Query.
5. 15 *Seen* results were stored when the user paged to the next page. When the user closed the browser, no results were marked as *Seen*.

In the following, let S_q, S_{qr} and S_r denote storage for Query, QueryRun and result. Let R denote number of results.

The estimated number of results for this single QueryRun is

$$R_{\text{QueryRun}} = 5\ \textit{Saved} + 10\ \textit{Discarded} + 5\ \textit{White-listed} + 15\ \textit{Seen} = 35$$

When the Query is executed for a year with a daily QueryRun, and with the same estimated usage of 60 results, the number of results is

$$R_{\text{Query}} = 365 \times R_{\text{QueryRun}} = 12\,775$$

If a user has 100 Queries, the yearly number of results for this user would be

$$100 \times 12\,775 = 1\,277\,500$$

With a size of 932.3 bytes per result, as outlined in section *Result storage* above, the total storage per user could be $932.3 \times 1\,277\,500 = 1\,277\,500\,000$ bytes, which amounts to 1.2 Gigabytes of storage per user per year.

Note that this is a maximised calculation for all Queries, most likely few users are this active, or follow this pattern. Usage will also vary according to type of Query, number of results paged by, saved or discarded, and how many times the Query is run, and the size of each returned result will vary, as well.

Still, this is storage for a single user, so if this scales to thousands or even millions of users, storage needs to be handled carefully.

IIR testing scenario storage

For test data, storage is different. 40 Queries have been set up, and were executed daily over a period of 54 days (QueryRuns). Assume maximum 1000 results per Query per day, and a result storage of 932.3 bytes as outlined in section *Result storage* above. This gives

$$40 \times 54 \times 1000 \times 932.3 = 2\,013\,768\,000$$

This means the maximum storage requirement is 2 013 768 000 bytes for the prototype testing, *i.e.*, 2 Gigabytes.

In chapter 5 there are concrete numbers for storage, and with a result size of 932.3 the total is $1\,444\,330 \times 932.3 = 1346529859$ bytes for the prototype testing, *i.e.*, 1.3 Gigabytes.

Note that this is an approximation, since the result size of 932.3 is an estimated size.

4.7 Selecting search engine to work with

The three major general purpose web search engine APIs have been examined, and the chosen candidate is the Bing Search API v2, for practical reasons.

The main argument is that Bing Search API v2 had an initial 5000 free searches per month, which would come in handy for development and initial testing. Bing Search API v2 also had the largest maximum number of results, 1000 results per query. Google Custom Search (GCS) also had free searches, but they were not as many as Bing, and GCS was made for searching limited number of sites anyway. The Yahoo BOSS API had costs associated from the very first query.

This project also has been designed in such a way that it is possible to change data source with a minimal amount of work. Ideally, the finished IIR solution

could have attached any number of APIs or other data sources, so that different result generators can be used. This means that the selected search engine could be swapped out for another source at some future point. So for the purpose of implementing the IIR prototype, the search engine selected was not a central concern, but more a practical issue.

4.8 Hypothetical search progression

In section 3.2 *Search example*, a progression for search results returned from the search engine was described, showing how search results accumulate over time for the same search query.

Using the different statuses described in section 4.4.1, we can hypothesise about how result statuses would be distributed for each consecutive search using a generic search text.

This hypothetical search progression is based on the data collected from Bing via Bing Search API v2. Data collection is described in greater detail in chapter 5, but for now it is important to recognise that the maximum number of results for each search scenario in this section, figures 4.5 - 4.8, is 1000 results.

Each of the searches in the figures 4.5, 4.6, 4.7 and 4.8 below have a start and an end scenario. For each figure, the start scenario is to the left, marked with an **a**. It shows how the search results distribute when fresh from the search engine.

The end scenario is to the right, marked with a **b**. It shows how the search results distribute after the user has used the IIR web service, working with the results. The end scenario can be regarded as a snapshot of the system just after the user presses the search button again to get more results from the search engine, but before the next search results arrive.

When the user searches again, all results still in the *new* column could theoretically be found again in the next search's *new* column. As an example consider new results from 4.5b's *new* column partly being found again in 4.6a's *new* column. This will happen for all "b" and next "a" figures shown below.

In figure 4.5a, the search query has been executed for the first time. Some results are auto-discarded as duplicates when different results have the same URL.

The user then looks at the results, decides that some results can be saved, and some discarded. The user has white-listed some domain name(s), and black-listed others, which updates the relevant results with the corresponding statuses *white-listed* and *black-listed*. The end scenario, after user browsing and user actions, can be seen to the right in figure 4.5b.

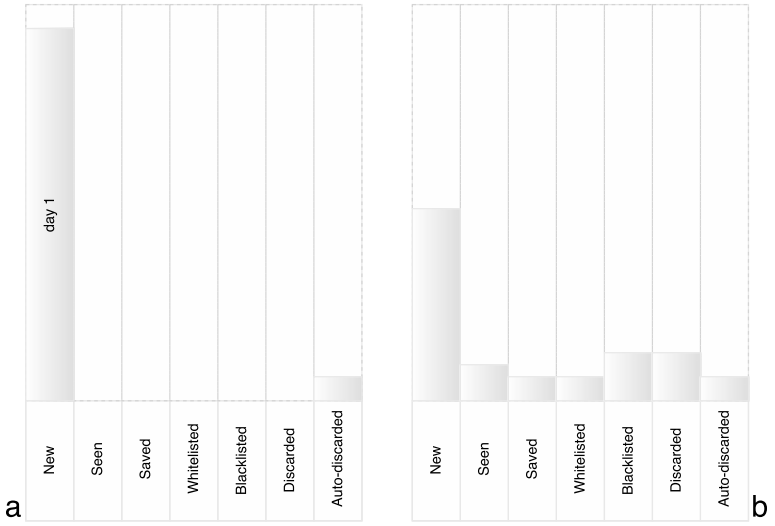


Figure 4.5: Hypothetical search progression, search 1. Results for day one are shown in grey. Start scenario is to the left, marked **a**, end scenario is to the right, marked **b**.

When the search query is executed for the second time, new results pour in and are compared to previous results, and also to white-listed and black-listed domain names. This can be seen in figure 4.6a, where many results are auto-discarded because they were seen in the first search, or they are automatically white-listed or black-listed.

In the same way as for the first search, the user works with the results, saves or discards some more results, and white- or black-lists even more domain names. This can be seen in figure 4.6b.

The same kind of progression follows for the third search, as shown in figure 4.7a. Even more results are automatically given appropriate statuses in the incoming results, based on results already seen or white- or black-listing.

After the user has worked with the third batch of results for a while, even more results have been given appropriate statuses, as shown in figure 4.7b.

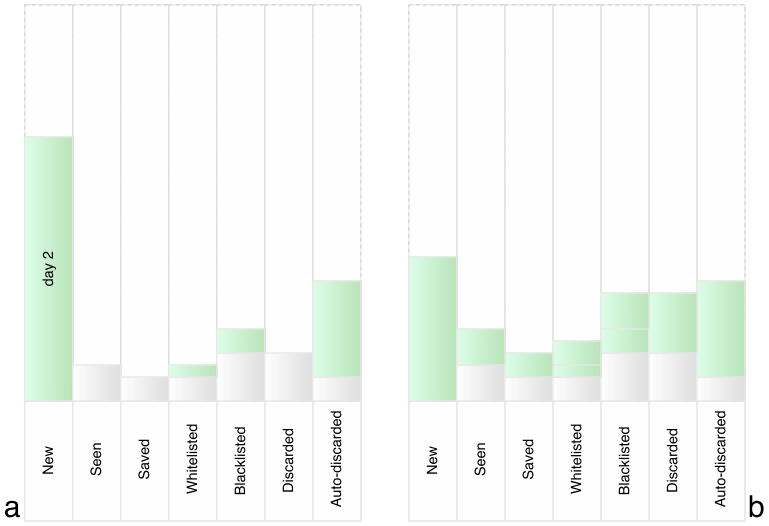


Figure 4.6: Hypothetical search progression, search 2. Results for day one are shown in grey. Results for day two are shown in green. Start scenario is to the left, marked **a**, end scenario is to the right, marked **b**.

In the fourth search, shown in figure 4.8a, the new results are not that many, since many of the results have been seen before.

After the user has worked with the results, shown in figure 4.8b, all *new* results have been classified according to the statuses in table 4.2 in section 4.4.1.

In the following searches, the user can expect even fewer results with the *new* status than what is shown in figure 4.8a (left). The number of results will most likely vary according to updates in the search engine's index.

This progression closely resembles the scenario in testing the prototype, and in any connection to a data source that supplies IIR with maximum 1000 results, or a similar small final number of results.

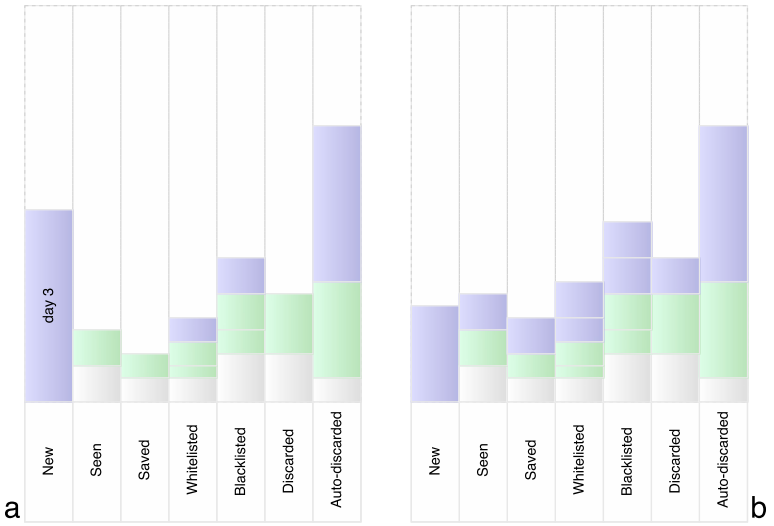


Figure 4.7: Hypothetical search progression, search 3. Results for day one are shown in grey. Results for day two are shown in green. Results for day three are shown in blue. Start scenario is to the left, marked **a**, end scenario is to the right, marked **b**.

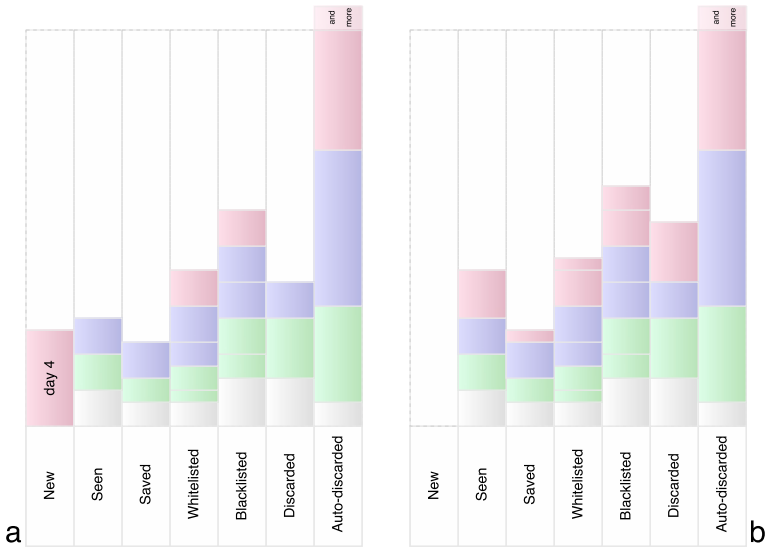


Figure 4.8: Hypothetical search progression, search 4. Results for day one are shown in grey. Results for day two are shown in green. Results for day three are shown in blue. Results for day four are shown in red. Start scenario is to the left, marked **a**, end scenario is to the right, marked **b**.

5

Data collection and analysis

In this chapter data collection is described. Refer to section 5.1 for the background on why data collection was necessary.

5.1 Why data collection

Bing search API *v2* was chosen as search tool for this thesis in section 4.7. After some time had gone into working on this thesis, news broke that Microsoft had decided to change the structure and price model of their search API. This would result in Microsoft shutting down the *v2* version of the search API, and remaking it as a new version (*v5*), with new functionality, reworked structure and increased cost. This new API offering is a part of Microsoft Cognitive Services[1].

This meant that a decision needed to be made; either discard a lot of work already done and start again with the new and radically updated version of the search engine API[2], or continue to use the *v2* search engine service

1. https://www.microsoft.com/cognitive-services
2. https://msdn.microsoft.com/en-US/library/mt707570.aspx

until its termination. In cooperation with the supervisor, the latter option was chosen.

Another point of view is that it is an advantage to having a fixed set of data as input to the testing process, as long as the data set is large enough. This would facilitate repeated tests of the IIR system. The data would be the same, so the main variable in the system would be the behaviour of the IIR system.

As a consequence of these two circumstances stated above, data was collected in the period from November 2016, beyond the official service termination 14th of December 2016 and until 10th of January 2017.

As a side note, the Bing service was finally closed down 31st of March 2017.

5.2 Queries

Friends have been recruited to contribute some information needs, to vary the results found.

A total of 20 Queries was collected. These would be used as search queries for the search API. Each Query was submitted both as a non-exact (free) query, and as an exact phrase query, to see if this would give variations in the results.

This made a total of 40 queries, one free and one exact for each of the 20 contributed queries. The full list of searches is displayed and explained in table A.1 in appendix A.

ID[3]	ID[4]	Search text	Contributor
1	2	Messerschmitt KR200 restoration	Person A
3	4	Web search API thesis	Person A
5	6	Web search thesis	Person A
7	8	Search API thesis	Person A
9	10	Messerschmitt TG500 for sale	Person A
11	12	winds of winter	Person A
13	14	promise of spring	Person A
15	16	terry pratchett	Person A
17	18	liverpool leeds efl	Person B
19	20	hillary clinton e-mail fbi	Person B
21	22	macbook pro 2016 touch bar problems	Person B
23	24	apple stock price	Person C

3. Odd-numbered IDs are "free" searches
4. Even-numbered IDs are exact searches

25	26	samsung note 8 release date	Person C
27	28	google self driving car	Person C
29	30	mobile application health sensor data	Person D
31	32	mobile phone body area network	Person D
33	34	mobile phone sensor research health	Person D
35	36	forest fairytales	Person E
37	38	tudor politics	Person E
39	40	jazz poetry	Person E

Table 5.1: Summary of automated searches, run for 54days.

5.3 The anatomy of a query

There are four main parts to collecting the data, the *Query*, the *QueryRun*, the *Transaction* and the *Result*.

Part	Description
Query	This is where the textual query sent to the search engine is stored. In addition to the actual query text, properties of a query include a numerical id, first run date, latest run date and if the query is searches for an exact match or not. See also section 3.1.
QueryRun	A QueryRun is created every time a Query is run, and contains properties like ID, date of run and how many results collected for this run. See also section 3.1.
Transaction	This is the actual call to the Bing Search API. Each transaction would return a set of maximum 50 results, a "page". There are 20 transactions run per QueryRun.
Result	This contains the actual response from the search engine. All results gets a Query ID and QueryRun ID in addition to essential properties returned from Bing, such as title, URL and description. See also figure 5.2.

Table 5.2: The "moving parts" of the data collection system.

As described in section 3.1, a Query is a container for the search query, a QueryRun represents a single run of a textual query, and is built from up to 20 API transactions, which could have a maximum of 50 results each.

Each transaction is called with an offset start value, like 51, 101, 151 etc., and the maximum offset allowed would be 951, to make the potential maximum number results of a thousand (1000) for the given Query.

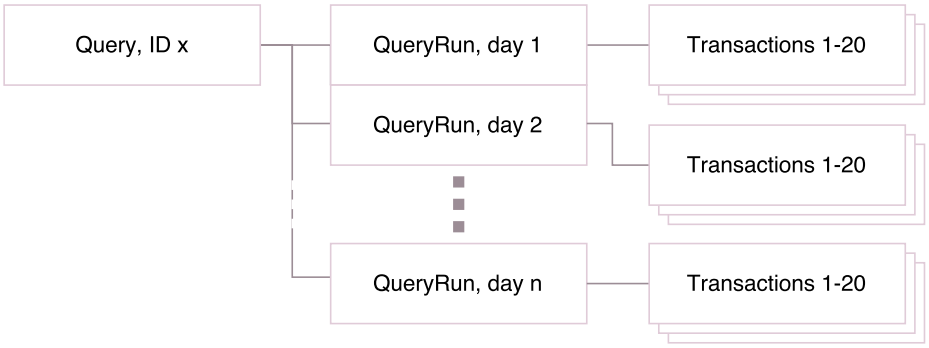


Figure 5.1: A Query consists of QueryRuns, which have of up to 20 API transactions, which in turn could have a maximum of 50 results each.

5.4 Data collection environment

For the data collection a program was run in batch, on a daily schedule. A collection of 40 searches (Queries) was run against the Bing Web Search API every day at 12:00 CET.

5.4.1 Hardware

All batch runs are done on a Microsoft Windows 10 Pro desktop computer with an Intel Core i7-5820K processor, 16 GB RAM and an SSD disk. It is connected to a 60 Mbit (in the last stages of the run a 100 Mbit) network from Canal Digital, a solid and trustworthy Internet Service Provider (ISP).

5.4.2 Database

The database is of the NoSQL[5] variety, see more about the database in section 6.1.1. There are two main collections in the data collection database, queries and results. Queries contains the search text, and for each time the query is run, an entry is added to the sub-collection of QueryRuns. The other collection contains the *results* which holds the results from the search, with added keys from Query and QueryRun to connect the two collections.

5. http://wiki.c2.com/?NoSql

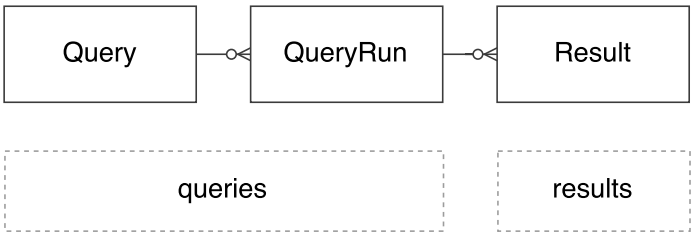


Figure 5.2: Data collection database entities, and corresponding database collections.

Listing 5.1: Go struct describing the data collection Query

```
// Query is the container for the search
type Query struct {
    // Query ID
    ID                bson.ObjectId    `bson:"_id"`

    // Simple numeric ID for the Query
    SimpleID          int              `bson:"simpleId,omitempty"`

    // Name of the Query
    Name              string           `bson:"name,omitempty"`

    // Actual search text to send to search engine
    Query             string           `bson:"query"`

    // If true, the search text will be enclosed by
    // quotation marks when sent to the search engine
    IsExactSearch     bool             `bson:"isExactSearch"`

    // When was the Query created
    CreatedDate       time.Time        `bson:"createdDate,omitempty"`

    // When was the Query last run against the search engine
    LastRunDate       time.Time        `bson:"lastRunDate,omitempty"`

    // Sub-collection containing QueryRuns
    QueryRuns         []QueryRun       `bson:"queryRuns,omitempty"`
}
```

Listing 5.2: Go struct describing the data collection QueryRun

```
// QueryRun contains information on when the query is run
type QueryRun struct {
    // QueryRun ID
    ID          bson.ObjectId `bson:"_id"`

    // Date of search execution
    Date        time.Time     `bson:"date,omitempty"`
}
```

Listing 5.3: Go struct describing the data collection Result

```
// Result is the actual result
type Result struct {
    // Result ID
    ID          bson.ObjectId    `bson:"_id"`

    // Query ID
    QueryID     bson.ObjectId    `bson:"queryId"`

    // Query simple ID
    SimpleID    int              `bson:"simpleId,omitempty"`

    // QueryRun ID
    QueryRunID  bson.ObjectId    `bson:"queryRunId"`

    // Bing meta-data, just the actual URI run, and result-type
    MetaData    bingapi.MetaData `bson:"metadata"`

    // A Guid returned from Bing to identify the result
    IDBing      string           `bson:"idbing"`

    // Title of result from search engine
    Title       string           `bson:"title"`

    // Description of result from search engine
    Description string           `bson:"description,omitempty"`

    // DisplayURL of result from search engine
    DisplayURL  string           `bson:"displayUrl,omitempty"`

    // URL of result from search engine
    URL         string           `bson:"url"`

    // URL hashed to make it easier to compare
    URLCRC      string           `bson:"urlcrc,omitempty"`

    // Time of search execution
    Time        int              `bson:"time,omitempty"`

    // Index holding order of returned result from search engine
    Index       int              `bson:"index,omitempty"`
}
```

5.4.3 The batch job

The batch runs were started from the Windows Task Scheduler which executed a Windows script command file that executed the data collection.

A Go program was developed and set up to execute one Query at a time, by referring to the Query's numeric *SimpleId*. The Query was run (*i.e.*, a QueryRun), performing 20 transactions to the Bing search engine API per Query. Each transaction would start at a 50 result offset from the previous, yielding 50 results for each transaction, with a potential total of 1000 results for the query.

This program would be called 40 times, each time with a different Query *SimpleId* as parameter.

Listing 5.4: Running the queries in a command file

```
@echo off
bingv2batch -id  1
bingv2batch -id  2
bingv2batch -id  3
...
bingv2batch -id 40
```

After all the queries were finished, a backup would be created of the full database by exporting the relevant collection to files, and then stored off-site. The actual command file was much more comprehensive than implied in listing 5.4, with logging and compression.

5.4.4 Updating the database

Before starting batch runs, there is only the *queries* collection, with the queries configured for the search. An example of one such query is outlined in listing 5.5.

Listing 5.5: The starting configuration of the query, before any searches are run

```
{
   "_id" : ObjectId("581647b0e1caad25dc08a1e5"),
   "simpleid" : 27,
   "query" : "google self driving car",
   "exactsearch" : false
}
```

After running 3 batch runs, this record is expanded with information on the queries that has been run, see listing 5.6. The QueryRun has an id and a date for the run.

Listing 5.6: The same query after 3 runs, with more information from the runs

```
{
   "_id" : ObjectId("581647b0e1caad25dc08a1e5"),
   "simpleid" : 27,
   "query" : "google self driving car",
   "exactsearch" : false,
   "queryruns" : [{
      "_id" : ObjectId("58187611e1caad4338e04ee1"),
      "date" : ISODate("2016-11-01T11:01:37.265+0000")
   }, {
      "_id" : ObjectId("5819c7a1e1caad23a4cff95f"),
      "date" : ISODate("2016-11-02T11:01:53.541+0000")
   }, {
      "_id" : ObjectId("581b1914e1caad3958a3b94a"),
      "date" : ISODate("2016-11-03T11:01:40.096+0000")
   }]
}
```

When the query is run for the first time, the *results* collection is created, and results from the search are inserted. See listing 5.7 for one such result.

There are 20 queries run twice, one free run and one exact run, as described in section 5.2. Each query consists of 20 transactions (or "pages") with a maximum of 50 results each, giving a potential maximum of 1000 results per query per run.

With q as queries and r as results, this means a potential maximum of

$$40q \times (20 \times 50r) = 40q \times 1000r = 40000$$

results for each day of the data collection period.

Preliminary tests have shown that results per run is not always close to the maximum, but normally somewhere between 500 - 1000 results per query per run, and even lower, depending on how the query is built, and the results available from Bing.

This means that for every run there would be maybe somewhere between 20,000 and 40,000 results added to the results collection. With a run period of 54 days this resulted in a collection of 1 444 330 results to run analysis and testing on. With 40 Queries, average number of results per Query is approximately 36108, and average number of results per QueryRun is approximately 669.

The first run for query 27 had 823 results, and the first of these results are shown in listing 5.6 (some fields edited for brevity).

Listing 5.7: The first out of the 823 results for query 27 on the first day

```
{
   "_id" : ObjectId("58187612e1caad4338e05187"),
   "queryid" : ObjectId("581647b0e1caad25dc08a1e5"),
   "queryrunid" : ObjectId("58187611e1caad4338e04ee1"),
   "simpleid" : 27,
   "metadata" : {
      "uri" : "https://api.datamarket.azure.com/Data.ashx/Bing/Search[...]",
      "resulttype" : "WebResult"
   },
   "idbing" : "ed2d4424-9c8f-4dad-b94c-d734d24c9ccf",
   "title" : "Google␣Self-Driving␣Car␣Project",
   "description" : "Our␣self-driving␣cars␣are␣designed␣to␣navigate[...]",
   "displayurl" : "https://www.google.com/selfdrivingcar",
   "url" : "https://www.google.com/selfdrivingcar/",
   "urlcrc" : BinData(0, "QIuP+XXSfMoAGpcqluNqNEzGEPA="),
   "index" : 10000
}
```

5.5 Data collection problems and errors

When running a procedure over a stretch of time to collect data for analysis, the main rule should be to not change the procedure during the data collection. This would change the data collected and thereby the basis of the analysis.

That said, some problems were discovered during the data collection period that could have potentially ruined the analysis.

Three distinct adjustments were done, to avoid bad data or wrong data. These are outlined in sections 5.5.1, 5.5.2 and 5.5.3.

Other errors that occurred are described in the last part of this section. All error codes are denoted in the data collection result tables in appendix A.

Note that when discussing days in this section (section 5.5), the whole data collection period of day 1 - 71 are discussed, see figure 5.3 in section 5.6.2 for more on this.

5.5.1 Typing error

Two of the queries were actually typed wrong, which meant that results from these queries would be wrong. This especially goes for exact search. Queries 1 and 9 and their respective exact queries (2 and 10) had typos. These typos were discovered after day 5 and were corrected before day 6.

See also figures A.3 and A.4 in appendix A.3.2. Errors are not so clearly visible, but for the eagle-eyed, both figures have a graph that goes below 500 results for the first five days.

5.5.2 Premature termination

Each of the 40 queries were run as a separate parametrised executable every day of the test period. In some cases one of the 20 API transactions per query run (see section 5.3 for details on how Query, QueryRun and transactions relate to each other) would terminate prematurely, and this would make the whole executable terminate, thus not yielding any results for that query that day (that QueryRun).

The batch executable was changed after the 12^{th} day and would be correct from the 13^{th} day onwards. After this, only the transaction that failed would give zero results, but the rest of the transactions would be ok.

Premature termination would be prevalent at the start of the period, but would happen very few times after the initial period of day 1 - 12. In the tables in appendix A.2, this is denoted by "—". As an example, see query 19 on December 1st - day 13 - in table A.13.

5.5.3 Exact search

On day 17 of the data collection period, more information on how to do exact search was exposed. The API parameter used as exact search is used by Bing as a way of preventing the search engine from altering the query, and not as a method of making the search exact. Coincidentally, information was uncovered on how the search engine would suppress more than two results from the same top level URL as a default[6].

DisableQueryAlterations Specifying this option prevents Bing from altering the query string. Such alteration may have been done in order to correct apparent spelling error in the original query string.

DisableHostCollapsing Specifying this option prevents Bing from suppressing more than two results from the same top-level URL for a request.

To actually do an exact search using the API, the exact queries would have to be surrounded by quotation marks in addition to the *DisableQueryAlteration* parameter. The parameter *DisableHostCollapsing* was also added to all free queries, to make the search engine return all possible results.

All this was corrected from day 18 onwards. This is visible in most result plots from the run, but is especially in the combined plot for exact searches, as seen in A.4.

5.5.4 Connection error

In some runs there were connection errors, one or more of the 20 transactions did not finish. The error message was "request canceled [sic] (Client.Timeout exceeded while awaiting headers)". Client time-out were set to 30 seconds, something that should be enough for a single transaction.

In all cases except one, only one transaction was cancelled out of the 20 transactions for the query marked with CT. The exception was Query 10 on day 48 of the run, where 9 out of 20 transactions failed.

6. `https://msdn.microsoft.com/en-us/library/dd250969.aspx`

This would probably be because of a connection problem between any of the nodes from the machine running the queries to the API endpoint.

These types of errors are denoted with a **CT** in the tables in appendix A.2.

5.5.5 Parameter incorrect

In one particular query - Query 26 on day 38 - one of the 20 transactions was perceived by Bing Search API as having incorrect parameters. The error message was "The parameter is incorrect." This seems odd, knowing that this exact transaction was run daily for weeks with no errors. There must be some other reason for this than incorrect parameters, typically communication errors where URI used or data sent were cut off or garbled.

These types of errors are denoted with a **PI** in the tables in appendix A.2.

5.5.6 Parse error

Parse errors would happen if one of the 20 transactions per query run failed, and returned HTML containing an error message instead of the expected JSON result. One such example would be if the Bing API suddenly decided that one of the transactions was illegal, and returned a HTML based error message. The reason for this is unknown, but - hazarding a guess - it could be the same explanation as for incorrect parameters, see 5.5.5.

These types of errors are denoted with a **PE** in the tables in appendix A.2.

5.6 Batch result analysis

The following sections outline a short analysis of the results from the batch run. More detail on results can be found in appendix A.

5.6.1 Data collection periods

The gravest of the problems described in 5.5 were mainly located to the days up to and including day 17 of the run period.

- Day 1-5: Typo on query 1,2,9 and 10

- Day 1-12: Some queries terminated without giving results.
- Day 1-17: No exact search, and Bing was configured to hold back multiple results from the same domain.

The test period can thus be divided into 4 parts: 1-5, 6-12, 13-17 and 18-71. From day 18 in the batch run, queries ran much more stable and gave results, except query 19, 25 and 39 which had one day each where they did not give any results after day 18.

5.6.2 Data selected for analysis and testing

For the full 71 days of data collection, a total of *1 981 006* results were collected.

Since the early days of data collection had many errors, the period selected for analysis and testing was from day 18 to day 71, in total 54 days of data, see figure 5.3.

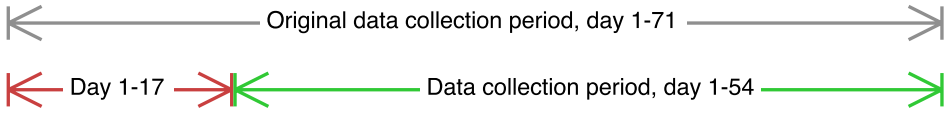


Figure 5.3: This shows the full period, the error-prone period of the first 17 days, and the period selected for analysis and testing.

A total of *1 444 330* results were collected in the selected period, for all 40 queries.

When referring to days of results in this thesis, the start is day 1 and end is day 54, unless otherwise noted.

The start day is referred to as day 1, but was really day 18 in the data collection period. The last day of data collection period is referred to as day 54, but was really day 71.

5.6.3 Short analysis of new results

Showing all queries in the same plot, can be used to see the trend in results. Figure 5.4 shows that results follows the suggestion of the predicted graph for results, outlined in figure 3.6 in chapter 3. The measured results shows

the same steep curve as in the graph. This means that the results have the characteristics shown in figures 3.2 to 3.5, with the first day having many new results, and the following day having few new results.

Note that figure 5.4 shows the percentage-wise number of new results for all collected results for each Query per day of data collection, not for the end-user experiences, as shown in figures 3.7 and 3.8.

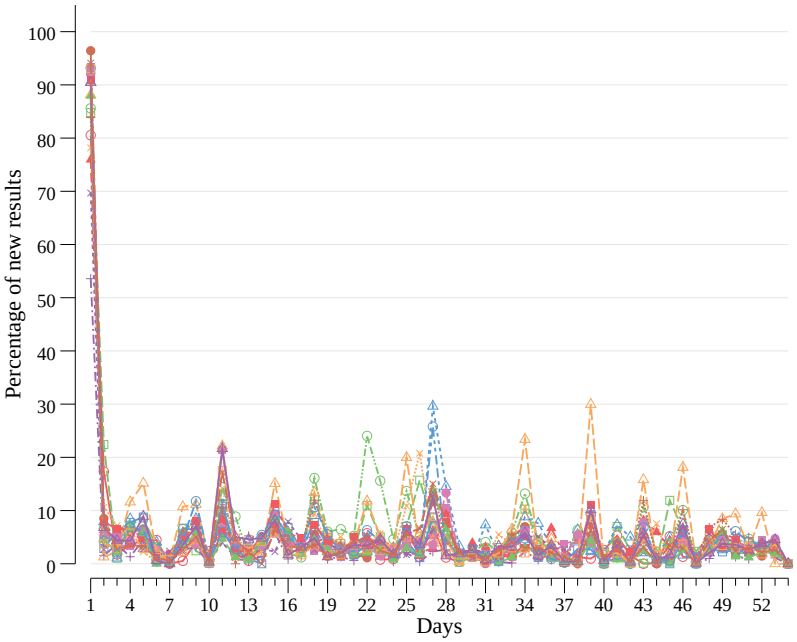


Figure 5.4: Plot of all 20 free queries (1,3,5 up until 39) from day 1.

This is visible in all plots, but is easiest to spot in the plot for all 20 free queries (odd-numbered; 1, 3, 5 up until 39), as shown in figure 5.4. This plot shows the percentage-wise number of new results each day compared to all the previous days' accumulated results.

The first day of data collection naturally gives a lot of new results, since this is the first time any data is collected, and there's nothing to compare it with. With the data collected the first day, one would expect the same number of *new* results the first day. This is not the case, and when analysing the data there are some duplicate URLs, see section 5.6.5 on duplicates.

After the first couple of days of results, the curve is almost at the bottom, and then flattens out. For each day there is normally a few new results, but it varies from day to day. Some days more, some days less. Some queries seem to vary wildly, but after the first or second day there is never more than 30% of new

results, and predominantly less than 10%.

On day 11 and day 27 among others, there are small peaks in the new results. One explanation could be adjustments to the Bing search engine content, since all Queries peaked.

5.6.4 Result overlap between days

By examining the data collected from Bing Search API v2 for the two first days of data collection, we can find how much overlap there was between them. Overlap in this context represent how many results from day one reappeared on day two.

The first 100 results for day one were compared to the first 100 results from day two. Results were ranked somewhat differently from day one to day two, but the top of the lists were very similar. It is therefore interesting to see how many results were exactly the same, when counted from the top of the lists. This is shown in column *Equal rank* in table 5.3. In the case of the Query "liverpool leeds efl", a user would have to browse through the 20 first results on day two before finding a result that is different from day one, see also figure 5.5.

Id	Query	Equal rank	New results	Overlap
11	winds of winter	29	1	99
15	terry pratchett	60	0	100
17	liverpool leeds efl	20	2	98
29	mobile application health sensor data	34	5	95
35	forest fairytales	2	6	94

Table 5.3: Overlap in the first 100 results. The table also shows number of equal results day 2, that were also equal in rank.

Table 5.3 shows that most results from day one reappeared day two. The column *New results* shows how many of the 100 first results on day two were different from the 100 first results on day one.

Assuming the page size of 15 when viewing, column *Equal rank* shows that for most of the Query more than one page were exactly equal from day one to day two. If a user were to browse through these results, presented to them with a page size of 15, the user would the second day (second search) see the exact same results as the first day, if only looking at the first page. For "terry pratchett" it would be 4 pages of 15 results each before changes emerges on page 5.

The exception is Query 35 "forest fairytales", which had 2 results at the top of the list on the first day that were equal to the second day. Still, out of the

100 first results both days, 94 of them were overlapping, the results just had different rankings.

As an illustration of the column *Equal rank* in table 5.3, figure 5.5 shows a screen capture of the Query 17 "liverpool leeds efl" comparison, displaying the first 30 results from day one and day two. The 20 first results were ranked equally on both day one and day two, after which some results with different ranking shows, but most of them are just ranked differently, therefore the results are mostly the same, albeit in a slightly different sort order.

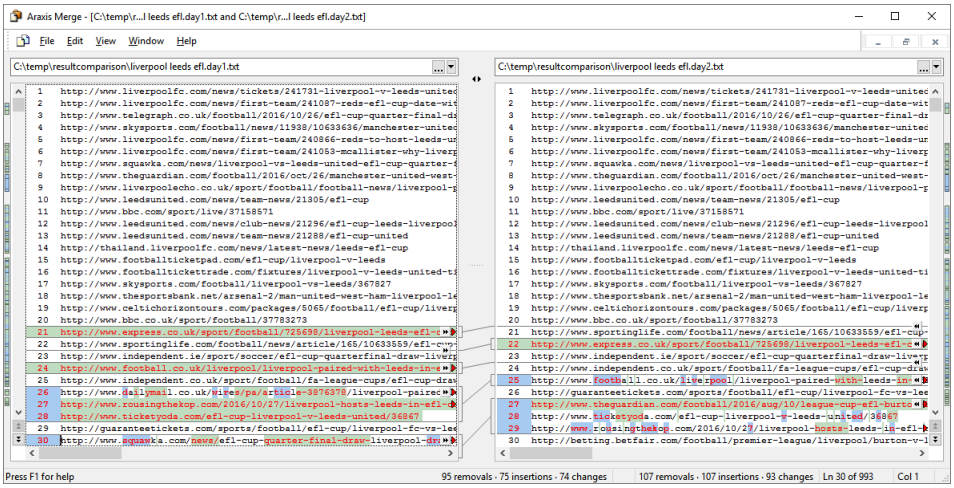


Figure 5.5: Screen capture of comparison between day one and day two for Query 17.

Queries analysed are those that are used as examples in this chapter and in chapter 7. Other Queries have not been analysed, but it is reasonable to think that they would show the same characteristics.

5.6.5 Duplicates

Duplicates in the data collection results stem from duplicates in the search engine index. It can be difficult for the search engine to distinguish between duplicates within the same site.

When analysing and finding the numbers for columns *New* and *New d2* in table A.3, only the individual result's URL has been used as a basis for finding which results are new.

Creating a checksum from the result's URL should be enough to identify the

result. Looking through some of the results, it is clear that duplicate URLs in some cases occur where the accompanying title and/or description is different from each other. These would be distinct results if the checksum is created from a combination of the result's URL, Title and Description. But then the content should really be the same anyway, and the URLs may for instance be indexed at different points in time.

One exception to this would be home pages for newspapers etc., see section 2.1.2.

It turns out that some URLs appeared as different results, with *http* and *https* as the only difference, and some with and without a trailing "/" as the only difference. So before taking a checksum of the URL, the following has been done to make sure even more duplicates could be ignored.

- it has been made lower case,
  so that *HTTP://WWW.EXAMPLE.COM* becomes *http://www.example.com*
- "http**s**://" has been replaced with "http://",
  so that ***https****://www.example.com* becomes ***http****://www.example.com*
- forward slash at the end of the URL has been removed,
  so that *http://www.example.com/* becomes *http://www.example.com*

This contributed to reducing the number of new results slightly, and would also add to duplicates in the result. Note that the duplicates was not removed from the result set, only its URL and checksum was updated.

When disregarding the first day of new results, the column *New d2 %* in table A.3 shows that the percentage of new results for the full period never surpasses 3.2%.

It's important to note that duplicates also occurred before turning *DisableHost-Collapsing* on, on day 18 in the original data collection series. See section 5.5.3 for more on this.

But after disabling host collapsing it's reasonable to assume that duplicates increased slightly. This is difficult to quantify, since the data collection method changed radically from the 18th day on.

Still, the upper limit of 1000 results per Query per run was never reached until *DisableHostCollapsing* was turned on, and then only for a limited number of queries.

Every single result from the Bing search API contains a GUID[7][8], a unique identifier, which is stored together with the result. These have been checked for duplicates as well, to rule out any insufficiencies in the data collection application. No duplicates were found.

5.7 Batch result patterns

Generally, free queries gave many results, and exact queries gave fewer results than free queries. This is also reflected in the number of new results, with more new results for free queries and fewer new results for exact queries. See also section 5.6.3 for a more thorough discussion on new results.

Some groups of results tended to follow certain patterns, which are described below.

5.7.1 Many free results - many exact

A "many-many" pattern can be seen in query 11 and 12, "Winds of winter". Figure 5.6 shows a high number of results when searching freely, and when searching for the exact phrase there are still quite a lot of results showing, as seen in figure 5.7.

New results since the start of the period however, is sparse. This means that the same results crop up again and again, and very few new results are found for each day. Even fewer new results are found each day for the exact query, due to fewer results in total for each day, as seen in figure 5.7.

7. Microsoft's implementation of UUID
8. `https://www.iso.org/standard/62795.html`

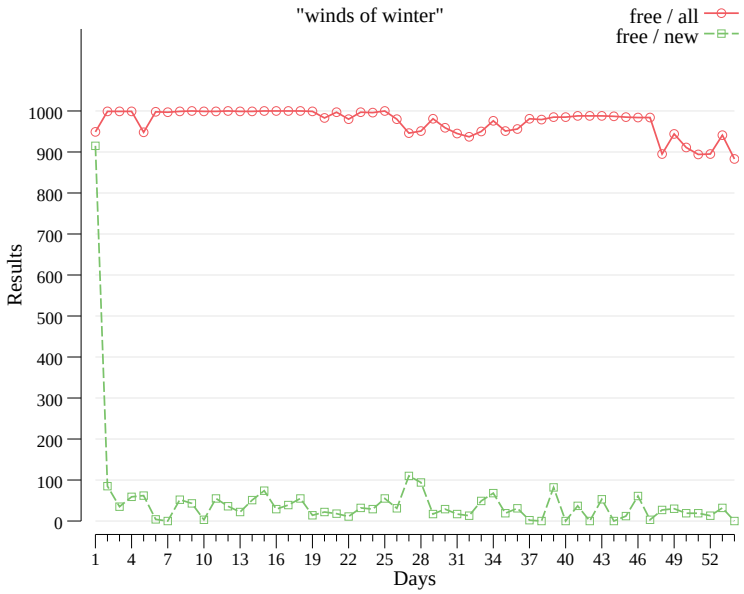


Figure 5.6: Plot of new results for free query "Winds of winter".

