Decentralized Social Data Sharing

by

Alan Davoust

A Thesis submitted to
the Faculty of Graduate Studies and Post-Doctoral Affairs
in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Ottawa-Carleton Institute for
Electrical and Computer Engineering
(OCIECE)

Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
2015

Copyright ©
2015 - Alan Davoust
Abstract

Many data sharing systems are open to arbitrary users on the Internet, who are independent and self-interested agents. Therefore, in addition to traditional design goals such as technical performance, data sharing systems should be designed to best support the strategic interactions of these agents.

Our research hypothesis is that designs that maximize the participants’ autonomy can produce useful data sharing systems.

We apply this design principle to both the system architecture and the functional design of a data sharing system, and study the resulting class of systems, which we call Decentralized Social Data Sharing ((DS)$^2$) systems.

We formally define this class of systems and provide a reference implementation and an example application: a distributed wiki system called P2Pedia. P2Pedia implements a decentralized collaboration model, where the users are not required to reach a consensus, and instead benefit from being exposed to multiple viewpoints. We demonstrate the value of this collaboration model through an extensive user study.

Allowing the users to autonomously control their data prevents the system architecture from being optimized for efficient query processing. We show that Regular Path Queries, a useful class of graph queries, can still be processed on the shared data: although in the worst case such queries are intractable, we propose a cost estimation technique to identify tractable queries from partial knowledge of the data.

Through simulation, we also show that the users’ control over network connections
allows them to self-organize and interact with other users with whom their interests are best aligned. This may result in less data being available, and we study cases where this is in fact demonstrably beneficial to the users, as the available data to each user is the most relevant to them.

This suggests that querying this reduced collection of shared data may lead to more tractable query processing without necessarily reducing the users’ utility.
Acknowledgments

First of all, I would like to acknowledge the valuable guidance of my thesis supervisor Dr. Babak Esfandiari, who helped me follow proper research methods while allowing me a significant amount of what I should call academic freedom. This freedom is one of the reasons why this thesis took so long, but I’m not sure I could have worked this hard on something I hadn’t largely chosen myself.

Dr. Esfandiari was also instrumental in connecting me with useful elements such as funding and valuable aspects of my academic training, including peer reviewing activities, a teaching contract, and an industrial internship.

I am indebted to Pr. Vincent Kazmierski of the Department of Law at Carleton University, who made possible the user study with P2Pedia. Pr. Kazmierski trusted us and fearlessly accepted to use our tool P2Pedia for academic writing tutorials in one of his courses, for two consecutive semesters, with close to 200 students each time, then participated in the data analysis.

My work also benefited from my collaboration with Drs. Hala Skaf, Pascal Molli and Khaled Aslan from the University of Nantes, France. While the results of that work only occupy a small place in this dissertation in terms of actual text, they greatly contributed to shaping my understanding of collaborative systems.

A number of undergraduate students also participated more or less directly to this research with their development work, including (in approximate chronological order) Alexander Craig, Denis Dionne, Matthew Smith, Cameron Blanchard, Jeremy
Dunsmore, and Ethan Aubuchon.

Finally, I could not have achieved this without the loving support of my wife Sylviana.
# Table of Contents

Abstract iii  

Acknowledgments v  

Table of Contents vii  

List of Tables xvi  

List of Figures xvii  

List of Acronyms xx  

1 Introduction 1  

1.1 Motivation ................................................. 1  
1.2 Research Problem .......................................... 5  
1.2.1 Definition .............................................. 6  
1.2.2 Efficiency: Data Availability and Query Processing .......... 7  
1.2.3 Self-Interested and Cooperative Behaviour .................. 9  
1.3 Contributions and Conclusions ................................ 12  
1.3.1 Contributions ............................................ 12  
1.3.2 Conclusions .............................................. 13  
1.4 Organization of the Thesis ................................. 14
2 State of the Art 16

2.1 Introduction ................................................. 16

2.2 The Web of Data ............................................ 18
  2.2.1 System Model ........................................... 18
  2.2.2 Functionality ........................................... 19
  2.2.3 Queries .................................................. 20

2.3 Online Social Networks ..................................... 21
  2.3.1 System Model ........................................... 21
  2.3.2 Functionality ........................................... 22
  2.3.3 Queries .................................................. 24

2.4 Collaborative Repositories ................................ 24
  2.4.1 System Model ........................................... 24
  2.4.2 Functionality ........................................... 25

2.5 Peer-to-peer Data Sharing Systems .......................... 26
  2.5.1 System Model ........................................... 27
  2.5.2 Functionality ........................................... 29
  2.5.3 Queries .................................................. 29

2.6 Comparison .................................................. 30
  2.6.1 Technical Costs ......................................... 30
  2.6.2 Conflicts over Shared Data .............................. 31

2.7 Conclusion .................................................. 36
  2.7.1 Summary ................................................ 36
  2.7.2 Decentralized Social Data Sharing (DS)² .................. 38

3 Decentralized Social Data Sharing Applications: Formal Definition 40

3.1 Introduction .................................................. 40

3.2 Conceptual Model of Social Data Sharing .................... 41
3.2.1 Social Network ........................................... 41
3.2.2 Data Resources and Graph of Data ......................... 41
3.2.3 Data Sharing .................................................. 43
3.2.4 System Model ............................................... 43
3.3 System Evolution ............................................. 44
  3.3.1 Modifying Operations ...................................... 44
  3.3.2 Deletion vs. Modification .................................. 46
  3.3.3 Queries ..................................................... 46
3.4 Model Refinement for a P2P Deployment ....................... 47
  3.4.1 Churn ...................................................... 48
  3.4.2 The Problem of Query Processing ......................... 49
  3.4.3 Query Semantics ......................................... 49
  3.4.4 Strategic Behaviour ....................................... 50
3.5 Conclusion ................................................... 50

4 P2Pedia, a (DS)² Wiki ......................................... 52
  4.1 Motivation .................................................... 52
  4.2 Related Work ............................................... 54
    4.2.1 Decentralized Content Management ...................... 54
    4.2.2 Decentralized Wikis ................................... 55
    4.2.3 Wikipedia Language Editions .......................... 56
  4.3 Functionality of P2Pedia .................................. 56
    4.3.1 Overview ................................................ 56
    4.3.2 Example Scenario ....................................... 58
    4.3.3 Use Cases ............................................... 59
    4.3.4 Regular Path Queries ................................... 62
  4.4 Trust Indicators in P2Pedia ................................ 63
5 A P2P Platform for social data-sharing applications

5.1 Introduction ........................................ 67
5.2 Related Work: Frameworks for Data Sharing Applications ........ 69
5.3 Background: REST .................................. 71
5.4 Layered Architecture ................................. 73
5.5 Base P2P Layer .................................. 74
  5.5.1 Layer Implementation ....................... 74
  5.5.2 Resources .................................. 75
5.6 Social Networking Layer .................................. 76
  5.6.1 Implementation ............................ 76
  5.6.2 Resources .................................. 76
5.7 Base Storage Layer .................................. 77
  5.7.1 Implementation ............................ 77
  5.7.2 Resources .................................. 78
5.8 Data-Sharing Layer .................................. 78
  5.8.1 Implementation ............................ 79
  5.8.2 REST resources .......................... 79
5.9 Structured Data Management Layer ..................... 81
  5.9.1 Data Model .................................. 81
  5.9.2 Modifying the Graph of Data ............... 81
  5.9.3 Queries .................................. 82
6 Processing Regular Path Queries in a (DS)²

6.1 Introduction ................................................. 84
6.2 Definitions and Algorithms for RPQs ........................ 88
   6.2.1 Notations .............................................. 88
   6.2.2 Definitions ............................................ 88
   6.2.3 Regular Path Query With Inverse ....................... 89
   6.2.4 Example ................................................ 90
   6.2.5 Basic RPQ Algorithm .................................... 91
   6.2.6 Application to RPQI .................................... 92
   6.2.7 Complexity ............................................. 92
   6.2.8 Optimizations .......................................... 93
6.3 RPQ Processing on Distributed Data ............................ 94
   6.3.1 Overview ............................................... 94
   6.3.2 Example Database and Query ............................ 97
   6.3.3 Query Processing Strategies for the (DS)² setting .... 98
   6.3.4 Complexity Comparison ................................ 103
   6.3.5 Conclusion ............................................. 106
6.4 Asynchronous Iterative Centralized Query Processing .......... 107
   6.4.1 Sequential Algorithm ................................... 108
   6.4.2 Example ................................................ 110
   6.4.3 Complexity ............................................. 111
   6.4.4 Parallel Algorithm ..................................... 112
6.5 Cost Comparison on Real-world Queries ......................... 118
   6.5.1 Dataset and queries .................................. 118
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4.2 Benefit of the activities for the students’ learning experience</td>
<td>151</td>
</tr>
<tr>
<td>7.4.3 Single vs. multiple solutions</td>
<td>152</td>
</tr>
<tr>
<td>7.5 Conclusion</td>
<td>152</td>
</tr>
<tr>
<td>8 Cooperative Behavior in a (DS)$^2$</td>
<td>154</td>
</tr>
<tr>
<td>8.1 Introduction</td>
<td>154</td>
</tr>
<tr>
<td>8.2 Participation in a (DS)$^2$ as a Prisoners’ Dilemma</td>
<td>157</td>
</tr>
<tr>
<td>8.2.1 The Prisoners’ Dilemma</td>
<td>157</td>
</tr>
<tr>
<td>8.2.2 Iterated PD Games and Social Networks</td>
<td>158</td>
</tr>
<tr>
<td>8.2.3 The Social Network PD Game</td>
<td>160</td>
</tr>
<tr>
<td>8.2.4 Connected Component Payoffs</td>
<td>164</td>
</tr>
<tr>
<td>8.2.5 Validation and Limitations</td>
<td>166</td>
</tr>
<tr>
<td>8.3 Data Relevance</td>
<td>167</td>
</tr>
<tr>
<td>8.3.1 Churn and Data Availability</td>
<td>167</td>
</tr>
<tr>
<td>8.3.2 Payoff models: the Value of Data</td>
<td>168</td>
</tr>
<tr>
<td>8.3.3 Introduction to the Case Studies</td>
<td>170</td>
</tr>
<tr>
<td>8.4 Data Availability in a Friend-to-Friend Network</td>
<td>172</td>
</tr>
<tr>
<td>8.4.1 Approach</td>
<td>173</td>
</tr>
<tr>
<td>8.4.2 Datasets</td>
<td>174</td>
</tr>
<tr>
<td>8.4.3 Reachable Peers and Available Data</td>
<td>175</td>
</tr>
<tr>
<td>8.4.4 Availability and Popularity</td>
<td>178</td>
</tr>
<tr>
<td>8.4.5 Assigning Value to the Available Data</td>
<td>180</td>
</tr>
<tr>
<td>8.4.6 Conclusions</td>
<td>182</td>
</tr>
<tr>
<td>8.5 Case Study: Cooperative Behaviour in Online Rating Systems</td>
<td>184</td>
</tr>
<tr>
<td>8.5.1 Introduction</td>
<td>184</td>
</tr>
<tr>
<td>8.5.2 Agreement Graph</td>
<td>186</td>
</tr>
<tr>
<td>8.5.3 Evaluation of the Agreement Graph</td>
<td>188</td>
</tr>
</tbody>
</table>
8.5.4 Social Network ............................................ 190
8.5.5 Similarity-based Filtering ................................. 191
8.5.6 Overall Comparison ...................................... 193