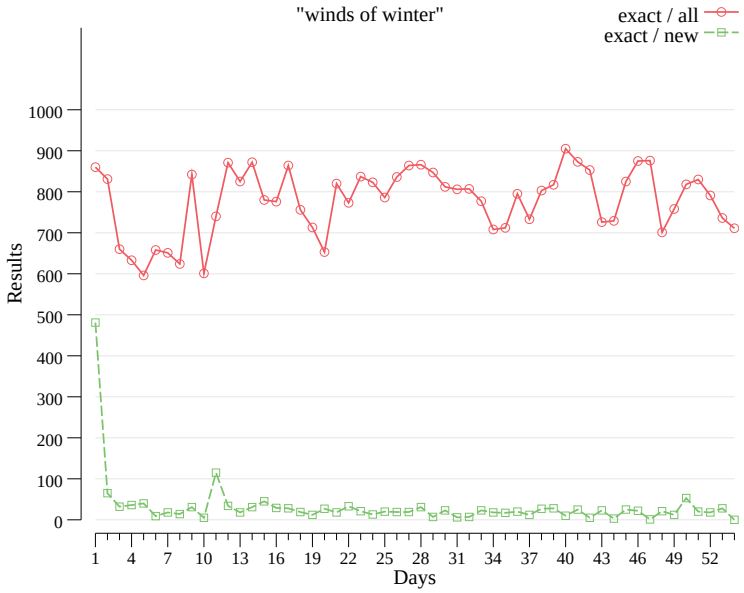


Figure 5.7: Plot of new results for exact query "Winds of winter".

The plots in figures 5.6 and 5.7 follows the series of results indicated by figures 3.2 to 3.5 in chapter 3, where the day's new results are compared to the accumulated previous days' results.

In contrast to this I can have a look at the results in a day to day fashion, comparing to the previous day's results only, as shown in figure 3.3 in chapter 3.

The day to day plots show more new results since the previous day's query run than the accumulated plots. This is evident in figures 5.8 and 5.9.

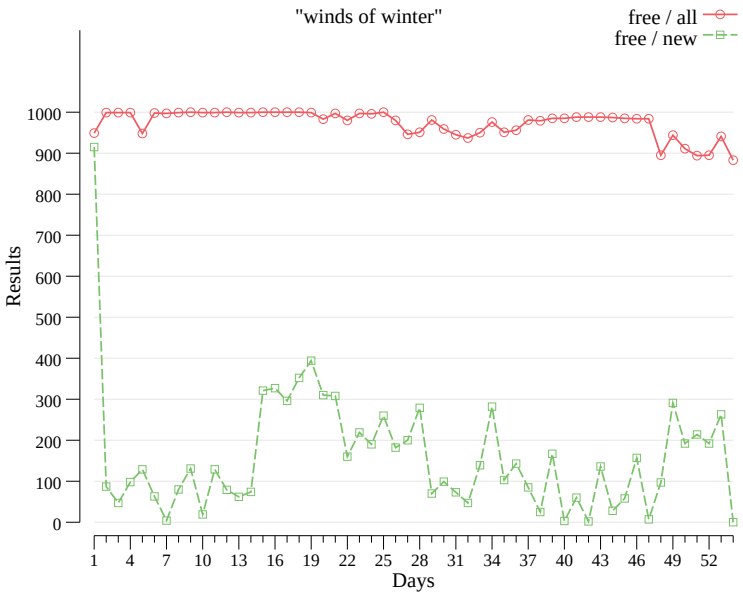


Figure 5.8: Day to day results of free query "Winds of winter".

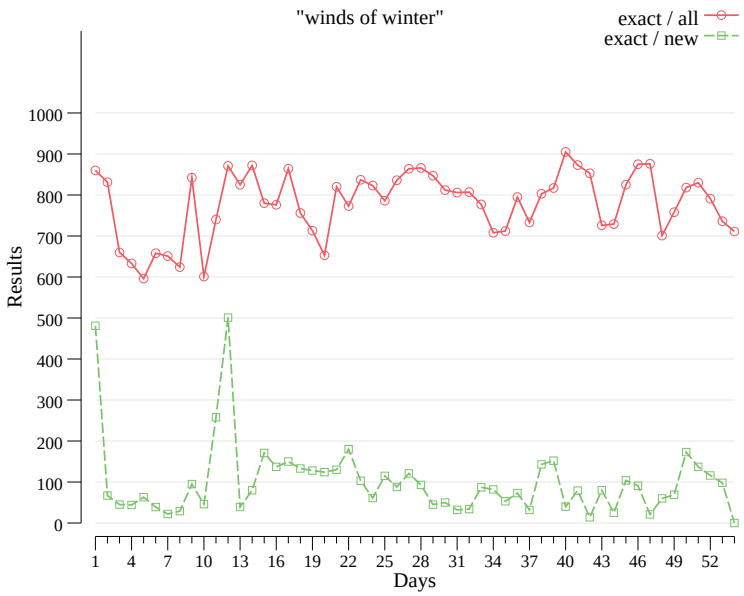


Figure 5.9: Day to day results of exact query "Winds of winter".

This means that when not comparing results to preceding accumulated results, the number of new results for each day is perceived as higher. This also means that a day to day measure of new results does not exclude all the previous seen results.

If the system was constructed to only look at the previous day's result, the user executing this query would have ended up with results that have been seen before.

Another matter is the actual content of the results. The query was meant to explore George R.R. Martin's not yet released novel "The Winds of Winter", but when looking at the actual content of the results, weather forecasts and reports feature heavily.

The occurrence of not related results, points to the problem of optimising queries when searching. This query in particular would benefit from adding "George R.R. Martin" to it. The following theoretical rewrite of the query could maybe have obtained better precision in the results.

"book" AND "Winds of Winter" AND
("George R.R. Martin" OR "George Martin" OR "Martin")

5.7.2 Many free results - no exact

This pattern can be seen in query 29 and 30, "mobile application health sensor data". In figure 5.10 I can see a high number of results when searching freely, but when searching for the exact phrase there are no results, as seen in figure 5.11.

The number of results for exact query is zero because of the way the query has been set up. The exact phrase "mobile application health sensor data" does not have corresponding results in the API of the search engine, so nothing is found.

This query could theoretically be rewritten as the following for better precision.

"*mobile application*" *AND* "*health sensor data*"
or
"*mobile application*" *AND* "*health sensor*" *AND* "*data*"

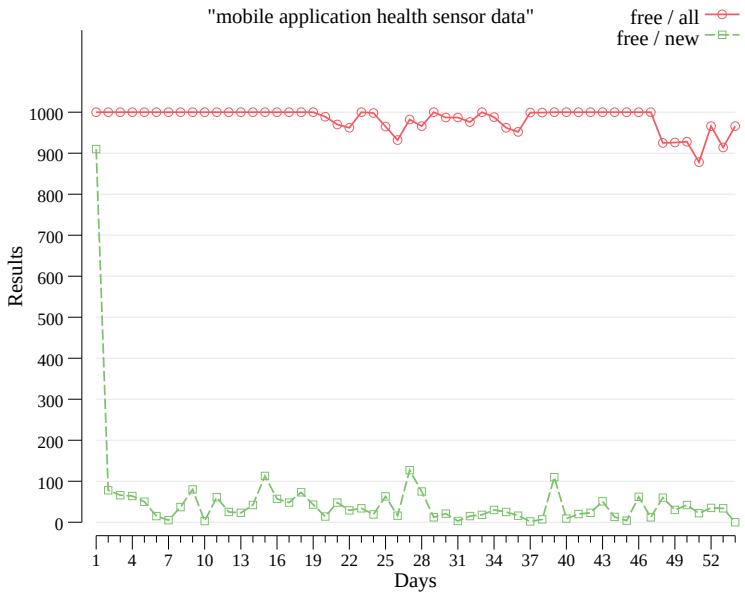


Figure 5.10: Plot of free query "mobile application health sensor data".

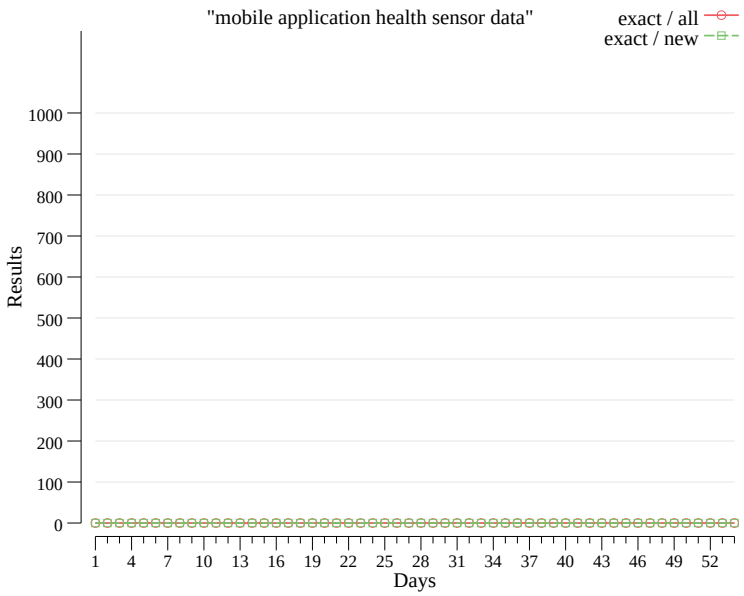


Figure 5.11: Plot of exact query "mobile application health sensor data".

5.8 Simulation of usage

We have compared one day's results with the accumulated previous results, as described in section 3.2 *Search example*, and seen as real data in figure 5.4. This has shown that the curve of new results is declining rapidly from day one.

During user testing, if all results from the first day of search were saved or discarded by the user, we would get a curve like the expected results curve in figure 3.6 in chapter 3.

However, when an end-user uses the system, the user would most likely not take the time to look through all 1000 of the first day's results, to find interesting or uninteresting results from the first day. So the user experience will likely follow a different type of progression.

Using the collected data it is possible to theorise about this usage, to be able to say something about the user's search progress when the user is viewing a certain number of results every day. The resulting graphs show how many days before the curve drops off. This would indicate that I am getting closer to a more manageable number of new results each day, and only the latest collected information done by the search engine would emerge as results.

Note that this simulation of usage is only for the collected data, which in essence is data retrieval through the Bing Search API v2, where the maximum number of results is 1000 for each search. This will simulate how the testers theoretically could use IIR, see also section 7.7.

Looking at the plot in figure 5.12, one can see that with viewing 25 results per day it will take a long time before there are fewer results. Many of the queries would have 25 new results per day throughout the test period as in figure 5.12. Viewing 25 per day is a low number, and users would probably view more than 25 results when searching.

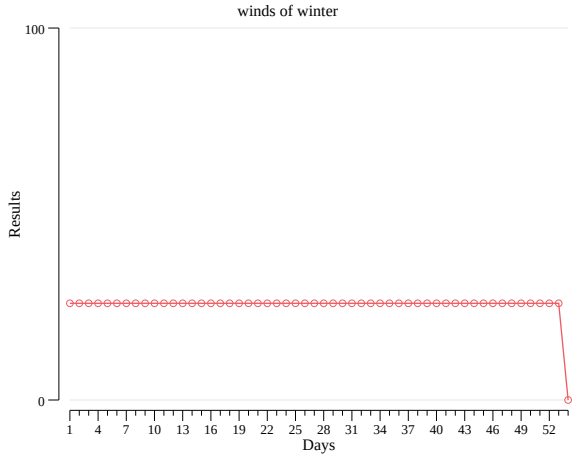


Figure 5.12: Progression when viewing 25 results per day, for query 11, "winds of winter".

The same reasoning can be used for 50 results per day, so the next step shown here is 100 results per day.

With a 100/day progress the number of days before any changes to the results is shown to be around 10 days, see figure 5.13. Sorting through 100 results per day does not take a very long time and can be done easily.

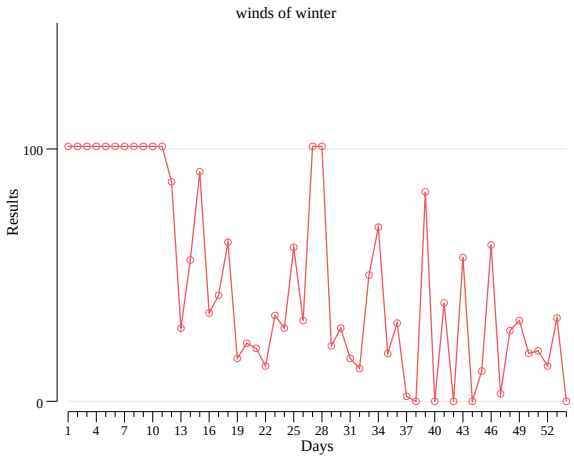


Figure 5.13: Progression when viewing 100 results per day, for query 11, "winds of winter".

The next level shown, in figure 5.14, is viewing 250 results per day. If the user can have the patience and view this amount of results per day, the time before fewer results is seen to be 1-4 days.

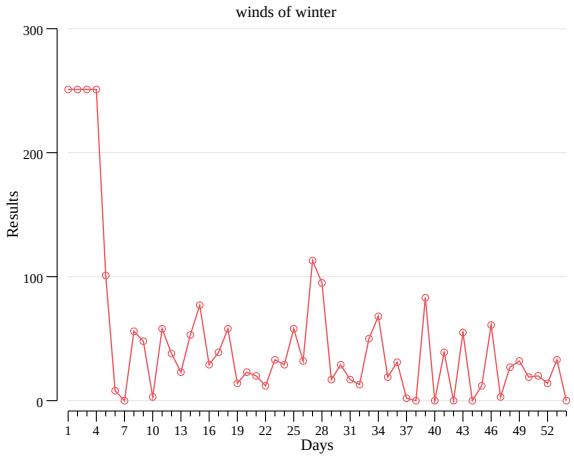


Figure 5.14: Progression when viewing 250 results per day, for query 11, "winds of winter".

With 500 results per day, as in figure 5.15, normally the second day will show fewer results.

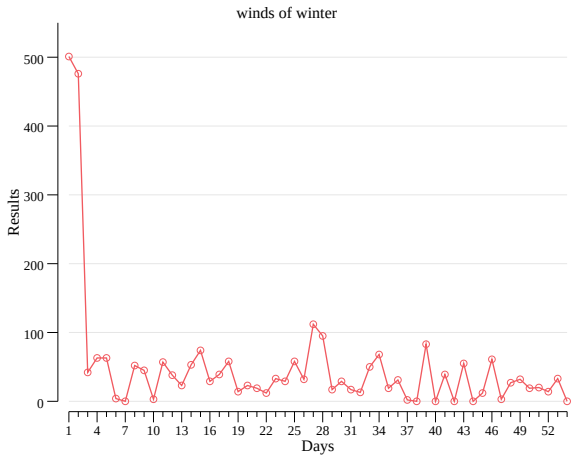


Figure 5.15: Progression when viewing 500 results per day, for query 11, "winds of winter".

The basis for this simulation is that even though there are only 1000 results for each day, this is still much for a user to go through. So lowering the number of results to a manageable number daily, *e.g.* 25, 50 or even 100 results, would help in the user search. With fewer results shown, it would be much easier to discover when the search engine supplies results that the user has not seen before.

At the same time it is not the objective to get to zero results per day - finding nothing would defeat the purpose of searching.

So as a general rule, viewing as many results as possible early on, would lead to a manageable number of results much faster. After that, new results will mostly stay fewer, and would be much easier to manage.

6

Implementation

This chapter outlines the implementation, including technology used, database architecture, server runtime environment and more details on the user interface.

6.1 Technology

Search engine API technology in general is created so that it can be used from a wide range of technology platforms. The relevant APIs are called by using the HTTP protocol, which makes them very easy to interface with almost any software.

This means that using the API is not dependent upon the development environment chosen, and so the process of choosing a development platform can look at other factors concerning suitability.

Server technology could easily have been from the LAMP[1] stack, or the open source ASP.NET[2] technology from Microsoft[3]. Both of these environments are perfectly capable of solving the server role.

1. LAMP stack is a web platform that consists of Linux, Apache, MySQL, and PHP/Python/Perl
2. `http://www.asp.net`
3. `http://www.microsoft.com`

The chosen implementation platform, however, is the Go language[41], because of concurrency, performance and scalability, see section 6.1.2 for more details.

6.1.1 MongoDB

When choosing a database, one of the issues is the amount of data to store. Since the goal of this thesis is to create a proof of concept type solution, it is not likely that significant amounts of data will be generated. Still, planning ahead is good, so a database that is scalable is preferable.

MongoDB [3] is an open-source document database that provides high performance, high availability, and automatic scaling, and this is the choice for database technology.

6.1.2 Go programming language

Developed at Google in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson, the Go language was originally developed as a systems programming language, and is now an open source general purpose programming language[38] [39]. It is strongly typed, has garbage collection and built-in support for concurrent programming, among other features.

Go is chosen because it is well suited for concurrency, batch routines, database access and for creating web services. A Go solution can also easily be deployed on several different platforms, in our case Linux and Windows. Go is also pleasant to work with, not to mention blazingly fast.

6.1.3 Go packages

The solution uses four packages outside the Go standard library.

Bingapi

This is a Go package for connecting to Bing Search API v2, made specially for this thesis[4], see section 6.5.2.

4. `https://godoc.org/github.com/borglefink/bingapi`

Gin

Gin[5] is a HTTP web framework written in Go (Golang). It is a minimalistic framework, with only the most essential features and libraries included, making it great for developing high-performance REST APIs.

Gorilla web toolkit

Gorilla[6] is a web toolkit for the Go programming language. Only *secure-cookie* is in use from this toolkit, it encodes and decodes authenticated and optionally encrypted cookie values.

mgo

mgo[7] is a MongoDB driver for the Go language that implements a rich and well tested selection of features under a very simple API following standard Go idioms. With the *mgo MongoDB driver*, accessing MongoDB from Go is quick and easy.

6.1.4 Web client

Go templates have been used to create web pages to send to the web client. To present results to the user, the web client will of course use HTML, CSS and JavaScript, and in addition JavaScript libraries like Angular.js[8] and jQuery[9]. Making the user interface look good is often a challenge, so to mitigate this, Bootstrap[10] is used.

6.1.5 Server

Server technology could easily have been from the LAMP[11] stack, or the open source ASP.NET[12] technology from Microsoft[13]. Both of these environments are perfectly capable of solving the server role.

The chosen server implementation platform, however, is the Go language, because of concurrency, performance and scalability. A Go+MongoDB solution

5. https://gin-gonic.github.io/gin/
6. http://www.gorillatoolkit.org/
7. https://labix.org/mgo
8. https://angularjs.org
9. https://jquery.com
10. http://getbootstrap.com/
11. LAMP stack is a web platform that consists of Linux, Apache, MySQL, and PHP/Python/Perl
12. http://www.asp.net
13. http://www.microsoft.com

can also easily be deployed on several different server platforms, both Unix- and Windows-based operating systems.

6.2 Database architecture

The database needs to reflect the entities that the IIR system is managing. A *User* can have many *Queries*, a *Query* can have many *QueryRuns*, and a *QueryRun* can have many *Results*.

These are relations that fit a non-relational database quite well, with every entity except results connected to a sub-entity, with a one-to-many type relationship, see figure 6.1.

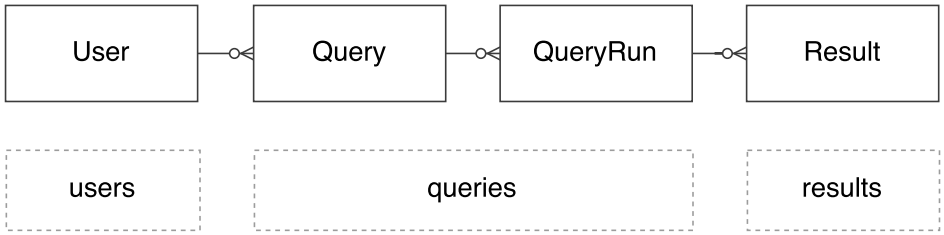


Figure 6.1: IIR database entities, and corresponding database collections.

Theoretically all these entities could fit into one collection, but every *User* would contain all this user's Queries, QueryRuns, and Results, which would reach record size limitations rather quickly. So for scalability and for practical reasons, they are divided into three collections; *users*, *queries*, and *results*.

The *queries* collection has user's ID as a foreign key, and contains a sub-collection of QueryRuns, where each QueryRun has its own ID. Each result has the user's, the Query's and the QueryRun's IDs as foreign keys. This makes it easy to access a Query's QueryRuns, and to fetch results across all QueryRuns.

The listings 6.1, 6.2, 6.3 and 6.4 are mostly self-explanatory, since they are commented.

Note that all fields have tags to marshal Binary JavaScript Objects Notation[14] (BSON) values into Go types. In the actual code these tags also contained entries for JSON marshalling, but are omitted for brevity here.

14. `http://bsonspec.org`

Listing 6.1: Go struct describing the User

```
// User is the logged-on user
type User struct {
    // User ID
    ID                  bson.ObjectId  `bson:"_id"`

    // User's name
    UserName            string         `bson:"userName"`

    // Last used Query, for the user to resume previous work after login
    CurrentQueryID      bson.ObjectId  `bson:"currentQueryId,omitempty"`
}
```

Listing 6.2: Go struct describing the Query

```
// Query is the container for the search
type Query struct {
    // Query ID
    ID                  bson.ObjectId    `bson:"_id"`

    // User ID
    UserID              bson.ObjectId    `bson:"userId,omitempty"`

    // Currently active QueryRun
    CurrentQueryRunID   bson.ObjectId    `bson:"currentQueryRunId,omitempty"`

    // Simple numeric ID for the Query
    SimpleID            int              `bson:"simpleId,omitempty"`

    // Name of the Query, for display in UI
    QueryName           string           `bson:"queryName"`

    // Actual search text to send to search engine
    Query               string           `bson:"query"`

    // If true, the search text will be enclosed by
    // quotation marks when sent to the search engine
    IsExactSearch       bool             `bson:"isExactSearch"`

    // When was the Query created
    CreatedDate         time.Time        `bson:"createdDate,omitempty"`

    // When was the Query last run against the search engine
    LastRunDate         time.Time        `bson:"lastRunDate,omitempty"`

    // List of black-listed domain names
    BlackList           []string         `bson:"blackList,omitempty"`

    // List of white-listed domain names
    WhiteList           []string         `bson:"whiteList,omitempty"`

    // Sub-collection containing QueryRuns
    QueryRuns           []QueryRun       `bson:"queryRuns,omitempty"`
}
```

Listing 6.3: Go struct describing the QueryRun

```
// QueryRun contains information on when the query is run
type QueryRun struct {
    // QueryRun ID
    ID          bson.ObjectId `bson:"_id"`

    // Updated when the user change the search text sent to search engine
    Query       string        `bson:"query"`

    // Date of search execution
    RunDate     time.Time     `bson:"runDate,omitempty"`
}
```

Listing 6.4: Go struct describing the Result

```
// Result is the actual result
type Result struct {
    // Result ID
    ID          bson.ObjectId `bson:"_id"`

    // User ID
    UserID      bson.ObjectId `bson:"userId"`

    // Query ID
    QueryID     bson.ObjectId `bson:"queryId"`

    // Query simple ID
    SimpleID    int           `bson:"simpleId,omitempty"`

    // QueryRun ID
    QueryRunID  bson.ObjectId `bson:"queryRunId"`

    // Title of result from search engine
    Title       string        `bson:"title"`

    // Description of result from search engine
    Description string        `bson:"description,omitempty"`

    // URL of result from search engine
    URL         string        `bson:"url"`

    // URL hashed to make it easier to compare
    URLHash     string        `bson:"urlHash,omitempty"`

    // DisplayURL of result from search engine
    DisplayURL  string        `bson:"displayUrl,omitempty"`

    // Index holding order of returned result from search engine
    Index       int           `bson:"index,omitempty"`

    // Domain name extracted from URL from search engine
    DomainName  string        `bson:"domainName,omitempty"`

    // Result status; new, seen, saved, discarded, white-listed
    Status      int           `bson:"status"`

    // Simple IIR ranking
    WordRank    int           `bson:"wordrank,omitempty"`
}
```

6.3 Front-end

The implementation of the web client for this solution follows the design described in section 4.5. The left side of the main area contains the queries, the right side of the main area contains results and tools to filter results.

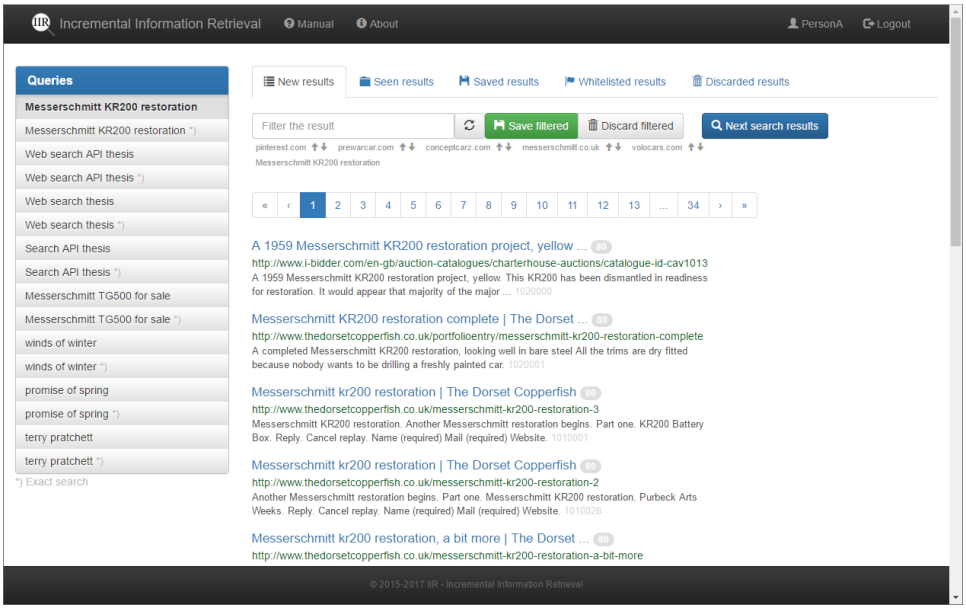


Figure 6.2: The main user interface, with the first query loaded

Angular and Bootstrap are libraries that work nicely together, and make the front-end easy to create. The Angular version used is the 1.x version, since work with the thesis started way before Angular v2.

The JavaScript code has been divided into controllers, services, and data containers, following the Angular 1 guidelines[15] created by John Papa and others. This makes the code follow good practices like the single responsibility principle [33].

15. https://github.com/johnpapa/angular-styleguide/blob/master/a1

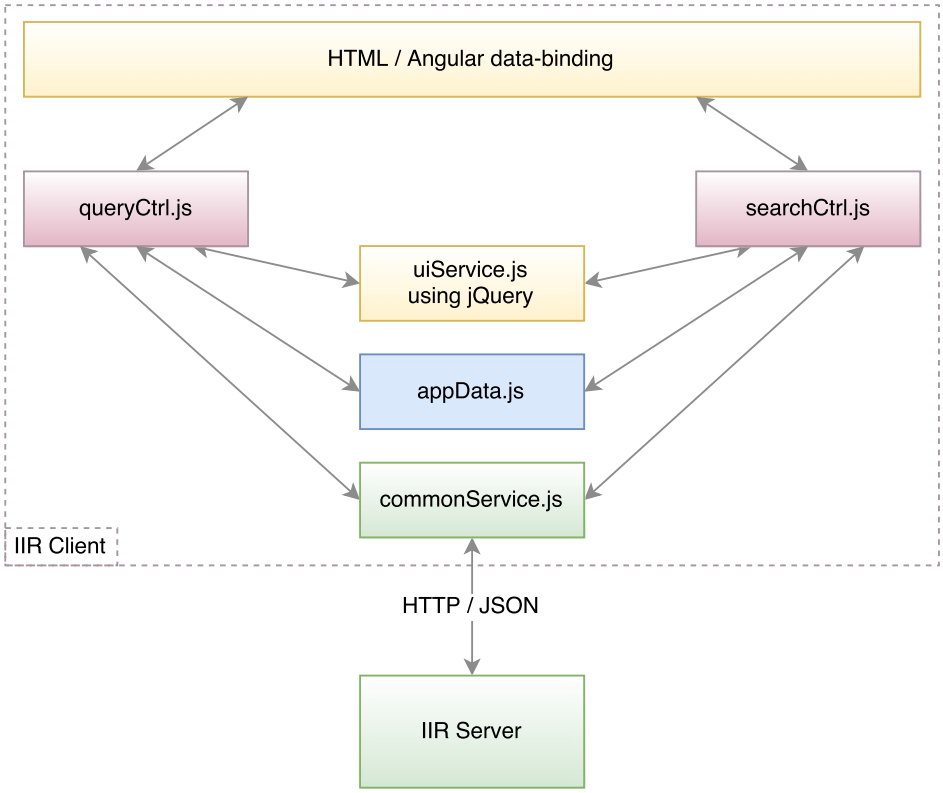


Figure 6.3: IIR client architecture.

Component	Description
HTML / View	HTML is generated by Go Templates and contains Angular data-binding to display and handle data in the user interface (UI).
queryCtrl	Handling most of the logic in the client, when clicking buttons or filtering results
searchCtrl	Handling the search button.
appData	Data used by both Angular controllers.
uiService	Contains methods to update the user interface via the Document Object Model (DOM)[16]. Used by both controllers. The only component that uses jQuery.
commonService	Handles all the communications with the IIR server, used by both controllers.
IIR Server	This is the IIR server, the back-end, detailed in section 6.4.

Table 6.1: IIR client components.

Figure 6.3 shows the two main Angular controllers, their shared data and routines, and how the client connects to the server, see component descriptions in table 6.1. The communication with the server is handled by Angular's $http

16. `https://www.w3.org/TR/WD-DOM/introduction.html`

service[17], primarily by HTTP POST[18], and the data received from the IIR server is formatted as JSON.

6.4 Back-end

The back-end is fully realised in Go. The architecture is inspired by a tutorial called "Building a REST Service with Golang"[43] by Steven White, but is further evolved and expanded upon.

The main program is setting up a closable session to MongoDB, and passing this session along to the *RegisterRoutes()* method, that sets up handlers for all requests to the server, and marks static files like JavaScript and CSS as special requests.

The main program then hands the control to the http engine of gin-gonic (see section 6.1.3), a wrapper for the Go http service. When the server gets a request from the web client, control is passed to the registered handler for the request.

Listing 6.5: Main program for IIR

```
package main

import (
    "controllers"
    "db"
)

func main() {
    // connect the database
    var mgosession = db.GetMgoSession()

    // make sure the database is closed when the main function exits
    defer mgosession.Close()

    // register all routes for http requests
    var routes = controllers.RegisterRoutes(mgosession)

    // start the http server on the given TCP port
    routes.Run(":80")
}
```

Routes are registered by the *RegisterRoutes()* function, see listing 6.6 for more details. Note the code is edited for brevity.

17. https://docs.angularjs.org/api/ng/service/%24http
18. https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.5

When registering a route, an optional middleware[19] can be added to it. Middleware used in IIR is a request logger and an authentication checker. When the server gets a request, it is first logged, then authenticated, then - provided the authentication is approved - the controller handle (function) for the route is called.

Listing 6.6: Registering routes for the IIR web service

```
func RegisterRoutes(mgosession *mgo.Session) *gin.Engine {
var router = gin.New()

router.Static("/scripts", "./public/scripts")
router.Static("/styles", "./public/styles")
router.Static("/images", "./public/images")

router.LoadHTMLGlob("templates/**/*.html")

var homeController = newHomeController()
router.GET("/",      [...] identity.Authentication, homeController.index)
router.GET("/about", [...] identity.Authentication, homeController.about)

// registering routes for resultController
var resultController = newResultController(mgosession)
router.POST("/resultlist", identity.Authentication,
                                    resultController.postResultList)

[...plus more routes and controllers...]

return router
}
```

Authentication is managed by an encrypted cookie, where Gorilla web toolkit's tool *securecookie*[20] (see also section 6.1.3) encodes and encrypts the user's authentication. The cookie exists as long as the user is logged on or the cookie times out, and it is sent to the server with every request. The server decrypts and decodes the cookie, and by this IIR knows who is logged on, and can retrieve the appropriate queries and results.

The controller handle has a database session, and can retrieve data or update the database. The controller returns in most cases JSON, with the exception of web pages like the index page, the about page and the IIR manual page.

Controller	Description
homeController	This controller handles the index page, and is also responsible for showing the manual page and the about page.
loginController	Responsible for user login and logout, *i.e.*, setting and removing the encrypted authentication cookie.
queryController	For listing, creating, updating and deleting Queries in the IIR database.

19. http://www.webopedia.com/TERM/M/middleware.html
20. http://www.gorillatoolkit.org/pkg/securecookie

resultController	This controller is responsible for showing lists of results saved in the IIR database.
searchController	This is where the API search and automatic filtering happens.
updateResultController	Responsible for updating result status (seen, saved, discarded, and white-listed).
userController	Responsible for registering new users.

Table 6.2: IIR server main controllers.

The controllers mostly do housekeeping in one form or other, like creating, updating or deleting users and Queries, and login and logout.

There are controllers worth extra mention; the *queryController*, the *searchController*, and the *updateResultController*.

The *queryController*'s main responsibility is to read from the IIR database and return a list of results to the web client. It also has the responsibility of adding the white- and black-list settings to the current Query. The controller also updates all results that are white-listed with the result status *White-listed*. During the test phase of this project, it also set the black-listed results to *Black-listed*, but normally these results would have been deleted.

The *searchController* fetches a new batch of results from the chosen data source. It connects to the search engine API, and performs transactions, described in section 6.5.1. In the testing period, due to the data collection situation described in section 5.1, this was done by reading results from a separate source database. This controller also does the automatic filtering of already seen results, white- and black-listing, and updating the results with IIR rank, as described in section 4.3. This part of the code must be refactored if there is any change in data provider. The code for filtering and ranking can be reused, but the connection and data retrieval will most likely have to be rewritten. The API connection and the initial filtering, like removing previously seen results, white-and black-listing, is located to this one controller.

The *updateResultController* is responsible for handling the user's saving and discarding of results. Single results can be updated, but bulk updates are also possible, see description of filtering in section 4.1.

6.5 Using Bing Search API

The part that communicates with the search API is the core of the web application.

6.5.1 Interfacing with the API

The application needs to get as many results as possible, to have an increased chance of finding new information on the subject being searched for. At the same time each call to the search API can only return a certain amount of results - a "page".

This means that the application needs to do multiple calls to the search API for each search query, requesting each new "page", one by one. There is also a limit to how many "pages" can be returned for each search API. Bing Search API v2, for instance, uses maximum page size 50 and maximum number of results 1000.

A page size of 50 and a total of 1000 results, will generate 20 calls ("Transactions") to the API for each query. 20 pages are received, *i.e.*, results 1-50, 51-100, 101-150 and so on until the last result "page", 951-1000. See figure 6.4.

All these calls are run concurrently, which means that the results most likely are returned on a different order than the call order. This needs to be addressed when the results are presented to the user.

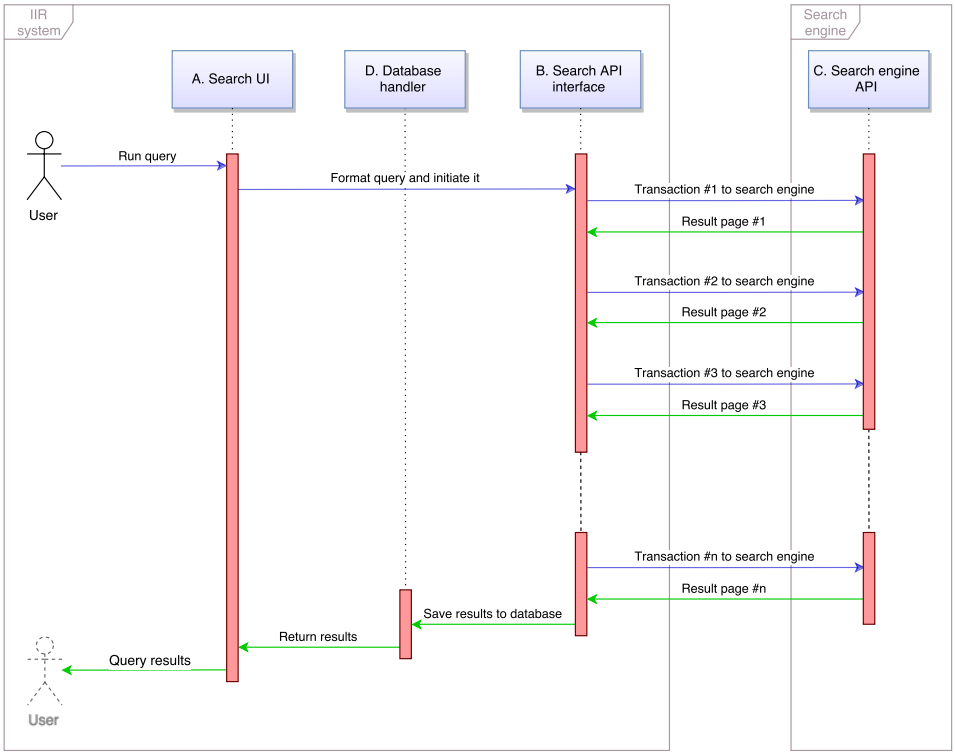


Figure 6.4: Search API call sequence diagram, implemented as 20 calls to the API because of limited number of results per call.

6.5.2 Go wrapper for Bing Search API

No Go package for connecting Bing Search API existed, so this had to be made from scratch. The Bing Search API can return different types of results, like web results, news, images, and videos. The Go package implemented could handle all result types that the Bing Search API could return, but this thesis only used the type "web results". See `https://godoc.org/github.com/borglefink/bingapi` for details on the implementation. This Go package was used when collecting data, as described in chapter 5.

6.5.3 Calling Bing Search API

The central code in calling the search API is shown below in listing 6.7. This is a call to a search API, written as a Go package. Calls to the search API is organised as a sequence of 20 API transactions, each with a "page" size of 50, to return the full set of 1000 results for each query. The different stages of the search are numbered as A to D, in ordered sequence. The connection to

the API and return of results are located in the *searchController*, described in 6.4.

Before the API in 6.7 is called, the URI needs to be constructed with the correct parameters for the call to the API. The search function would then be called with the URI, and would return a Go *struct* with all results.

Parameters used when calling the Bing Search API are listed in table 6.3. These are mostly self explanatory, but more information on *WebSearchOptions* can be found in section 5.5.3.

Parameter	**Value**
Base URI	https://api.datamarket.azure.com/Data.ashx/Bing/SearchWeb/v1/Web
Query	[text to search for]
Market	'en-US'
WebSearchOptions	'DisableHostCollapsing' 'DisableQueryAlterations'
$skip	[number of results to skip]
$top	[number of results to return]

Table 6.3: Parameters used in Bing Search API transactions

Listing 6.7: Main function for accessing the Bing Search API from Go

```
// search - the main search method,
// fills target struct with data from Bing API.
func (client Client) search(uri string, target interface{}) error {
   var httpClient = \&http.Client{}
   httpClient.Timeout = time.Duration(client.ReqTimeout) * time.Millisecond

   var req, errReq = http.NewRequest(client.Method, uri, nil)
   if errReq != nil {
      return errReq
   }

   req.Header.Set("User-Agent", client.UserAgent)
   req.SetBasicAuth(client.AccKey, client.AccKey)

   var res, errDo = httpClient.Do(req)
   if errDo != nil {
      return errDo
   }
   defer res.Body.Close()

   return json.NewDecoder(res.Body).Decode(target)
}
```

6.5.4 Search during the test phase

Because of the situation with Bing Search API being restructured, as outlined in section 5.1, the test phase solution could not access a live API. Instead a separate source database has been created, which contains the results collected in the data collection phase. In the solution, when the user presses the search button, the code that would have accessed the live API would instead read results from the source database and insert them into the IIR database.

As mentioned in section 6.4, this functionality is located to a single Go controller. Switching back to a live API access implementation will only concern this single controller, the rest of IIR need not be changed.

The architecture of the source database is almost identical to the data collection database architecture described in section 5.4.2. The only difference is that as source database, the QueryRun has a boolean field, *isDone*, to make it easier for the *searchController* to know which source data has been shown to the user.