8.6 Conclusions .................................................. 195
  8.6.1 Cooperation in the PD game ............................. 195
  8.6.2 Impact of Churn ........................................... 195
  8.6.3 Agreement and Cooperation .............................. 197

9 Conclusions ................................................... 199
  9.1 Introduction ................................................ 199
  9.2 Contributions .............................................. 200
    9.2.1 A Blueprint for (DS)² applications .................... 200
    9.2.2 Decentralized Cooperation with P2Pedia .............. 201
    9.2.3 Efficiency of Query Answering ........................ 202
    9.2.4 Costs and Benefits of Decentralization ............... 204
  9.3 Implications for the Design of Data Sharing Systems .... 206
    9.3.1 Centralized Systems ................................... 206
    9.3.2 Distributed Systems ................................... 207
    9.3.3 Collaborative Repositories ............................ 208
  9.4 Limitations and Future Work ............................... 209
    9.4.1 Query Processing with Limited Cost ................... 209
    9.4.2 Directed Social Networks ............................... 210
    9.4.3 Economics of Trust Indicators ........................ 211
  9.5 Derived Publications and Tools ........................... 212
    9.5.1 Conference and Journal Articles ....................... 212
    9.5.2 Tools .................................................. 213

List of References ............................................. 214
Appendix A  DEVS Simulation of (DS)$^2$  230

A.1  Introduction .................................................. 230
A.2  DEVS Model Framework ...................................... 231
    A.2.1  Network Model ........................................... 231
    A.2.2  Peer Model .............................................. 232
    A.2.3  A Behavioural Model for Peers ....................... 234
A.3  Random Surfers in a P2P Web  ................................ 237
A.4  Conclusion .................................................... 238
    A.4.1  Reusing the Model ..................................... 239
# List of Tables

6.1 RPQ for $(DS)^2$ data: summary of asymptotic query processing costs. 112

6.2 Biological queries: regular expression, number of solutions (multi-source), number of valid start nodes 119

7.1 Evaluation of the sample documents by the class instructor and by automatic scoring functions. The letters A, B, ..., J identify the documents of the sample. 146

8.1 Comparison of different ratings filtering methods 194
List of Figures

1.1 Sharing graph data in a decentralized social data sharing system. . . 7
2.1 The Web of Data. A graph of data $G_R$ is statically shared by servers separate from the users that consume the data. . . . . . . . . . . . . 19
2.2 Social Media for sharing a web page. . . . . . . . . . . . . . . . . . . 21
2.3 Online social network. The users are related in a social graph $G_S$, and share a set of unstructured data resources $R$. . . . . . . . . . . . . . . 22
2.4 A collaborative data repository. The set of users $U$ is sharing a graph of data $G_R$. There are no individual associations between users and data resources. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
2.5 Peer-to-peer data sharing system. The users are not organized in a socially meaningful graph, but they have associations with the data that they share. The data is structured and may form a graph. . . . 28
2.6 The Decentralized Social Data Sharing model combines properties mainly of OSN and P2P networks to manage structured data. . . . . 37
3.1 A graph of data, made up of multiple resources. . . . . . . . . . . . . 42
3.2 Sharing graph data in a Decentralized Social Data Sharing system. . 43
3.3 System model with online peers and offline peers, available and un-available data. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
4.1 The different versioning processes available in a centralized and de-
centralized collaboration model produce different versioning hierarchy
structures. Each figure shows an example hierarchy of versions for a
single page. ................................. 57
4.2 Instantiation of the (DS)$^2$ conceptual model, for an example scenario. 59
4.3 Use Cases of P2Pedia .................................. 60
5.1 Layered Architectural Model for (DS)$^2$ applications. .................... 73
6.1 Localized vs. non-localized distributed graph of data. .................... 86
6.2 An example graph of data. .................................. 91
6.3 A query automaton. .................................. 91
6.4 A graph of data arbitrarily distributed and replicated between 3 sites. 97
6.5 NFA for regular expression $a^*bb$. .................................. 110
6.6 Amount of data to be transferred by broadcast /unicast for the evalu-
ation of the biological queries. .................................. 122
6.7 Optimality of S1 and S4 depending on $k$, $d$ and the query-dependent
discriminating function. ................................. 125
6.8 Tail Distributions for different queries: true distribution, and estimates
based on the two types of statistical graph models. ........................ 131
7.1 P2Pedia screenshot: “Diff” highlighting of the changes between two
versions. .................................. 140
7.2 Frequency of the different atomic edits .................................. 143
7.3 Pairwise similarity of documents: CDF. The dashed line shows the
similarity of edits excluding their “included” edits (as discussed in sec-
tion 7.2.2), and the continuous line shows the similarity of edits after
the included edits were added. .................................. 144
7.4 Similarity of the edits on the end of the documents, plotted as a func-
tion of the similarity of the edits on the beginning of the documents. 150
8.1 Social Network PD game simulation: mean node degree, by strategy: simulation with 50 nodes using each strategy. 162
8.2 Social Network PD game simulation: size of largest connected component. Simulation with 100 nodes implementing each strategy. 163
8.3 Social Network PD game simulation: mean scores of players, by strategy. (100 players per strategy) 164
8.5 Distributions of the relative sizes of the reachable network and the available resource set, for online peers. 176
8.6 Probability of a peer being in the giant connected component, depending on the peers’ degree in the social network (number of friends). 177
8.7 Probability that a resource $r$ is available to an arbitrary user $u$, depending on the popularity (replication) of $r$, and on the users’ number of connections in the social network. 179
8.8 Value of the data available in the network, relative to total value of all resources (CDF). 181
8.9 Node degree distribution in the graph of agreeing and disagreeing users. 187
8.10 Evaluation of the agreement graph as a ratings filter, against Yelp’s filter. The top plot indicates the number of users in the GCC, and its y scale is on the left axis. The other plots indicate the MAE for different filtering criteria, and their scale is on the right axis. 189
8.11 Performance of K-NN vs. average rating. Threshold and $k$ are varied, $\text{minsupport} = 2$ fixed. If $k$ neighbours are not available for prediction, the algorithm falls back to average rating. 192
A.1 DEVS Model Framework, showing the structure and interconnections of the network and peer models. 231
A.2 A peer’s stateful behaviour. 236
A.3 Replication of Web Pages in a P2P Web Surfing Application. 238
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BFS</td>
<td>Breadth-First Search (graph search algorithm)</td>
</tr>
<tr>
<td>CCI</td>
<td>Causality, Consistency, and Intention preservation</td>
</tr>
<tr>
<td>CDF</td>
<td>Cumulative Distribution Function</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>DEVS</td>
<td>Discrete-EVent Simulation</td>
</tr>
<tr>
<td>DFS</td>
<td>Depth-First Search</td>
</tr>
<tr>
<td>DHT</td>
<td>Distributed Hash Table</td>
</tr>
<tr>
<td>DVCS</td>
<td>Decentralized Version Control System</td>
</tr>
<tr>
<td>F2F</td>
<td>Friend to Friend</td>
</tr>
<tr>
<td>FOAF</td>
<td>Friend Of A Friend (a web ontology)</td>
</tr>
<tr>
<td>GCC</td>
<td>Giant Connected Component</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IR</td>
<td>Information Retrieval</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>k-NN</td>
<td>k Nearest Neighbours</td>
</tr>
<tr>
<td>MAE</td>
<td>Mean Absolute Error</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translators</td>
</tr>
<tr>
<td>NFA</td>
<td>Non-deterministic Finite Automaton</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OSN</td>
<td>Online Social Network</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-To-Peer</td>
</tr>
<tr>
<td>PA</td>
<td>Product Automaton</td>
</tr>
<tr>
<td>PAA</td>
<td>Product Automaton Algorithm</td>
</tr>
<tr>
<td>PD</td>
<td>Prisoners’ Dilemma</td>
</tr>
<tr>
<td>PDMS</td>
<td>Peer Data Management System</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>REST</td>
<td>REpresentational State Transfer</td>
</tr>
<tr>
<td>RPQ</td>
<td>Regular Path Query</td>
</tr>
<tr>
<td>RPQI</td>
<td>Regular Path Query with Inverse</td>
</tr>
<tr>
<td>SFW</td>
<td>Smallest Federated Wiki</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL Protocol and RDF Query Language</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

Data sharing systems are a general class of networked information systems, where users can store data and make it accessible for others to retrieve through queries.

A few prominent examples of data sharing systems include online social networks (OSN), peer-to-peer (P2P) file-sharing networks, P2P data management systems, and collaborative repositories such as Wikipedia or Wikidata [1], a more recent Wikimedia Foundation project intended to complement Wikipedia with structured data.

These systems implement different architectures, which can be evaluated with respect to different non-functional requirements, such as their efficiency, scalability, or reliability. Accordingly, most research relevant to their design has focused on optimizing these different properties.

However, the technical performance of a data sharing system alone does not guarantee that the system is useful and viable.

Data sharing systems should also be seen as multi-agent systems, where the users participate in order to fulfill their own goals, and in doing so may have beneficial or detrimental interactions with other users. When the participants’ interests are closely aligned, these interactions can be expected to be mutually beneficial. On the other
hand, in the open context of the Internet, where the participants are independent
and self-interested\(^1\) agents, there is no guarantee that their interests are aligned.

The functional design of a data sharing system defines the mechanisms of the
users’ interactions, and therefore plays an important role in determining how users
with conflicting interests can affect each others’ ability to fulfill their goals. Accom-
modating conflicting interests among the participants is therefore an important design
goal for open data sharing systems.

It is also important to note that this concern is not separable from the architectural
design of the system. Architectural design choices are intended to optimize specific
properties of the system, and often involve trade-offs with the participants’ \textit{autonomy}.

For example, structured P2P networks (e.g. Chord [2]) optimize communication
costs and data availability by algorithmically determining the network structure and
allocating the shared data to the different participants. This implies that participants
must store data from arbitrary other peers, and cannot choose which other peers they
are connected to.

Depending on each participant’s interests and goals, the interactions imposed by
the algorithm can be either beneficial, in which case it is in their interest to implement
the algorithm, or detrimental, in which case their interest is to “cheat” and circumvent
the constraints imposed by the algorithm, or else to leave the system altogether.

A notorious example of users circumventing algorithms occurred with the P2P file-
sharing system Kazaa, which had a simple reputation mechanism that determined a
user’s priority in downloading queues. Each client managed its own reputation, and
soon enough this code was reverse-engineered, and a modified version of the P2P
software was distributed, allowing users to set their own reputation [3].