6.6 Development environment

Application development has mainly been done on 2-3 different machines with the operating system Windows 10 Pro. Some of the development has also been done on a Linux laptop (operating system Ubuntu, versions 14.04 - 16.10).

Several different open sourced editors have been used. Adobe's Brackets[21], Microsoft's Visual Studio Code[22] (VSCode) and GitHub's Atom[23] have been utilised at different stages of development, but VSCode and Atom have been the main editors. Especially VSCode has been nice for Go debugging. Both editors available have been used in Windows and Linux operating systems.

As graphical user interface GUI for MongoDB (aside from the MongoDB CLI that also have been in use), MongoChef[24] from Studio3T has been used on Windows. Robomongo[25] has been used as well, both on Windows and Linux.

As repository, Bitbucket[26] has been invaluable for all code, and also for the thesis itself.

21. `https://brackets.io/`
22. `https://code.visualstudio.com/`
23. `https://atom.io/`
24. `https://studio3t.com/`
25. `https://robomongo.org`
26. `https://bitbucket.org`

6.7 Code summary

In listing 6.8 is a summary of the code used in the IIRsolution. Libraries like jQuery, Angular and Bootstrap are excluded from the code summary. This is also true for Go packages used, only IIR Go code is counted.

filetype	#files	#lines	line%	size	size%
.go	22	1 660	30.1	46 932	22.9
.html	21	895	16.2	38 218	18.7
.js	13	2 031	36.8	61 182	29.9
.json	1	26	0.5	484	0.2
.scss	4	906	16.4	14 278	7.0
.gif	1			10 144	5.0
.png	3			31 929	15.6
Total:	65	5 518	100.0	204 735	100.0

Listing 6.8: This is a summary of code made by the utility countsource[27], showing code used in the IIR solution.

See appendix F for more on code used with IIR.

27. https://github.com/borglefink/countsource

7 Testing

Using user experience with the solution is a good way to get feedback on how well it is perceived to work.

User testing as a measure of how successful the project is, also has drawbacks, such as it being a subjective experience. User interface layout and functionality may in some cases become the focus of attention, and actually prevent the user from relating to results. If the user is having trouble looking beyond the user interface, this will influence user experience assessments.

7.1 User testing

As mentioned in section 5.2, friends were recruited to contribute their queries to the data collection phase. It would only be fitting if these friends could test the results of their own queries. Ideally IIR would have been a live system, not a static setup, so the testers could adjust their queries when they produced too few relevant results. Due to circumstances described in 5.1, this was not implemented.

User testing was carried out between April 1st and April 7th 2017.

Four people contributed to the testing, their ages ranging from 25 to 57. Three of them are IT knowledgeable, having worked in the IT industry, one of them was

not an IT professional, but is using a computer on a daily basis as their primary tool. One tester is a female, three are men. This makes the age distribution reasonably good, male/female balance not quite so even, and the number of testers rather limited.

The solution was deployed on Microsoft Azure as a web site for the contributors to use, and a Questback[1] questionnaire was created for giving feedback to the author after testing the solution. The questionnaire is shown in appendix D.

The testers would get 6 queries each to test, 3 free queries and 3 exact queries. The testers were instructed to run each query at least 10 times, to simulate a minimum of 10 days' worth of searches. Furthermore the instructions were to save some interesting results, and discard some uninteresting ones. They could also optionally white-list domain names that would always give interesting results, and black-list domain names that would never give interesting results.

The aim of testing the solution was to get feedback from testers via the accompanying questionnaire, but also to analyse how they used the solution, by running some statistics on the results they leafed through, saved or discarded.

The author tested $8 + 8 = 16$ Queries, and answered the questionnaire. The author's experiences are mainly found in section 8.5.3, and in chapter 9.

7.2 Manual for using IIR

To help users test the solution, a manual was made to explain certain aspects and concepts. The full manual can be found in appendix C, it contains the main points shown below.

- What is IIR, what does it do, and how can it help the user.
- Main features of the application.
- Terms like Query, QueryRun and Result explained.
- Result statuses explained.
- How result retrieval works.
- About folders and filters.

1. `http://www.questback.com`

- Ranking the results.
- Reset ("panic button").
- Summary, simple instructions.

7.3 Instructions to the testers

The testers were given more explicit instructions on how to make the test phase more enjoyable for them and for me to get more useful results for post-test analysis. See the original instructions in appendix B.

- Use Google Chrome, it's not tested with other browsers.
- Use a screen width larger than 1200px, Bootstrap development isn't finalised.
- Open the IIR solution at http://iir-test.northeurope.cloudapp.azure.com
- Log on with your user (select your name from the dropdown on the login page), and you will see your queries displayed to the left.
- Read the online manual to be more aware what's going on.
- Try out some functionality on one of the searches first, to get a feel for the application. Then use the reset button at the bottom of the Discarded results folder, and start for real.

Then:

- For each search, go through more than ten sets of results for the search, really as many as you have the patience for. Upper limit is 54.
- Make sure you save some results you find interesting, and discard some results you find uninteresting. Save or discard single results or filtered results, use what you think works best.
- Optionally black-list domains that never have interesting results or white-list domains that always have interesting results for the search you are viewing.

After the list of instructions they were asked to open the Questback questionnaire and give feedback on the user experience of using IIR.

When you are finished testing, there are a few questions I would like you to answer. There is a Questback survey with a handful of questions, and comment fields for feedback and suggesting improvements.

7.4 Questionnaire

The testers would rate the following statements with a range of 5 possible ratings from a value of 1 for "completely disagree" to a value of 5 for "completely agree". This would place the value of 3 in the middle as a sort of "I do not know" or "I do not care" kind of answer.

- It was useful to be able to save results I found interesting
- It was useful to be able to discard results and never see them again
- It was useful to be able to white-list domain names
- It was useful to be able to black-list domain names
- This kind of application is useful to have, as a tool for searching

In addition to rating the above statements, testers were asked to give their comments on three feedback themes.

- How did you like having a system where you can save and discard results?
- Please add your comments on enhancements
- How useful do you think a fully implemented IIR system might be?

7.5 Questionnaire result

All participants filled out the questionnaire fully, rating the five statements and giving responses in the open text fields.

7.5.1 Statement scores

The score for each statement in the questionnaire is listed below.

#	Statement	B	C	D	E	Score
1	It was useful to be able to save results I found interesting	5	3	5	5	4.5
2	It was useful to be able to discard results and never see them again	5	2	5	5	4.25
3	It was useful to be able to white-list domain names	5	5	2	5	4.25
4	It was useful to be able to black-list domain names	5	4	5	5	4.75
5	This kind of application is useful to have, as a tool for searching	4	2	5	5	4
	Average score for each person, and total	4.8	3.2	4.4	5	4.35

Table 7.1: Statement score for the questionnaire. Possible score per statement is from 1 to 5. Columns B - E refers to test person B through E. Person A is the author.

7.5.2 How did you like having a system where you can save and discard results?

Person	How did you like having a system where you can save and discard results?
B	I really enjoyed using the system. The thing I liked the most was being able to blacklist different sites and having the opportunity to see all of my saved results.
C	The feature of saving results is useful. For this I have used a couple of services like f.ex Pocket (either as an extension to chrome or as an app on my phone) or just keep adding results to my bookmarks. Can't say I have felt a need for discarding results as Google usually brings up expected results for searches I do. Usually I visually discard results when looking through the results and trust that Google's algorithm actually brings the most relevant results to the first few result pages.
D	I like the idea of being able to save/discard.
E	I liked being able to discard and, most of all, blacklist domain names especially as they were repeated several times and made my search confusing. At the same time I liked whitelisting those domain names that I wanted to look at leisurely. That way I did not have to go back another day and start my search all over again as I do with other search engines.

Table 7.2: Replies in the first comment field

7.5.3 Please add your comments on enhancements

Person	Please add your comments on enhancements
B	Only minor GUI stuff, I had some difficulty understanding how to use the blacklist/white-list function at first. Overall did not use much time to get a hang of the system.
C	Obviously the search results only get as good as the search engine the queries are run against, so querying against google would enhance it a lot. Other than that it might have been nice to filter the queries by date to f.ex get the most current hits on a query.
D	- When hovering web page title, show full title as tool tip - Option to show more of the web page "content" - Show web page date somewhere (if the gray text after web "content" is a date, it has a bug) - Ability to see what the "score" is made up of on each hit (some sort of tool tip perhaps). It may make a difference. - List of domains to whitelist/ban, was not complete. I routinely felt I missed a domain I wanted to white list/ban. - Although this is a testing feature, it was unclear that the "New search result" would fetch the next day's result set. (Maybe I read the documentation to quickly) - Nice to have an option to discard from "all queries". I.e. Never show me this link again, regardless which query it appear in.
E	I was quite satisfied with the IIR system once I understood how I could use it. At this point I haven't got suggestions for enhancements.

Table 7.3: Replies in the second comment field

7.5.4 How useful do you think a fully implemented IIR system might be?

Person	How useful do you think a fully implemented IIR system might be?
B	I don't see myself using it often. I'm mostly searching for sites that that I can't remember the URL of. But for searching for complex or uncommon stuff the IIR can be super useful. And I think it can also be used in a commercial way for companies to scan the internet over time for any negative review about the company or any products they make, in order for them to fix the problems and contact users before the problems scale to the media.
C	From my experience I usually don't have the time or capacity to go through saved results other than the most important things which I need to have easy access to - and for this I use the good old bookmark functionality, which seems good enough. For the ad-hoc searches done through the day Google gives me the results I need the next day if I choose to search for the topic again, so I might not utilize an application like IIR. The simplicity of the search screen and scrolling through results from Google is more or less sufficient for me I think.
D	I think it potentially can be very useful. However, I felt I had to do a lot of clicking, and as a normal user I expect I would be tired of it quickly. I say go for it. I will use it if it exists.

E	Yes I find this application a useful tool for searching as it helps me narrow down my search to exactly what I want to find. It saves my time and that is a very significant thing about any system. It serves as a memory bank of my choices as well.

Table 7.4: Replies in the third comment field

7.5.5 Questionnaire result summary

This section will summarise results from the questionnaire, and also look at data behind the testing.

Scores

There were 4 users and 5 statements in the questionnaire to agree or disagree with, in total 20 scores. Out of these 20 scores were 14 scores of 5, 2 scores of 4, 1 score of 3, and 3 scores of 2.

The questionnaire had a total score of 4.35 out of 5, see table 7.1. This is 87% of the maximum score, which can be considered quite good.

Low scores are often interesting to examine.

Statement 1, about the usefulness of saving results, got a score of 4.5. One tester gave it a 3, commenting that there are other mechanisms for saving results, like Pocket[40].

Statement 2, with a score of 4.25, about the usefulness of discarding results got a score of 2 from the same tester that scored 3 on statement 1, which also is consistent with the comments about saving and discarding, see section 7.5.2.

Statement 3 about the usefulness of white-listing domain names, got a score of 4.25. The comments seem to indicate that the tester giving a score of 2 felt black-listing was more useful than white-listing.

Statement 4 about the usefulness of black-listing, got a score of 4.75, the strongest individual score of all the statements.

Statement 5 about the usefulness of this kind of application, got the overall lowest score of 4. The tester that gave the 2, commented that saving and discarding results did not seem useful, as Google should be trusted to give relevant results. The tester that gave a 4, did not really see the need personally for such a system, but suggested that the system could still be useful.

Comments

There were mostly positive comments on the question of saving and discarding search results. White-listing and especially black-listing also got thumbs up. As mentioned in section 7.5.5, one tester found it less useful. One tester found it useful to be able to continue a search without starting from scratch.

On enhancements, many valid points were listed. Some of them were; using another search engine source, being able to filter on date - and show - result date, show more of the results on mouse hover, better functionality around white- and black-listing.

On the usefulness of a fully implemented IIR system, opinions differed. Only two out of four testers would use this kind of system themselves. One tester found that it could be useful in some circumstances, and one tester found that it was not really useful for them at all.

7.5.6 Author testing

The author/creator of the IIR system also tested the system and answered the questionnaire. This would naturally not count as a part of the test result, since the author is biased, even if attempting to be neutral. The author's test results are consequently kept out of the questionnaire score and comments. See section 8.5.3 for a short discussion on the author's experience using IIR. Most of the author's improvements have made it into chapter 9 Future work.

7.6 Testing statistics

Table 7.6 shows totals for each query, and indicates how the testers used the system. The table columns are abbreviated to make the table more readable, so they need some explaining.

Column	Explanation
P	Short for "Person", the tester.
Query	The query tested, the *) marker signifies exact search.
QR	Total number of QueryRuns ("searches" done) for this query. Testers were instructed to do more than 10 searches per query.
WDN	Total number of white-listed domain names.
BDN	Total number of black-listed domain names.
New	Total number of new results across all QueryRuns that have the status *New*, see also explanation in table 4.2 in chapter 4.
Saved	Total number of saved results. The user have explicitly pushed a save button to make results appear here.

Seen	Total number of seen results. See explanation in table 4.2 in chapter 4.
Wlist	Total number of white-listed results. When the user white-listed a domain name, the new results in the current query would get the white-listed status. This would also happen to the folowing query runs.
Blist	Total number of black-listed results, based on black-listing domain names, similar to white-listing.
AutoD	Total number of results automatically discarded by IIR during search. Duplicates of already seen, saved or discarded results would end up here.
Disc	Total number of results, manually discarded by the user.
Total	Total number of results in Query, fetched by searching.

Table 7.5: Explanation for columns in table 7.6

See also detailed test results for each query in appendix E.

ID	Query	P	QR	WDN	BDN	New	Seen	Saved	Wlist	Blist	AutoD	Disc	Total
1	Messerschmitt KR200 restoration	A	54	6	21	1 951	529	23	67	426	36 781	417	40 194
2	Messerschmitt KR200 restoration *)	A	54	5	0	0	4	5	9	0	20 409	24	20 451
3	Web search API thesis	A	54	0	27	5 758	640	29	0	225	42 837	1 430	50 919
4	Web search API thesis *)	A	54	0	0	0	0	0	0	0	0	0	0
5	Web search thesis	A	54	1	29	2 631	584	11	2	232	46 945	1 738	52 143
6	Web search thesis *)	A	54	1	7	0	72	10	3	11	15 396	18	15 510
7	Search API thesis	A	54	1	18	1 543	565	12	2	135	42 184	1 462	45 903
8	Search API thesis *)	A	54	0	0	0	0	0	0	0	0	0	0
9	Messerschmitt TG500 for sale	A	54	0	18	1 144	452	32	0	215	36 827	330	39 000
10	Messerschmitt TG500 for sale *)	A	54	0	5	0	5	9	0	4	13 397	6	13 421
11	winds of winter	A	54	4	22	9 048	787	2	62	847	41 288	531	52 565
12	winds of winter *)	A	54	1	20	638	728	6	5	59	39 910	689	42 035
13	promise of spring	A	54	0	0	0	0	5	0	0	49 998	2 695	52 698
14	promise of spring *)	A	54	0	6	103	0	15	0	13	42 020	2 465	44 616
15	terry pratchett	A	54	4	18	709	506	33	58	211	42 545	1 375	45 437
16	terry pratchett *)	A	54	8	37	658	683	46	16	86	39 021	725	41 235
17	liverpool leeds efl	B	2	2	1	1 230	279	8	28	13	416	10	1 984
18	liverpool leeds efl *)	B	1	0	0	11	45	1	0	0	214	3	274
19	hillary clinton e-mail fbi	B	1	2	1	545	315	9	22	10	74	2	977
20	hillary clinton e-mail fbi *)	B	1	0	0	21	0	5	0	0	480	0	506
21	macbook pro 2016 touch bar problems	B	1	4	2	476	74	14	29	16	83	8	700
22	macbook pro 2016 touch bar problems *)	B	2	0	0	0	0	0	0	0	0	0	0
23	apple stock price	C	1	0	0	570	15	1	0	0	255	0	841
24	apple stock price *)	C	1	0	0	160	30	4	0	0	104	0	298
25	samsung note 8 release date	C	1	1	1	734	0	2	13	9	146	1	905
26	samsung note 8 release date *)	C	1	0	0	26	0	1	0	0	549	2	578
27	google self driving car	C	1	5	1	808	15	0	48	10	74	1	956
28	google self driving car *)	C	1	0	0	580	0	0	0	0	136	0	716
29	mobile application health sensor data	D	25	1	8	143	30	5	13	114	22 825	1 754	24 884
30	mobile application health sensor data *)	D	1	0	0	0	0	0	0	0	0	0	0
31	mobile phone body area network	D	37	3	74	0	0	0	7	378	34 538	1 567	36 490
32	mobile phone body area network *)	D	1	0	0	0	0	0	0	0	0	0	0
33	mobile phone sensor research health	D	54	0	14	2 495	223	8	0	230	47 696	2 344	52 996
34	mobile phone sensor research health *)	D	1	0	0	0	0	0	0	0	0	0	0
35	forest fairytales	E	19	1	1	8 530	270	39	25	12	10 065	32	18 973
36	forest fairytales *)	E	17	0	13	1 108	158	11	0	82	6 623	111	8 093
37	tudor politics	E	11	1	0	6 696	163	14	50	0	4 036	24	10 983
38	tudor politics *)	E	11	1	0	3 514	150	19	12	0	4 374	8	8 077
39	jazz poetry	E	13	3	0	7 266	180	20	145	0	5 252	30	12 893
40	jazz poetry *)	E	11	1	0	3 960	270	13	15	0	4 408	25	8 691

Table 7.6: Test totals for query 1 - 40. See explanation of columns in table 7.5.

7.7 Data analysis

Looking at table 7.6, especially looking at the column QR, we can see how much testing was done by the testers. Testers were instructed to do the search at least 10 times, covering 10 days worth of results. What the testers did varied quite a lot, with only the author, and test persons D and E, actually searching beyond the required 10 searches. The author searched all 54 days for all 16 queries, and test person D reached 54 days for one of his searches.

7.7.1 Test result plot details

Plots in appendix E are used as a basis for discussing the results in this chapter.

It is important to note that these plots show the *end state* of user testing per day. This means that the numbers plotted correspond to the hypothetical end state shown in figures 4.5b, 4.6b, 4.7b and 4.8b in section 4.8.

7.7.2 Few QueryRuns

Test person C only ran a single search for each query available. This means that trends over several days of search results could not be produced for query 23 - 28.

The same occurred for test person B's queries 18 - 22, except that this tester ran a second day of search for query 17, see figure 7.1. Figure 7.1 has only two nodes for each measurement, nevertheless trends that were described in section 3.2 and detailed in section 4.8 seem to be there. The data for figure 7.1 can be seen in table 7.7.

We can see both in the figure and in the table that more results were auto-discarded the second day than the first (from 143 to 273), and the number of new results were decreased from the first to the second day (from 796 to 434). There are some saved, discarded, white- or black-listed results, but the number of *Seen* results increased from 24 to 255 from the first to the second day. This means that the tester has been paging around in the results the second day.

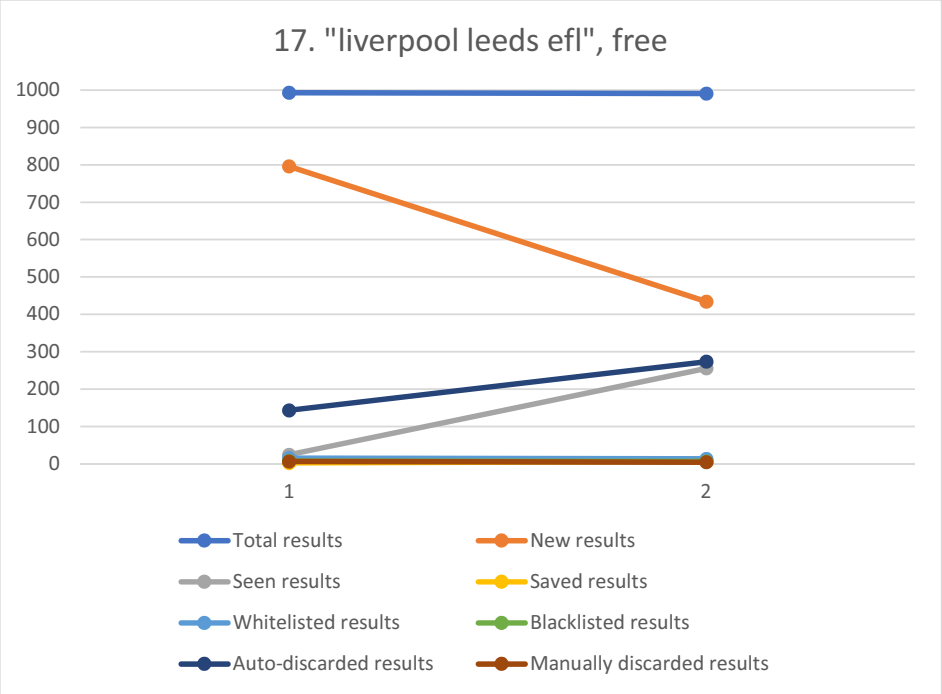


Figure 7.1: Detailed test results for query 17, only 2 days of results examined by tester.

Day	Total	New	U.New	Seen	Saved	W List	B List	A Disc	M Disc
1	993	850	796	24	2	15	7	143	6
2	991	66	434	255	6	13	6	273	4
Sum	1 984	916	1 230	279	8	28	13	416	10

Table 7.7: Detailed test results for Query 17, "liverpool leeds efl".

A short explanation of the columns in table 7.7 may be needed. Gray columns are; Day, Total, New results for Query 17 (from table A.13). The white columns contain data from the user test, shown in figure 7.1; User's New (New after user actions), Seen, Saved, White-listed, Black-listed, Automatically Discarded, and Manually Discarded results.

7.7.3 Energetic usage pattern

This kind of usage can be seen in many of the author's plots, but it is more appropriate to look at test person D's first query, see figure 7.2.

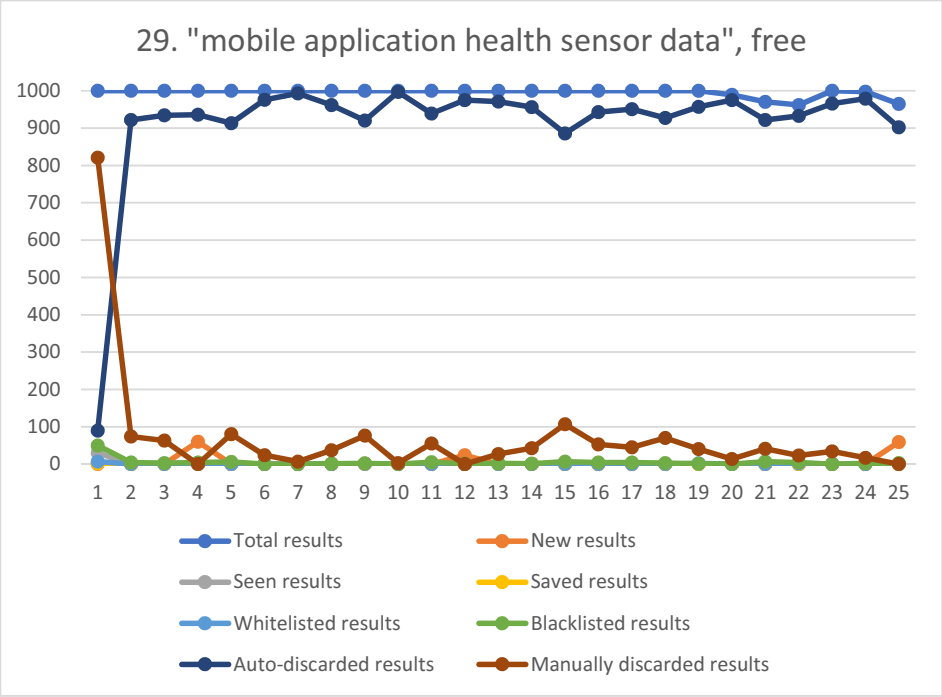


Figure 7.2: Detailed test results for query 29.

In figure 7.2 it is evident that a lot of work has gone into day one. According to table 7.8, 821 results have been manually discarded from day one to the second day, and on day two, most results have been auto-discarded as a consequence of this work. This has led to much fewer new results each following day.

We can also see that manual discarding of results continued throughout the period of 25 days (QueryRuns), and that almost all results each day were auto-discarded, as a consequence of this.

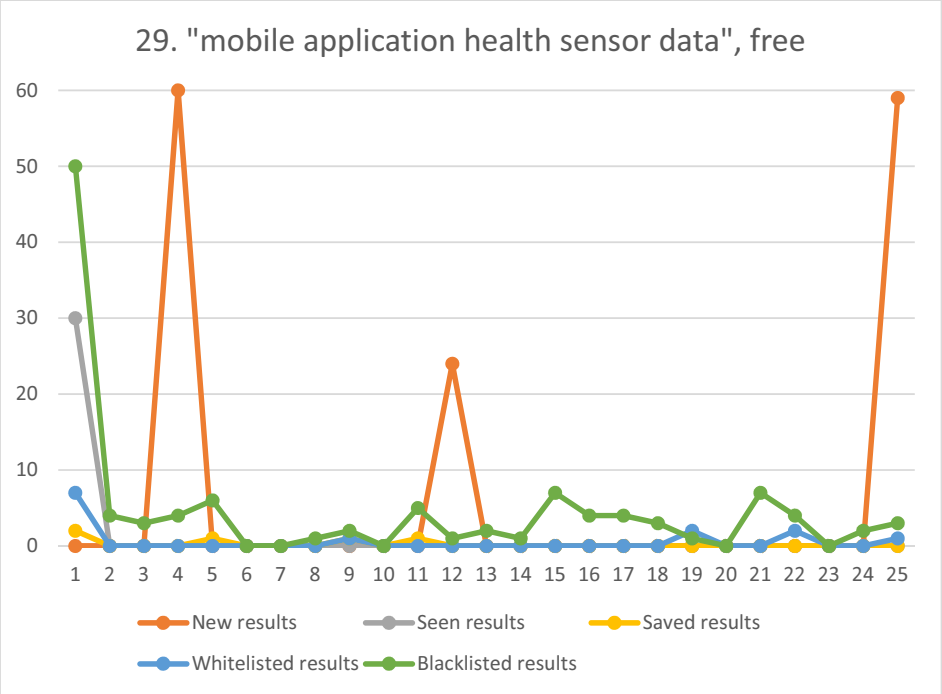


Figure 7.3: Detailed test results for query 29, showing the lower part of figure 7.2.

In figure 7.3 it becomes clear that during the period of twenty-five days, the results were saved, white- and black-listed, in addition to the manually discarded results shown in figure 7.2, so that all results classified. This has led to no results with the *Seen* status.

Day	Total	New	U.New	Seen	Saved	W List	B List	A Disc	M Disc
1	1000	910	0	30	2	7	50	90	821
2	1000	78	0	0	0	0	4	922	74
3	1000	66	0	0	0	0	3	934	63
4	1000	64	60	0	0	0	4	936	0
5	1000	50	0	0	1	0	6	913	80
6	1000	15	0	0	0	0	0	976	24
7	1000	5	0	0	0	0	0	993	7
8	1000	37	0	0	0	0	1	962	37
9	1000	80	0	0	1	1	2	920	76
10	1000	3	0	0	0	0	0	997	3
11	1000	61	0	0	1	0	5	939	55
12	1000	25	24	0	0	0	1	975	0
13	1000	23	0	0	0	0	2	971	27
14	1000	42	0	0	0	0	1	956	43
15	1000	113	0	0	0	0	7	886	107
16	1000	57	0	0	0	0	4	943	53
17	1000	48	0	0	0	0	4	951	45
18	1000	73	0	0	0	0	3	927	70

19	1000	43	0	0	0	2	1	957	40
20	989	14	0	0	0	0	0	975	14
21	970	48	0	0	0	0	7	922	41
22	962	29	0	0	0	2	4	933	23
23	1000	34	0	0	0	0	0	966	34
24	998	19	0	0	0	0	2	979	17
25	965	63	59	0	0	1	3	902	0
Sum	24 884	2 000	143	30	5	13	114	22 825	1 754

Table 7.8: Detailed test results for Query 29, "mobile application health sensor data".

A short explanation of the columns in table 7.8 may be needed. Gray columns are; Day, Total, New results for Query 29 (from table A.19). The white columns contain data from the user test, shown in figure 7.2; User's New (New after user actions), Seen, Saved, White-listed, Black-listed, Automatically Discarded, and Manually Discarded results.

7.7.4 Relaxed usage pattern

Another pattern emerges in test person E's first query, see 7.4.

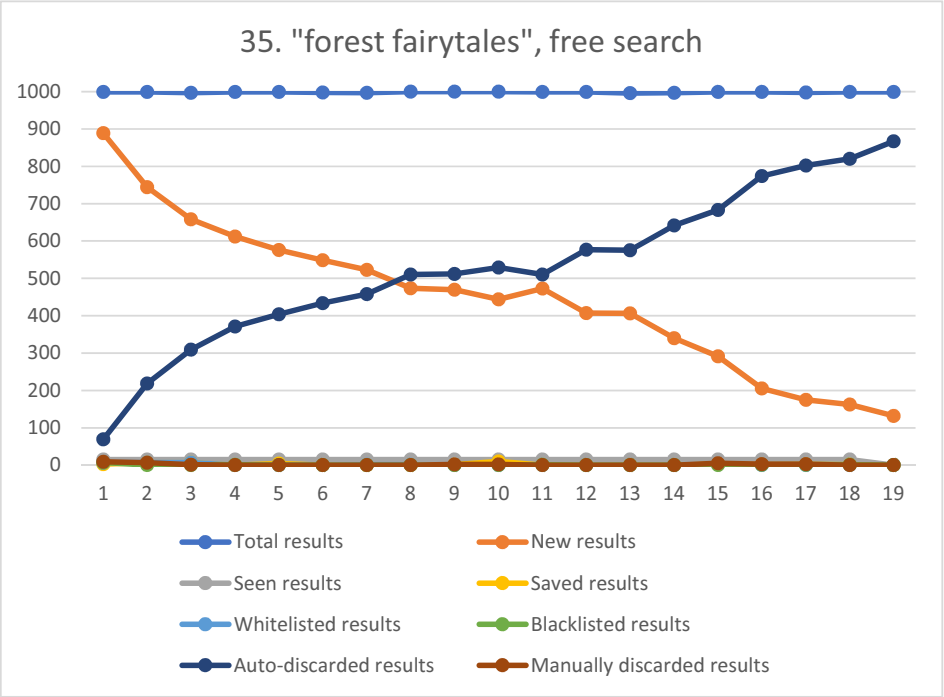


Figure 7.4: Detailed test results for query 35.

This is a period of nineteen days, where the tester has seen some results, saved some, white-and blacklisted a few domain names, and discarded some results.

The tester has processed few results for each day, but still the number of new results shown in figure 7.4 decreases steadily, and the number of auto-discarded results increases for each day in the period.

See figure 7.5 for clearer details on the lower part of figure 7.4. In figure 7.5 the user's actions become more apparent. During the whole period of nineteen days, the user has saved and manually discarded results. This points to the fact that there are interesting results to find even towards the end of the nineteen days. As the numbers in table 7.6 for Query 35 points to, the user has saved 39 results and manually discarded 32 results.

Figure 7.5 also contains white- and black-listed results including day one and throughout the nineteen days period. Table 7.6 shows only a single domain name white-listed, and a single domain name black-listed for Query 35. This means that the user on day one in the period white-listed one domain name, and black-listed another. These were applied as filtering, as described in section 4.3.1, and came into effect the whole rest of the period. Automatic white-listing was applied on day 2, 3, 7, 8, and 15, and automatic black-listing on day 4, 11, 16, 17, and 18.

For each day, the number of seen results is constant, at 15, the same as the number of results displayed per the page in the IIR web client. This means that the user did not find it interesting to page around in the result list, which probably would have been a double-digit number of pages. So when the search button was pressed to find the next day's results, the displayed results on the shown page was saved as *Seen*. This is also why there is nothing for day nineteen, the user just stopped looking at Query 35. This can be verified by looking at the *Seen* column for Query 35 in table 7.6. The number of seen results is 270, the page size is 15. This gives $270 \div 15 = 18$, which is the exact number of days before the user stopped testing Query 35.

The actions revealed in figure 7.5 shows that IIR helps the user by automatically hiding from view, results that the user normally would have to relate to.

Day	Total	New	U.New	Seen	Saved	W List	B List	A Disc	M Disc
1	999	930	889	15	3	7	7	69	9
2	999	37	744	15	7	7	0	219	7
3	997	29	658	15	7	7	0	309	1
4	999	58	612	15	0	0	1	371	0
5	999	29	576	15	4	0	0	404	0
6	998	12	549	15	0	0	0	434	0
7	997	8	523	15	0	1	0	458	0

8	1000	23	474	15	0	1	0	510	0
9	1000	45	470	15	1	0	0	512	2
10	1000	3	444	15	10	0	0	529	2
11	999	175	473	15	0	0	1	510	0
12	999	48	407	15	0	0	0	577	0
13	996	29	406	15	0	0	0	575	0
14	997	42	340	15	0	0	0	642	0
15	999	58	291	15	3	2	0	683	5
16	999	22	205	15	1	0	1	774	3
17	998	17	175	15	2	0	1	802	3
18	999	35	162	15	1	0	1	820	0
19	999	17	132	0	0	0	0	867	0
Sum	18 973	1 617	8 530	270	39	25	12	10 065	32

Table 7.9: Detailed test results for Query 35, "forest fairytales".

A short explanation of the columns in table 7.9 may be needed. Gray columns are; Day, Total, New results for Query 35 (from table A.22). The white columns contain data from the user test, shown in figure 7.4; User's New (New after user actions), Seen, Saved, White-listed, Black-listed, Automatically Discarded, and Manually Discarded results.

7.7.5 Other patterns

In other queries tested, we can see from most plots in appendix E that the testers followed one or the other of the patterns described in sections 7.7.3 and 7.7.4, or a combination of these patterns.

In many plots, the curve for auto-discarded results closely follows the curve for total number of results.

This can be seen both in energetic and relaxed usage patterns. Figure 7.6 shows a relaxed usage pattern, where curves for auto-discarded and total converge slowly, and then follow each other closely for the rest of the tested period.

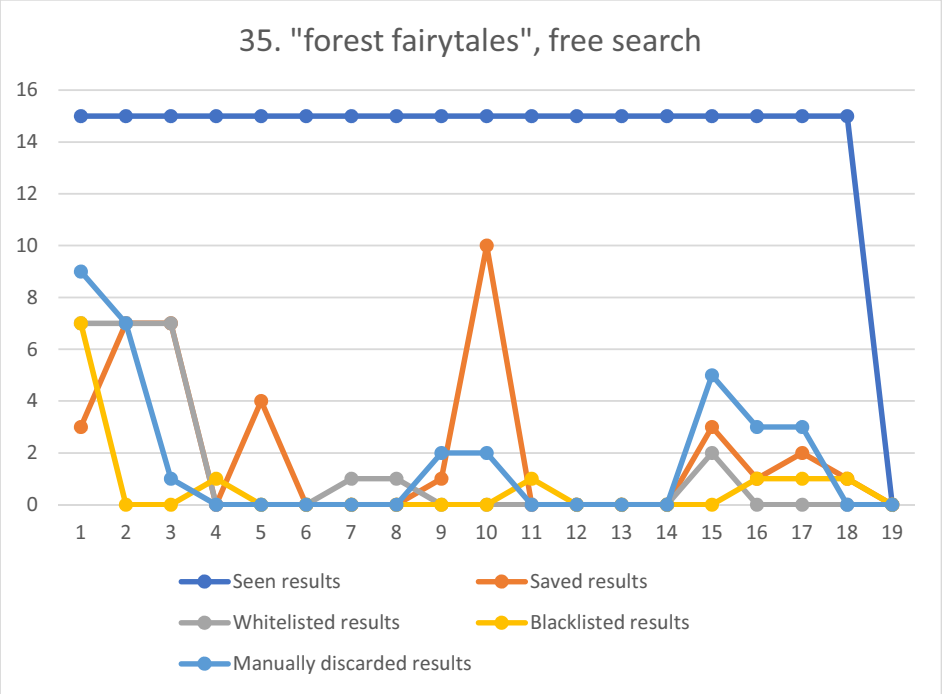


Figure 7.5: Detailed test results for query 35, showing the lower part of figure 7.4.

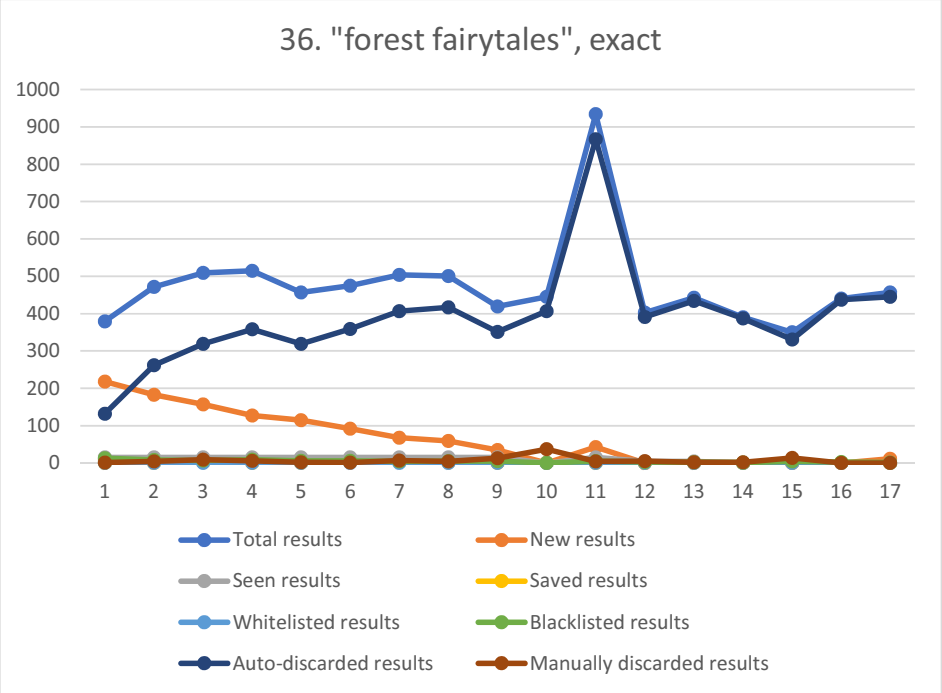


Figure 7.6: Detailed test results for query 36.

The same can be seen in the energetic patterned query shown in figure 7.7, where curves for auto-discarded and total follow each other closely from the second day of the tested period.

Convergence of result total and auto-discarded results may happen when the user is very active and saves or discards almost all results for one day in the period. It can also happen when the total number of results for one Query are very low, and the user follows a relaxed pattern, as shown in figure 7.6.

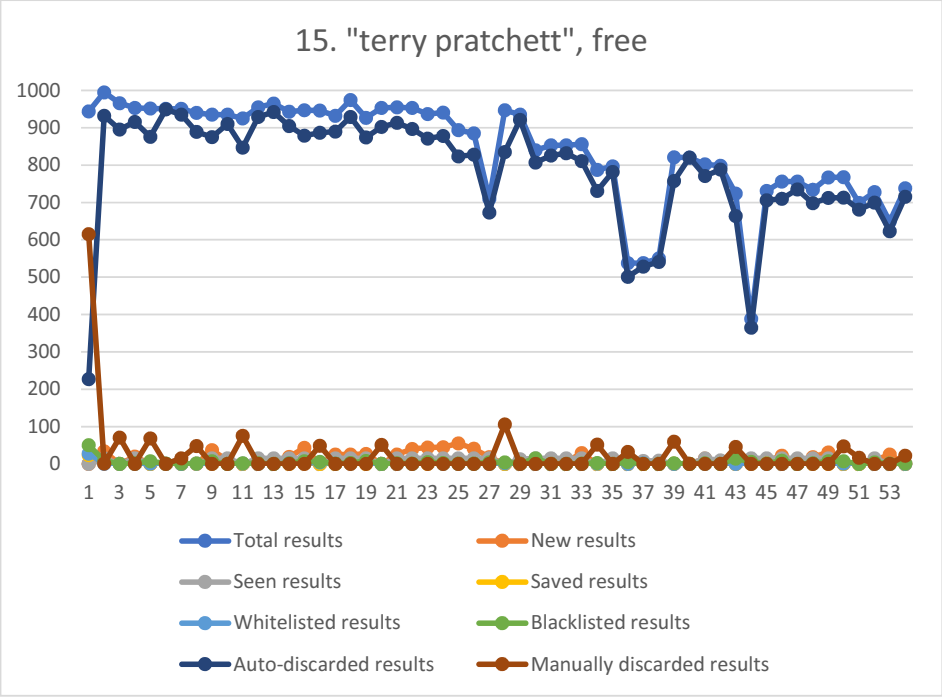


Figure 7.7: Detailed test results for query 15.

8

Discussion

This chapter examines choices made in setting up the IIR service, its environment and technology, and inspects findings and implications. Some of the limitations of this study will also be addressed.