In much existing research, this problem is treated as a \textit{security} problem, in the

\(^1\)in the economic sense of having their own objectives in participating
sense of ensuring that participants implement the prescribed algorithms and identifying the “malicious” participants that don’t (e.g. [4–7]).

While the malicious/honest dichotomy is justifiable in some contexts (e.g. denial of service attacks), there are other situations where different participants each have legitimate motivations in using the system, but their interaction as prescribed by the system is detrimental to one or several of them.

For example, participants of a distributed storage system such as Oceanstore [8] may see their data algorithmically allocated to arbitrary participants in arbitrary geographical locations, which may cause legitimate concerns since this location determines which laws are applicable to their data [9].

It would be desirable to allow the participants to autonomously choose which other participants they cooperate with, but this would typically interfere with optimization algorithms and reduce their performance.

This problem extends to other aspects of data sharing systems, such as the coordination processes to manage the shared data in collaboratively curated repositories.

Different users may have competing views over what data should be stored in the repository, for example what amount of pornographic material should be allowed in the Wikimedia Commons repository [10], what topics should be the subject of Wikipedia articles, what facts should be claimed as “true” in Wikidata [1], etc. These disagreements are resolved through complex processes, that essentially amount to algorithms taking the different participants’ preferences as inputs, and producing a decision as output [11, 12]. In other words, a social choice [13].

These coordination processes are costly (e.g. in 2007 40% of edits in Wikipedia were found to be indirect work such as developing procedures and building consensus, and the proportion was increasing [11]) and their outcome is arguably not optimal for every participant. Existing collaborative systems are largely centralized, and rely on the central authority’s administrative control of the infrastructure to enforce the
processes and decisions of these processes. While the coordination processes can be implemented in a distributed setting (e.g. there are many distributed wikis [14, 15]), it is an open question as to how they might be applied in a decentralized environment where the participants are self-interested agents, i.e. not necessarily obedient, but rather strategic [16].

In fact, reaching a decision through a distributed algorithm, in a context where not all the participants implement the prescribed algorithm (due to faults or to strategic decisions not to cooperate) is essentially the Byzantine Generals problem [17], for which it is known that a majority of two-thirds of cooperative participants is needed for these participants to be able to reach their decision (which does not even address the problem of enforcing the decision).

This motivates us to explore designs where the need for cooperation between participants is kept to a minimum. In particular, we study a design where the peers autonomously manage their own data, and autonomously choose which other peers they will interact with.

This essentially amounts to a “friend-to-friend” (F2F) peer-to-peer network [18, 19] (i.e. a file-sharing network where the participants only connect to their trusted acquaintances), or a Peer Data Management System (PDMS) such as Hyperion [20] or Piazza [21], where database management systems are interconnected in an ad-hoc network.

As such, it is not an entirely new design proposal. However, F2F networks are not designed to handle structured data, and have received limited academic attention. PDMS have primarily been studied in scenarios where institutions (e.g. universities or scientific institutions) wish to share their data to mutual benefit, and as such have arguably addressed primarily the problem of data integration [22–24], i.e. how their data can be queried despite schema heterogeneity, as if it were all integrated in one large repository. Handling conflicting interests between the participants is in
this case limited to automatically resolving inconsistencies across repositories [25,26]. Therefore, the adequacy of such systems to support autonomous and self-interested participants has not really been evaluated.

1.2 Research Problem

In this thesis, we address the general problem of designing a data sharing system to accommodate the participation of autonomous and self-interested users.

In this context, we formulate the hypothesis that applying design choices that maximize the users’ autonomy can produce useful and viable systems for sharing structured data.

In order to validate this hypothesis, we address the following sub-questions:

• **Definition**: How can the principle of maximizing the users’ autonomy be applied to the design of a data sharing system?

• **Efficiency**: Can expressive queries be efficiently processed on the data shared in such a system?

• **Data Availability**: If some of the data is unavailable to query, what is the cost of this problem?

• **Usefulness**: Can useful applications be built according to this model?

• **Cooperation**: What are the users’ incentives in such a system? How can the users cooperate when they have conflicting interests?

Our answers to these questions are presented in detail in the following subsections. We note that in addition to the above issues, several security issues are left outside the scope of this thesis: these include user identity management (in particular authentication), as well as privacy. These issues are primarily of interest to lower-level
protocols, which are not addressed by our design.

1.2.1 Definition

Our first main research question is the following:

How can the principle of maximizing the users’ autonomy be applied to the design of a data sharing system?

We define and characterize a general design for sharing structured data in a peer-to-peer network, obtained by applying the following design choices:

- the data-sharing system should be physically decentralized, i.e. it should have a peer-to-peer (P2P) architecture, with fully independent peers.
- in this P2P architecture, the participants should be able to freely choose which other peers they connect to and cooperate with: we represent this as an explicit social network of participants.
- the contributor of a data resource\(^2\) should retain control over this data resource, in the sense that other users cannot delete it. For this purpose, the participants store their own data locally.

This design, which we call the (DS)\(^2\) model, for Decentralized Social Data Sharing, is illustrated in figure 1.1.

The users are organized in a social graph. The connections between users are a first-class concept in this model, since modifying these connections is one of the main affordances available to the users.

The data is potentially structured, which we express by representing the data as a graph, where the resources are connected by relationships. We do not commit to

\(^2\)In this thesis, we will use the term data resource to designate an atomic piece of digital information, that the users can contribute, delete, or query for. This can be a file, an RDF triple, etc. We do not commit at this level to any particular data model.
a specific data model such as the relational model or RDF, as these are technical details of the storage solution, and are irrelevant at this level of abstraction. Instead, we simply consider that any data resource (e.g. a tuple or an RDF triple) is potentially related to other neighbouring resources, stored by the same peer or by others.