8.1 Findings

This study set out to find if a system could be implemented to help the user find search results they had not seen before. The research question, asked in section 1.2, was "*How can a long term search service be created to discover previously unseen search results, regularly concealed in traditional on-line search?*".

The study has shown that search engines, by trying to give the most relevant or concise answers, repeatedly give the same or similar results.

By analysing data collected from Bing, figure 5.4 in chapter 5 shows that the first day of results has many new results, but the second and subsequent days have many results that were seen the first day, and very few new results. The same pattern is also shown in table 5.3 in chapter 5, when comparing the 100 first results from day one and day two in the data collected.

When looking at the testing phase, *e.g.* figure 7.4 in chapter 7, the progression of user operated searches behave differently, since users do not interact with

all results from day one, as the data collection analysis did. By examining the test data it is clear that such an IIR system will work with minimal amount of work from the user's side. The more the user uses the system, the more results are discarded as already seen or black-listed.

8.2 The approach to search

When researching ways of searching at the start of this project, different search engines and various technologies were examined.

Collecting data by way of "web scraping"[1], parsing results directly from the HTML returned from an on-line search, is not the way to go. Content based companies do not like[2] that their content is "stolen"[3], search engine companies least of all - they live by showing ads.

Setting up a search engine for the project by using open source search engine software like Lucene[4], Sphinx[5], Xapian[6], Indri[7], or Zettair[8], would complicate the project unnecessarily.

So, very early in the process it became clear that to limit the work of this thesis, choosing a well known search engine with a useable API was the right way to go. The API, at the time of choosing, that seemed to fit the task best was Bing Search API v2, as described in 4.7. It had a maximum of 1000 results returned as JSON per search, had 5000 free transactions per month, and had a reasonable set of parameters that the API could be configured with.

As mentioned in section 2.4.2 *On-line search vs search APIs*, the on-line and API versions of the search engines differ in which and how many results they return. This will give different results when searching with our IIR system, but this will only be evident in the very first batch of search results from a query. The subsequent searches will differ substantially, because the IIR system will compare the results with the database, and only present results that have not already been seen, as described in chapter 3.

But there is another important difference between on-line search and API-based

1. http://wiki.c2.com/?WebScraping
2. https://techcrunch.com/2016/08/15/linkedin-sues-scrapers/
3. http://blog.icreon.us/advise/web-scraping-legality
4. https://lucene.apache.org/
5. http://sphinxsearch.com/
6. https://xapian.org/
7. https://www.lemurproject.org/indri/
8. http://www.seg.rmit.edu.au/zettair/

search. All the main search APIs have a limited number of results they are willing to return to the consumer. In Bing's case this was 1000 results. This means that the search is only so good as the content that the search engine returns as the 1000 results. There are generally a lot more results available on the Internet, and in the search engine's database. A quick search for "terry pratchett" on Bing on-line search gave "618000 results". So the Bing Search API has to decide which 1000 of the those 618000 results to return.

This situation reduces the usefulness of the service consuming the API. What if we are looking for a result that is found as number 1001 - or number 600001. The results are ranked in Bing, if the ranking Bing uses does not suit the user running the query, the desired results may never be returned as one of the 1000 results.

Also, the Bing Search API v2 had limited parameter options, and no time period selections. It had no parameter based domain name limitation features, but this could probably be mitigated in a live search situation by experimenting with adding domain names to the search text string at the time of building the URI to send to the API. No such parametrisation was done due to the data collection situation, see description in chapter 5 and discussion in section 8.4.

There has been a reduction in available search engine APIs since the start of working with this thesis. Table 2.1 in chapter 2 shows that the big companies seem to find less value in offering their search index to the public. At the time of selecting API to work with, the top three search engines to choose from, Google, Bing and Yahoo, had in total four possible API candidates, though in reality Google's two candidates were only meant for searching a limited number of sites. Google has now announced the death of one of their two offerings, Google Site Search, and Yahoo BOSS API was shut down in 2016. Bing is the only major search engine that has kept their general purpose search engine API, but in renewing it Bing created a problem for this thesis, as described in section 5.1.

8.3 Implementing the prototype

The IIR solution was in total worked on over a period of two years. The intention was to create IIR as an API based on-line system for user testing, and a separate batch system for comparing IIR results with the raw results from the API.

The server and search API parts of it was originally implemented in nodejs, and

this was perfect for the web server part of the solution. However, nodejs with its asynchronous nature, was not suited for the offline (batch) routine.

This led to re-implementing everything in Go, which was an good fit for both the on-line and off-line system. There was no wrapper or adapter in Go for connecting to the Bing Search API, so this had to be implemented as well, as described in section 6.5.2. Using Go for the IIR server and utilities (see appendix F) turned out to be a delight. The snappy, sub-millisecond localhost web request and database access timings were especially nice.

8.3.1 Storage

If the user-base expands significantly, storage needs to be managed. In the IIR implementation, everything is stored for all results, even duplicates, of which are many. This has been necessary in the prototype, but in a fully implemented system this should change.

All results stored in the IIR database that the user will never want to see again, like auto-discarded and manually discarded results, could be stripped of superfluous information. They could *e.g.* after some time be compacted, in that only the URL hash, and possibly the URL itself remains.

Other methods of compacting storage could be to experiment with compressing text fields in results, like Title, Description, DisplayURL and URL.

It is difficult to estimate exactly how much storage can be recovered in this way, especially since usage patterns probably will vary a lot between different users. Section 4.6.2 does some calculation on this, but does not include the suggestion mentioned in this section. The storage strategy will also depend on the final implementation, see sections 9.3 and 9.4.

8.3.2 Usability

The IIR user interface had a limited number of features, but it had enough features to make users find it usable, despite its limitations. This is after all a prototype implementation, and feedback on how the UI works, is a part of the evaluation of the prototype.

As discussed in section 8.5, some criticisms include "a lot of clicking", initial difficulty understanding white- and black-listing. Much clicking was needed to go through the instructed amount of results. This was also the author's experience, when testing the 16 Queries. The suggested solution for this is creating rules. More on rules in section 9.1.2. The UI can be much improved

upon, many suggestions have found their way into chapter 9.

8.3.3 Reliability

Users have to trust IIR, even after using it for a long time. The full IIR system consist of several moving parts, the main two are the IIR web service (web client and web server), and the connected data provider. If any of these should fail, the IIR would fail. Through the testing phase this did not happen, but then again the testing was done with collected data, not a full API-based IIR system, as described in 6.5.4.

The IIR web client needs to be rock solid and bug-free to make the solution work. It also needs either to support many different versions of web browsers, or preferably demand that the user has a modern web browser. The IIR server including database needs to be always online, to be able to support the web client. The search API needs to always be on-line to support IIR. If the data source is unavailable, IIR stops being useful, and can only show data stored in IIR. The reliability of the system is dependent on these parts.

8.3.4 Scalability

Scaling of the solution will depend on several factors. One is the matter of storage, discussed in section 8.3.1, and storage is dependent on how search is finalised, see section 9.4.

Deploying the web service in the cloud as suggested in section 4.6.1 would make it easy to scale the infrastructure. As Go is the chosen language, Google seems like a good fit, though AWS and Azure probably could provide good service as well.

As described in section 9.1.5, the application needs to handle the user's growing number of queries.

8.4 Data collection

In section 5.1 the reason for using data collection was explained, that version 2 of Bing Search API service would be terminated. This made the data collection necessary, and data was collected in a period of 54 days.

The actual data collection was a satisfactory experience, once it was scripted

and set up as a scheduled task. It performed very well, and got data collected daily through the whole period.

But there were errors. Data collection errors are described in section 5.5. Most of the errors experienced, shaved off a bit of the collected data. In most cases only a "page" was missing, meaning somewhere between 0 and 50 results, for the relevant query on the day the error happened.

8.4.1 Ranking

As described in section 4.4.2, IIR has an optional proprietary ranking of results, based on if the searched for text query was found in the title or description in each result. IIR ranking has the advantage that if your query text is exactly what you are looking for, as in the "terry pratchett" query mentioned in section 8.2, the ranking gives all these results at the top of the result list.

On the other hand, the search engine's ranking is made for actually helping the user to find results, so in some cases the results ranked high by the search engine would fall behind on the IIR ranking. One example of this is the search for "messerschmitt KR200 restoration", where results that linked to selling spare parts were ranked low, since they did not have the exact words in the result.

This feature should have been tested properly, to see if IIR ranking has merit, but because of time constraints, this was not done. This feature could be a useful addition to the IIR features, but should perhaps have been implemented as a part of a set of sorting mechanisms for the results. As an example of such a set of sort orders; sort by IIR rank, sort by search engine rank, and sort by modified date.

8.4.2 Query quality

The way the queries are created influence what results are returned from the search engine API. Search engine techniques like stemming and similar techniques, described in the section 2.1.2, influence what results the search engine returns through the API.

Collecting data to use for search as described in chapter 5, gives a very static set of results to search for and show in IIR. When using an on-line search engine, the user will change the query when the relevant results are not found. The user had originally the possibility to adjust the query text in IIR, but this was abandoned in the final prototype because of the data collection situation.

When using a static set of data collected for IIR, adjusting the query text is less useful, since the changed search is performed on a stored (unchanging) set of data retrieved based on the original search text.

8.5 Test results

IIR was tested by four users during the course of a week. The users had contributed the search tests in advance, and the fact that they were testing their own Queries, would make the results more relevant for them.

There are several issues with the test scenario.

Time The testers contributed their queries late October 2016, the test period was the first week of April 2017. This is a time period of 5 whole months. The contributed searches that were current, would be old news after 5 months. This is the case for some of the Queries, *e.g.* Query 17 "liverpool leeds efl" and Query 19 "hillary clinton e-mail fbi". So in a way, the search results would not be relevant for these testers after such a long time.

Few testers More people to test IIR would have been an advantage. There is not much statistical significance in such a small number, and could potentially skew the findings.

Collected data Ideally the testing should have been done on live data, by connecting to a live search engine. If this was the case, the users could adjust their queries, to get more relevant results for their queries if they wanted.

Two of the testers did not follow the instructions given, or did not fully understand them. This was unfortunate, in that they did not generate enough test data to analyse their testing process properly. Another consequence was that their impression of the IIR system might be coloured by not having the full experience of having a search progression over several days of data, like the other testers did. The instructions were given in several different ways, but maybe they could have been improved upon, or emphasised more. IIR is a different approach to search, after all.

Another view could be that they really did not see the need for this type of search, as one tester commented through the questionnaire. It could also be that the searches they had contributed did not have that much relevance for them at the time of the test, five months after contributing them.

Still, even for Query 17 that was run for two days, see figure 7.1 in chapter 7,

there was the suggestion of a positive outcome. And for the testers that tested more than ten days, there was a clear trend, whether they tested energetically as shown in figure 7.2, or leisurely as shown in figure 7.4.

IIR also had a limited UI implementation, something that also could have confused or discouraged the testers, as hinted at in the introduction to chapter 7.

8.5.1 Exact search

With exact search, the total number of results in a QueryRun is often a lot fewer than the total number of search results from a "free" QueryRun. One could imagine this as normal, but at the same time, it is unusual for a search engine to return less than 1000 results, for a correctly worded Query. The reason for this is unclear, one could speculate if this is one of the differences between on-line and API based search.

In my opinion the exact versions of the Queries did not work that well. They yielded markedly fewer results than the free queries, and contained many results that were duplicates.

8.5.2 Analysing new results

When plots from the data collection show new results for the whole period since the first day, it is clear that the first day of results have most of the new results. The following days in the period have fewer new results, since all previous runs are taken into account when finding new results.

When it comes to implementing a user based search via UI, this way of counting new results has limitations. It relies on the user going through all the up to 1000 results from the first search. This would ensure that all the results from the first search can be counted as seen.

In practice this would rarely be the case, as described in section 3.3. The user would be going through *some* of the results from the first search, but not all. This would mean that many of the results from the first search would not be filtered out in the second and third search, and would appear in the second and third search as well. So if the user does not take any action on these reappearing results, they could possibly hide more relevant results, and in this respect be regarded as "noise" in the result.

8.5.3 Author's comments on testing IIR

It was nice to be able to sort of bookmark results. Discarding results and never seeing them again is a very useful mechanism, and so is black-listing. White-listing, not so much.

But the user interface needs a lot of work. As one of the testers commented, "clicking" is a keyword here. Knowing the system more intimately than the other testers made it easy to use the system more efficiently.

Most of the author's suggestions have made it into chapter 9 Future work.

8.6 Problems, bugs and errors

Data collection errors were described in section 5.5. The major errors were fixed, and data collection worked well afterwards. The minor errors that occurred after the initial error period will change the number of results collected, but will not significantly alter the number of results collected. Errors are not many, and each time only 0-50 results out of a potential 1000 are missing. If considering the number of results that overlap between days of the data collection period, as described in section 5.6.4, the number of previously unseen results missing could in fact be closer to zero.

No bugs have been reported in the IIR test phase.

There was one problem that was not reported by any of the testers, but was discovered during the author's testing. The test data collected were examined for duplicates, as described in section 5.6.5. Search result URLs were lowercased and then hashed and compared, and in this process the lower-cased URLs were by mistake saved back to the database. This would not be a problem for most URLs, but when testing I found that some results were not possible to preview by opening them in the web browser. Some YouTube[9] and Imgur[10] links did not work.

On closer inspection the problem turned out to be that identification keys, for videos and images respectively, are case sensitive. An an example, try `http://imgur.com/r/golang/wVOArpG` (works) versus `http://imgur.com/r/golang/wvoarpg` (not found). This issue would have little impact on how IIR works, and the thesis as such.

9. `http://www.youtube.com`
10. `http://imgur.com/`

There is a slim chance that an URL in lower-case actually existed for the relevant Query. The user would then accept that the lower-cased result URL was correct, since the link works when opened. The mixed-case URL would then be obscured by the lower-cased link, and never appear as a consequence of this. If content previews as described in section 9.1.1 had been implemented, the problem of lower-cased URLs would in some cases have prevented content previews, hindering users in deciding if these results were relevant or not.

8.7 Is it commercially viable?

In its current implementation IIR is set up to search on the user's behalf by reading from existing search engines via their web search APIs. These services are not free, and some financial model needs to be researched to be able to make a system like this available to the public.

In addition to the financial aspects, there is also a matter of storage, see discussion in section 8.3.1. Storage is also mentioned in section 9.1.5.

9

Future work

Personal experience has shown that an application in most cases *never* reaches a finished state, and there is always possibility for improvement.

The aim of a proof-of-concept type application is to show that something is possible, or show how something can be solved. This kind of development involves a good amount of trial and error, technical changes, code patching and quick fixes. In the end, an answer emerges for how this application can be implemented, but the proof-of-concept implementation itself is often riddled with technical debt[1][2].

A full rewrite of the application is recommended now that different features have been explored. The application is a fairly small one, and it is not deployed to production, so a rewrite is feasible.

However, a refactoring[3] of IIR is also possible. Details around refactoring are discussed in section 9.2.

1. http://wiki.c2.com/?TechnicalDebt
2. https://martinfowler.com/bliki/TechnicalDebt.html
3. http://wiki.c2.com/?WhatIsRefactoring

9.1 New features

This section discusses what new features would improve the existing solution. All the described improvements should be considered experimental, and would have to be specified, implemented and tested to see if they make sense in the context of this application.

9.1.1 Content preview

The content for each result in the solution is limited to what the search engine can provide.

This was in Bing Search API's case a result title of 65 characters, and description of 170 characters. This is in many cases not enough to assess if the result is interesting or not, so the result becomes "more noise than nice".

One feature that could mitigate this would be to show a preview of the content that the result's URL refers to. This could possibly be solved as an *<iframe/>*[4] html tag. There are some security concerns[5] with using an *<iframe/>* tag, though, like cross-site scripting (XSS), that needs to be handled.

9.1.2 Rules or filters

Through feedback from the testing phase, it became evident that creating some kind of rules would benefit the solution. These would be filters that are applied on new results before showing results to the user. This would automatically save results that the user already knows would be interesting or discard results that the user knows would be uninteresting, relieving the user of unnecessary work. Rules would be applied while in the process of fetching new results from the search engine.

The rule could be set up to act on the result's fields, which would depend on what fields would be available from the search engine. Examples of such filters could be:

- "if result.title contains [notinterestingtext], discard result automatically"
- "if result.url contains [urltowhite-list], white-list result automatically"

4. `https://www.w3.org/TR/html5/embedded-content-0.html#the-iframe-element`
5. `https://www.w3.org/Security/wiki/Cross_Site_Attacks`

- "if result.domainname equals [urltoblack-list], discard result automatically"

The rule would in its simplest form contain a criteria and an action. If the criteria is true, the action would be performed.

The criteria could for instance contain some comparison between fields in the result and a user specified text to compare against. Comparison operators could as a start be *equals* and *contains*.

The fields to test against could be *title*, *description*, *URL*, *domain name*, and also *date*, if available.

The possible actions could be to save or discard results.

A rule could also be set to *global* for this user, meaning that the rule could work across all the user's current and future queries. The user should use global rules with caution, as these kind of filters could remove results that would have been interesting in other or future queries.

Creating rules would also diminish the need for the user to white-list or black-list domain names. The solution should instead help the user create a new rule in an easy way.

Rules could also be used to add other features like custom ranking of results instead of saving or discarding them.

9.1.3 White- or black-listing

White- or black-listing domain names could be toned down. Still, there should be a way to suppress many domain names in one go. While testing the query "Terry Pratchett" (simpleId 15), there were literally thousands of results like the following, that clearly had nothing to do with "Terry Pratchett".

- http://ageing-body.review/skin/terry.pratchett.wrinkles
- http://ageing-body.review/skin/terry-pratchett-wrinkles
- http://ageing-calm.review/skincare/terry-pratchett-wrinkles
- http://ageing-data.review/antiageing/terry-pratchett-face-cream
- http://ageing-feel.review/antiageing/terry-pratchett-face-cream
- ... lots and lots more ...

The number of unique domain names for this particular case were somewhere in the hundreds. To be able to add many of these to some sort of black-list *at the same time*, would be very valuable. This could be implemented by creating

a special case of a rule, or one rule for each selected domain name, see section 9.1.2.

As found in the testing phase, there would be a problem with internationalised domain names, like *.blogspot.com, *.blogspot.se, *.blogspot.in, etcetera. These would show exactly the same content with only the root domain (.com, .se, .in) as the difference between them. Handling these domain names may be more difficult to do in a general way, without storing internationalised domain names somewhere in the solution. Storing default internationalised domain names is not be recommended for a more generic application like IIR, but some mechanism for handling those domain names would help the solution.

9.1.4 Analysis of user interactions

There could be implemented analysis of the user's interactions with IIR, and automated creation of rules, described in section 9.1.2.

This could for instance be based on analysing the number of times the user has discarded results for a domain name. Such an action could result in an automatic creation of a rule that blocks this domain name.

Additionally, analysis could have been made on for how long a user previews content, see section 9.1.1. This could be automatically saved if previewed longer than a threshold.

If the user always finds results from a particular domain name interesting, maybe a rule should automatically be created for that domain name, so that results from that domain name are always saved.

9.1.5 Handling many queries and results

When the number of queries accumulate, the need arises to manage them in some way.

Refer to figure 6.2 in chapter 6. The list of queries to the left should probably only contain the latest active queries, and the less active queries should be listed in some kind of archive.

When a query is selected, the folders that show results from all QueryRuns (seen, saved, white-listed, discarded) should be enhanced with sorting, grouping, search and selection criterias, to make navigation easier. This can *e.g.* be realised as a separate view mode, showing a data grid with columns, in addition to the standard search result view mode like today.

Since all Queries and their results take up storage, some scheme should be devised for letting Queries expire, removing their uninteresting results while keeping their interesting results. This could be done automatically after some time of inactivity, or storage use could be brought to the attention of the user at some stage, and the application could make the user decide what to do with these inactive queries.

9.1.6 Miscellaneous features

As a help to handling storage, ref section 9.1.5, there should be a way of presenting statistics to the user. The number of each status like seen, saved, white-listed, discarded could be displayed, and estimated storage used, for instance.

In addition to saving results to IIR, a feature could be implemented to save the result to Pocket[40]. This would immediately update all Pocket-ready devices or applications with the chosen result.

9.2 Refactoring

There are many important technical improvements that can and should be done to this application. Refactoring is the process of changing a software system without changing its external behavior[18], and all software has, at some stage, need for refactoring.

Being a temporary type of implementation, many things can be done in higher quality. As soon as a piece of code is written, many developers will already refer to it as legacy code[17]. See also the introduction to this chapter on page 111.

9.2.1 Front-end

The UI should first and foremost be expanded to handle mobile clients. This is work for a Bootstrap layout designer, but this should be reflected in client code.

The JavaScript front-end will benefit from being rewritten in a AngularJS version greater than 1.x. Version 2 was released fall 2016, and come March 2017 the version number is already 4. The newer versions are superior to version

1.x[6] [7]

Implementing the JavaScript part of the client using TypeScript[8] gives many benefits, arguably chief among them type safety in JavaScript.

Go templates for HTML should be rewritten to be more modular and support Angular and Bootstrap better.

Error handling should be improved, see section 9.2.4.

9.2.2 Back-end

Smaller, shorter, more dedicated Gin controllers would make the back-end more stable and easier to maintain.

Error handling should be improved, see section 9.2.4.

9.2.3 Unit testing

This proof of concept solution has no unit tests neither in server nor client. This should be implemented, preferably from scratch with test-driven development (TDD[9]), but tests should be added no matter which style of programming is used.

9.2.4 Error handling

Error handling can be much improved. When the web service has a problem, it should be able to recover from it and continue to serve users smoothly. If the error was from a user error, the user should be notified. If a system error occurred, the user should know about it if the user experience is interrupted or influenced.

One relatively easy way of showing errors to the user is by using the JavaScript library toastr[10].

6. http://blog.angular-university.io/introduction-to-angular2-the-main-goals/
7. https://www.quora.com/What-are-the-advantages-of-angular2-over-angular1
8. https://www.typescriptlang.org/
9. https://www.agilealliance.org/glossary/tdd
10. https://codeseven.github.io/toastr

9.3 Change in type of application

Implementing the application, with thousands of users, puts demands on how storage is dealt with, as implied in section 4.6.2.

One way of solving this is to create IIR as a desktop application, using the storage space on the machine it is installed on. This way, all storage problems would instantly go away. The user would then be responsible for limiting their usage, removing queries, or expanding available storage for the application.

There are many downsides to having a desktop application, like software maintenance, bugfixes and versioning. But mainly that all sorts of different operating systems suddenly needs to be catered to. Windows, OSX, Linux, iOS and Android are just the headlines.

Another way of storing data could have been to use Web Storage[11], but it is a cookie like key-value type storage, that does not suit the IIR storage. There is also a suggested storage limit per origin of 5 MB, which is too small. And there would also be privacy and security concerns with this kind of solution.

So the recommendation for now is to keep the solution as a web based solution, similar to how IIR was implemented.

9.4 Search Engine

All results in IIR were found during data collection through interfacing with a search engine API. This has drawbacks, as discussed in section 8.2. More work needs to be done in finding an appropriate way of gathering search results for IIR.

Microsoft's Bing Search API version 2 has been used in this thesis, and it finally closed down March 31st 2017. Their replacement service is Bing Search API version 5[12] and is a part of Microsoft Cognitive Services[13]. No in-depth evaluation of v5's suitability for IIR has been done, but at a glance it seems better, with more meta data like date returned with its results.

11. https://www.w3.org/TR/webstorage/
12. https://www.microsoft.com/cognitive-services/en-us/bing-web-search-api
13. https://www.microsoft.com/cognitive-services/en-us/apis

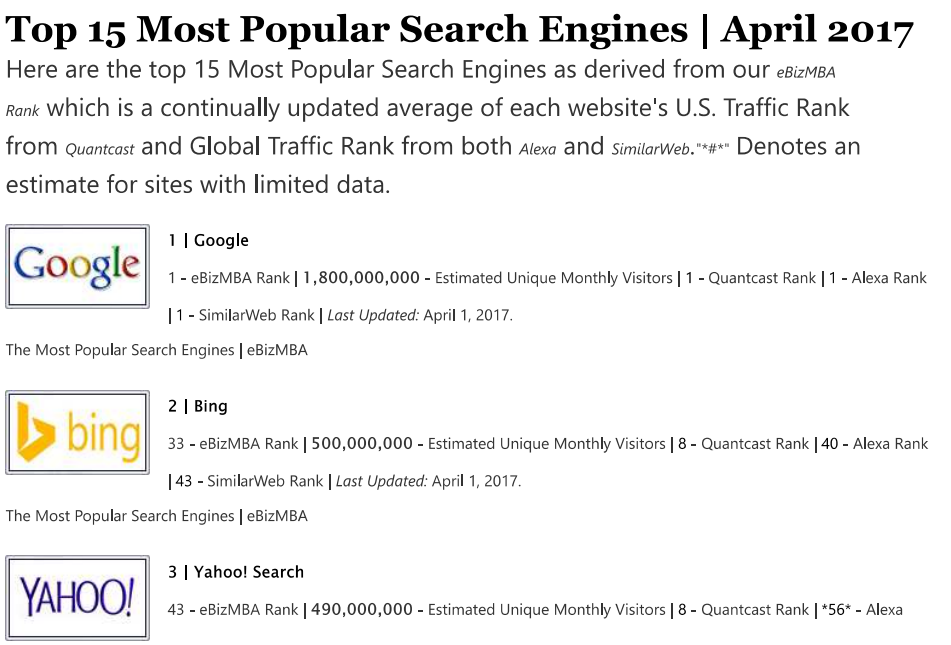


Figure 9.1: Top 3 search engines shown, out of 15 listed, courtesy of eBizMBA (see footnote).

Google would on the surface seem like a good candidate to use for interfacing with, since their search engine is widely regarded as the top on-line search engine[14], see figure 9.1. But their web search API was deprecated in 2010, and finally closed down in 2014[15]. The replacement service is called Google Custom Search, and is meant for implementing search on a limited number of specific websites[16].

Search engine "web scraping"[17] is not an option, since this violates the terms of service.

One avenue that has not been explored but could turn out interesting, is creating an IIR web crawler, to find results without using traditional search engines. One way such a solution could work is by letting the user enter their query, and then the web crawler could try to find information for the user, which could be presented *e.g.* the next day. The user could be notified when there were new results.

14. `http://www.ebizmba.com/articles/search-engines`
15. `https://developers.google.com/web-search/docs/`
16. `https://developers.google.com/custom-search/`
17. `http://wiki.c2.com/?WebScraping`

This is not an ideal way of searching, it would take much longer time to get answers, but there are a couple of benefits. Firstly, the user using IIR is in this for the long haul, immediacy is not necessarily expected or required. Also, a web crawler based solution could do the search for the user, without the user needing to be logged in to IIR. In this way it would superficially resemble a publish/subscribe[18] type solution.

There could be other possible avenues to explore, like combining different sources, using P2P solutions, DuckDuckGo, etcetera, but this will need to be researched separately.

18. http://wiki.c2.com/?PublishSubscribeModel

10

Conclusion

The research question of this thesis asked "*How can a long term search service be created to discover previously unseen search results, regularly concealed in traditional on-line search?*" An Incremental Information Retrieval (IIR) prototype was built to examine this scenario.

Experience shows that search engines give very similar results for the same search query. In fact, this is the whole basis for search today, to find precise answers to precise questions.

User testing demonstrated that the application had a good user satisfaction with total score of 4.35 out of 5. What was also evident, was that long-term type of search does not suit everyone. One tester gave a score of 2 on usefulness where other testers gave 4 and 5.

This study shows that search engines generally show the same or similar set of results every time a search is executed. The implementation of the IIR prototype system also shows that by suppressing already seen results users can find new and fresh results when running the same search query repeatedly.

It is important to note that this IIR system is not intended to replace ordinary on-line search. Still, this kind of search has its place alongside the usual search engines. It solves a problem that has not been solved today, and with growing content on the Internet, this may be important.

Bibliography

[1] Anh, V. N., and Moffat, A. Pruned query evaluation using pre-computed impacts. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2006), SIGIR '06, ACM, pp. 372–379.

[2] Baeza-Yates, R., and Ribeiro-Neto, B. *Modern Information Retrieval: The Concepts and Technology Behind Search*. Addison Wesley, 2011.

[3] Banker, K. *MongoDB in action: covers MongoDB version 3.0. Mongo DB in action; 2nd ed.* Manning Publ., Shelter Island, NY, 2016.

[4] Bar-Yossef, Z., Keidar, I., and Schonfeld, U. Do not crawl in the dust: Different urls with similar text. *ACM Trans. Web 3*, 1 (Jan. 2009), 3:1–3:31.

[5] Brin, S., Motwani, R., Page, L., and Winograd, T. What can you do with a web in your pocket? *IEEE Data Eng. Bull. 21*, 2 (1998), 37–47.

[6] Brin, S., and Page, L. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer networks 56*, 18 (2012), 3825–3833.

[7] Cambazoglu, B. B., and Aykanat, C. Performance of query processing implementations in ranking-based text retrieval systems using inverted indices. *Information Processing & Management 42*, 4 (2006), 875–898.

[8] Chakrabarti, S., Van den Berg, M., and Dom, B. Focused crawling: a new approach to topic-specific web resource discovery. *Computer networks 31*, 11 (1999), 1623–1640.

[9] Chang, W. Y., Abu-Amara, H., and Sanford, J. F. *Transforming enterprise cloud services*. Springer Science & Business Media, 2010.

[10] Chitika. The value of google result positioning, June 2013.
`http://chitika.com/google-positioning-value`
Since Chitika Insights' last report in May 2010 about the value of Google result positioning, the Google search algorithm has changed hundreds of

times. With these changes in mind, the Insights team sought to update our Google result valuation statistics. Accessed: 2017.03.11.

[11] Cho, J., and Garcia-Molina, H. The evolution of the web and implications for an incremental crawler. Tech. rep., Stanford, 1999.

[12] Clarke, C. L., and Cormack, G. V. Dynamic inverted indexes for a distributed full-text retrieval system. *Ws ec u R eport CS-95-o 10* (1995).

[13] Clarke, C. L., Cormack, G. V., and Tudhope, E. A. Relevance ranking for one to three term queries. *Information processing & management 36*, 2 (2000), 291–311.

[14] Cormack, G. *Information Retrieval–Implementing and Evaluating Search Engines*. MIT Press, Cambridge, 2010.

[15] eBizMBA Inc. Top 15 most popular search engines | april 2017, April 2017.
http://www.ebizmba.com/articles/search-engines
Here are the top 15 Most Popular Search Engines as derived from our eBizMBA Rank which is a continually updated average of each website's Alexa Global Traffic Rank, and U.S. Traffic Rank from both Compete and Quantcast.

[16] Edwards, J., McCurley, K., and Tomlin, J. An adaptive model for optimizing performance of an incremental web crawler. In *Proceedings of the 10th international conference on World Wide Web* (2001), ACM, pp. 106–113.

[17] Feathers, M. *Working Effectively with Legacy Code*. Robert C. Martin Series. Pearson Education, 2004.

[18] Fowler, M., and Beck, K. *Refactoring: Improving the Design of Existing Code*. Component software series. Addison-Wesley, 1999.

[19] Frakes, W. Stemming algorithms. In *Information retrieval* (1992), Prentice-Hall, Inc., pp. 131–160. http://orion.lcg.ufrj.br/Dr.Dobbs/books/book5/chap08.htm.

[20] Gantz, J., and Reinsel, D. Idc: The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east.

[21] Ghorab, M. R., Zhou, D., O'connor, A., and Wade, V. Personalised information retrieval: Survey and classification. *User Modeling and User-Adapted Interaction 23*, 4 (September 2013), 381–443.

[22] Hannak, A., Sapiezynski, P., Molavi Kakhki, A., Krishnamurthy, B., Lazer, D., Mislove, A., and Wilson, C. Measuring personalization of web search. In *Proceedings of the 22Nd International Conference on World Wide Web* (New York, NY, USA, 2013), WWW '13, ACM, pp. 527–538.

[23] Harry, D. How search engines rank web pages | search engine watch. https://searchenginewatch.com/sew/news/2064539/how-search-engines-rank-web-pages, 2013. Accessed: 14.05.2017.

[24] Hauff, C., Hiemstra, D., and de Jong, F. A survey of pre-retrieval query performance predictors. In *Proceedings of the 17th ACM conference on Information and knowledge management* (2008), ACM, pp. 1419–1420.

[25] Henzinger, M. Finding near-duplicate web pages: A large-scale evaluation of algorithms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2006), SIGIR '06, ACM, pp. 284–291.

[26] Internet-Live-Stats. Internet usage & social media statistics. http://www.internetlivestats.com/. Internet Live Stats is part of the Real Time Statistics Project (Worldometers and 7 Billion World). We are an international team of developers, researchers, and analysts with the goal of making statistics available in a dynamic and time relevant format to a wide audience around the world. Accessed: 11.05.2017.

[27] Karlsen, R., Morell, J. E. B., and Salcedo, V. T. Are trustworthy health videos reachable on youtube? *BIOSTEC 2017* (2017), 17.

[28] Kliman-Silver, C., Hannak, A., Lazer, D., Wilson, C., and Mislove, A. Location, location, location: The impact of geolocation on web search personalization. In *Proceedings of the 2015 Internet Measurement Conference* (New York, NY, USA, 2015), IMC '15, ACM, pp. 121–127.

[29] Kritzinger, W. T., and Weideman, M. Search engine optimization and pay-per-click marketing strategies. *Journal of Organizational Computing and Electronic Commerce 23*, 3 (2013), 273–286.

[30] Kumar, H., and Kang, S. Another face of search engine: Web search api's. In *Proceedings of the 21st International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems: New Frontiers in Applied Artificial Intelligence* (Berlin, Heidelberg, 2008), IEA/AIE '08, Springer-Verlag, pp. 311–320.

[31] Kumaran, G., and Carvalho, V. R. Reducing long queries using query quality predictors. In *Proceedings of the 32nd international ACM SIGIR*

conference on Research and development in information retrieval (2009), ACM, pp. 564–571.

[32] Lu, Y., Chau, M., and Chau, P. Y. K. Are sponsored links effective? investigating the impact of trust in search engine advertising. *ACM Trans. Manage. Inf. Syst. 7*, 4 (Jan. 2017), 12:1–12:33.

[33] Martin, R. *Agile Software Development: Principles, Patterns, and Practices*. Alan Apt series. Pearson Education, 2003.

[34] Mayr, P., and Tosques, F. Google web apis - an instrument for webometric analyses? *CoRR abs/cs/0601103* (2006).

[35] Munzert, S., Rubba, C., Meissner, P., and Nyhuis, D. *Automated data collection with R: A practical guide to web scraping and text mining*. John Wiley & Sons, 2014.

[36] Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[37] Pasi, G. Contextual search: Issues and challenges. In *Proceedings of the 7th Conference on Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society: Information Quality in e-Health* (Berlin, Heidelberg, 2011), USAB'11, Springer-Verlag, pp. 23–30.

[38] Pike, R. Go at google: Language design in the service of software engineering, October 2012.
`https://talks.golang.org/2012/splash.article`
This is a modified version of the keynote talk given by Rob Pike at the SPLASH 2012 conference in Tucson, Arizona, on October 25, 2012. Accessed: 2017.05.03.

[39] Pike, R. The go programming language faq, what is the purpose of the project?, October 2012.
`https://golang.org/doc/faq#What_is_the_purpose_of_the_project`
No major systems language has emerged in over a decade, but over that time the computing landscape has changed tremendously. [...] We believe it's worth trying again with a new language, a concurrent, garbage-collected language with fast compilation. Accessed: 2017.05.03.

[40] Read it Later, I. Pocket. `https://getpocket.com/`, 2007. The application allows the user to save an article or web page to remote servers for later reading. Accessed: 10.05.2017.

[41] Robert Griesemer, R. P., and Thompson, K. The go programming language, May 2017.
https://golang.org/doc
The Go programming language is an open source project to make programmers more productive. Accessed: 2017.05.03.

[42] Sullivan, D. How search engines rank web pages. *Search Engine Watch 31* (2003).

[43] White, S. Building a rest service with golang, December 2014.
Building a REST Service with Golang, in 3 parts.
https://stevenwhite.com/building-a-rest-service-with-golang-1
https://stevenwhite.com/building-a-rest-service-with-golang-2
https://stevenwhite.com/building-a-rest-service-with-golang-3
https://github.com/swhite24/go-rest-tutorial
Accessed: 2017.05.09.

[44] Williams, H. E. Query rewriting in search engines | hugh e. williams. https://hughewilliams.com/2012/03/19/query-rewriting-in-search-engines/, March 2012. Accessed:14.05./2017.

A

Data collection results

This appendix contains a summary of the results from the data collection phase, described in chapter 5.

The queries were run from 1st of November 2016 up until 10th of January 2017, 71 days in total. We keep the results from 18th of November up until 10th of January 2017, 54 days in total. See also figure 5.3 in chapter 5.

A.1 Queries and the reasoning behind them

As mentioned earlier, friends were recruited to contribute their searches, to vary the results found. All query "owners" are listed in the column P in the table below as person A - E.

ID[1]	ID[2]	P	Search text	Comments
1	2	A	Messerschmitt KR200 restoration	Interested in this topic.
3	4	A	Web search API thesis	Interesting to see if results could shed light on other theses around this kind of web search.
5	6	A	Web search thesis	Interesting to see if results could shed light on other theses around this kind of web search.
7	8	A	Search API thesis	Interesting to see if results could shed light on other theses around this kind of web search.
9	10	A	Messerschmitt TG500 for sale	Interested in this topic.
11	12	A	winds of winter	"The Winds of winter" is the title of George R.R. Martin's 6th book in the "A song of ice and fire" series. Not yet published, and has been "delayed for years".
13	14	A	promise of spring	George R.R. Martin's 7th book in the "A song of ice and fire" series is called "A Dream of Spring", and is not yet published. What would happen if I wanted to find this, but erroneously searched for "A Promise of spring"?
15	16	A	terry pratchett	Favourite author Terry Pratchett sadly died in spring 2015, see quotes at the start of this thesis.
17	18	B	liverpool leeds efl	Liverpool would play Leeds United in the Football League Cup Quarter-final in the middle of the test period (29/11 2016), and it would be interesting to see if this match would generate more results.
19	20	B	hillary clinton e-mail fbi	The US election would happen during the test period, so "Hillary Clinton" would generate more results.
21	22	B	macbook pro 2016 touch bar problems	Just before the start of the test period, Apple released their latest MacBook Pro with touch bar.
23	24	C	apple stock price	Interested in this topic.
25	26	C	samsung note 8 release date	Samsung Note 7 exploded in users' hands, and was removed from the market.
27	28	C	google self driving car	Interested in this topic.
29	30	D	mobile application health sensor data	Interested in this topic.
31	32	D	mobile phone body area network	Interested in this topic.
33	34	D	mobile phone sensor research health	Interested in this topic.
35	36	E	forest fairytales	Interested in this topic.
37	38	E	tudor politics	The Elizabethan era, and before that the reign of Henry VIII, are interesting topics.
39	40	E	jazz poetry	Performances with the combination of jazz and poetry is not that widespread.

Table A.1: Queries, query "owners" and a short reasoning behind the queries.

1. Odd-numbered IDs are "free" searches
2. Even-numbered IDs are exact searches

A.2 Full data collection results

This section contains tables of results from the data collection phase.

A.2.1 Full data collection results, totals

The table in figure A.3 needs some explanation. The numbers columns in table A.2 and table A.3 are divided into three types.

Type A Statistics for the full result.
Type B Statistics for new results.
Type C Statistics for new results, if using new results from day 2 - 54.

The reasoning behind type **C** is that day 1 in the data collection period always contains almost entirely new results, and this skews the average number of new results per day. So these statistics shows how many new results appear *after* day one.

Column	Type	Explanation
ID		ID, also called SimpleId, of the Query.
Query		Query text search for.
Exact		Is this query an exact query or not?
Results	A	Total number of results, collected for this query in the 54 day period.
Avg	A	Average number of results per day in the 54 day period.
Min	A	Minimum number of daily results in the period.
Max	A	Maximum number of daily results in the period.
New	B	Total number of new results in the period.
New %	B	Total number of new results in the period, percentage-wise.
Avg	B	Average number of new results per day in the 54 day period.
Min	B	Minimum number of new daily results in the period.
Max	B	Maximum number of new daily results in the period.
New d2	C	Total number of new results in the period from day 2.
New d2 %	C	Total number of new results in the period from day 2, percentage-wise
Avg d2	C	Average number of results per day in the 53 day period, from day 2.