The relationships between the users and the data resources express the notion that users are in control of specific resources, as discussed above.

1.2.2 Efficiency: Data Availability and Query Processing

One the main functional requirements of a data sharing system is to allow the users to access the shared data through queries. In our proposed model, the users can freely choose their connections and host their own data, with the following implications:

- The network topology is unknown;
- The location of the data in the network depends on which users choose to share it;
- If the peers sharing the data are unreachable or offline, then the data is unavailable altogether.

In other words, the network topology is not guaranteed to be connected, and data resources may be found in zero to many arbitrary locations. This has problematic
consequences for the users’ access to data.

A first consequence is that part of the data may be unreachable (due to the network not being connected).

The second consequence is that it is particularly challenging to process queries over whatever data is available: in order to process a query over the distributed graph of data, the necessary data must be either collected locally, or else the query must be decomposed into appropriate subqueries, and these subqueries sent to the locations of the data. If these locations are arbitrary and unknown, the message cost of distributing the queries and collecting the data may easily become prohibitively expensive.

This leads us to the following questions:

- Can any useful (classes of) queries be processed on this data? How?

- Can we quantify the cost (i.e. the utility loss) due to some data being unavailable?

**Processing Expressive Queries**

In order to answer the first of these questions, we studied in chapter 6 a simple but powerful class of graph queries, namely *single-source Regular Path Queries (RPQ)*, which can be used to implement several functionalities of our example application P2Pedia.

We showed that RPQ can be executed at a reasonable cost, in some cases, on data shared in a (DS)². We proposed a query processing technique that performs well for more *selective* queries, in cases where baseline techniques can be prohibitively costly, and proposed novel techniques to evaluate the selectivity of queries, based on statistical models of the graph of data.
We evaluated our techniques on a dataset and meaningful queries from the biomedical domain, that had previously been used to study the processing of RPQ.

**Data Availability**

In order to answer the second question above, we simulated the effects of *churn* (the intermittent presence of peers in a P2P network) on a network made up of social connections. This study is detailed in section 8.4, and is based on data from the online music sharing site Last.fm, combined with availability traces from the Overnet P2P network [27].

Our experiments show that churn may cause peers with few social connections to be isolated, but is unlikely to break up the network significantly. We then evaluate the impact on the users’ *utility*, by associating value to the available data in several ways. This impact is significantly reduced when the data resources are assumed to have different values, subjective or objective.

### 1.2.3 Self-Interested and Cooperative Behaviour

As we have discussed above, the (DS)$^2$ model is designed to require minimal cooperation from the participants, motivated by the realization that cooperative behaviours are difficult to *enforce* among independent and self-interest peers.

So if there is no way to enforce cooperative behaviour, can data sharing systems still be useful? The rationale of not restricting the users’ autonomy is that it will increase their utility, in an economic sense, by allowing them to act exclusively in their own interest.

However, if the other participants also act in their own interest, then will the users actually derive any benefit from participating in the system?

More precisely, we ask the following questions:

---

3[^1]: [http://www.last.fm](http://www.last.fm)
• How do the participants of a (DS)\(^2\) system benefit from their participation?

• Could we build collaborative systems on this model? Would such applications be useful?

• What are the key features of the (DS)\(^2\) system that would make such a system useful?

We addressed these questions by an in-depth study of an example application, and a simulation-based study exploring several economic views of the users’ utility in participating in a (DS)\(^2\).

**Case Study: P2Pedia**

In order to understand the benefits of this model, we implemented and evaluated an example application, called P2Pedia (presented in chapter 4), obtained by applying the (DS)\(^2\) design to the concept of a wiki system. This unconventional distributed wiki allows the different users’ contributions to be shared and co-exist in the system, without the requirement of a consensus.

We evaluated P2Pedia and its collaboration model through a user study, presented in chapter 7. In this study, P2Pedia was used to support academic writing exercises, for tutorial sessions of an early-year undergraduate course in the Department of Law at Carleton University. We analyzed the students’ contributions and collected their feedback through an online survey.

The students were able to share their work, and were exposed to alternative solutions to the given problem. This successful user study partially answered the above questions, for the specific case of P2Pedia:

• This demonstrates that valuable collaborative applications can be built using our decentralized interaction model;
• The participants benefit by being exposed to multiple points of view, which is difficult in a traditional wiki [28];

• The key features of P2Pedia of value in this study were:
  
  – The ability for each participant to maintain their own version of the work product,
  
  – The ability to search and view the other available versions in the network.

**Economic Simulation of (DS)$^2$ Systems**

In order to acquire a more general perspective of the value of the (DS)$^2$ model, we simulated the (DS)$^2$ model as an economic game. This study is presented in chapter 8.

We first represented the participation in a (DS)$^2$ system as a classic Prisoners’ Dilemma game, or more specifically an iterated $n$-player version of the game, where the players are organized in a social network and can strategically modify their connections. We simulated this game using the NetLogo simulation tool [29], and found that the users’ control over their network connections was a key feature to foster collaborative behaviour by isolating uncooperative users, confirming similar findings by evolutionary biologists [30, 31].

We then studied the case where the users’ payoff (the benefit of participating) depended on the possibly subjective value of the data that they had access to. This is the case for example in an online rating system, where the available ratings can help the user choose between available options (e.g. restaurants listed on Yelp). Using a dataset of Yelp ratings, we simulated the result of the users strategically managing their connections, and showed that the data available from the resulting social network was significantly more valuable that the full set of data, in the sense that it better predicted each user’s own ratings. We also showed that a comparable performance improvement was found for Yelp’s existing social network (i.e. the existing
“friend” connections). In addition, we found that these “social” filtering mechanisms significantly outperformed Yelp’s own filter.

These analyses provide additional answers to our research questions:

- The participants benefit from the other users’ contributions, but will also indirectly benefit from their own contributions, as these contributions can represent a cooperative behaviour that will lead other users to include them in their social network;

- This shows that a wider range of (DS)$^2$ applications can be useful, such as a system to share ratings, i.e. a decentralized version of Yelp.

- This study demonstrates that the users’ direct control over the social network is a key in providing value to the users.

1.3 Contributions and Conclusions

1.3.1 Contributions

The main contributions of this research are the following:

- We identify and formally define the (DS)$^2$ model, a useful design for data sharing systems, with a reference implementation and an example application;

- We present a novel distributed wiki application implementing the (DS)$^2$ model, and demonstrate its value through an extensive user study and an in-depth analysis of the data contributed by the participants;

- For a class of expressive queries over the shared data, we evaluate the complexity of several distributed query processing algorithms, and propose a new technique to evaluate the actual cost of queries with partial knowledge of the data;
• We evaluate the (DS)$^2$ model in an economic perspective, and contrast our findings with a purely performance-oriented evaluation in terms of available data. Our results show that accessing all of the shared data is not necessarily optimal, and that the users’ control over their social connections is a key feature that users can leverage to increase their utility in using a (DS)$^2$ system.

1.3.2 Conclusions

These results generally validate our hypothesis that maximizing the users’ autonomy can produce useful data sharing systems for structured data.

The users benefit from the (DS)$^2$ model in two ways:

• by not being required to reach a consensus on the shared data, they do not need to compromise their autonomy but can still benefit from the points of view expressed by other users’ contributions,

• their control over the network allows them to self-organize and interact with other users with whom their interests are best aligned, which in many cases will be the contributors of relevant data, in their subjective assessment.

The main downside of this model is the cost of expressive queries over the data, which are in general intractable with existing techniques. However, we have demonstrated that this problem is mitigated in three ways:

• for one class of expressive queries, we provide a technique to identify which queries may be tractable,

• our query processing technique can be interrupted anytime, providing incomplete query answers,

• the fact that only a subset of the users may be cooperative implies that from an economic perspective, it may not be optimal to collect answers from the full
graph of data, as some of it may be contributed by less trustworthy participants: in other words, incomplete query results are not necessarily sub-optimal.

These results highlight the importance of mechanisms of social or trust-based connections.

1.4 Organization of the Thesis

In Chapter 2, we survey the main classes of data sharing systems, and provide a schematic view of their architecture and functionality.

These data sharing systems include the emerging Web of Data, viewed as a single complex system, then Online Social Networks, P2P data sharing systems, and Collaborative Repositories, i.e. wikis and wiki-like systems.

We defer related work on the topics of distributed wikis and the distributed processing of RPQs to chapters 4 and 6, respectively, where they appear more relevant.

In Chapter 3, we formally introduce and define the class of Decentralized Social Data Sharing systems. Such a system consists of a social network of users, who share and query a graph of data.

As an example, we apply this design to the concept of a wiki, and obtain P2Pedia, the decentralized wiki system discussed above. We describe the functionality of this application in Chapter 4.

In Chapter 5 we present an implemented platform that allows various such applications to be built, by exposing the data-sharing functionalities through an API following the REST architectural style.

In Chapter 6, we describe the processing of Regular Path Queries (RPQ) on data shared in a \((DS)^2\) system, and show how the cost of processing a particular query can be estimated.
In Chapter 7, we describe our user study of P2Pedia, and in Chapter 8, we study the abstract problem of cooperation in a (DS)$^2$ system through simulation.

We conclude this thesis with a discussion of the main results, limitations and perspectives of future work in chapter 9.
Chapter 2

State of the Art

2.1 Introduction

In this chapter, we review existing classes of data-sharing systems. We model their logical structure, describe their functionality, then compare how these different classes of systems can facilitate the interaction of self-interested agents.

The basic functional requirements of a data sharing system are to store the data that the users contribute, and provide access to this data through queries. From this definition, we can identify two main aspects that distinguish classes of systems at a functional level.

The first is the distribution of control over the shared data: once data is published, who has control over this data, i.e. which stakeholders can modify it, delete it, or prevent its deletion? From an agent-oriented perspective, users may have conflicting preferences regarding the collection of shared data (e.g. the contents of a wiki-encyclopedia), and this aspect is therefore key in determining how such conflicts can be resolved.

The second is the logical structure of the data sharing system, i.e. the associations between users and data resources (or lack thereof). The possible associations included in our models are the following:
• associations between users: typically, these associations constitute the “social
  network” of online social networks.

• associations between users and data resources: these are typically “ownership”
  associations, but may include other types of associations;

• associations between data resources: these distinguish unrelated resources (such
  as files) from structured data, where the structure is meaningful and accessible
  to queries.

When a user wants to query the available data, the logical structure of the model
provides leverage for the users to better choose which data is returned by a query:
they can either directly query the logical structure (e.g., “which data does user X
own?”) or modify the structure to influence query results (e.g. by adding or removing
P2P connections, they modify the set of data resources that will be queried).

In the rest of this chapter, we briefly present four existing general data sharing
paradigms that the (DS)$^2$ idea borrows from, then discuss and compare their proper-
ties.