Table A.2: Explanation for columns in table A.3

ID	Query	Exact	Results	Avg	Min	Max	New	New %	Avg	Min	Max	New d2	New d2 %	Avg d2
1	Messerschmitt KR200 restoration	—	40 194	566.1	612	846	1596	4.0	22.5	0	509	534	1.3	7.6
2	Messerschmitt KR200 restoration	Yes	20 451	288.0	285	478	42	0.2	0.6	0	18	9	0.0	0.1
3	Web search API thesis	—	50 919	717.2	646	1000	3111	6.1	43.8	0	770	1087	2.1	15.5
4	Web search API thesis	Yes	0	0.0	0	0	0	0.0	0.0	0	0	0	0.0	0.0
5	Web search thesis	—	52 143	734.4	809	998	3164	6.1	44.6	0	871	1090	2.1	15.6
6	Web search thesis	Yes	15 510	218.5	126	411	114	0.7	1.6	0	62	25	0.2	0.4
7	Search API thesis	—	45 903	646.5	620	941	2582	5.6	36.4	0	655	955	2.1	13.6
8	Search API thesis	Yes	0	0.0	0	0	0	0.0	0.0	0	0	0	0.0	0.0
9	Messerschmitt TG500 for sale	—	39 000	549.3	532	967	1097	2.8	15.5	0	419	344	0.9	4.9
10	Messerschmitt TG500 for sale	Yes	13 421	189.0	150	291	24	0.2	0.3	0	14	0	0.0	0.0
11	winds of winter	—	52 565	740.4	883	1000	2668	5.1	37.6	0	915	810	1.5	11.6
12	winds of winter	Yes	42 035	592.0	596	905	1717	4.1	24.2	0	481	635	1.5	9.1
13	promise of spring	—	52 698	742.2	863	1000	2685	5.1	37.8	0	931	906	1.7	12.9
14	promise of spring	Yes	44 616	628.4	455	912	2492	5.6	35.1	0	777	931	2.1	13.3
15	terry pratchett	—	45 437	640.0	388	995	2401	5.3	33.8	0	717	777	1.7	11.1
16	terry pratchett	Yes	41 235	580.8	586	963	1752	4.2	24.7	0	511	660	1.6	9.4
17	liverpool leeds efl	—	51 214	721.3	866	999	3614	7.1	50.9	0	850	1160	2.3	16.6
18	liverpool leeds efl	Yes	33 504	471.9	114	901	277	0.8	3.9	0	60	78	0.2	1.1
19	hillary clinton e-mail fbi	—	50 729	714.5	0	1000	2576	5.1	36.3	0	903	824	1.6	11.8
20	hillary clinton e-mail fbi	Yes	27 647	389.4	431	620	68	0.2	1.0	0	26	28	0.1	0.4
21	macbook pro 2016 touch bar problems	—	40 435	569.5	0	900	3099	7.7	43.6	0	617	1290	3.2	18.4
22	macbook pro 2016 touch bar problems	Yes	0	0.0	0	0	0	0.0	0.0	0	0	0	0.0	0.0
23	apple stock price	—	39 449	555.6	393	983	1736	4.4	24.5	0	586	524	1.3	7.5
24	apple stock price	Yes	36 366	512.2	264	1000	1098	3.0	15.5	0	194	482	1.3	6.9
25	samsung note 8 release date	—	47 934	675.1	0	995	2893	6.0	40.7	0	759	973	2.0	13.9
26	samsung note 8 release date	Yes	42 507	598.7	235	946	181	0.4	2.5	0	29	62	0.1	0.9
27	google self driving car	—	51 224	721.5	851	997	2540	5.0	35.8	0	882	752	1.5	10.7
28	google self driving car	Yes	40 914	576.3	614	906	1669	4.1	23.5	0	580	609	1.5	8.7
29	mobile application health sensor data	—	53 117	748.1	878	1000	2894	5.4	40.8	0	910	846	1.6	12.1
30	mobile application health sensor data	Yes	0	0.0	0	0	0	0.0	0.0	0	0	0	0.0	0.0
31	mobile phone body area network	—	52 870	744.6	882	1000	2206	4.2	31.1	0	876	510	1.0	7.3
32	mobile phone body area network	Yes	0	0.0	0	0	0	0.0	0.0	0	0	0	0.0	0.0
33	mobile phone sensor research health	—	52 996	746.4	825	1000	3019	5.7	42.5	0	932	823	1.6	11.8
34	mobile phone sensor research health	Yes	0	0.0	0	0	0	0.0	0.0	0	0	0	0.0	0.0
35	forest fairytales	—	52 958	745.9	877	1000	2542	4.8	35.8	0	930	832	1.6	11.9
36	forest fairytales	Yes	28 986	408.3	0	934	704	2.4	9.9	0	247	248	0.9	3.5
37	tudor politics	—	52 314	736.8	815	1000	2894	5.5	40.8	0	905	1044	2.0	14.9
38	tudor politics	Yes	39 589	557.6	526	966	1278	3.2	18.0	0	535	338	0.9	4.8
39	jazz poetry	—	51 367	723.5	0	1000	2640	5.1	37.2	0	940	802	1.6	11.5
40	jazz poetry	Yes	42 083	592.7	574	951	2102	5.0	29.6	0	663	791	1.9	11.3
	Sum		**1 444 330**				**65 475**					**21 779**		

Table A.3: Totals and averages for query 1 - 40, for the 54 day run period from day 18.

A.2.2 Full data collection results, details

Each and every table in this section contains *Date of run*, *Day in period* and two query results side by side. These are the free and the exact version of the same query.

For each query the columns *ID*, all results and *New* results are shown. All results have column name *Free* for free queries, and *Exact* for exact queries. Percentage of new results are also shown in the column *New %*.

If a query run terminated prematurely without any results, a "—" is shown, if a run ended normally but did not give results, a zero (0) is shown. Only the IDs of the queries are shown, see preceding tables A.1 and A.4 for description of the queries and error codes.

For each table is a sum of results and new results for the whole period. See chapter 5 for more details on data collection.

In the tables of results there are some codes next to some of the numbers, that needs explaining. Below is a short reference. These errors are also discussed in more detail in section 5.5.

Code	Meaning	Explanation
CT	Client Timeout	Message "Client.Timeout exceeded while awaiting headers". See chapter 5.5.4 for an explanation.
PE	Parse Error	HTML was returned instead of the expected JSON result. See chapter 5.5.6 for an explanation.
PI	Parameter Incorrect	Message "The parameter is incorrect". See chapter 5.5.5 for an explanation.
—	Run failed	Run terminated before any results could be stored for the query run. See chapter 5.5.2 for an explanation.

Table A.4: Error codes for the QueryRuns.

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	1	632	509	80.5	2	338	18	5.3
2016.11.19	2	1	681	118	17.3	2	CT 323	0	0.0
2016.11.20	3	1	690	12	1.7	2	340	0	0.0
2016.11.21	4	1	744	27	3.6	2	400	3	0.8
2016.11.22	5	1	763	33	4.3	2	400	0	0.0
2016.11.23	6	1	784	35	4.5	2	CT 380	0	0.0
2016.11.24	7	1	784	0	0.0	2	393	0	0.0
2016.11.25	8	1	755	4	0.5	2	CT 380	0	0.0
2016.11.26	9	1	772	31	4.0	2	CT 380	0	0.0
2016.11.27	10	1	722	17	2.4	2	404	1	0.2
2016.11.28	11	1	832	86	10.3	2	CT 285	0	0.0
2016.11.29	12	1	714	13	1.8	2	CT 380	0	0.0
2016.11.30	13	1	721	4	0.6	2	CT 361	0	0.0
2016.12.01	14	1	736	22	3.0	2	418	11	2.6
2016.12.02	15	1	770	50	6.5	2	478	0	0.0
2016.12.03	16	1	784	19	2.4	2	CT 417	1	0.2
2016.12.04	17	1	836	20	2.4	2	410	0	0.0
2016.12.05	18	1	823	27	3.3	2	CT 405	0	0.0
2016.12.06	19	1	826	31	3.8	2	CT 416	0	0.0
2016.12.07	20	1	846	22	2.6	2	CT 394	0	0.0
2016.12.08	21	1	844	15	1.8	2	356	0	0.0
2016.12.09	22	1	802	46	5.7	2	361	0	0.0
2016.12.10	23	1	771	6	0.8	2	360	0	0.0
2016.12.11	24	1	794	7	0.9	2	359	0	0.0
2016.12.12	25	1	836	24	2.9	2	378	1	0.3
2016.12.13	26	1	781	10	1.3	2	392	0	0.0
2016.12.14	27	1	804	56	7.0	2	388	0	0.0
2016.12.15	28	1	773	9	1.2	2	397	0	0.0
2016.12.16	29	1	769	3	0.4	2	400	0	0.0
2016.12.17	30	1	720	10	1.4	2	392	0	0.0
2016.12.18	31	1	700	1	0.1	2	386	0	0.0
2016.12.19	32	1	676	9	1.3	2	386	0	0.0
2016.12.20	33	1	703	10	1.4	2	389	0	0.0
2016.12.21	34	1	726	27	3.7	2	358	2	0.6
2016.12.22	35	1	739	10	1.4	2	358	0	0.0
2016.12.23	36	1	708	9	1.3	2	361	0	0.0
2016.12.24	37	1	719	4	0.6	2	371	2	0.5
2016.12.25	38	1	719	9	1.3	2	372	0	0.0
2016.12.26	39	1	720	7	1.0	2	360	0	0.0
2016.12.27	40	1	696	18	2.6	2	380	0	0.0
2016.12.28	41	1	719	38	5.3	2	380	0	0.0
2016.12.29	42	1	762	12	1.6	2	380	0	0.0
2016.12.30	43	1	759	0	0.0	2	380	0	0.0
2016.12.31	44	1	759	1	0.1	2	400	1	0.2
2017.01.01	45	1	698	36	5.2	2	364	2	0.5
2017.01.02	46	1	687	9	1.3	2	380	0	0.0
2017.01.03	47	1	680	7	1.0	2	380	0	0.0
2017.01.04	48	1	612	28	4.6	2	372	0	0.0
2017.01.05	49	1	734	25	3.4	2	368	0	0.0
2017.01.06	50	1	639	15	2.3	2	371	0	0.0
2017.01.07	51	1	693	19	2.7	2	365	0	0.0
2017.01.08	52	1	701	21	3.0	2	370	0	0.0
2017.01.09	53	1	733	15	2.0	2	357	0	0.0
2017.01.10	54	1	803	0	0.0	2	378	0	0.0
Sum		**1**	**40194**	**1596**		**2**	**20451**	**42**	

Table A.5: Full and new results, for queries 1 - 2, "Messerschmitt KR200 restoration".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	3	CT 908	770	84.8	4	CT 0	0	0.0
2016.11.19	2	3	964	216	22.4	4	0	0	0.0
2016.11.20	3	3	964	10	1.0	4	0	0	0.0
2016.11.21	4	3	965	70	7.3	4	0	0	0.0
2016.11.22	5	3	991	85	8.6	4	CT 0	0	0.0
2016.11.23	6	3	992	32	3.2	4	0	0	0.0
2016.11.24	7	3	991	4	0.4	4	CT 0	0	0.0
2016.11.25	8	3	994	57	5.7	4	0	0	0.0
2016.11.26	9	3	CT 944	23	2.4	4	0	0	0.0
2016.11.27	10	3	CT 895	7	0.8	4	0	0	0.0
2016.11.28	11	3	994	107	10.8	4	0	0	0.0
2016.11.29	12	3	995	22	2.2	4	0	0	0.0
2016.11.30	13	3	993	23	2.3	4	0	0	0.0
2016.12.01	14	3	CT 944	37	3.9	4	0	0	0.0
2016.12.02	15	3	994	75	7.5	4	0	0	0.0
2016.12.03	16	3	990	46	4.6	4	0	0	0.0
2016.12.04	17	3	995	28	2.8	4	0	0	0.0
2016.12.05	18	3	CT 942	114	12.1	4	0	0	0.0
2016.12.06	19	3	995	49	4.9	4	0	0	0.0
2016.12.07	20	3	994	38	3.8	4	0	0	0.0
2016.12.08	21	3	986	24	2.4	4	0	0	0.0
2016.12.09	22	3	993	109	11.0	4	0	0	0.0
2016.12.10	23	3	CT 930	47	5.1	4	0	0	0.0
2016.12.11	24	3	981	12	1.2	4	CT 0	0	0.0
2016.12.12	25	3	995	70	7.0	4	0	0	0.0
2016.12.13	26	3	980	154	15.7	4	0	0	0.0
2016.12.14	27	3	964	70	7.3	4	0	0	0.0
2016.12.15	28	3	998	53	5.3	4	CT 0	0	0.0
2016.12.16	29	3	998	3	0.3	4	0	0	0.0
2016.12.17	30	3	947	12	1.3	4	0	0	0.0
2016.12.18	31	3	997	8	0.8	4	CT 0	0	0.0
2016.12.19	32	3	985	30	3.0	4	0	0	0.0
2016.12.20	33	3	998	43	4.3	4	0	0	0.0
2016.12.21	34	3	985	101	10.3	4	0	0	0.0
2016.12.22	35	3	999	45	4.5	4	0	0	0.0
2016.12.23	36	3	998	23	2.3	4	0	0	0.0
2016.12.24	37	3	998	8	0.8	4	0	0	0.0
2016.12.25	38	3	998	10	1.0	4	0	0	0.0
2016.12.26	39	3	998	63	6.3	4	0	0	0.0
2016.12.27	40	3	995	8	0.8	4	0	0	0.0
2016.12.28	41	3	999	70	7.0	4	0	0	0.0
2016.12.29	42	3	CT 950	25	2.6	4	0	0	0.0
2016.12.30	43	3	1000	1	0.1	4	0	0	0.0
2016.12.31	44	3	998	4	0.4	4	0	0	0.0
2017.01.01	45	3	646	77	11.9	4	0	0	0.0
2017.01.02	46	3	683	44	6.4	4	0	0	0.0
2017.01.03	47	3	690	16	2.3	4	0	0	0.0
2017.01.04	48	3	830	44	5.3	4	0	0	0.0
2017.01.05	49	3	731	40	5.5	4	0	0	0.0
2017.01.06	50	3	821	15	1.8	4	0	0	0.0
2017.01.07	51	3	900	20	2.2	4	0	0	0.0
2017.01.08	52	3	810	26	3.2	4	0	0	0.0
2017.01.09	53	3	859	23	2.7	4	0	0	0.0
2017.01.10	54	3	835	0	0.0	4	0	0	0.0
Sum		**3**	**50919**	**3111**		**4**	**0**	**0**	

Table A.6: Full and new results, for queries 3 - 4, "Web search API thesis".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	5	963	871	90.4	6	277	62	22.4
2016.11.19	2	5	986	78	7.9	6	278	0	0.0
2016.11.20	3	5	CT 937	10	1.1	6	CT 284	1	0.4
2016.11.21	4	5	995	84	8.4	6	251	3	1.2
2016.11.22	5	5	CT 939	27	2.9	6	250	0	0.0
2016.11.23	6	5	984	40	4.1	6	259	0	0.0
2016.11.24	7	5	982	1	0.1	6	259	0	0.0
2016.11.25	8	5	994	56	5.6	6	259	0	0.0
2016.11.26	9	5	993	84	8.5	6	250	2	0.8
2016.11.27	10	5	992	13	1.3	6	259	2	0.8
2016.11.28	11	5	997	125	12.5	6	126	4	3.2
2016.11.29	12	5	997	42	4.2	6	221	0	0.0
2016.11.30	13	5	993	25	2.5	6	231	0	0.0
2016.12.01	14	5	994	34	3.4	6	238	0	0.0
2016.12.02	15	5	996	73	7.3	6	186	0	0.0
2016.12.03	16	5	993	53	5.3	6	155	2	1.3
2016.12.04	17	5	998	32	3.2	6	176	1	0.6
2016.12.05	18	5	995	67	6.7	6	187	3	1.6
2016.12.06	19	5	985	23	2.3	6	191	0	0.0
2016.12.07	20	5	992	31	3.1	6	205	2	1.0
2016.12.08	21	5	983	32	3.3	6	211	0	0.0
2016.12.09	22	5	979	23	2.3	6	226	1	0.4
2016.12.10	23	5	994	20	2.0	6	237	2	0.8
2016.12.11	24	5	992	15	1.5	6	239	1	0.4
2016.12.12	25	5	993	27	2.7	6	224	2	0.9
2016.12.13	26	5	894	15	1.7	6	299	6	2.0
2016.12.14	27	5	956	283	29.6	6	308	1	0.3
2016.12.15	28	5	937	136	14.5	6	372	1	0.3
2016.12.16	29	5	996	33	3.3	6	391	0	0.0
2016.12.17	30	5	974	12	1.2	6	387	0	0.0
2016.12.18	31	5	809	59	7.3	6	395	2	0.5
2016.12.19	32	5	947	15	1.6	6	CT 385	0	0.0
2016.12.20	33	5	974	26	2.7	6	386	3	0.8
2016.12.21	34	5	CT 920	81	8.8	6	324	1	0.3
2016.12.22	35	5	962	73	7.6	6	297	1	0.3
2016.12.23	36	5	968	33	3.4	6	286	1	0.3
2016.12.24	37	5	982	12	1.2	6	297	0	0.0
2016.12.25	38	5	987	63	6.4	6	384	3	0.8
2016.12.26	39	5	987	23	2.3	6	393	2	0.5
2016.12.27	40	5	988	19	1.9	6	411	0	0.0
2016.12.28	41	5	998	73	7.3	6	372	1	0.3
2016.12.29	42	5	996	50	5.0	6	373	0	0.0
2016.12.30	43	5	974	20	2.1	6	354	1	0.3
2016.12.31	44	5	979	17	1.7	6	354	0	0.0
2017.01.01	45	5	984	49	5.0	6	353	0	0.0
2017.01.02	46	5	996	16	1.6	6	CT 319	0	0.0
2017.01.03	47	5	984	9	0.9	6	335	1	0.3
2017.01.04	48	5	921	37	4.0	6	325	0	0.0
2017.01.05	49	5	880	23	2.6	6	310	1	0.3
2017.01.06	50	5	879	28	3.2	6	302	0	0.0
2017.01.07	51	5	899	23	2.6	6	300	1	0.3
2017.01.08	52	5	949	19	2.0	6	284	0	0.0
2017.01.09	53	5	850	31	3.6	6	287	0	0.0
2017.01.10	54	5	927	0	0.0	6	248	0	0.0
Sum		**5**	**52143**	**3164**		**6**	**15510**	**114**	

Table A.7: Full and new results, for queries 5 - 6, "Web search thesis".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	7	CT 838	655	78.2	8	0	0	0.0
2016.11.19	2	7	894	95	10.6	8	0	0	0.0
2016.11.20	3	7	870	62	7.1	8	0	0	0.0
2016.11.21	4	7	874	57	6.5	8	0	0	0.0
2016.11.22	5	7	884	20	2.3	8	CT 0	0	0.0
2016.11.23	6	7	896	3	0.3	8	0	0	0.0
2016.11.24	7	7	893	18	2.0	8	0	0	0.0
2016.11.25	8	7	908	50	5.5	8	0	0	0.0
2016.11.26	9	7	877	51	5.8	8	0	0	0.0
2016.11.27	10	7	880	0	0.0	8	0	0	0.0
2016.11.28	11	7	864	123	14.2	8	0	0	0.0
2016.11.29	12	7	894	30	3.4	8	0	0	0.0
2016.11.30	13	7	877	20	2.3	8	0	0	0.0
2016.12.01	14	7	844	34	4.0	8	0	0	0.0
2016.12.02	15	7	829	47	5.7	8	0	0	0.0
2016.12.03	16	7	823	31	3.8	8	0	0	0.0
2016.12.04	17	7	842	26	3.1	8	0	0	0.0
2016.12.05	18	7	831	78	9.4	8	0	0	0.0
2016.12.06	19	7	867	29	3.3	8	0	0	0.0
2016.12.07	20	7	833	42	5.0	8	0	0	0.0
2016.12.08	21	7	900	18	2.0	8	0	0	0.0
2016.12.09	22	7	874	35	4.0	8	0	0	0.0
2016.12.10	23	7	919	52	5.7	8	0	0	0.0
2016.12.11	24	7	893	19	2.1	8	0	0	0.0
2016.12.12	25	7	CT 847	92	10.9	8	0	0	0.0
2016.12.13	26	7	833	173	20.8	8	0	0	0.0
2016.12.14	27	7	865	38	4.4	8	0	0	0.0
2016.12.15	28	7	840	16	1.9	8	0	0	0.0
2016.12.16	29	7	890	5	0.6	8	0	0	0.0
2016.12.17	30	7	834	13	1.6	8	0	0	0.0
2016.12.18	31	7	887	7	0.8	8	0	0	0.0
2016.12.19	32	7	861	47	5.5	8	0	0	0.0
2016.12.20	33	7	CT 716	26	3.6	8	0	0	0.0
2016.12.21	34	7	823	88	10.7	8	0	0	0.0
2016.12.22	35	7	867	22	2.5	8	0	0	0.0
2016.12.23	36	7	862	13	1.5	8	0	0	0.0
2016.12.24	37	7	871	11	1.3	8	0	0	0.0
2016.12.25	38	7	871	5	0.6	8	0	0	0.0
2016.12.26	39	7	801	76	9.5	8	0	0	0.0
2016.12.27	40	7	802	6	0.7	8	0	0	0.0
2016.12.28	41	7	938	39	4.2	8	0	0	0.0
2016.12.29	42	7	941	3	0.3	8	0	0	0.0
2016.12.30	43	7	909	25	2.8	8	0	0	0.0
2016.12.31	44	7	863	65	7.5	8	0	0	0.0
2017.01.01	45	7	869	4	0.5	8	0	0	0.0
2017.01.02	46	7	876	36	4.1	8	0	0	0.0
2017.01.03	47	7	888	8	0.9	8	0	0	0.0
2017.01.04	48	7	CT 756	36	4.8	8	0	0	0.0
2017.01.05	49	7	791	32	4.0	8	0	0	0.0
2017.01.06	50	7	740	33	4.5	8	0	0	0.0
2017.01.07	51	7	686	19	2.8	8	0	0	0.0
2017.01.08	52	7	CT 620	21	3.4	8	0	0	0.0
2017.01.09	53	7	839	28	3.3	8	0	0	0.0
2017.01.10	54	7	813	0	0.0	8	0	0	0.0
Sum		7	**45903**	**2582**		**8**	**0**	**0**	

Table A.8: Full and new results, for queries 7 - 8, "Search API thesis".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	9	782	419	53.6	10	CT 261	14	5.4
2016.11.19	2	9	784	12	1.5	10	261	0	0.0
2016.11.20	3	9	795	29	3.6	10	260	0	0.0
2016.11.21	4	9	905	12	1.3	10	269	0	0.0
2016.11.22	5	9	CT 868	39	4.5	10	276	0	0.0
2016.11.23	6	9	901	3	0.3	10	280	0	0.0
2016.11.24	7	9	901	2	0.2	10	280	0	0.0
2016.11.25	8	9	731	19	2.6	10	280	0	0.0
2016.11.26	9	9	567	18	3.2	10	271	0	0.0
2016.11.27	10	9	566	0	0.0	10	280	0	0.0
2016.11.28	11	9	780	73	9.4	10	240	0	0.0
2016.11.29	12	9	532	5	0.9	10	280	0	0.0
2016.11.30	13	9	548	3	0.5	10	277	0	0.0
2016.12.01	14	9	797	11	1.4	10	272	1	0.4
2016.12.02	15	9	748	43	5.7	10	281	4	1.4
2016.12.03	16	9	672	6	0.9	10	282	0	0.0
2016.12.04	17	9	693	10	1.4	10	281	0	0.0
2016.12.05	18	9	673	27	4.0	10	289	0	0.0
2016.12.06	19	9	682	6	0.9	10	284	0	0.0
2016.12.07	20	9	686	9	1.3	10	291	0	0.0
2016.12.08	21	9	665	4	0.6	10	271	0	0.0
2016.12.09	22	9	697	17	2.4	10	269	0	0.0
2016.12.10	23	9	745	18	2.4	10	276	0	0.0
2016.12.11	24	9	747	3	0.4	10	278	0	0.0
2016.12.12	25	9	767	16	2.1	10	279	0	0.0
2016.12.13	26	9	752	3	0.4	10	272	0	0.0
2016.12.14	27	9	610	14	2.3	10	255	0	0.0
2016.12.15	28	9	603	16	2.7	10	265	0	0.0
2016.12.16	29	9	604	4	0.7	10	279	1	0.4
2016.12.17	30	9	589	8	1.4	10	264	0	0.0
2016.12.18	31	9	669	1	0.1	10	CT 150	0	0.0
2016.12.19	32	9	636	2	0.3	10	255	0	0.0
2016.12.20	33	9	619	1	0.2	10	251	0	0.0
2016.12.21	34	9	967	22	2.3	10	239	0	0.0
2016.12.22	35	9	767	6	0.8	10	241	0	0.0
2016.12.23	36	9	543	4	0.7	10	236	0	0.0
2016.12.24	37	9	635	3	0.5	10	240	0	0.0
2016.12.25	38	9	597	0	0.0	10	240	0	0.0
2016.12.26	39	9	630	57	9.0	10	219	0	0.0
2016.12.27	40	9	630	1	0.2	10	220	0	0.0
2016.12.28	41	9	665	7	1.1	10	220	0	0.0
2016.12.29	42	9	666	0	0.0	10	220	0	0.0
2016.12.30	43	9	951	26	2.7	10	195	0	0.0
2016.12.31	44	9	953	0	0.0	10	200	0	0.0
2017.01.01	45	9	950	3	0.3	10	203	1	0.5
2017.01.02	46	9	859	27	3.1	10	220	0	0.0
2017.01.03	47	9	859	0	0.0	10	198	0	0.0
2017.01.04	48	9	808	8	1.0	10	206	0	0.0
2017.01.05	49	9	668	31	4.6	10	211	2	0.9
2017.01.06	50	9	681	9	1.3	10	CT 210	1	0.5
2017.01.07	51	9	645	6	0.9	10	232	0	0.0
2017.01.08	52	9	737	26	3.5	10	216	0	0.0
2017.01.09	53	9	720	8	1.1	10	CT 195	0	0.0
2017.01.10	54	9	755	0	0.0	10	201	0	0.0
Sum		**9**	**39000**	**1097**		**10**	**13421**	**24**	

Table A.9: Full and new results, for queries 9 - 10, "Messerschmitt TG500 for sale".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	11	PE 949	915	96.4	12	PE 860	481	55.9
2016.11.19	2	11	999	85	8.5	12	831	65	7.8
2016.11.20	3	11	999	35	3.5	12	660	32	4.8
2016.11.21	4	11	999	59	5.9	12	633	36	5.7
2016.11.22	5	11	CT 948	62	6.5	12	CT 596	40	6.7
2016.11.23	6	11	998	4	0.4	12	658	9	1.4
2016.11.24	7	11	997	0	0.0	12	651	18	2.8
2016.11.25	8	11	999	52	5.2	12	624	14	2.2
2016.11.26	9	11	1000	43	4.3	12	842	31	3.7
2016.11.27	10	11	999	3	0.3	12	601	5	0.8
2016.11.28	11	11	999	55	5.5	12	740	115	15.5
2016.11.29	12	11	1000	36	3.6	12	871	34	3.9
2016.11.30	13	11	999	22	2.2	12	825	18	2.2
2016.12.01	14	11	999	51	5.1	12	872	31	3.6
2016.12.02	15	11	1000	74	7.4	12	780	45	5.8
2016.12.03	16	11	1000	29	2.9	12	776	29	3.7
2016.12.04	17	11	1000	39	3.9	12	864	28	3.2
2016.12.05	18	11	1000	55	5.5	12	756	19	2.5
2016.12.06	19	11	999	14	1.4	12	713	12	1.7
2016.12.07	20	11	983	22	2.2	12	653	27	4.1
2016.12.08	21	11	997	18	1.8	12	820	18	2.2
2016.12.09	22	11	980	11	1.1	12	773	33	4.3
2016.12.10	23	11	997	32	3.2	12	837	21	2.5
2016.12.11	24	11	996	29	2.9	12	823	13	1.6
2016.12.12	25	11	1000	55	5.5	12	786	20	2.5
2016.12.13	26	11	980	31	3.2	12	836	19	2.3
2016.12.14	27	11	946	110	11.6	12	864	19	2.2
2016.12.15	28	11	951	94	9.9	12	866	31	3.6
2016.12.16	29	11	981	17	1.7	12	847	7	0.8
2016.12.17	30	11	959	29	3.0	12	812	23	2.8
2016.12.18	31	11	945	17	1.8	12	806	6	0.7
2016.12.19	32	11	937	13	1.4	12	807	7	0.9
2016.12.20	33	11	950	49	5.2	12	777	23	3.0
2016.12.21	34	11	976	68	7.0	12	708	18	2.5
2016.12.22	35	11	951	19	2.0	12	712	17	2.4
2016.12.23	36	11	956	31	3.2	12	795	20	2.5
2016.12.24	37	11	981	2	0.2	12	733	12	1.6
2016.12.25	38	11	979	0	0.0	12	803	27	3.4
2016.12.26	39	11	985	82	8.3	12	817	28	3.4
2016.12.27	40	11	985	0	0.0	12	905	10	1.1
2016.12.28	41	11	988	37	3.7	12	873	25	2.9
2016.12.29	42	11	988	0	0.0	12	853	5	0.6
2016.12.30	43	11	988	53	5.4	12	726	23	3.2
2016.12.31	44	11	987	0	0.0	12	729	3	0.4
2017.01.01	45	11	985	12	1.2	12	825	25	3.0
2017.01.02	46	11	984	61	6.2	12	875	22	2.5
2017.01.03	47	11	984	3	0.3	12	876	1	0.1
2017.01.04	48	11	895	27	3.0	12	701	21	3.0
2017.01.05	49	11	944	30	3.2	12	758	12	1.6
2017.01.06	50	11	911	19	2.1	12	818	53	6.5
2017.01.07	51	11	894	19	2.1	12	830	20	2.4
2017.01.08	52	11	895	13	1.5	12	791	18	2.3
2017.01.09	53	11	941	32	3.4	12	736	28	3.8
2017.01.10	54	11	883	0	0.0	12	711	0	0.0
Sum		**11**	**52565**	**2668**		**12**	**42035**	**1717**	

Table A.10: Full and new results, for queries 11 - 12, "winds of winter".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	13	998	931	93.3	14	871	777	89.2
2016.11.19	2	13	967	54	5.6	14	867	30	3.5
2016.11.20	3	13	998	35	3.5	14	893	41	4.6
2016.11.21	4	13	998	33	3.3	14	887	32	3.6
2016.11.22	5	13	CT 947	50	5.3	14	901	35	3.9
2016.11.23	6	13	998	26	2.6	14	893	3	0.3
2016.11.24	7	13	997	18	1.8	14	899	26	2.9
2016.11.25	8	13	997	32	3.2	14	902	33	3.7
2016.11.26	9	13	1000	37	3.7	14	891	33	3.7
2016.11.27	10	13	1000	2	0.2	14	895	23	2.6
2016.11.28	11	13	999	213	21.3	14	901	195	21.6
2016.11.29	12	13	1000	31	3.1	14	887	55	6.2
2016.11.30	13	13	1000	27	2.7	14	883	24	2.7
2016.12.01	14	13	999	35	3.5	14	893	31	3.5
2016.12.02	15	13	999	60	6.0	14	897	44	4.9
2016.12.03	16	13	992	34	3.4	14	886	22	2.5
2016.12.04	17	13	997	35	3.5	14	904	25	2.8
2016.12.05	18	13	1000	24	2.4	14	910	21	2.3
2016.12.06	19	13	999	25	2.5	14	PE 855	5	0.6
2016.12.07	20	13	1000	13	1.3	14	891	16	1.8
2016.12.08	21	13	932	28	3.0	14	900	24	2.7
2016.12.09	22	13	995	34	3.4	14	889	24	2.7
2016.12.10	23	13	999	15	1.5	14	912	30	3.3
2016.12.11	24	13	977	12	1.2	14	896	9	1.0
2016.12.12	25	13	1000	41	4.1	14	905	39	4.3
2016.12.13	26	13	945	42	4.4	14	782	69	8.8
2016.12.14	27	13	992	53	5.3	14	817	62	7.6
2016.12.15	28	13	925	97	10.5	14	809	23	2.8
2016.12.16	29	13	998	17	1.7	14	799	14	1.8
2016.12.17	30	13	985	17	1.7	14	792	22	2.8
2016.12.18	31	13	971	23	2.4	14	455	45	9.9
2016.12.19	32	13	971	10	1.0	14	479	14	2.9
2016.12.20	33	13	959	37	3.9	14	809	36	4.4
2016.12.21	34	13	895	50	5.6	14	860	52	6.0
2016.12.22	35	13	958	26	2.7	14	858	25	2.9
2016.12.23	36	13	986	18	1.8	14	846	18	2.1
2016.12.24	37	13	1000	38	3.8	14	847	4	0.5
2016.12.25	38	13	1000	44	4.4	14	728	56	7.7
2016.12.26	39	13	1000	48	4.8	14	856	40	4.7
2016.12.27	40	13	999	4	0.4	14	844	15	1.8
2016.12.28	41	13	1000	13	1.3	14	659	82	12.4
2016.12.29	42	13	1000	3	0.3	14	650	0	0.0
2016.12.30	43	13	1000	44	4.4	14	704	51	7.2
2016.12.31	44	13	997	13	1.3	14	701	8	1.1
2017.01.01	45	13	996	4	0.4	14	700	26	3.7
2017.01.02	46	13	998	54	5.4	14	702	37	5.3
2017.01.03	47	13	998	4	0.4	14	726	11	1.5
2017.01.04	48	13	949	22	2.3	14	823	53	6.4
2017.01.05	49	13	953	43	4.5	14	906	42	4.6
2017.01.06	50	13	886	24	2.7	14	856	22	2.6
2017.01.07	51	13	863	22	2.5	14	745	18	2.4
2017.01.08	52	13	924	42	4.5	14	818	32	3.9
2017.01.09	53	13	868	28	3.2	14	856	18	2.1
2017.01.10	54	13	894	0	0.0	14	881	0	0.0
Sum		**13**	**52698**	**2685**		**14**	**44616**	**2492**	

Table A.11: Full and new results, for queries 13 - 14, "promise of spring".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	15	PE 944	717	76.0	16	PE 892	511	57.3
2016.11.19	2	15	995	63	6.3	16	963	20	2.1
2016.11.20	3	15	966	45	4.7	16	791	24	3.0
2016.11.21	4	15	953	34	3.6	16	653	30	4.6
2016.11.22	5	15	952	59	6.2	16	698	44	6.3
2016.11.23	6	15	951	1	0.1	16	609	12	2.0
2016.11.24	7	15	951	15	1.6	16	633	29	4.6
2016.11.25	8	15	940	51	5.4	16	619	28	4.5
2016.11.26	9	15	935	60	6.4	16	596	37	6.2
2016.11.27	10	15	935	2	0.2	16	615	27	4.4
2016.11.28	11	15	925	75	8.1	16	910	89	9.8
2016.11.29	12	15	955	25	2.6	16	667	31	4.6
2016.11.30	13	15	965	17	1.8	16	665	13	2.0
2016.12.01	14	15	943	33	3.5	16	668	20	3.0
2016.12.02	15	15	947	64	6.8	16	801	14	1.7
2016.12.03	16	15	946	39	4.1	16	889	9	1.0
2016.12.04	17	15	932	29	3.1	16	924	9	1.0
2016.12.05	18	15	974	39	4.0	16	897	34	3.8
2016.12.06	19	15	927	46	5.0	16	907	19	2.1
2016.12.07	20	15	953	26	2.7	16	898	16	1.8
2016.12.08	21	15	955	40	4.2	16	888	31	3.5
2016.12.09	22	15	953	40	4.2	16	893	10	1.1
2016.12.10	23	15	937	43	4.6	16	650	32	4.9
2016.12.11	24	15	941	30	3.2	16	586	18	3.1
2016.12.12	25	15	894	49	5.5	16	697	53	7.6
2016.12.13	26	15	885	18	2.0	16	658	21	3.2
2016.12.14	27	15	710	21	3.0	16	691	52	7.5
2016.12.15	28	15	947	88	9.3	16	672	14	2.1
2016.12.16	29	15	935	11	1.2	16	671	8	1.2
2016.12.17	30	15	840	33	3.9	16	659	13	2.0
2016.12.18	31	15	853	27	3.2	16	673	18	2.7
2016.12.19	32	15	853	11	1.3	16	670	15	2.2
2016.12.20	33	15	856	39	4.6	16	651	37	5.7
2016.12.21	34	15	787	39	5.0	16	648	43	6.6
2016.12.22	35	15	797	15	1.9	16	637	14	2.2
2016.12.23	36	15	538	36	6.7	16	657	25	3.8
2016.12.24	37	15	538	9	1.7	16	628	8	1.3
2016.12.25	38	15	550	9	1.6	16	874	31	3.5
2016.12.26	39	15	821	60	7.3	16	929	19	2.0
2016.12.27	40	15	820	0	0.0	16	944	1	0.1
2016.12.28	41	15	802	30	3.7	16	955	32	3.4
2016.12.29	42	15	798	0	0.0	16	896	10	1.1
2016.12.30	43	15	724	59	8.1	16	808	22	2.7
2016.12.31	44	15	388	23	5.9	16	834	19	2.3
2017.01.01	45	15	731	23	3.1	16	904	36	4.0
2017.01.02	46	15	756	40	5.3	16	644	20	3.1
2017.01.03	47	15	756	0	0.0	16	944	18	1.9
2017.01.04	48	15	734	28	3.8	16	812	17	2.1
2017.01.05	49	15	767	44	5.7	16	797	15	1.9
2017.01.06	50	15	768	36	4.7	16	817	29	3.5
2017.01.07	51	15	699	14	2.0	16	824	17	2.1
2017.01.08	52	15	728	26	3.6	16	745	22	3.0
2017.01.09	53	15	649	20	3.1	16	758	16	2.1
2017.01.10	54	15	738	0	0.0	16	826	0	0.0
Sum		**15**	**45437**	**2401**		**16**	**41235**	**1752**	

Table A.12: Full and new results, for queries 15 - 16, "terry pratchett".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	17	993	850	85.6	18	CT 274	60	21.9
2016.11.19	2	17	991	66	6.7	18	294	1	0.3
2016.11.20	3	17	999	41	4.1	18	278	0	0.0
2016.11.21	4	17	998	71	7.1	18	291	6	2.1
2016.11.22	5	17	999	61	6.1	18	297	0	0.0
2016.11.23	6	17	997	15	1.5	18	297	0	0.0
2016.11.24	7	17	997	4	0.4	18	297	2	0.7
2016.11.25	8	17	999	65	6.5	18	265	2	0.8
2016.11.26	9	17	997	71	7.1	18	183	4	2.2
2016.11.27	10	17	998	6	0.6	18	183	0	0.0
2016.11.28	11	17	993	143	14.4	18	240	6	2.5
2016.11.29	12	17	991	88	8.9	18	681	20	2.9
2016.11.30	13	17	CT 942	14	1.5	18	541	1	0.2
2016.12.01	14	17	992	46	4.6	18	494	11	2.2
2016.12.02	15	17	996	84	8.4	18	625	11	1.8
2016.12.03	16	17	999	20	2.0	18	625	3	0.5
2016.12.04	17	17	999	12	1.2	18	601	0	0.0
2016.12.05	18	17	931	150	16.1	18	880	38	4.3
2016.12.06	19	17	929	56	6.0	18	863	2	0.2
2016.12.07	20	17	970	63	6.5	18	822	4	0.5
2016.12.08	21	17	935	49	5.2	18	739	3	0.4
2016.12.09	22	17	949	228	24.0	18	641	19	3.0
2016.12.10	23	17	974	152	15.6	18	480	2	0.4
2016.12.11	24	17	953	44	4.6	18	481	0	0.0
2016.12.12	25	17	981	134	13.7	18	835	20	2.4
2016.12.13	26	17	929	47	5.1	18	820	2	0.2
2016.12.14	27	17	978	65	6.6	18	706	0	0.0
2016.12.15	28	17	968	56	5.8	18	686	0	0.0
2016.12.16	29	17	987	8	0.8	18	696	2	0.3
2016.12.17	30	17	970	18	1.9	18	690	1	0.1
2016.12.18	31	17	934	38	4.1	18	682	0	0.0
2016.12.19	32	17	911	20	2.2	18	680	0	0.0
2016.12.20	33	17	919	54	5.9	18	678	3	0.4
2016.12.21	34	17	887	117	13.2	18	688	5	0.7
2016.12.22	35	17	911	39	4.3	18	674	0	0.0
2016.12.23	36	17	903	26	2.9	18	701	1	0.1
2016.12.24	37	17	909	9	1.0	18	694	1	0.1
2016.12.25	38	17	928	59	6.4	18	639	4	0.6
2016.12.26	39	17	884	35	4.0	18	687	9	1.3
2016.12.27	40	17	884	0	0.0	18	693	3	0.4
2016.12.28	41	17	889	34	3.8	18	722	0	0.0
2016.12.29	42	17	889	1	0.1	18	734	0	0.0
2016.12.30	43	17	940	100	10.6	18	824	9	1.1
2016.12.31	44	17	938	21	2.2	18	829	1	0.1
2017.01.01	45	17	981	37	3.8	18	856	6	0.7
2017.01.02	46	17	952	96	10.1	18	114	3	2.6
2017.01.03	47	17	952	0	0.0	18	901	0	0.0
2017.01.04	48	17	880	33	3.8	18	784	2	0.3
2017.01.05	49	17	900	37	4.1	18	788	6	0.8
2017.01.06	50	17	891	36	4.0	18	884	2	0.2
2017.01.07	51	17	875	41	4.7	18	861	0	0.0
2017.01.08	52	17	866	29	3.3	18	866	2	0.2
2017.01.09	53	17	935	25	2.7	18	848	0	0.0
2017.01.10	54	17	922	0	0.0	18	872	0	0.0
Sum		**17**	**51214**	**3614**		**18**	**33504**	**277**	

Table A.13: Full and new results, for queries 17 - 18, "liverpool leeds efl".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	19	977	903	92.4	20	506	26	5.1
2016.11.19	2	19	976	28	2.9	20	520	0	0.0
2016.11.20	3	19	980	60	6.1	20	520	0	0.0
2016.11.21	4	19	990	77	7.8	20	520	3	0.6
2016.11.22	5	19	991	87	8.8	20	519	0	0.0
2016.11.23	6	19	991	11	1.1	20	520	0	0.0
2016.11.24	7	19	991	5	0.5	20	520	0	0.0
2016.11.25	8	19	1000	67	6.7	20	524	1	0.2
2016.11.26	9	19	1000	63	6.3	20	519	1	0.2
2016.11.27	10	19	1000	1	0.1	20	520	0	0.0
2016.11.28	11	19	968	108	11.2	20	440	1	0.2
2016.11.29	12	19	996	27	2.7	20	520	0	0.0
2016.11.30	13	19	996	9	0.9	20	498	0	0.0
2016.12.01	14	19	—	—	0.0	20	483	1	0.2
2016.12.02	15	19	997	100	10.0	20	462	3	0.6
2016.12.03	16	19	994	68	6.8	20	466	1	0.2
2016.12.04	17	19	996	28	2.8	20	474	0	0.0
2016.12.05	18	19	992	77	7.8	20	514	2	0.4
2016.12.06	19	19	988	33	3.3	20	528	1	0.2
2016.12.07	20	19	977	31	3.2	20	534	1	0.2
2016.12.08	21	19	960	16	1.7	20	539	0	0.0
2016.12.09	22	19	955	34	3.6	20	516	0	0.0
2016.12.10	23	19	983	24	2.4	20	500	0	0.0
2016.12.11	24	19	982	18	1.8	20	497	0	0.0
2016.12.12	25	19	973	42	4.3	20	468	0	0.0
2016.12.13	26	19	936	11	1.2	20	464	0	0.0
2016.12.14	27	19	960	83	8.6	20	472	0	0.0
2016.12.15	28	19	927	42	4.5	20	456	0	0.0
2016.12.16	29	19	972	12	1.2	20	471	0	0.0
2016.12.17	30	19	951	26	2.7	20	456	0	0.0
2016.12.18	31	19	973	16	1.6	20	456	1	0.2
2016.12.19	32	19	975	4	0.4	20	459	1	0.2
2016.12.20	33	19	935	37	4.0	20	474	3	0.6
2016.12.21	34	19	928	58	6.2	20	571	12	2.1
2016.12.22	35	19	923	11	1.2	20	571	0	0.0
2016.12.23	36	19	959	27	2.8	20	569	0	0.0
2016.12.24	37	19	955	3	0.3	20	560	0	0.0
2016.12.25	38	19	934	26	2.8	20	541	1	0.2
2016.12.26	39	19	960	26	2.7	20	601	3	0.5
2016.12.27	40	19	960	0	0.0	20	600	0	0.0
2016.12.28	41	19	961	24	2.5	20	610	1	0.2
2016.12.29	42	19	963	14	1.5	20	620	0	0.0
2016.12.30	43	19	949	44	4.6	20	580	1	0.2
2016.12.31	44	19	950	7	0.7	20	580	0	0.0
2017.01.01	45	19	940	0	0.0	20	575	0	0.0
2017.01.02	46	19	944	41	4.3	20	560	0	0.0
2017.01.03	47	19	944	0	0.0	20	560	0	0.0
2017.01.04	48	19	860	25	2.9	20	487	2	0.4
2017.01.05	49	19	895	20	2.2	20	469	2	0.4
2017.01.06	50	19	941	24	2.6	20	480	0	0.0
2017.01.07	51	19	816	18	2.2	20	474	0	0.0
2017.01.08	52	19	893	28	3.1	20	441	0	0.0
2017.01.09	53	19	889	32	3.6	20	432	0	0.0
2017.01.10	54	19	883	0	0.0	20	431	0	0.0
Sum		**19**	**50729**	**2576**		**20**	**27647**	**68**	

Table A.14: Full and new results, for queries 19 - 20, "hillary clinton e-mail fbi".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	21	700	617	88.1	22	0	0	0.0
2016.11.19	2	21	706	9	1.3	22	0	0	0.0
2016.11.20	3	21	698	16	2.3	22	0	0	0.0
2016.11.21	4	21	672	78	11.6	22	0	0	0.0
2016.11.22	5	21	608	92	15.1	22	0	0	0.0
2016.11.23	6	21	612	14	2.3	22	0	0	0.0
2016.11.24	7	21	612	3	0.5	22	0	0	0.0
2016.11.25	8	21	681	73	10.7	22	0	0	0.0
2016.11.26	9	21	897	100	11.1	22	0	0	0.0
2016.11.27	10	21	900	9	1.0	22	0	0	0.0
2016.11.28	11	21	655	145	22.1	22	0	0	0.0
2016.11.29	12	21	622	23	3.7	22	0	0	0.0
2016.11.30	13	21	618	7	1.1	22	0	0	0.0
2016.12.01	14	21	649	14	2.2	22	0	0	0.0
2016.12.02	15	21	723	109	15.1	22	0	0	0.0
2016.12.03	16	21	683	28	4.1	22	0	0	0.0
2016.12.04	17	21	657	13	2.0	22	0	0	0.0
2016.12.05	18	21	664	89	13.4	22	0	0	0.0
2016.12.06	19	21	765	43	5.6	22	0	0	0.0
2016.12.07	20	21	674	23	3.4	22	0	0	0.0
2016.12.08	21	21	735	28	3.8	22	0	0	0.0
2016.12.09	22	21	764	90	11.8	22	0	0	0.0
2016.12.10	23	21	744	38	5.1	22	0	0	0.0
2016.12.11	24	21	724	17	2.3	22	0	0	0.0
2016.12.12	25	21	781	156	20.0	22	0	0	0.0
2016.12.13	26	21	726	19	2.6	22	0	0	0.0
2016.12.14	27	21	859	37	4.3	22	0	0	0.0
2016.12.15	28	21	809	36	4.4	22	0	0	0.0
2016.12.16	29	21	712	3	0.4	22	0	0	0.0
2016.12.17	30	21	827	15	1.8	22	0	0	0.0
2016.12.18	31	21	870	3	0.3	22	0	0	0.0
2016.12.19	32	21	869	20	2.3	22	0	0	0.0
2016.12.20	33	21	806	52	6.5	22	0	0	0.0
2016.12.21	34	21	753	176	23.4	22	0	0	0.0
2016.12.22	35	21	752	15	2.0	22	0	0	0.0
2016.12.23	36	21	809	38	4.7	22	0	0	0.0
2016.12.24	37	21	822	1	0.1	22	0	0	0.0
2016.12.25	38	21	822	3	0.4	22	0	0	0.0
2016.12.26	39	21	845	253	29.9	22	0	0	0.0
2016.12.27	40	21	846	0	0.0	22	0	0	0.0
2016.12.28	41	21	867	25	2.9	22	0	0	0.0
2016.12.29	42	21	867	0	0.0	22	0	0	0.0
2016.12.30	43	21	848	134	15.8	22	0	0	0.0
2016.12.31	44	21	848	4	0.5	22	0	0	0.0
2017.01.01	45	21	846	11	1.3	22	0	0	0.0
2017.01.02	46	21	817	148	18.1	22	0	0	0.0
2017.01.03	47	21	817	2	0.2	22	0	0	0.0
2017.01.04	48	21	782	27	3.5	22	0	0	0.0
2017.01.05	49	21	734	62	8.4	22	0	0	0.0
2017.01.06	50	21	854	80	9.4	22	0	0	0.0
2017.01.07	51	21	880	25	2.8	22	0	0	0.0
2017.01.08	52	21	790	76	9.6	22	0	0	0.0
2016.01.09	53	21	—	—	0.0	22	0	0	0.0
2017.01.10	54	21	814	0	0.0	22	0	0	0.0
Sum		**21**	**40435**	**3099**		**22**	**0**	**0**	

Table A.15: Full and new results, for queries 21 - 22, "macbook pro 2016 touch bar problems".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	23	841	586	69.7	24	298	194	65.1
2016.11.19	2	23	959	26	2.7	24	423	18	4.3
2016.11.20	3	23	938	42	4.5	24	399	16	4.0
2016.11.21	4	23	964	39	4.0	24	387	17	4.4
2016.11.22	5	23	800	22	2.8	24	471	16	3.4
2016.11.23	6	23	735	11	1.5	24	508	10	2.0
2016.11.24	7	23	927	22	2.4	24	264	43	16.3
2016.11.25	8	23	893	20	2.2	24	512	10	2.0
2016.11.26	9	23	771	44	5.7	24	615	19	3.1
2016.11.27	10	23	775	3	0.4	24	652	19	2.9
2016.11.28	11	23	656	26	4.0	24	PE 904	46	5.1
2016.11.29	12	23	589	7	1.2	24	652	30	4.6
2016.11.30	13	23	842	14	1.7	24	740	22	3.0
2016.12.01	14	23	822	23	2.8	24	752	39	5.2
2016.12.02	15	23	657	15	2.3	24	662	14	2.1
2016.12.03	16	23	695	55	7.9	24	679	18	2.7
2016.12.04	17	23	702	32	4.6	24	726	16	2.2
2016.12.05	18	23	733	19	2.6	24	816	16	2.0
2016.12.06	19	23	715	25	3.5	24	842	6	0.7
2016.12.07	20	23	717	20	2.8	24	850	20	2.4
2016.12.08	21	23	686	26	3.8	24	695	20	2.9
2016.12.09	22	23	662	35	5.3	24	673	14	2.1
2016.12.10	23	23	966	25	2.6	24	614	18	2.9
2016.12.11	24	23	625	10	1.6	24	599	11	1.8
2016.12.12	25	23	706	12	1.7	24	601	12	2.0
2016.12.13	26	23	902	19	2.1	24	539	17	3.2
2016.12.14	27	23	425	27	6.4	24	608	20	3.3
2016.12.15	28	23	672	50	7.4	24	740	31	4.2
2016.12.16	29	23	968	14	1.4	24	728	13	1.8
2016.12.17	30	23	677	20	3.0	24	807	14	1.7
2016.12.18	31	23	644	11	1.7	24	836	16	1.9
2016.12.19	32	23	651	19	2.9	24	956	15	1.6
2016.12.20	33	23	680	17	2.5	24	823	14	1.7
2016.12.21	34	23	393	20	5.1	24	900	11	1.2
2016.12.22	35	23	689	35	5.1	24	690	19	2.8
2016.12.23	36	23	696	21	3.0	24	389	14	3.6
2016.12.24	37	23	734	20	2.7	24	974	22	2.3
2016.12.25	38	23	745	4	0.5	24	305	26	8.5
2016.12.26	39	23	682	35	5.1	24	881	23	2.6
2016.12.27	40	23	682	0	0.0	24	708	3	0.4
2016.12.28	41	23	692	24	3.5	24	859	20	2.3
2016.12.29	42	23	670	20	3.0	24	807	11	1.4
2016.12.30	43	23	983	21	2.1	24	973	16	1.6
2016.12.31	44	23	978	12	1.2	24	1000	22	2.2
2017.01.01	45	23	965	8	0.8	24	767	16	2.1
2017.01.02	46	23	402	23	5.7	24	933	17	1.8
2017.01.03	47	23	419	1	0.2	24	863	10	1.2
2017.01.04	48	23	649	37	5.7	24	832	18	2.2
2017.01.05	49	23	637	28	4.4	24	722	8	1.1
2017.01.06	50	23	655	29	4.4	24	699	18	2.6
2017.01.07	51	23	649	19	2.9	24	398	7	1.8
2017.01.08	52	23	720	27	3.8	24	407	6	1.5
2017.01.09	53	23	699	16	2.3	24	441	7	1.6
2017.01.10	54	23	715	0	0.0	24	447	0	0.0
Sum		**23**	**39449**	**1736**		**24**	**36366**	**1098**	

Table A.16: Full and new results, for queries 23 - 24, "apple stock price".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	25	905	759	83.9	26	578	29	5.0
2016.11.19	2	25	947	46	4.9	26	717	10	1.4
2016.11.20	3	25	942	54	5.7	26	720	0	0.0
2016.11.21	4	25	956	63	6.6	26	718	3	0.4
2016.11.22	5	25	966	61	6.3	26	737	3	0.4
2016.11.23	6	25	963	13	1.3	26	742	1	0.1
2016.11.24	7	25	963	4	0.4	26	761	2	0.3
2016.11.25	8	25	940	58	6.2	26	777	7	0.9
2016.11.26	9	25	909	75	8.3	26	816	3	0.4
2016.11.27	10	25	898	10	1.1	26	780	0	0.0
2016.11.28	11	25	877	92	10.5	26	580	4	0.7
2016.11.29	12	25	—	—	0.0	26	777	1	0.1
2016.11.30	13	25	935	49	5.2	26	740	2	0.3
2016.12.01	14	25	892	41	4.6	26	812	12	1.5
2016.12.02	15	25	933	86	9.2	26	779	5	0.6
2016.12.03	16	25	947	45	4.8	26	764	0	0.0
2016.12.04	17	25	915	30	3.3	26	785	0	0.0
2016.12.05	18	25	896	107	11.9	26	778	1	0.1
2016.12.06	19	25	905	46	5.1	26	789	3	0.4
2016.12.07	20	25	897	33	3.7	26	819	3	0.4
2016.12.08	21	25	888	15	1.7	26	PI 789	3	0.4
2016.12.09	22	25	907	42	4.6	26	871	3	0.3
2016.12.10	23	25	914	31	3.4	26	919	2	0.2
2016.12.11	24	25	933	21	2.3	26	946	2	0.2
2016.12.12	25	25	901	61	6.8	26	491	6	1.2
2016.12.13	26	25	797	14	1.8	26	604	0	0.0
2016.12.14	27	25	853	120	14.1	26	918	1	0.1
2016.12.15	28	25	927	36	3.9	26	935	5	0.5
2016.12.16	29	25	927	23	2.5	26	905	1	0.1
2016.12.17	30	25	903	19	2.1	26	910	0	0.0
2016.12.18	31	25	902	8	0.9	26	920	2	0.2
2016.12.19	32	25	926	15	1.6	26	944	1	0.1
2016.12.20	33	25	925	52	5.6	26	837	2	0.2
2016.12.21	34	25	850	52	6.1	26	861	1	0.1
2016.12.22	35	25	847	21	2.5	26	718	5	0.7
2016.12.23	36	25	872	17	1.9	26	829	0	0.0
2016.12.24	37	25	888	6	0.7	26	854	1	0.1
2016.12.25	38	25	885	46	5.2	26	876	2	0.2
2016.12.26	39	25	900	67	7.4	26	875	5	0.6
2016.12.27	40	25	900	0	0.0	26	900	3	0.3
2016.12.28	41	25	937	39	4.2	26	902	2	0.2
2016.12.29	42	25	939	5	0.5	26	894	0	0.0
2016.12.30	43	25	995	118	11.9	26	936	4	0.4
2016.12.31	44	25	994	10	1.0	26	940	0	0.0
2017.01.01	45	25	992	12	1.2	26	710	3	0.4
2017.01.02	46	25	877	89	10.1	26	235	2	0.9
2017.01.03	47	25	878	4	0.5	26	929	1	0.1
2017.01.04	48	25	910	62	6.8	26	635	2	0.3
2017.01.05	49	25	887	74	8.3	26	722	5	0.7
2017.01.06	50	25	872	48	5.5	26	671	2	0.3
2017.01.07	51	25	849	42	4.9	26	724	1	0.1
2017.01.08	52	25	788	17	2.2	26	637	21	3.3
2017.01.09	53	25	769	35	4.6	26	881	4	0.5
2017.01.10	54	25	816	0	0.0	26	820	0	0.0
Sum		**25**	**47934**	**2893**		**26**	**42507**	**181**	

Table A.17: Full and new results, for queries 25 - 26, "samsung note 8 release date".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	27	956	882	92.3	28	716	580	81.0
2016.11.19	2	27	947	50	5.3	28	692	25	3.6
2016.11.20	3	27	958	50	5.2	28	713	8	1.1
2016.11.21	4	27	950	41	4.3	28	742	32	4.3
2016.11.22	5	27	940	57	6.1	28	742	53	7.1
2016.11.23	6	27	935	13	1.4	28	741	6	0.8
2016.11.24	7	27	941	3	0.3	28	736	12	1.6
2016.11.25	8	27	926	36	3.9	28	744	21	2.8
2016.11.26	9	27	934	40	4.3	28	776	31	4.0
2016.11.27	10	27	932	1	0.1	28	774	8	1.0
2016.11.28	11	27	966	64	6.6	28	808	111	13.7
2016.11.29	12	27	995	34	3.4	28	776	15	1.9
2016.11.30	13	27	952	30	3.2	28	783	13	1.7
2016.12.01	14	27	986	34	3.4	28	777	24	3.1
2016.12.02	15	27	989	91	9.2	28	756	28	3.7
2016.12.03	16	27	981	34	3.5	28	746	25	3.4
2016.12.04	17	27	978	41	4.2	28	725	19	2.6
2016.12.05	18	27	946	44	4.7	28	752	28	3.7
2016.12.06	19	27	950	26	2.7	28	747	14	1.9
2016.12.07	20	27	945	23	2.4	28	735	15	2.0
2016.12.08	21	27	975	22	2.3	28	732	8	1.1
2016.12.09	22	27	922	32	3.5	28	733	10	1.4
2016.12.10	23	27	947	21	2.2	28	684	16	2.3
2016.12.11	24	27	954	27	2.8	28	735	7	1.0
2016.12.12	25	27	985	33	3.4	28	759	11	1.4
2016.12.13	26	27	947	37	3.9	28	751	15	2.0
2016.12.14	27	27	900	32	3.6	28	726	54	7.4
2016.12.15	28	27	894	119	13.3	28	700	33	4.7
2016.12.16	29	27	938	12	1.3	28	702	2	0.3
2016.12.17	30	27	917	11	1.2	28	682	3	0.4
2016.12.18	31	27	928	1	0.1	28	686	0	0.0
2016.12.19	32	27	928	5	0.5	28	680	2	0.3
2016.12.20	33	27	907	32	3.5	28	677	6	0.9
2016.12.21	34	27	966	60	6.2	28	661	50	7.6
2016.12.22	35	27	916	23	2.5	28	637	7	1.1
2016.12.23	36	27	941	12	1.3	28	649	12	1.8
2016.12.24	37	27	967	11	1.1	28	648	2	0.3
2016.12.25	38	27	974	53	5.4	28	672	25	3.7
2016.12.26	39	27	983	40	4.1	28	614	35	5.7
2016.12.27	40	27	982	1	0.1	28	775	4	0.5
2016.12.28	41	27	979	14	1.4	28	895	59	6.6
2016.12.29	42	27	980	0	0.0	28	899	16	1.8
2016.12.30	43	27	996	77	7.7	28	847	48	5.7
2016.12.31	44	27	996	5	0.5	28	833	8	1.0
2017.01.01	45	27	997	10	1.0	28	833	15	1.8
2017.01.02	46	27	984	41	4.2	28	835	19	2.3
2017.01.03	47	27	982	5	0.5	28	832	9	1.1
2017.01.04	48	27	959	33	3.4	28	852	21	2.5
2017.01.05	49	27	877	38	4.3	28	906	28	3.1
2017.01.06	50	27	903	41	4.5	28	898	19	2.1
2017.01.07	51	27	924	24	2.6	28	858	12	1.4
2017.01.08	52	27	851	31	3.6	28	856	24	2.8
2017.01.09	53	27	912	43	4.7	28	855	21	2.5
2017.01.10	54	27	906	0	0.0	28	831	0	0.0
Sum		**27**	**51224**	**2540**		**28**	**40914**	**1669**	

Table A.18: Full and new results, for queries 27 - 28, "google self driving car".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	29	1000	910	91.0	30	0	0	0.0
2016.11.19	2	29	1000	78	7.8	30	0	0	0.0
2016.11.20	3	29	1000	66	6.6	30	0	0	0.0
2016.11.21	4	29	1000	64	6.4	30	0	0	0.0
2016.11.22	5	29	1000	50	5.0	30	0	0	0.0
2016.11.23	6	29	1000	15	1.5	30	0	0	0.0
2016.11.24	7	29	1000	5	0.5	30	0	0	0.0
2016.11.25	8	29	1000	37	3.7	30	0	0	0.0
2016.11.26	9	29	1000	80	8.0	30	0	0	0.0
2016.11.27	10	29	1000	3	0.3	30	0	0	0.0
2016.11.28	11	29	1000	61	6.1	30	0	0	0.0
2016.11.29	12	29	1000	25	2.5	30	0	0	0.0
2016.11.30	13	29	1000	23	2.3	30	0	0	0.0
2016.12.01	14	29	1000	42	4.2	30	0	0	0.0
2016.12.02	15	29	1000	113	11.3	30	0	0	0.0
2016.12.03	16	29	1000	57	5.7	30	0	0	0.0
2016.12.04	17	29	1000	48	4.8	30	0	0	0.0
2016.12.05	18	29	1000	73	7.3	30	0	0	0.0
2016.12.06	19	29	1000	43	4.3	30	0	0	0.0
2016.12.07	20	29	989	14	1.4	30	0	0	0.0
2016.12.08	21	29	970	48	4.9	30	0	0	0.0
2016.12.09	22	29	962	29	3.0	30	0	0	0.0
2016.12.10	23	29	1000	34	3.4	30	0	0	0.0
2016.12.11	24	29	998	19	1.9	30	0	0	0.0
2016.12.12	25	29	965	63	6.5	30	0	0	0.0
2016.12.13	26	29	932	16	1.7	30	0	0	0.0
2016.12.14	27	29	982	127	12.9	30	0	0	0.0
2016.12.15	28	29	966	75	7.8	30	0	0	0.0
2016.12.16	29	29	1000	12	1.2	30	0	0	0.0
2016.12.17	30	29	987	21	2.1	30	0	0	0.0
2016.12.18	31	29	987	3	0.3	30	0	0	0.0
2016.12.19	32	29	976	15	1.5	30	0	0	0.0
2016.12.20	33	29	1000	18	1.8	30	0	0	0.0
2016.12.21	34	29	988	30	3.0	30	0	0	0.0
2016.12.22	35	29	962	25	2.6	30	0	0	0.0
2016.12.23	36	29	952	16	1.7	30	0	0	0.0
2016.12.24	37	29	999	2	0.2	30	0	0	0.0
2016.12.25	38	29	999	7	0.7	30	0	0	0.0
2016.12.26	39	29	1000	110	11.0	30	0	0	0.0
2016.12.27	40	29	1000	9	0.9	30	0	0	0.0
2016.12.28	41	29	1000	20	2.0	30	0	0	0.0
2016.12.29	42	29	1000	23	2.3	30	0	0	0.0
2016.12.30	43	29	1000	51	5.1	30	0	0	0.0
2016.12.31	44	29	1000	13	1.3	30	0	0	0.0
2017.01.01	45	29	1000	4	0.4	30	0	0	0.0
2017.01.02	46	29	1000	62	6.2	30	0	0	0.0
2017.01.03	47	29	1000	12	1.2	30	0	0	0.0
2017.01.04	48	29	925	60	6.5	30	0	0	0.0
2017.01.05	49	29	926	30	3.2	30	0	0	0.0
2017.01.06	50	29	928	42	4.5	30	0	0	0.0
2017.01.07	51	29	878	22	2.5	30	0	0	0.0
2017.01.08	52	29	966	35	3.6	30	0	0	0.0
2017.01.09	53	29	914	34	3.7	30	0	0	0.0
2017.01.10	54	29	966	0	0.0	30	0	0	0.0
Sum		**29**	**53117**	**2894**		**30**	**0**	**0**	

Table A.19: Full and new results, for queries 29 - 30, "mobile application health sensor data".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	31	996	876	88.0	32	0	0	0.0
2016.11.19	2	31	996	44	4.4	32	0	0	0.0
2016.11.20	3	31	995	22	2.2	32	0	0	0.0
2016.11.21	4	31	997	67	6.7	32	0	0	0.0
2016.11.22	5	31	999	57	5.7	32	0	0	0.0
2016.11.23	6	31	997	0	0.0	32	0	0	0.0
2016.11.24	7	31	997	2	0.2	32	0	0	0.0
2016.11.25	8	31	999	34	3.4	32	0	0	0.0
2016.11.26	9	31	1000	56	5.6	32	0	0	0.0
2016.11.27	10	31	1000	0	0.0	32	0	0	0.0
2016.11.28	11	31	998	47	4.7	32	0	0	0.0
2016.11.29	12	31	998	12	1.2	32	0	0	0.0
2016.11.30	13	31	997	8	0.8	32	0	0	0.0
2016.12.01	14	31	998	40	4.0	32	0	0	0.0
2016.12.02	15	31	997	84	8.4	32	0	0	0.0
2016.12.03	16	31	997	61	6.1	32	0	0	0.0
2016.12.04	17	31	998	18	1.8	32	0	0	0.0
2016.12.05	18	31	996	45	4.5	32	0	0	0.0
2016.12.06	19	31	992	26	2.6	32	0	0	0.0
2016.12.07	20	31	995	18	1.8	32	0	0	0.0
2016.12.08	21	31	999	17	1.7	32	0	0	0.0
2016.12.09	22	31	977	20	2.0	32	0	0	0.0
2016.12.10	23	31	998	26	2.6	32	0	0	0.0
2016.12.11	24	31	989	8	0.8	32	0	0	0.0
2016.12.12	25	31	996	29	2.9	32	0	0	0.0
2016.12.13	26	31	917	16	1.7	32	0	0	0.0
2016.12.14	27	31	965	134	13.9	32	0	0	0.0
2016.12.15	28	31	963	63	6.5	32	0	0	0.0
2016.12.16	29	31	996	10	1.0	32	0	0	0.0
2016.12.17	30	31	962	22	2.3	32	0	0	0.0
2016.12.18	31	31	921	13	1.4	32	0	0	0.0
2016.12.19	32	31	963	2	0.2	32	0	0	0.0
2016.12.20	33	31	982	12	1.2	32	0	0	0.0
2016.12.21	34	31	973	30	3.1	32	0	0	0.0
2016.12.22	35	31	970	22	2.3	32	0	0	0.0
2016.12.23	36	31	984	10	1.0	32	0	0	0.0
2016.12.24	37	31	993	1	0.1	32	0	0	0.0
2016.12.25	38	31	993	2	0.2	32	0	0	0.0
2016.12.26	39	31	999	44	4.4	32	0	0	0.0
2016.12.27	40	31	996	0	0.0	32	0	0	0.0
2016.12.28	41	31	999	8	0.8	32	0	0	0.0
2016.12.29	42	31	995	0	0.0	32	PI 0	0	0.0
2016.12.30	43	31	999	25	2.5	32	0	0	0.0
2016.12.31	44	31	997	5	0.5	32	0	0	0.0
2017.01.01	45	31	998	0	0.0	32	0	0	0.0
2017.01.02	46	31	1000	27	2.7	32	0	0	0.0
2017.01.03	47	31	990	3	0.3	32	0	0	0.0
2017.01.04	48	31	894	23	2.6	32	0	0	0.0
2017.01.05	49	31	895	55	6.1	32	0	0	0.0
2017.01.06	50	31	882	12	1.4	32	0	0	0.0
2017.01.07	51	31	943	11	1.2	32	0	0	0.0
2017.01.08	52	31	908	20	2.2	32	0	0	0.0
2017.01.09	53	31	964	19	2.0	32	0	0	0.0
2017.01.10	54	31	928	0	0.0	32	0	0	0.0
Sum		**31**	**52870**	**2206**		**32**	**0**	**0**	

Table A.20: Full and new results, for queries 31 - 32, "mobile phone body area network".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	33	1000	932	93.2	34	0	0	0.0
2016.11.19	2	33	999	42	4.2	34	0	0	0.0
2016.11.20	3	33	997	21	2.1	34	0	0	0.0
2016.11.21	4	33	997	68	6.8	34	0	0	0.0
2016.11.22	5	33	997	63	6.3	34	0	0	0.0
2016.11.23	6	33	997	7	0.7	34	0	0	0.0
2016.11.24	7	33	997	3	0.3	34	0	0	0.0
2016.11.25	8	33	998	46	4.6	34	0	0	0.0
2016.11.26	9	33	995	117	11.8	34	0	0	0.0
2016.11.27	10	33	996	8	0.8	34	0	0	0.0
2016.11.28	11	33	999	60	6.0	34	0	0	0.0
2016.11.29	12	33	998	28	2.8	34	0	0	0.0
2016.11.30	13	33	998	42	4.2	34	0	0	0.0
2016.12.01	14	33	998	54	5.4	34	0	0	0.0
2016.12.02	15	33	1000	92	9.2	34	0	0	0.0
2016.12.03	16	33	998	51	5.1	34	0	0	0.0
2016.12.04	17	33	1000	30	3.0	34	0	0	0.0
2016.12.05	18	33	993	106	10.7	34	0	0	0.0
2016.12.06	19	33	954	53	5.6	34	0	0	0.0
2016.12.07	20	33	996	25	2.5	34	0	0	0.0
2016.12.08	21	33	977	23	2.4	34	0	0	0.0
2016.12.09	22	33	999	63	6.3	34	0	0	0.0
2016.12.10	23	33	998	43	4.3	34	0	0	0.0
2016.12.11	24	33	999	20	2.0	34	0	0	0.0
2016.12.12	25	33	979	69	7.0	34	0	0	0.0
2016.12.13	26	33	996	24	2.4	34	0	0	0.0
2016.12.14	27	33	966	249	25.8	34	0	0	0.0
2016.12.15	28	33	983	18	1.8	34	0	0	0.0
2016.12.16	29	33	1000	14	1.4	34	0	0	0.0
2016.12.17	30	33	987	13	1.3	34	0	0	0.0
2016.12.18	31	33	988	5	0.5	34	0	0	0.0
2016.12.19	32	33	948	5	0.5	34	0	0	0.0
2016.12.20	33	33	986	36	3.7	34	0	0	0.0
2016.12.21	34	33	949	45	4.7	34	0	0	0.0
2016.12.22	35	33	982	25	2.5	34	0	0	0.0
2016.12.23	36	33	999	9	0.9	34	0	0	0.0
2016.12.24	37	33	999	2	0.2	34	0	0	0.0
2016.12.25	38	33	999	5	0.5	34	0	0	0.0
2016.12.26	39	33	1000	92	9.2	34	0	0	0.0
2016.12.27	40	33	1000	5	0.5	34	0	0	0.0
2016.12.28	41	33	1000	17	1.7	34	0	0	0.0
2016.12.29	42	33	1000	0	0.0	34	0	0	0.0
2016.12.30	43	33	999	68	6.8	34	0	0	0.0
2016.12.31	44	33	999	9	0.9	34	0	0	0.0
2017.01.01	45	33	999	6	0.6	34	0	0	0.0
2017.01.02	46	33	999	63	6.3	34	0	0	0.0
2017.01.03	47	33	999	12	1.2	34	0	0	0.0
2017.01.04	48	33	932	32	3.4	34	0	0	0.0
2017.01.05	49	33	898	41	4.6	34	0	0	0.0
2017.01.06	50	33	932	57	6.1	34	0	0	0.0
2017.01.07	51	33	909	37	4.1	34	0	0	0.0
2017.01.08	52	33	825	30	3.6	34	0	0	0.0
2017.01.09	53	33	932	34	3.6	34	0	0	0.0
2017.01.10	54	33	932	0	0.0	34	0	0	0.0
Sum		**33**	**52996**	**3019**		**34**	**0**	**0**	

Table A.21: Full and new results, for queries 33 - 34, "mobile phone sensor research health".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	35	999	930	93.1	36	379	247	65.2
2016.11.19	2	35	999	37	3.7	36	472	2	0.4
2016.11.20	3	35	997	29	2.9	36	509	10	2.0
2016.11.21	4	35	999	58	5.8	36	515	20	3.9
2016.11.22	5	35	999	29	2.9	36	457	10	2.2
2016.11.23	6	35	998	12	1.2	36	475	1	0.2
2016.11.24	7	35	997	8	0.8	36	504	1	0.2
2016.11.25	8	35	1000	23	2.3	36	501	11	2.2
2016.11.26	9	35	1000	45	4.5	36	419	15	3.6
2016.11.27	10	35	1000	3	0.3	36	445	1	0.2
2016.11.28	11	35	999	175	17.5	36	934	64	6.9
2016.11.29	12	35	999	48	4.8	36	403	3	0.7
2016.11.30	13	35	996	29	2.9	36	443	8	1.8
2016.12.01	14	35	997	42	4.2	36	390	3	0.8
2016.12.02	15	35	999	58	5.8	36	350	18	5.1
2016.12.03	16	35	999	22	2.2	36	440	3	0.7
2016.12.04	17	35	998	17	1.7	36	457	10	2.2
2016.12.05	18	35	999	35	3.5	36	664	10	1.5
2016.12.06	19	35	999	17	1.7	36	429	5	1.2
2016.12.07	20	35	997	17	1.7	36	312	4	1.3
2016.12.08	21	35	966	34	3.5	36	368	4	1.1
2016.12.09	22	35	992	27	2.7	36	447	8	1.8
2016.12.10	23	35	982	25	2.5	36	481	6	1.2
2016.12.11	24	35	993	28	2.8	36	356	0	0.0
2016.12.12	25	35	997	50	5.0	36	696	5	0.7
2016.12.13	26	35	924	48	5.2	36	745	9	1.2
2016.12.14	27	35	982	87	8.9	36	741	35	4.7
2016.12.15	28	35	946	17	1.8	36	783	6	0.8
2016.12.16	29	35	998	8	0.8	36	794	1	0.1
2016.12.17	30	35	982	12	1.2	36	810	3	0.4
2016.12.18	31	35	981	4	0.4	36	815	2	0.2
2016.12.19	32	35	986	20	2.0	36	811	0	0.0
2016.12.20	33	35	973	29	3.0	36	826	14	1.7
2016.12.21	34	35	984	20	2.0	36	798	16	2.0
2016.12.22	35	35	967	38	3.9	36	754	8	1.1
2016.12.23	36	35	980	25	2.6	36	—	—	0.0
2016.12.24	37	35	995	5	0.5	36	702	7	1.0
2016.12.25	38	35	995	3	0.3	36	745	2	0.3
2016.12.26	39	35	998	87	8.7	36	652	31	4.8
2016.12.27	40	35	998	19	1.9	36	638	1	0.2
2016.12.28	41	35	997	20	2.0	36	616	4	0.6
2016.12.29	42	35	997	1	0.1	36	616	0	0.0
2016.12.30	43	35	998	45	4.5	36	514	18	3.5
2016.12.31	44	35	998	24	2.4	36	504	1	0.2
2017.01.01	45	35	998	23	2.3	36	464	5	1.1
2017.01.02	46	35	998	37	3.7	36	421	19	4.5
2017.01.03	47	35	997	5	0.5	36	433	4	0.9
2017.01.04	48	35	877	29	3.3	36	411	5	1.2
2017.01.05	49	35	906	31	3.4	36	419	14	3.3
2017.01.06	50	35	936	29	3.1	36	383	4	1.0
2017.01.07	51	35	904	29	3.2	36	465	5	1.1
2017.01.08	52	35	918	17	1.9	36	436	16	3.7
2017.01.09	53	35	882	32	3.6	36	419	5	1.2
2017.01.10	54	35	963	0	0.0	36	425	0	0.0
Sum		**35**	**52958**	**2542**		**36**	**28986**	**704**	

Table A.22: Full and new results, for queries 35 - 36, "forest fairytales".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	37	1000	905	90.5	38	645	535	82.9
2016.11.19	2	37	1000	68	6.8	38	611	11	1.8
2016.11.20	3	37	999	44	4.4	38	620	14	2.3
2016.11.21	4	37	1000	43	4.3	38	667	24	3.6
2016.11.22	5	37	1000	89	8.9	38	901	17	1.9
2016.11.23	6	37	1000	6	0.6	38	966	7	0.7
2016.11.24	7	37	1000	0	0.0	38	811	6	0.7
2016.11.25	8	37	1000	36	3.6	38	774	14	1.8
2016.11.26	9	37	1000	71	7.1	38	732	18	2.5
2016.11.27	10	37	1000	3	0.3	38	737	1	0.1
2016.11.28	11	37	984	213	21.6	38	613	71	11.6
2016.11.29	12	37	1000	50	5.0	38	699	2	0.3
2016.11.30	13	37	1000	46	4.6	38	735	8	1.1
2016.12.01	14	37	1000	48	4.8	38	814	17	2.1
2016.12.02	15	37	1000	82	8.2	38	889	46	5.2
2016.12.03	16	37	1000	17	1.7	38	881	8	0.9
2016.12.04	17	37	998	31	3.1	38	785	17	2.2
2016.12.05	18	37	997	48	4.8	38	837	25	3.0
2016.12.06	19	37	997	13	1.3	38	893	4	0.4
2016.12.07	20	37	1000	20	2.0	38	890	8	0.9
2016.12.08	21	37	994	35	3.5	38	832	8	1.0
2016.12.09	22	37	1000	34	3.4	38	815	15	1.8
2016.12.10	23	37	999	46	4.6	38	814	6	0.7
2016.12.11	24	37	996	18	1.8	38	770	10	1.3
2016.12.12	25	37	995	63	6.3	38	849	22	2.6
2016.12.13	26	37	947	34	3.6	38	880	27	3.1
2016.12.14	27	37	958	119	12.4	38	718	42	5.8
2016.12.15	28	37	943	14	1.5	38	722	5	0.7
2016.12.16	29	37	962	19	2.0	38	723	5	0.7
2016.12.17	30	37	950	25	2.6	38	728	6	0.8
2016.12.18	31	37	943	7	0.7	38	741	10	1.3
2016.12.19	32	37	934	31	3.3	38	776	5	0.6
2016.12.20	33	37	947	30	3.2	38	774	17	2.2
2016.12.21	34	37	945	55	5.8	38	773	22	2.8
2016.12.22	35	37	942	15	1.6	38	783	9	1.1
2016.12.23	36	37	939	32	3.4	38	764	5	0.7
2016.12.24	37	37	952	4	0.4	38	856	8	0.9
2016.12.25	38	37	953	4	0.4	38	878	15	1.7
2016.12.26	39	37	995	101	10.2	38	789	20	2.5
2016.12.27	40	37	990	6	0.6	38	719	2	0.3
2016.12.28	41	37	990	35	3.5	38	625	12	1.9
2016.12.29	42	37	989	3	0.3	38	636	7	1.1
2016.12.30	43	37	951	52	5.5	38	608	18	3.0
2016.12.31	44	37	949	7	0.7	38	623	5	0.8
2017.01.01	45	37	949	16	1.7	38	623	16	2.6
2017.01.02	46	37	976	73	7.5	38	526	13	2.5
2017.01.03	47	37	976	1	0.1	38	545	4	0.7
2017.01.04	48	37	878	18	2.1	38	603	18	3.0
2017.01.05	49	37	886	33	3.7	38	603	16	2.7
2017.01.06	50	37	815	29	3.6	38	622	10	1.6
2017.01.07	51	37	882	23	2.6	38	PI 551	8	1.5
2017.01.08	52	37	898	34	3.8	38	573	15	2.6
2017.01.09	53	37	998	45	4.5	38	654	24	3.7
2017.01.10	54	37	918	0	0.0	38	593	0	0.0
Sum		**37**	**52314**	**2894**		**38**	**39589**	**1278**	

Table A.23: Full and new results, for queries 37 - 38, "tudor politics".

Date	Day in period	ID	Free	New	New %	ID	Exact	New	New %
2016.11.18	1	39	999	940	94.1	40	776	663	85.4
2016.11.19	2	39	999	76	7.6	40	784	43	5.5
2016.11.20	3	39	CT 949	26	2.7	40	768	26	3.4
2016.11.21	4	39	998	35	3.5	40	760	21	2.8
2016.11.22	5	39	1000	33	3.3	40	792	36	4.5
2016.11.23	6	39	999	11	1.1	40	799	30	3.8
2016.11.24	7	39	999	6	0.6	40	790	20	2.5
2016.11.25	8	39	1000	26	2.6	40	779	13	1.7
2016.11.26	9	39	1000	50	5.0	40	831	24	2.9
2016.11.27	10	39	CT 950	3	0.3	40	786	8	1.0
2016.11.28	11	39	1000	172	17.2	40	826	188	22.8
2016.11.29	12	39	1000	44	4.4	40	747	41	5.5
2016.11.30	13	39	1000	23	2.3	40	758	25	3.3
2016.12.01	14	39	—	—	0.0	40	899	58	6.5
2016.12.02	15	39	1000	64	6.4	40	802	42	5.2
2016.12.03	16	39	998	35	3.5	40	779	22	2.8
2016.12.04	17	39	998	28	2.8	40	819	12	1.5
2016.12.05	18	39	1000	35	3.5	40	896	26	2.9
2016.12.06	19	39	999	15	1.5	40	859	6	0.7
2016.12.07	20	39	981	21	2.1	40	810	11	1.4
2016.12.08	21	39	917	39	4.3	40	865	15	1.7
2016.12.09	22	39	996	48	4.8	40	881	45	5.1
2016.12.10	23	39	999	25	2.5	40	899	22	2.4
2016.12.11	24	39	978	16	1.6	40	899	10	1.1
2016.12.12	25	39	999	57	5.7	40	793	35	4.4
2016.12.13	26	39	926	61	6.6	40	807	32	4.0
2016.12.14	27	39	546	82	15.0	40	757	49	6.5
2016.12.15	28	39	957	75	7.8	40	710	13	1.8
2016.12.16	29	39	999	21	2.1	40	707	4	0.6
2016.12.17	30	39	976	11	1.1	40	732	25	3.4
2016.12.18	31	39	981	9	0.9	40	787	2	0.3
2016.12.19	32	39	979	26	2.7	40	684	29	4.2
2016.12.20	33	39	948	38	4.0	40	705	21	3.0
2016.12.21	34	39	997	30	3.0	40	759	33	4.3
2016.12.22	35	39	978	29	3.0	40	767	12	1.6
2016.12.23	36	39	920	19	2.1	40	759	34	4.5
2016.12.24	37	39	996	13	1.3	40	794	6	0.8
2016.12.25	38	39	995	1	0.1	40	820	44	5.4
2016.12.26	39	39	1000	63	6.3	40	844	24	2.8
2016.12.27	40	39	998	13	1.3	40	796	4	0.5
2016.12.28	41	39	999	27	2.7	40	624	55	8.8
2016.12.29	42	39	995	5	0.5	40	678	15	2.2
2016.12.30	43	39	997	31	3.1	40	882	31	3.5
2016.12.31	44	39	997	2	0.2	40	890	8	0.9
2017.01.01	45	39	999	35	3.5	40	951	23	2.4
2017.01.02	46	39	999	29	2.9	40	574	34	5.9
2017.01.03	47	39	999	1	0.1	40	657	13	2.0
2017.01.04	48	39	928	40	4.3	40	696	38	5.5
2017.01.05	49	39	853	55	6.4	40	704	32	4.5
2017.01.06	50	39	937	23	2.5	40	735	23	3.1
2017.01.07	51	39	905	27	3.0	40	662	23	3.5
2017.01.08	52	39	955	21	2.2	40	709	11	1.6
2017.01.09	53	39	956	25	2.6	40	707	22	3.1
2017.01.10	54	39	894	0	0.0	40	789	0	0.0
Sum		**39**	**51367**	**2640**		**40**	**42083**	**2102**	

Table A.24: Full and new results, for queries 39 - 40, "jazz poetry".

A.3 Plots for results

For completeness and reference, all plots are shown in this section (A.3.1). For values in each plot, refer to the corresponding table in the previous section, see A.2.2.

A.3.1 Summary plots for all queries

In figures A.1 and A.2 are two summary plots, made to get an overview of trends in the results in the data collection period.

What is shown here is the percentage of new results for each day since the start of the period, as described in chapter 3.

Combining the queries in one plot, make trends in results emerge clearly. They follow the same pattern; first day has many results, the days following have fewer results.

In figure A.1 all 20 free queries are shown, the queries that were numbered with odd numbers from 1 - 39.

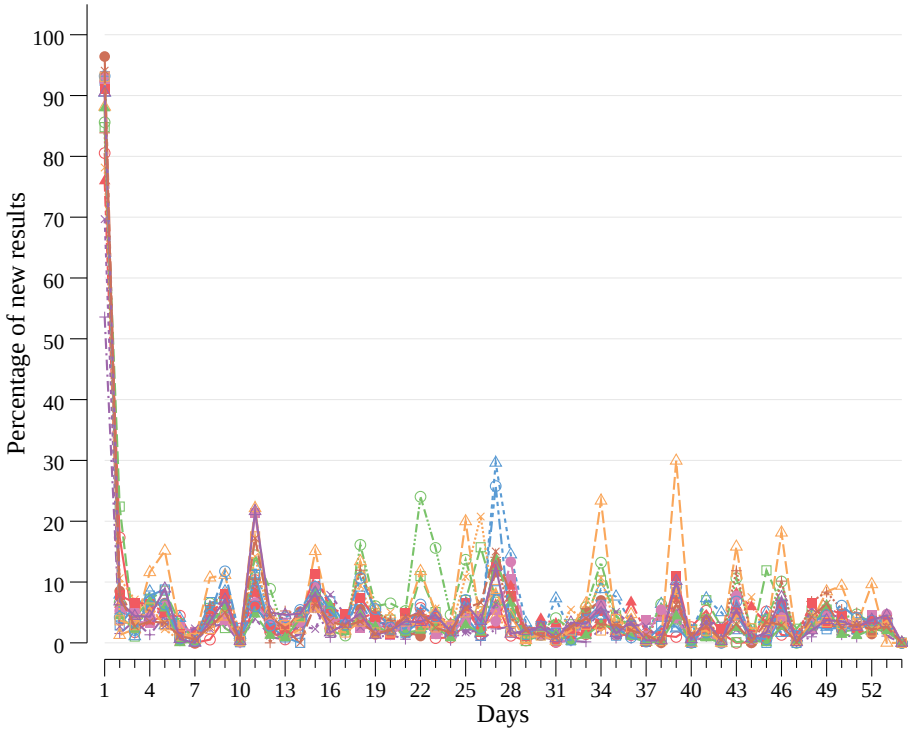


Figure A.1: Combined plots for all free searches.

In figure A.2 all 20 exact queries are shown, the queries that were numbered with even numbers from 2 - 40.

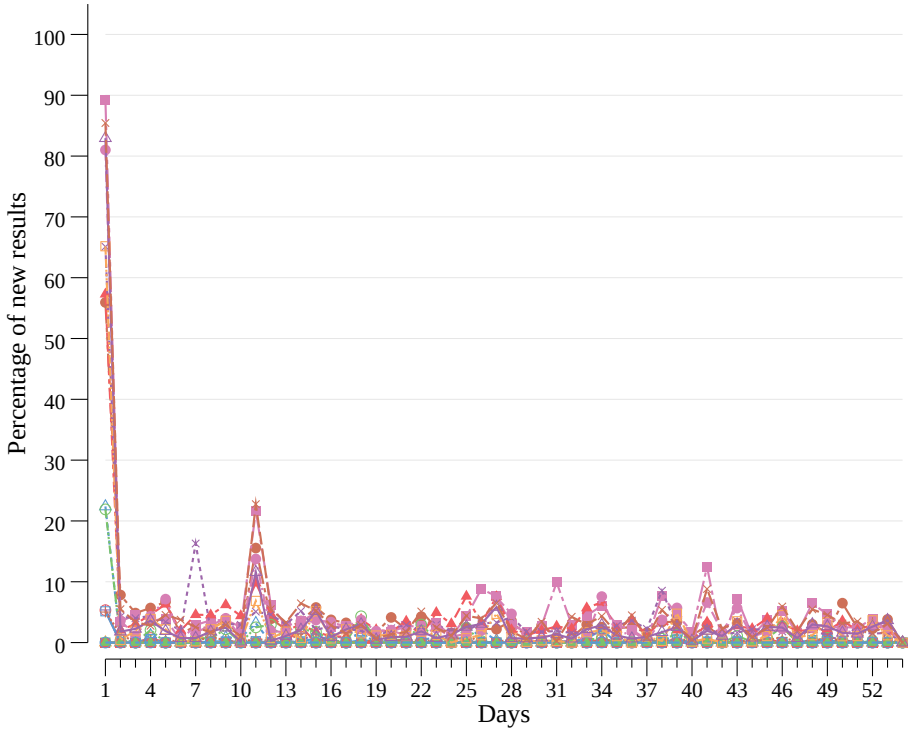


Figure A.2: Combined plots for all exact searches.

Some days have more new results than others. The search engine is continuously collecting results, see 2.1.1 *Building and updating the index*, to be able to provide topical results to its users.

One key difference between the free search plot (A.1) and the exact search plot (A.2) is that there are fewer results overall in the exact search plot. This is because there were fewer results in total for exact results, and by that also fewer new results. As an example of this, see for instance figure A.6 in section A.3.3, and compare the left (free) and the right (exact) plot.

A.3.2 Summary plots for all queries, including days with errors

The full period was originally a period of 71 days, but the first 17 days contained data collection errors as described in section 5.5.

The summary plots below illustrate why day 18 was selected as the starting point for the data selected for analysis. See figures A.3 and A.4 below.

The numbers shown here is the number of results per day, and also in the same plot, the number of new results per day. The data points at the top signifies all results for that query that day. The data points at the bottom are the number of new results for that query that day.

Data in figures A.3 and A.4 are actual number of results per day, not percentage of new results as in section A.3.1.

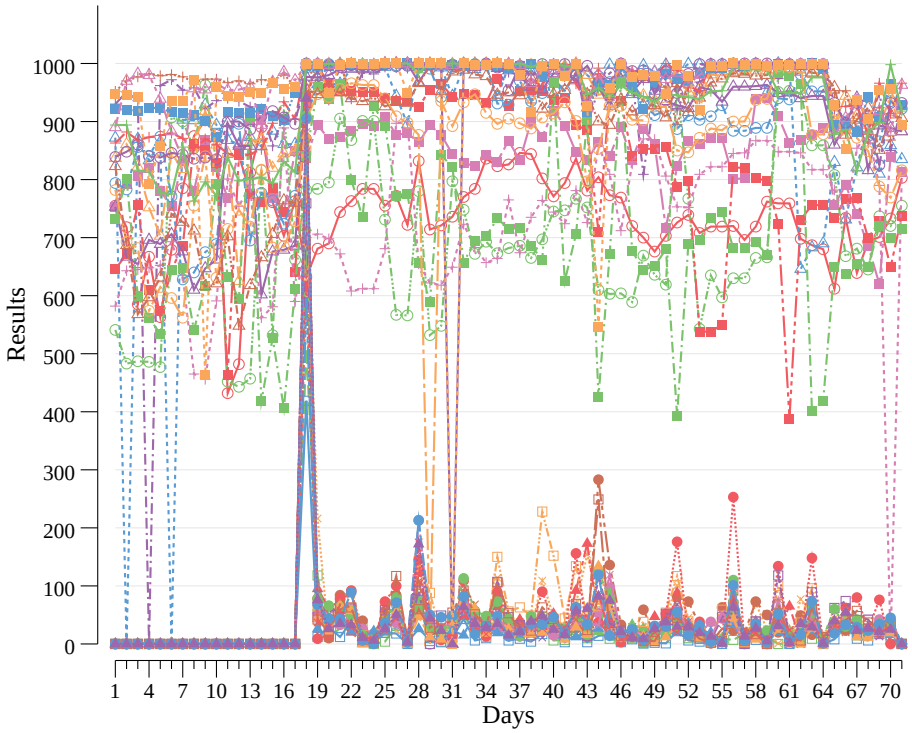


Figure A.3: Combined plots for free searches for the full data collection period. Y-axis shows number of results, not percentage.

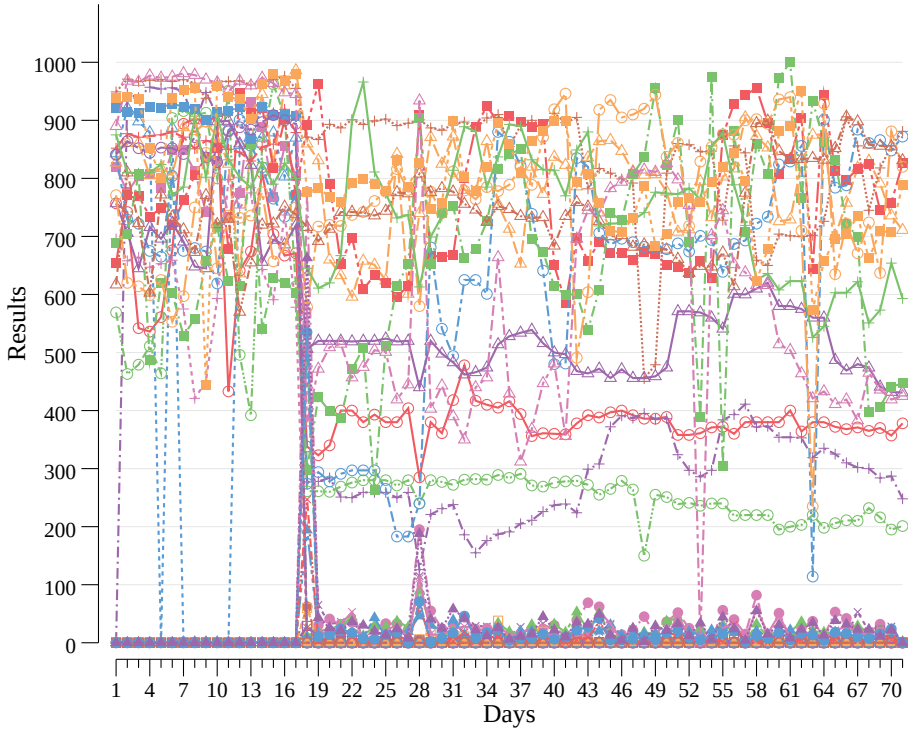


Figure A.4: Combined plots for exact searches for the full data collection period. Y-axis shows number of results, not percentage.

All 40 queries are shown, divided into free and exact searches.

In the left plot all 20 free queries are shown, the queries that were numbered with odd numbers from 1 - 39. We can see a clear change at day 18, which is when *DisableHostCollapsing* was introduced, see section 5.5.3. After this change, the number of returned results went up, closer to the maximum for many of the queries.

In the right plot all 20 exact queries are shown, the queries that were numbered with even numbers from 2 - 40. We can see a clear change at day 18 here, as well. This is when the query searched for was changed to include quotation marks, to make it an exact query. See section 5.5.3 for more on this. After this change, the number of returned results went down, due to fewer exact hits for the query text.

A.3.3 Summary plots for each query

The following plots shows data collection results for the selected period. Plots are shown first as percent-wise plots, then the raw data plots are shown. The

left plot is always the free query, the right plot is always the exact query. See example below.

Figure A5 left Free query "Messerschmitt KR200 restoration", showing new results percentage-wise.

Figure A5 right Exact query "Messerschmitt KR200 restoration", showing new results percentage-wise.

Figure A6 left Free query "Messerschmitt KR200 restoration", showing total number of results, and number of new results.

Figure A6 right Exact query "Messerschmitt KR200 restoration", showing total number of results, and number of new results.

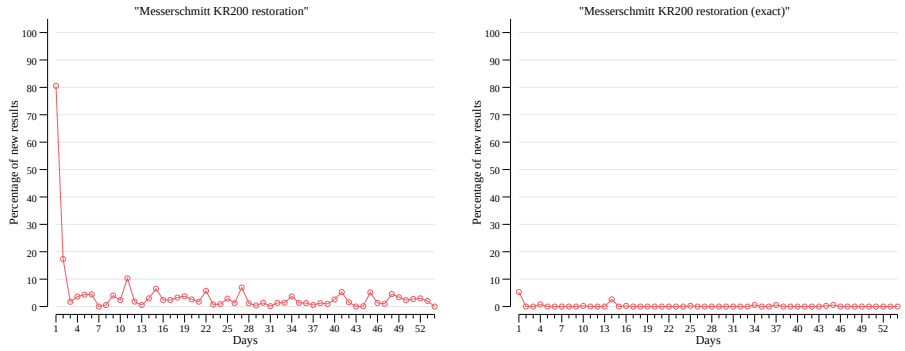


Figure A.5: Percentage-wise plots for queries 1 and 2.

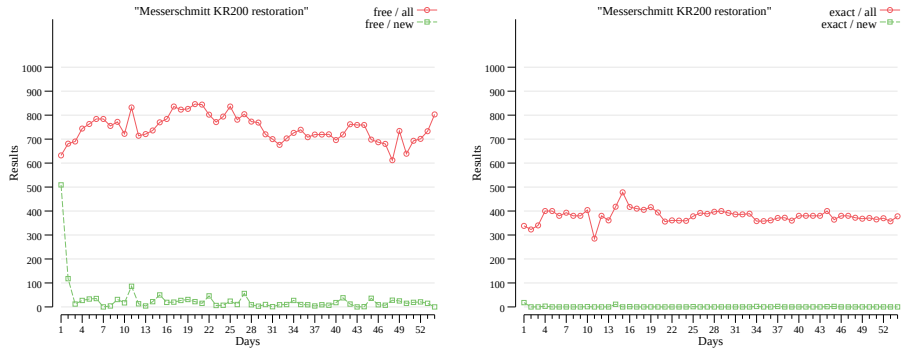


Figure A.6: Data plots for queries 1 and 2.

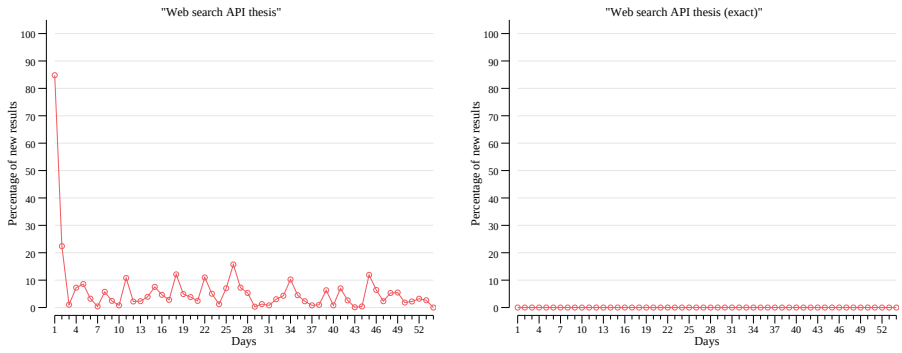


Figure A.7: Percentage-wise plots for queries 3 and 4.

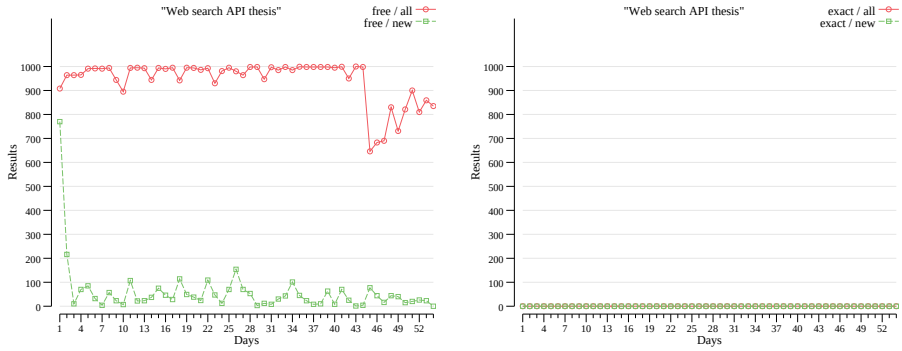


Figure A.8: Data plots for queries 3 and 4.

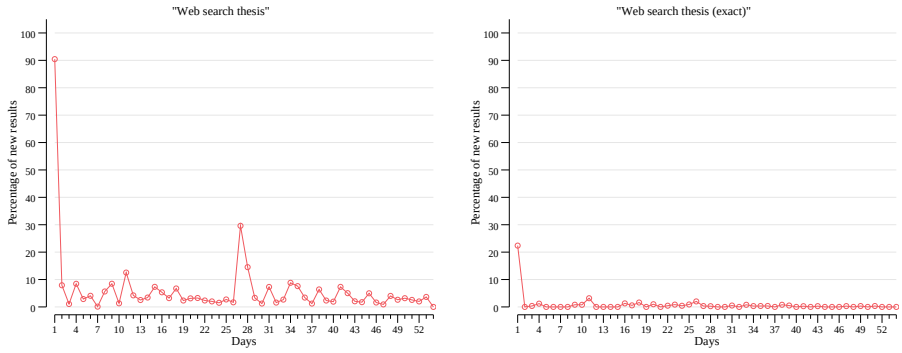


Figure A.9: Percentage-wise plots for queries 5 and 6.

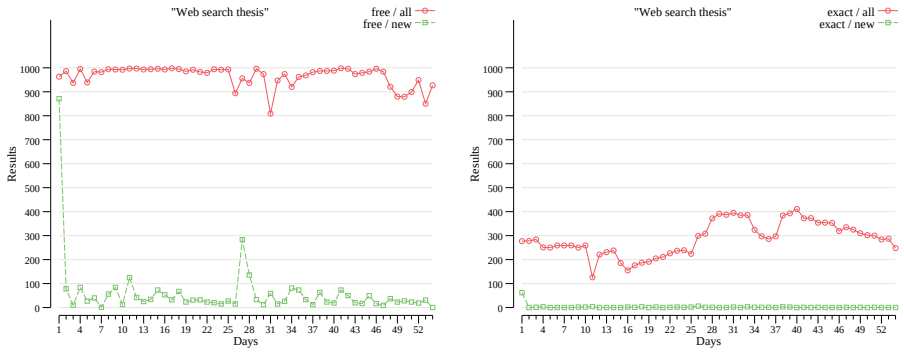


Figure A.10: Data plots for queries 5 and 6.

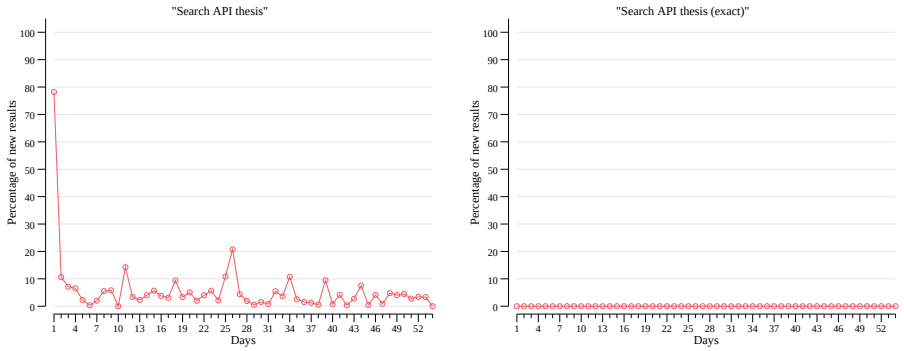


Figure A.11: Percentage-wise plots for queries 7 and 8.

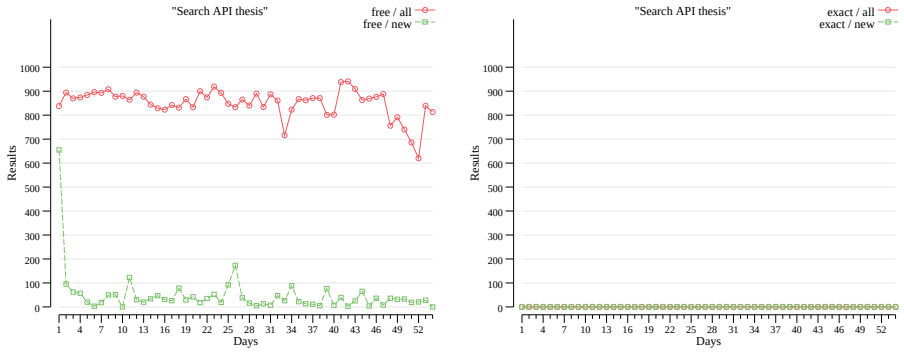


Figure A.12: Data plots for queries 7 and 8.

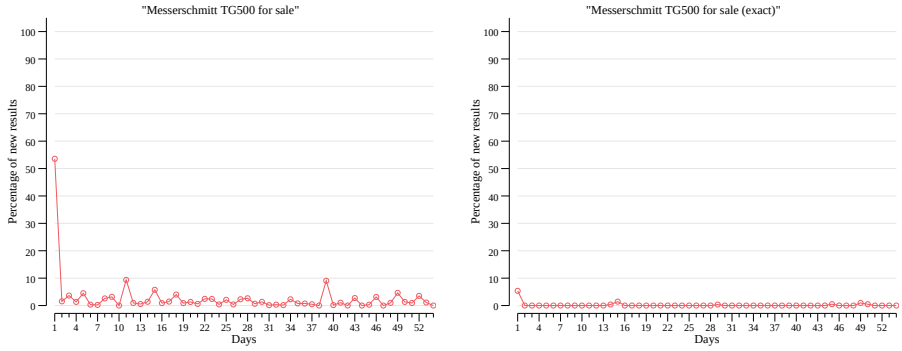


Figure A.13: Percentage-wise plots for queries 9 and 10.

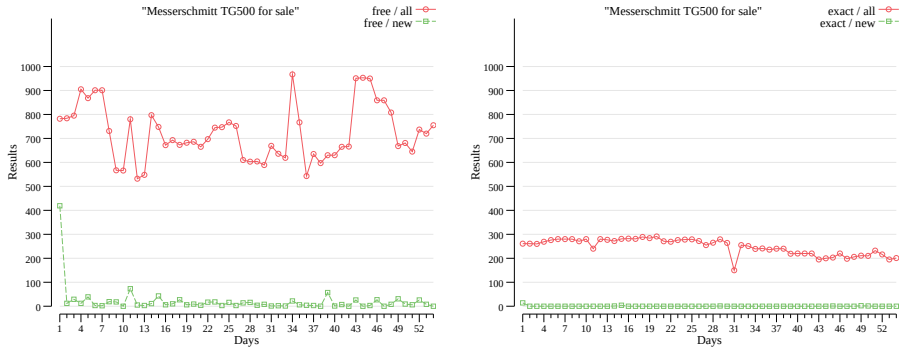


Figure A.14: Data plots for queries 9 and 10.

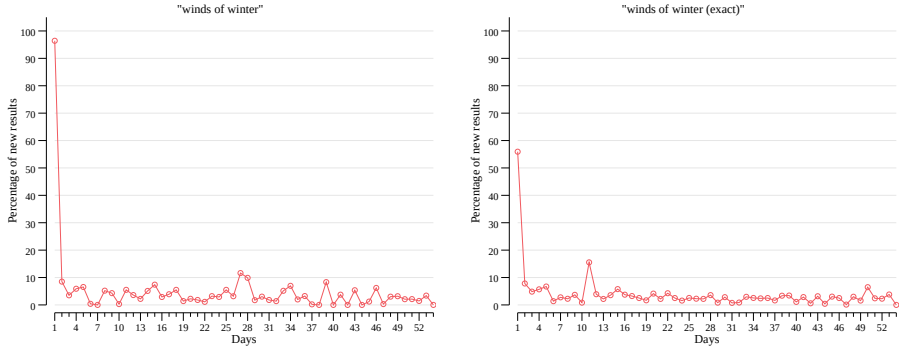


Figure A.15: Percentage-wise plots for queries 11 and 12.

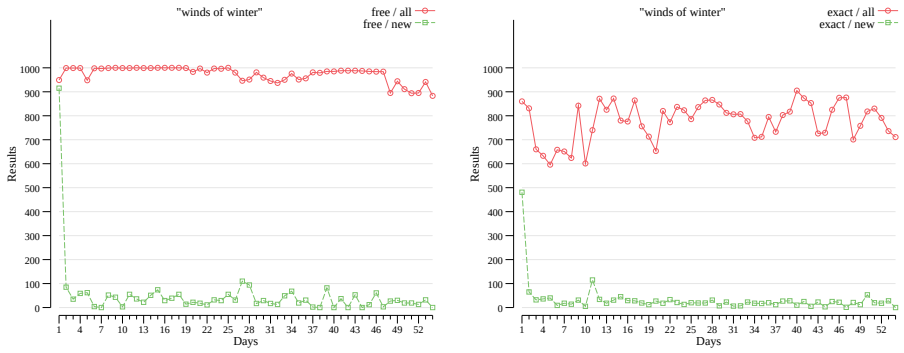


Figure A.16: Data plots for queries 11 and 12.

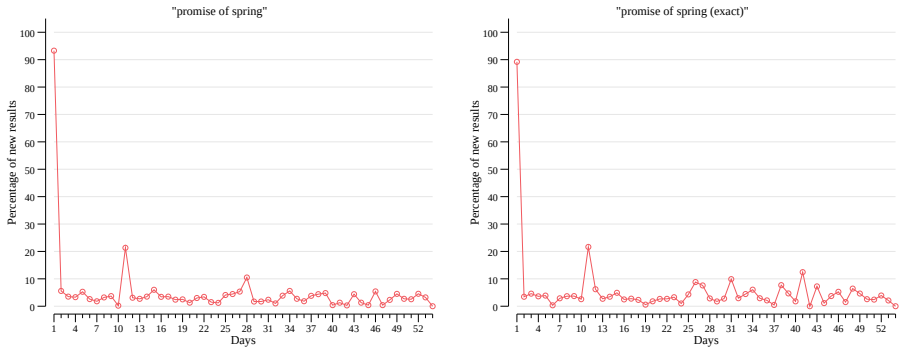


Figure A.17: Percentage-wise plots for queries 13 and 14.

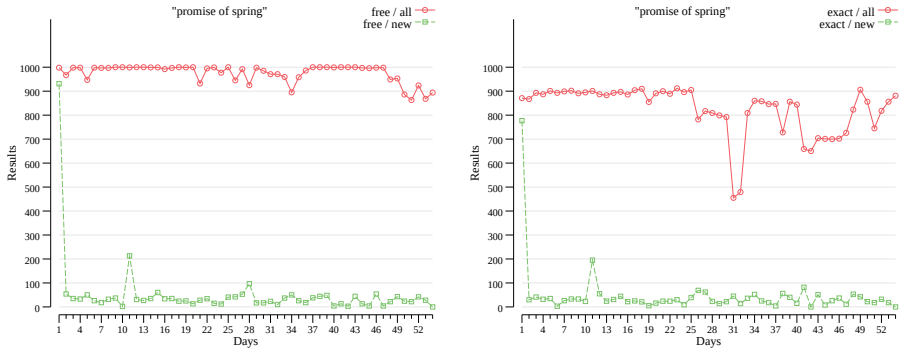


Figure A.18: Data plots for queries 13 and 14.

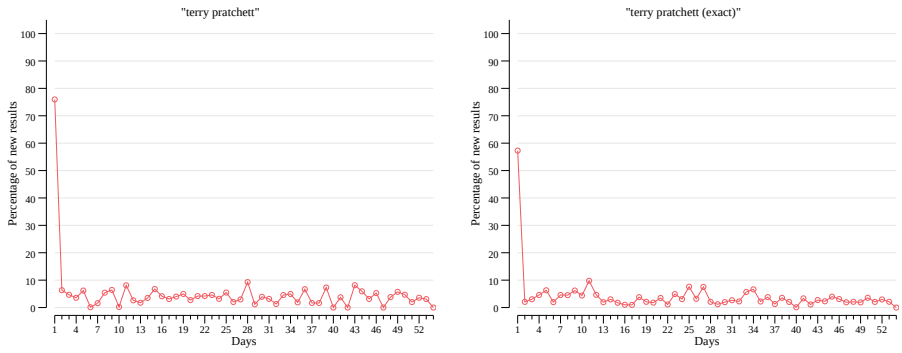


Figure A.19: Percentage-wise plots for queries 15 and 16.

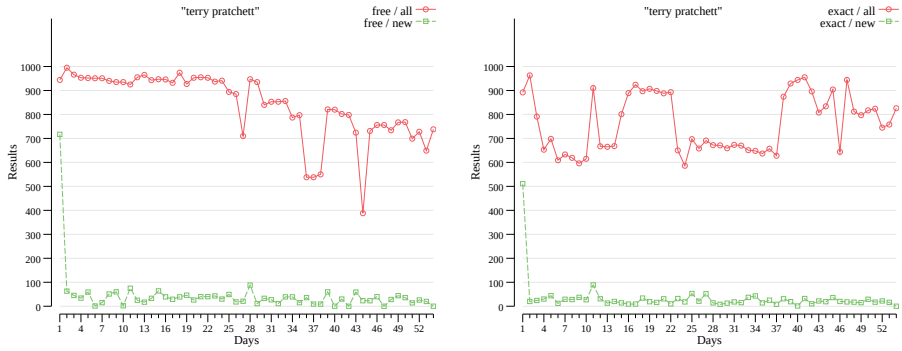


Figure A.20: Data plots for queries 15 and 16.

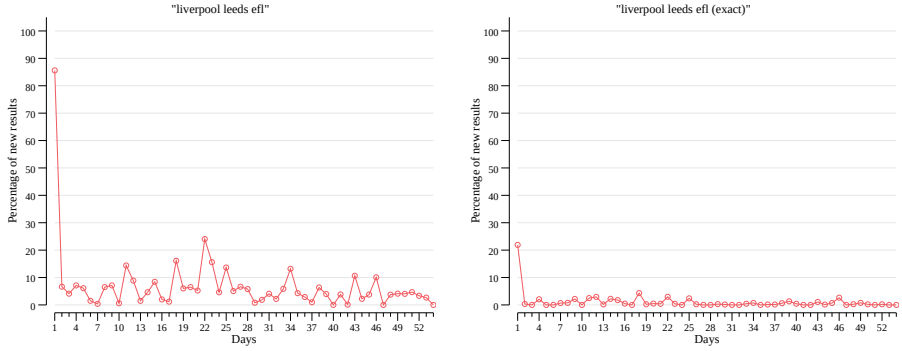


Figure A.21: Percentage-wise plots for queries 17 and 18.

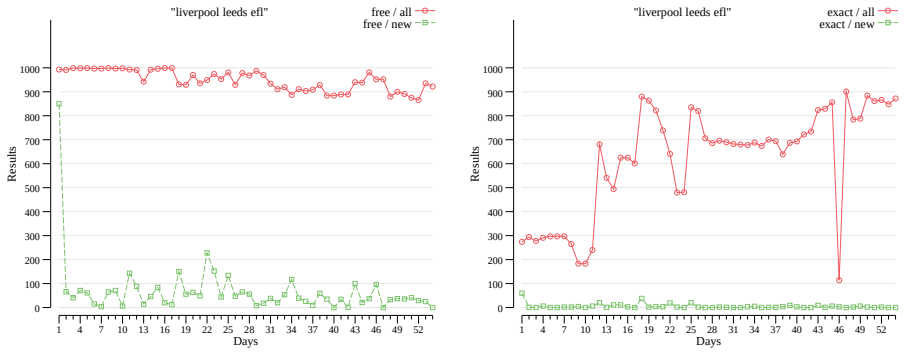


Figure A.22: Data plots for queries 17 and 18.

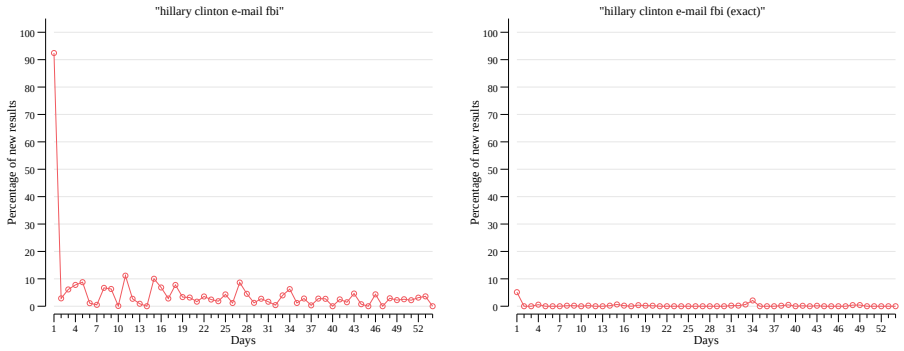


Figure A.23: Percentage-wise plots for queries 19 and 20.

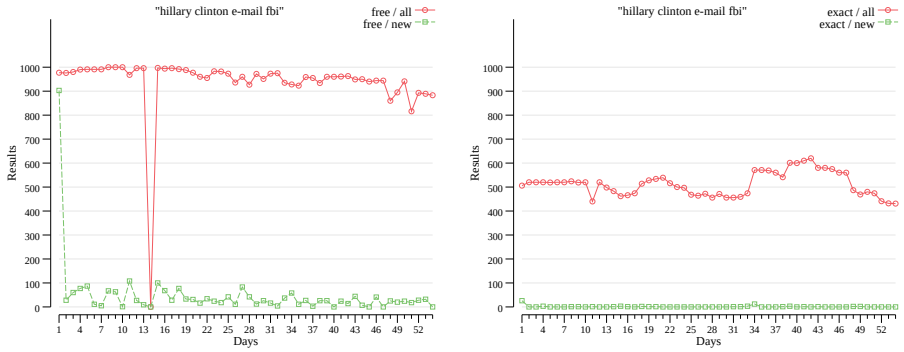


Figure A.24: Data plots for queries 19 and 20.

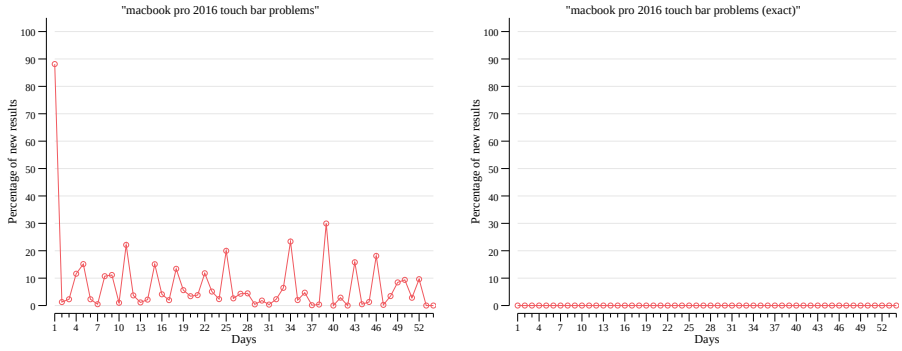


Figure A.25: Percentage-wise plots for queries 21 and 22.

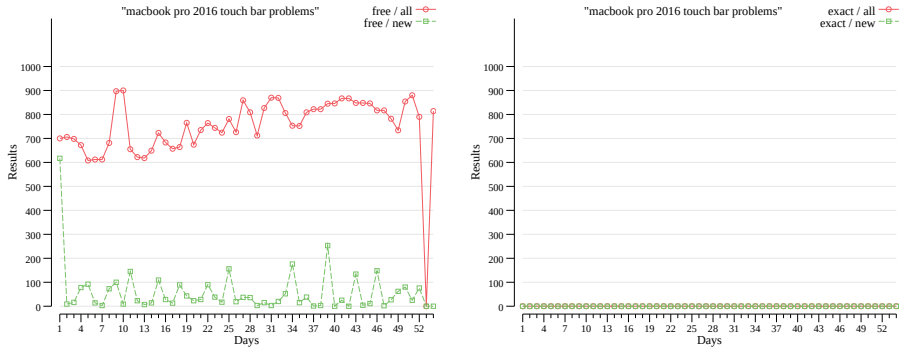


Figure A.26: Data plots for queries 21 and 22.

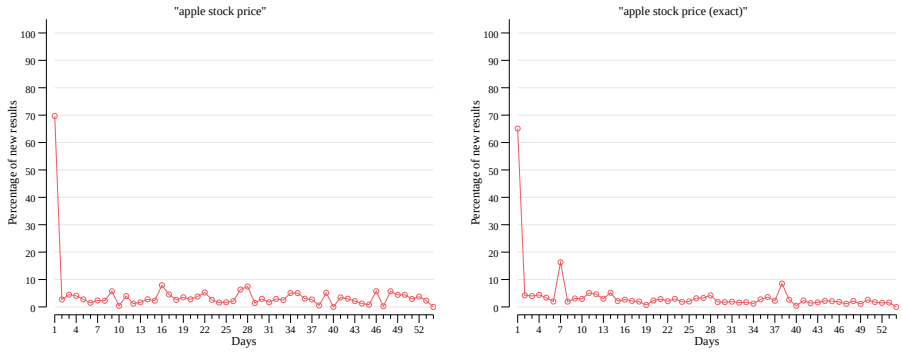


Figure A.27: Percentage-wise plots for queries 23 and 24.

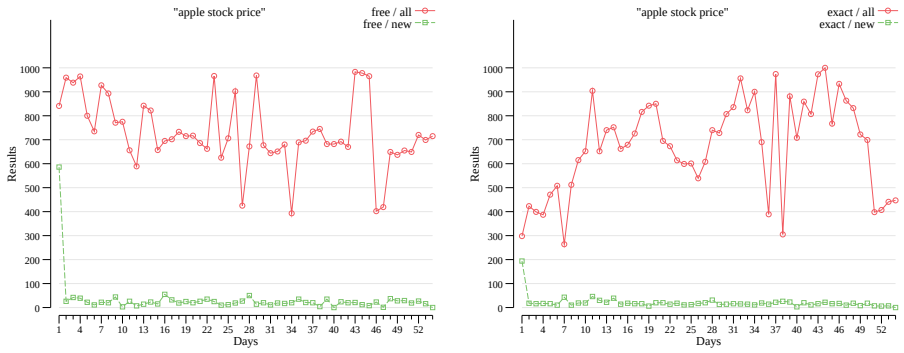


Figure A.28: Data plots for queries 23 and 24.

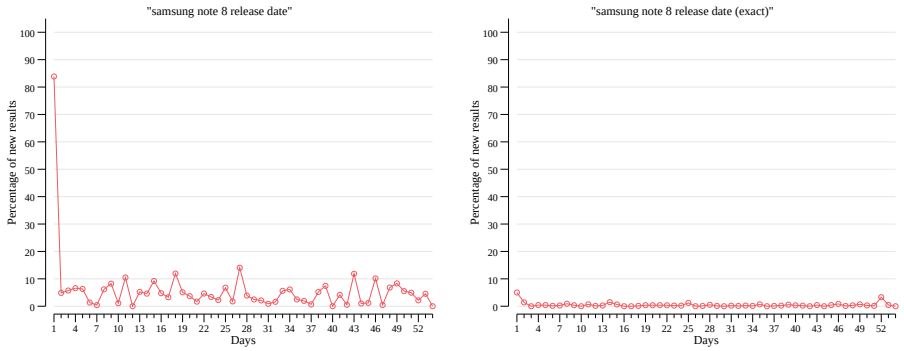


Figure A.29: Percentage-wise plots for queries 25 and 26.

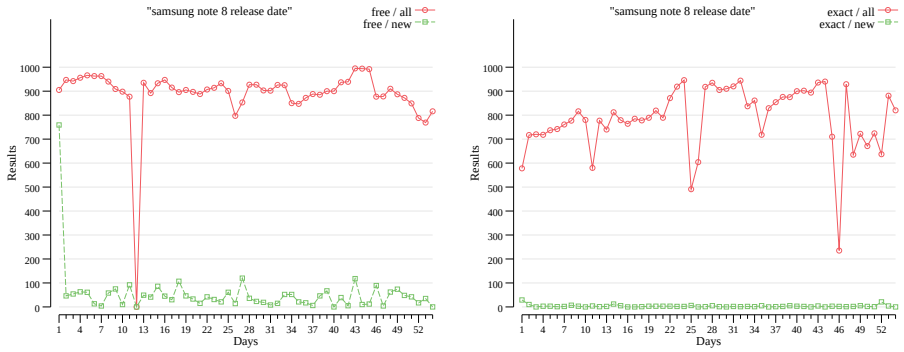


Figure A.30: Data plots for queries 25 and 26.

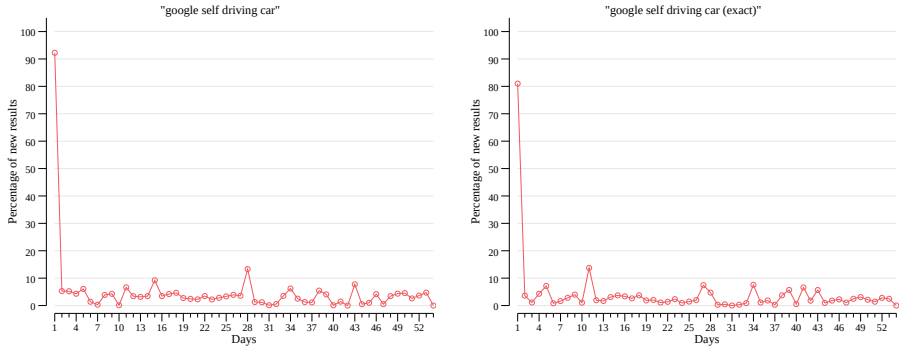


Figure A.31: Percentage-wise plots for queries 27 and 28.

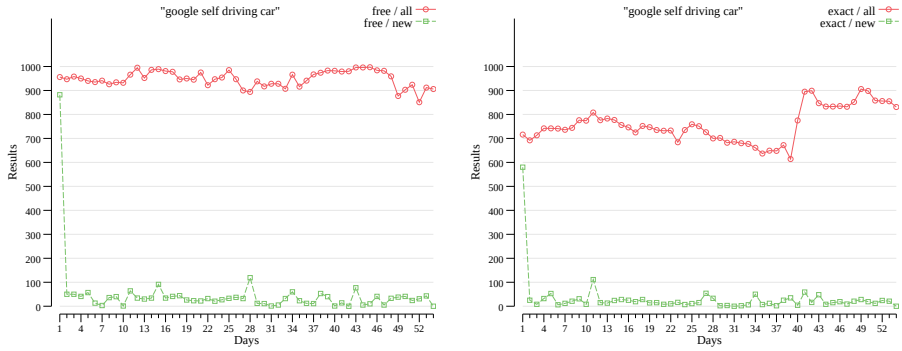


Figure A.32: Data plots for queries 27 and 28.

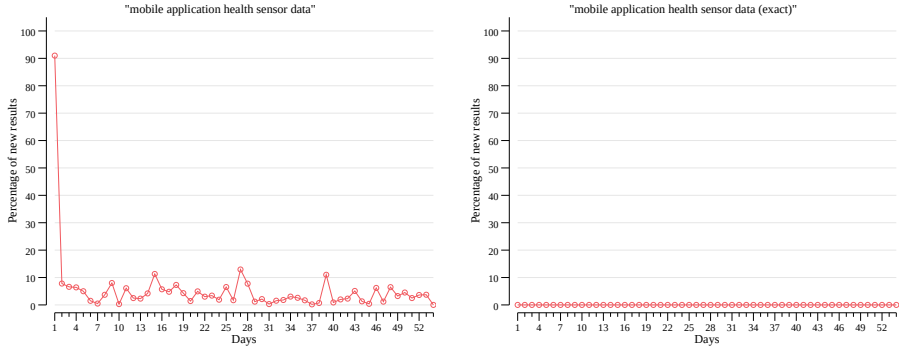


Figure A.33: Percentage-wise plots for queries 29 and 30.

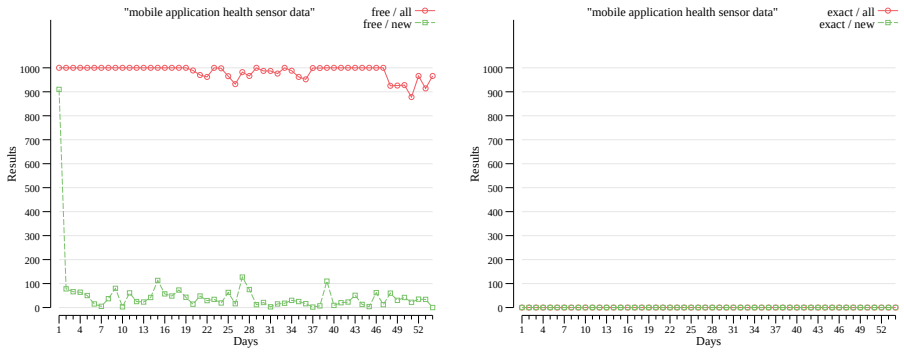


Figure A.34: Data plots for queries 29 and 30.

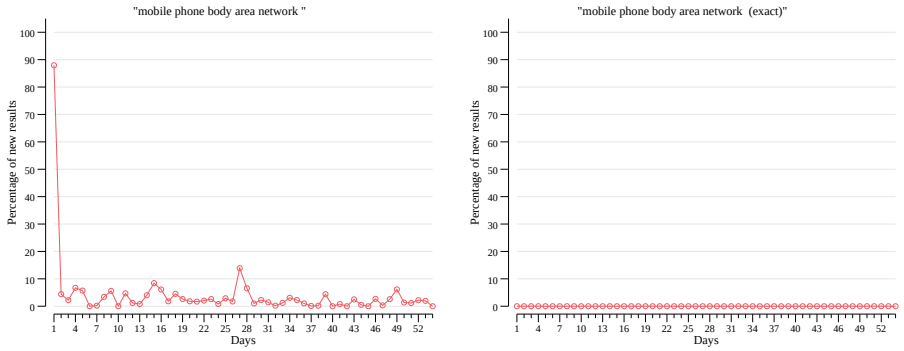


Figure A.35: Percentage-wise plots for queries 31 and 32.

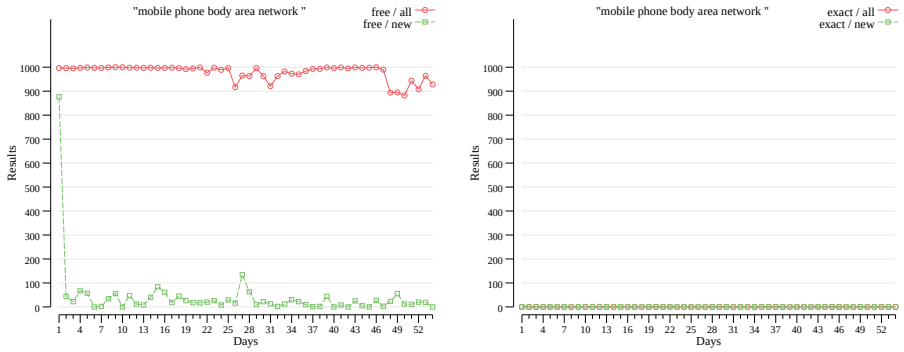


Figure A.36: Data plots for queries 31 and 32.

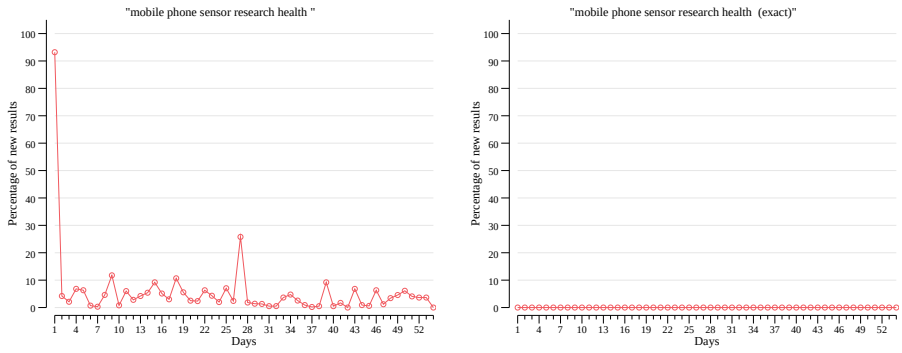


Figure A.37: Percentage-wise plots for queries 33 and 34.

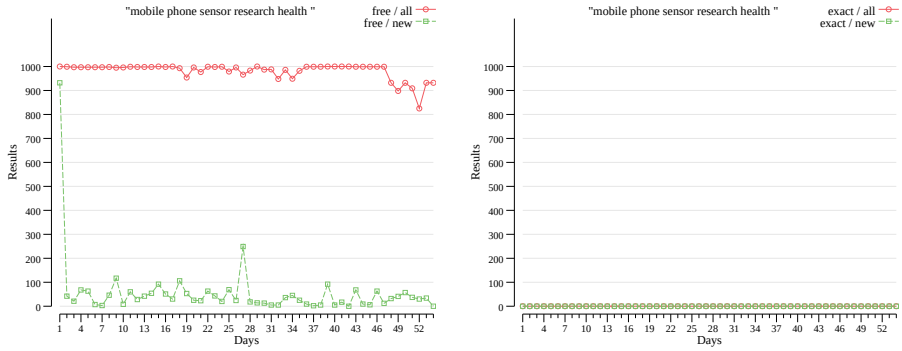


Figure A.38: Data plots for queries 33 and 34.

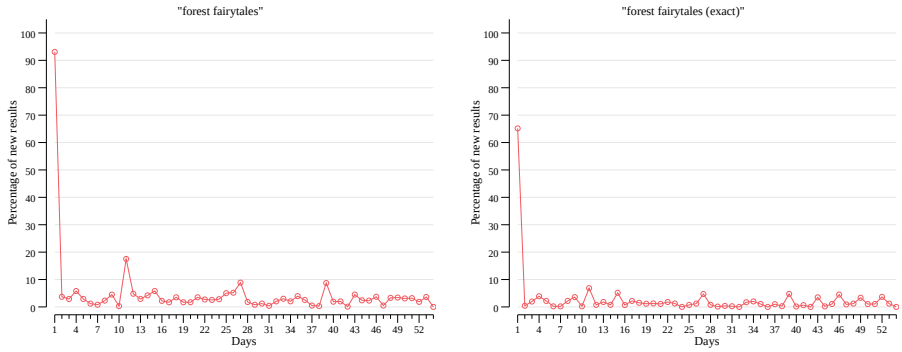


Figure A.39: Percentage-wise plots for queries 35 and 36.

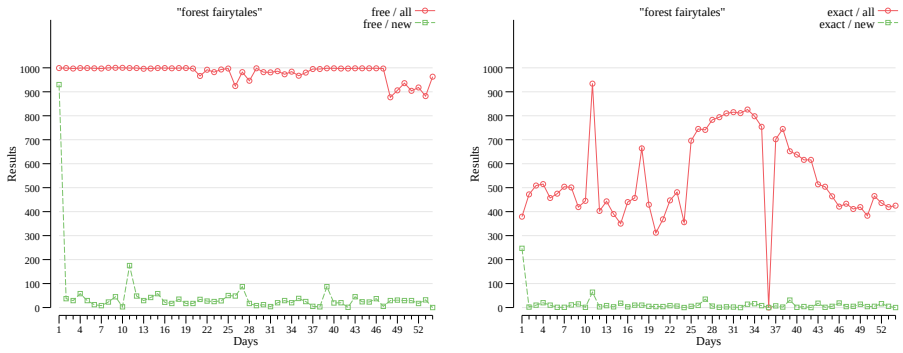


Figure A.40: Data plots for queries 35 and 36.

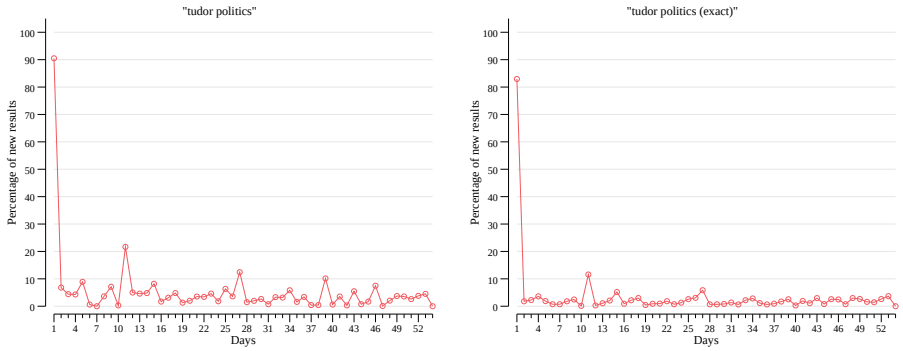


Figure A.41: Percentage-wise plots for queries 37 and 38.

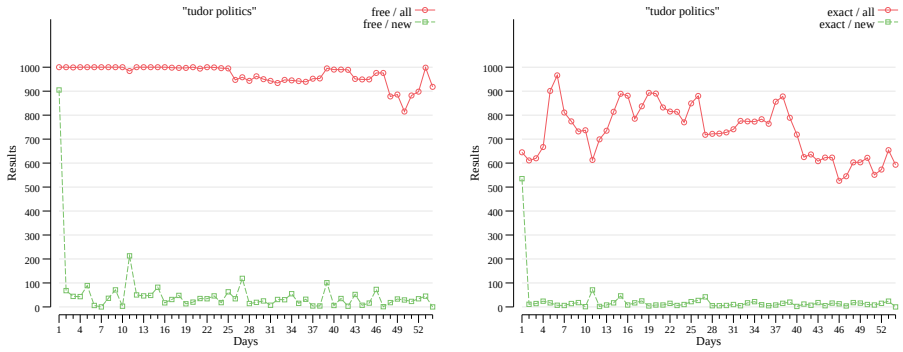


Figure A.42: Data plots for queries 37 and 38.

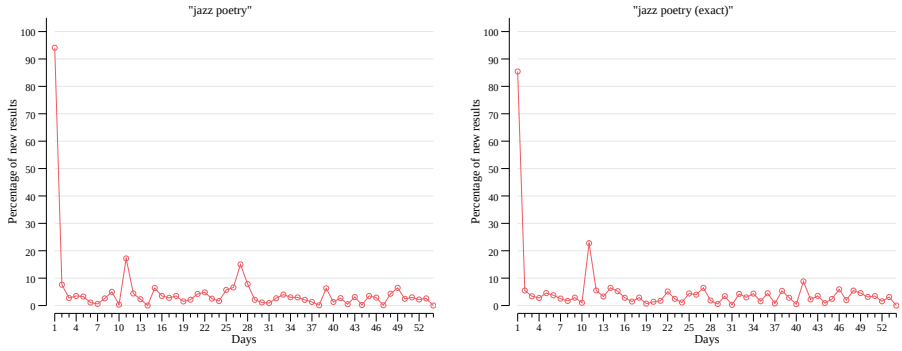


Figure A.43: Percentage-wise plots for queries 39 and 40.

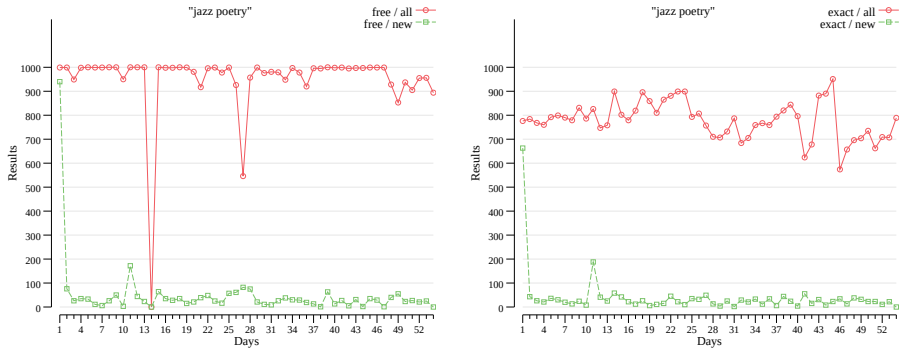


Figure A.44: Data plots for queries 39 and 40.

B

IIR testing instructions

The testers were given these instructions on how to test the solution.

IIR – Incremental Information Retrieval

Test instructions for André, Easterine, Jørgen and Stephan.

Introduction

TL;DR: Weeeell, it's not that important.

Ideally, I wanted to be able to provide a live API testing environment for you guys, I really did. But Microsoft cut that short by closing down Bing v2, which I used initially. They reworked their search API, giving it a new price and service structure.

So, I had the choice of starting again, rewriting the Go version of the Bing Search API that I had created, to support the new version - or continuing, collecting my data from the last spasms of the Bing v2 API.

I chose the latter option, too much work had gone into it to stop. I gathered data from 1st of November 2016 until the end of January 2017. From these data, 54 consecutive days' worth of data is used, from 18th of November till 10th of January. My aim is to get some usage statistics from your test usage.

Short "user guide" for testing

All of you have 3 searches each, which has been run as free search and exact search – which totals 6 searches each. **What I want you to do**, is to:

1) Use **Google Chrome**, it's not tested with other browsers
2) Use a screen width larger than **1200px**, Bootstrap development isn't finalised.
3) Open the IIR solution at http://iir-test.northeurope.cloudapp.azure.com
4) Log on with **your user** (select your name from the dropdown on the login page), and you will see your queries displayed to the left.
5) Read the **online manual** to be more aware what's going on.
6) **Try out** some functionality on one of the searches first, to get a feel for the application. Then use the **reset button** at the bottom of the *Discarded results* folder, and start for real.
   THEN:
7) **For each search**, go through **more than ten** sets of results for the search, really as many as you have the patience for. Upper limit is 54, of course.
8) Make sure you **save** some results you find interesting, and **discard** some results you find uninteresting. Save or discard single results or filtered results, use what you think works best.
9) Optionally **black-list** domains that never has interesting results or **white-list** domains that always has interesting results for the search you are viewing.

Questions

When you are finished testing, there are a few questions I would like you to answer. There is a **Questback survey** with a handful of questions, and comment fields for feedback and suggesting improvements.

Open the Questback survey at https://response.questback.com/erlendjohannessen/dz8mxj8c5j, and use the password **DvZoBrzlQ8** to start the survey.

Happy hunting! 😊
31st of March 2017,
Erlend

Figure B.1: Instructions to testers for testing IIR

C

IIR online manual

The following is a printout of the manual for the Incremental Information Retrieval system. It was created as an on-line help page in the IIR solution, so that it would always be available if the users (testers) had questions.

The testers were also encouraged to read the manual before starting test of a new query.

Manual

When searching the internet today we want immediate answers. We often search for a person, or a solution to a problem, or some topic we are interested in. The result quality off this kind of search is pretty good, most of the time we get the answers we need. The results, though, seem to be minor variations on the same results.

But what if the search for information is of a different nature, more like exploring. A typical case would be when a person has a hobby, and wants to search for information about it. Very soon all the quickly accessed information has already been seen, and is not that interesting in the context of new information.

IIR will help you ignore what you have already seen, and give you relevant results that you haven't seen before.

IIR

This application is a sort of search engine. When searching, it connects to a regular search engine, does the given search, and then manipulates the results coming out of that search engine. The goal of this application is to help the user to ignore results that have been seen before.

The application includes the following features.

- Shows results from the search in a standard search result card fashion.
- Duplicate results are automatically discarded.
- Results that have already been seen is automatically discarded.
- Results the user page by without interacting with, is marked as seen.
- The user can save results for later inspection.
- Results are ranked (sorted) according to how many of search words found.
- Results can be sorted by the search engine's original sort order.
- Always approve certain internet domain names (e.g. www.alwaysinteresting.com)
- Always suppress certain internet domain names (e.g. www.neverinteresting.com)

Terms

When IIR is used for searching, every result is saved so that the application can know if it has been seen before or not. That also means that each search needs to be kept separate from each other. So, there are three main terms that needs an explanation; Query, QueryRun, and Result.

Query
This acts as a container for a web search. The query contains the actual text, e.g. “apples”, sent to the search engine. If one wanted to search for “pears”, for instance, a new Query needs to be created.
QueryRun
Within each Query there is a set of QueryRuns. Every time a search is run against the search engine, a QueryRun is created. This is in our case once each day for a period of 54 days.
Result
Within each QueryRun there is a set of results. This is the actual response from the search engine. Every result has a title, an URL and a description. There can be up to 1000 results per QueryRun.

Result status

Each result can have one of several statuses.

New
When results come back from the search engine, they have the new status.
Seen
If a result is browsed past without doing anything to it, it is archived as seen.
Saved
The user can at any time save a result for further inspection.
Whitelisted
The user can approve a domain name (e.g. www.alwaysinteresting.com). All results from this domain will get the status whitelisted.
Blacklisted
If the user finds a domain name (e.g. www.neverinteresting.com) that never give relevant results for this Query, it can be blacklisted. Results from this domain will always be discarded, and will appear in the discarded folder.
Discarded
The user can at any time discard a result. This result will never again appear in a list of new results after being discarded.

How result retrieval works

When searching, events happen the following order:

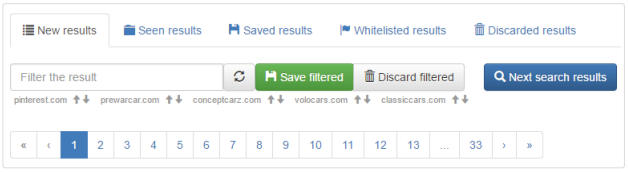
1. The search button is clicked
2. New search results are retrieved, and are checked and filtered:
    a. Duplicates are set to discarded – we only want one instance of each result
    b. If the user have seen a result before (i.e. status is either seen, saved, whitelisted or discarded), the result is discarded
    c. Results where the domain name is in the blacklist is automatically discarded.
    d. Results where the domain name is in the whitelist is automatically whitelisted.
    e. Results get a ranking score based on how many of the search words are found in title and description

Figure C.1: IIR on-line manual, page 1

3. After filtering, the remaining results are presented to the user in the *New results* folder sorted by descending rank (and then by title)

Folders and filters

The folders are a way of categorising the results by status. In the New results and Seen results folder there is a filter where the user can save or discard results. This can also be done for individual results.



Below the search filter is a row of domain names. When clicking on the domain name, the results will be filtered by this domain name.

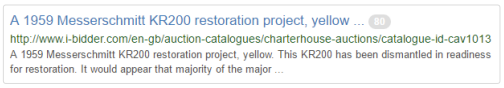
By clicking on the up-arrow, the domain name is whitelisted, and results will from now on automatically be moved to the *Whitelisted results* folder. By clicking on the down-arrow, the domain name is blacklisted, and all results from this domain will from now on automatically be discarded when searching.

When navigating to another page by clicking the paging buttons or searching for more results, the results that are now visible will get the seen status, and will henceforth be available in the Seen results folder. The seen mechanism only works in the *New results* folder, when showing other folders this mechanism is turned off.

Ranking

All results are given a simple rank, based on the search text and how it appears in the results. The result title and description of the result are analysed when read, and are given numeric values, which are summarised to givc the full rank for the result. These are:

50
The title of the result contains the exact phrase search for.
30
The description of the result contains the exact phrase search for.
15
If the exact phrase is not found in the title, the title is checked for all words appearing, though not in the exact order.
10
If the exact phrase is not found in the description, the description is checked for all words appearing, though not in the exact order.
1
If none of the above, each search word actually appearing in title or description is given one point.



The maximum rank is **80**, which means that the exact phrase searched for is found both in title and in description. Furthermore, a result without any points does not contain any of the words searched for (though the result might still be interesting).

Reset

At the bottom of the discarded results folder there is a button that will reset the current query. All results are removed, and the Query can be started again.

So...

All the user needs to do is

1. click the search button to make search results appear,
2. save or discard results, either by filter or just individual search results,
3. optionally white- or blacklist domain names, and
4. optionally page through to some other pages.

...then redo from the start for the next set of results.

Figure C.2: IIR on-line manual, page 2

D

Questionnaire

The questionnaire contains the statements that was scored, and the comment fields used by the testers for comments on IIR. See section 7.5 for answers to the questionnaire.

In order to show the questionnaire on a single page, some compacting was applied to it, see figure D.1.

Evaluation of Incremental Information Retrieval service

The Incremental Information Retrieval (IIR) service was built upon a static set of data, collected from 18th of November 2016 until 10th of January 2017. Please give your evaluation by responding to the statements below, and add comments about usefulness and possible improvements.

	1 Disagree completely	Agree completely 5
It was useful to be able to save results I found interesting	○	
It was useful to be able to discard results and never see them again	○	
It was useful to be able to whitelist domain names	○	
It was useful to be able to blacklist domain names	○	
This kind of application is useful to have, as a tool for searching	○	

In today's search engines we are not able to discard and save results. How did you like being able to do this in the IIR system?

2) * How did you like having a system where you can save and discard results?

Was there anything you felt was missing from the application? Something that would enhance the use of the application? Please add your comments below.

3) * Please add your comments on enhancements

IIR is not a fully implemented system. The full system would have the possibility of reformulating search queries, while keeping the seen/saved/discarded results, including blacklist and whitelist. How useful do you think such a system might be?

4) * How useful do you think a fully implemented IIR system might be?

Figure D.1: IIR questionnaire

E

Detailed test results

The following plots are detailed representations of how the users tested their queries. See section 7.6 for the summary of statistics. Each query has a corresponding detailed plot of test results. The x-axis shows days tested, and the number of days depends on how the user tested the query.

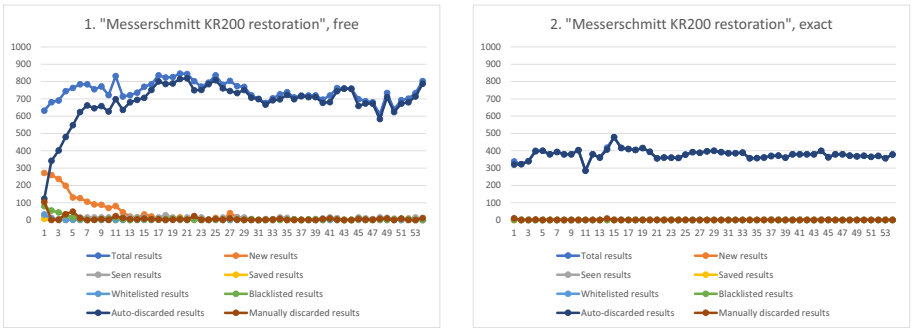


Figure E.1: Detailed test results for queries 1 and 2.

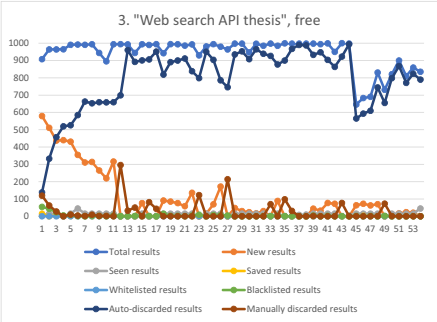


Figure E.2: Detailed test results for query 3, query 4 had no results.

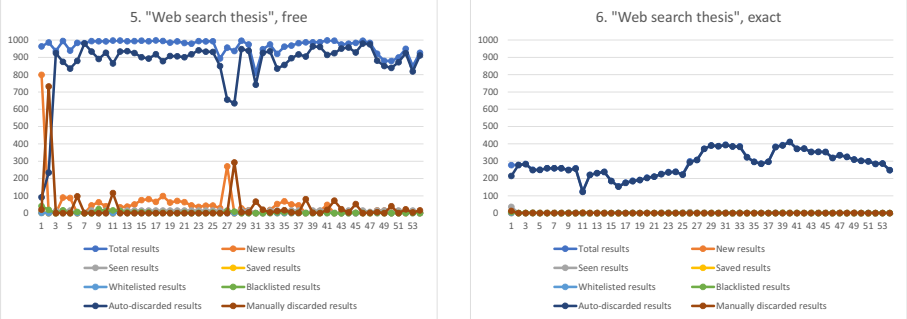


Figure E.3: Detailed test results for queries 5 and 6.

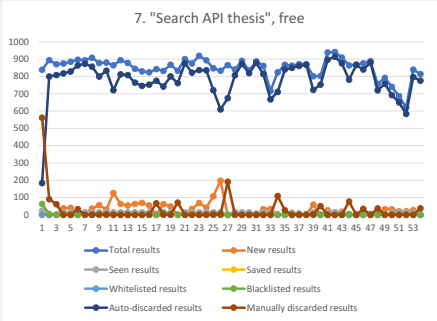


Figure E.4: Detailed test results for query 7, query 8 had no results.

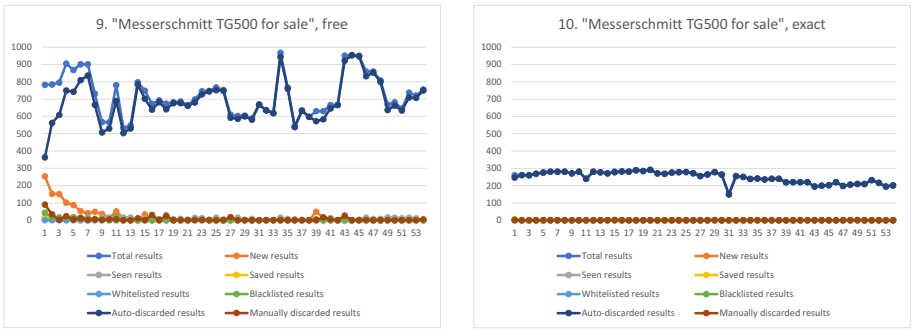


Figure E.5: Detailed test results for queries 9 and 10.

Figure E.6: Detailed test results for queries 11 and 12.

Figure E.7: Detailed test results for queries 13 and 14.

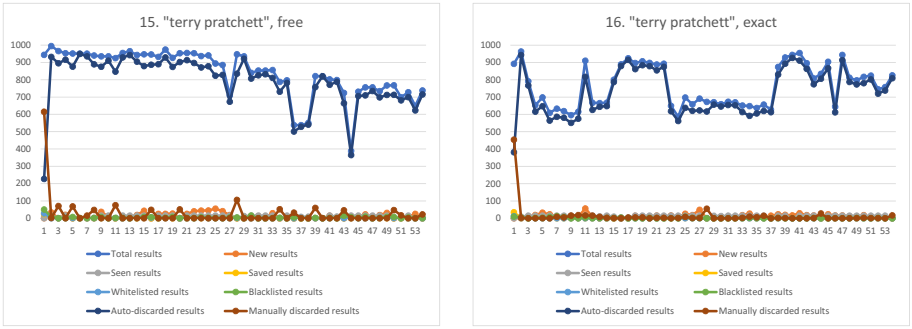


Figure E.8: Detailed test results for queries 15 and 16.

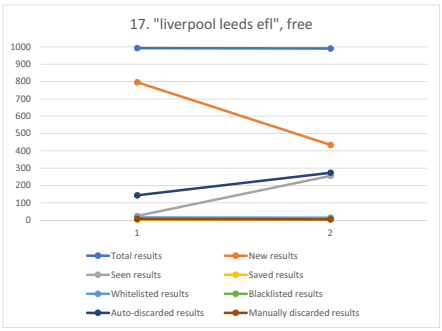


Figure E.9: Detailed test results for query 17, only 2 days of results examined by tester. Query 18 - 28 had only one day of results from testing.

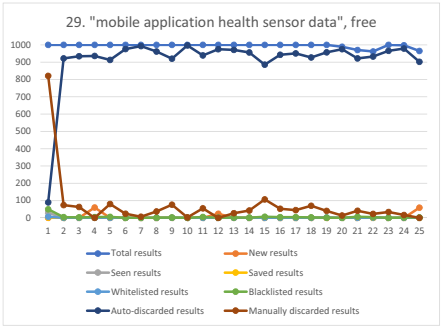


Figure E.10: Detailed test results for query 29, query 30 had no results.

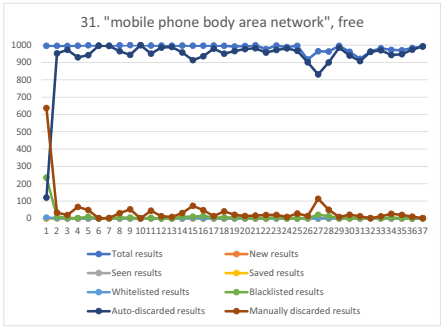


Figure E.11: Detailed test results for query 31, query 32 had no results.

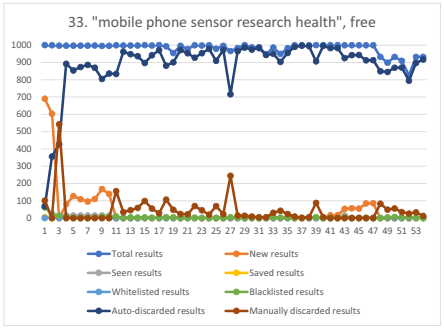


Figure E.12: Detailed test results for query 33, query 34 had no results.

Figure E.13: Detailed test results for queries 35 and 36.

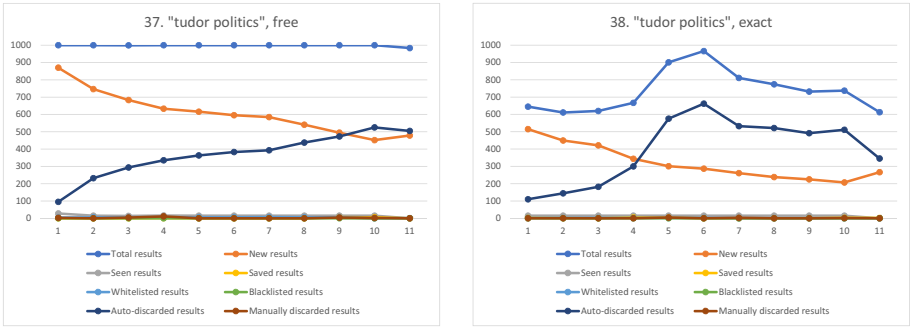


Figure E.14: Detailed test results for queries 37 and 38.

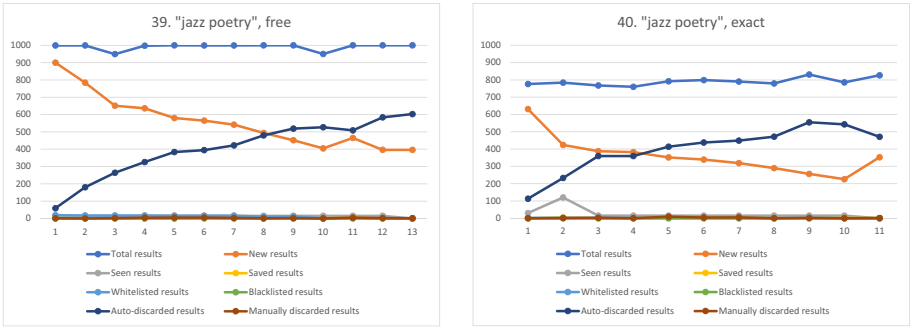


Figure E.15: Detailed test results for queries 39 and 40.

F

IIR code and utilities

This appendix contains a summary of all code used in the project. A command line interface (CLI) utility called countsource[1] has been used to count the number of source code lines. All command line utilities are written in Go[2], only the web service contains more diverse code, since it also contains web client code and images.

F.1 bingv2batch

This is the data collection application, that was used to collect data through Bing Search API. See also chapters 4 and 5.

filetype	#files	#lines	line%	size	size%
.go	17	1 004	90.7	30 184	90.7
.md	2	103	9.3	3 093	9.3
Total:	19	1 107	100.0	33 277	100.0

Listing F.1: Data collection application code.

1. https://github.com/borglefink/countsource
2. https://golang.org

F.2 bingv2analysis

This is a CLI utility made to analyse data from the data collection phase. It has produced most of the plots for this thesis, used in appendix A and chapter 5, among others. It has been created using the Go package *plot*[3]. This utility has also created miscellaneous LaTeX tables, based on the collected data, see appendix A.

```
------------------------------------------------------------------
filetype        #files        #lines  line%          size  size%
------------------------------------------------------------------
.go                  9         1 756  100.0        49 893  100.0
------------------------------------------------------------------
Total:               9         1 756  100.0        49 893  100.0
```

Listing F.2: Data collection analysis CLI.

F.3 bingv2analysispercent

This is a CLI utility very similar to bingv2analysis. The difference is that all plots created from this utility shows percentages, used in appendix A. Also, no LaTeX tables produced.

```
------------------------------------------------------------------
filetype        #files        #lines  line%          size  size%
------------------------------------------------------------------
.go                  8         1 549  100.0        42 878  100.0
------------------------------------------------------------------
Total:               8         1 549  100.0        42 878  100.0
```

Listing F.3: Data collection analysis CLI, to create percentage-wise plots.

3. github.com/gonum/plot

F.4 bingv2convert

Utility to prepare the data for the IIR web service. Used early and midway through the project. Abandoned in the end, when a different data usage strategy was chosen for the IIR web service, but was very useful for creating test data.

filetype	#files	#lines	line%	size	size%
.go	7	813	100.0	22 617	100.0
Total:	7	813	100.0	22 617	100.0

Listing F.4: Data collection conversion if collected data into IIR web service data.

F.5 iirweb

This is the actual web service.

filetype	#files	#lines	line%	size	size%
.go	22	1 660	30.1	46 932	22.9
.html	21	895	16.2	38 218	18.7
.js	13	2 031	36.8	61 182	29.9
.json	1	26	0.5	484	0.2
.scss	4	906	16.4	14 278	7.0
.gif	1			10 144	5.0
.png	3			31 929	15.6
Total:	65	5 518	100.0	204 735	100.0

Listing F.5: Summary of code used in IIR web service.

F.6 iiranalysis

Made to read test result data from the test database and create a CSV file, used for creating plots to show / analyse results.

filetype	#files	#lines	line%	size	size%
.go	11	685	100.0	18 955	100.0
Total:	11	685	100.0	18 955	100.0

Listing F.6: Test results analysis code.

F.7 Summary

The IIR project contains quite a lot of code, and there has been a substantial amount of coding work put into this thesis.

Code	Lines of code
bingv2batch	1 107
bingv2analysis	1 756
bingv2analysispercent	1 549
bingv2convert	813
iirweb	5 518
iiranalysis	685
Total	**11 428**

Table F.1: Summary of number of code lines

Side note:
At the start of the project, the web service was written with nodejs[4] including expressjs[5], handlebarsjs[6] and mongoosejs[7]. This nodejs based code actually adds to the server side code, but is not included here.

4. `https://nodejs.org`
5. `https://expressjs.com`
6. `https://handlebarsjs.com`
7. `https://mongoosejs.com`