In section 2.2 we present the Web of Data, a developing component of the In-
ternet, where Web servers publish structured data following Web and Semantic Web
standards (HTTP, RDF, SPARQL).

In section 2.3 we discuss the general class of Online Social Networks (OSN), such
as Facebook or Flickr. OSN are services where users publish content to personal
accounts, and establish explicit connections with other users [33].

The third class of interest, presented in section 2.4, is that of collaborative data
management systems. These systems include wikis and wiki-like systems, where users
interact with a single collection of data over which they have a priori equal access
permissions.

In section 2.5, we will discuss peer-to-peer (P2P) data sharing systems, networks
of autonomous peers that share files (file-sharing networks, e.g. Gnutella, eMule) or structured data (P2P database systems).

For each class of systems, we provide a general system model, schematically representing the logical structure of the data sharing system, and detail the functionality available to the users, including the types of queries that these systems typically support.

In section 2.6, we summarize and compare the relevant properties of these different systems. Finally, we conclude in section 2.7.

2.2 The Web of Data

As a starting point of this review, it seems appropriate to present the general data-sharing system constituted by the Web itself. We will focus on the emerging Web of Data, designed to provide data to applications, rather than human users.

The Web of Data is a global collection of structured data published following the principles of Linked Data [34,35]. These principles include a set of standards for the protocol, data model, and query language (HTTP, XML, RDF, SPARQL), as well as guidelines to connect data across independent repositories.

2.2.1 System Model

Data is published on the Web of Data using the RDF format, and each RDF file published by a server contains one or several RDF triples. These triples are the edges of a graph, where the nodes and edges are identified by URIs (Uniform Resource Identifiers). These URIs may reference other servers, and dereferencing them results in the servers returning other data resources1.

1The URIs may not in fact reference these data resources directly, but instead some non-information resources: a request for the non-information resource typically redirects to another graph of data on the same server.
We represent these resources and links by a graph $G_R$: the nodes of this graph are the data resources $R$ (the RDF files, or any subset that can be identified and retrieved independently), and an edge connects two resources $(r_1, r_2)$ if a URI in $r_1$ dereferences to $r_2$.

The data is published on Web Servers $S$, i.e. fixed network locations, and is identified using HTTP URIs. This identification by location implies that each data resource is related to a single server. Web servers are not linked to each other: although URIs in their data may constitute links, these links cannot be managed independently of the published data, and therefore we consider that they should not be represented separately either.

The data is accessed by a set of users $U$, disjoint from $S$.

The resulting model is illustrated in figure 2.1.

2.2.2 Functionality

The functionality available to the servers $S$ is the following:

\footnote{In practice, data providers can also be consumers in the sense that they are real people who may go home and use their web browser, but those are different roles that should conceptually be associated with different identities.}
• publish \((r: \text{Data Resource})\): The data resource \(r\) is published by a specific server.

• remove \((r: \text{Data Resource})\): A server removes a data resource \(r\) that it has published.

The functionality available to the data consumers \(U\) is reduced to querying the shared data:

• query \((q: \text{Query}, s: \text{source})\): Submit a query \(q\) to data source \(s\). The server will return local data matching the query.

### 2.2.3 Queries

The standard query language for the Web of Data is SPARQL, and applies to the graph of RDF data included within the data resources \(R\). Typically, servers offer an interface that supports queries over all of their local RDF data, i.e. the union of their local resources.

The Semantic Web envisions agents that are able to query multiple data sources (servers), which involves decomposing each query into subqueries, that are then sent to individual servers. The servers then return the resources matching the query, and the querying agent computes the query locally from the union of these resources.

Query agents are also expected to be able to discover and dynamically handle new data sources during the query processing, which amounts to following links between resources \([36, 37]\).

Semantic Web search engines (Swoogle \([38]\) and Sindice \([39]\)) even “crawl” the full Semantic Web this way, independently of any queries. They either build a local copy of the collection, over which queries can be processed, or else they build an index, which can then later be used to identify relevant sources for a given query.
2.3 Online Social Networks

Figure 2.2: Social Media for sharing a web page.

The past decade has seen much of our electronic communication patterns shift towards so-called social media. Social Media, such as those shown in the screenshot of Figure 2.2, are web-based services where users store data in personal accounts, and explicitly declare social relationships with other users of the same service. They are therefore also called Online Social Networks (OSN).

Different services encourage users to express different types of relationships, e.g. friendship (Facebook), trust (epinions.com), or a simple subscription to the user’s published information (twitter).

2.3.1 System Model

The users are organized in a social graph. This graph may be directed or undirected, and the users may in some cases declare several types of relationships, which can be modelled by labelled edges. Generally, we will refer to a users’ neighbours as “friends”. They may also sometimes form higher-order associations such as groups.

OSN users publish specific data resources to their accounts, over which they have exclusive control (or quasi-exclusive: system administrators also have control over all accounts, as we discuss below).
The users therefore have an ownership association with the data resources published to their account, but OSN can also include other associations, such as Facebook’s “like”. With these other associations, several users can be associated with the same data resource: for example, multiple users may have “liked” the same picture on Facebook, or “re-tweeted” a Twitter “tweet”.

As the data shared in OSN is typically unstructured [33], we model it as a set of unrelated objects.

The resulting model is represented in figure 2.3.

Figure 2.3: Online social network. The users are related in a social graph $G_S$, and share a set of unstructured data resources $R$.

### 2.3.2 Functionality

OSN users have access to functions to manage their data, and to manage their social network:

- **publish** ($r$: Data resource): add the data resource $r$ to the set of shared data. A link is implicitly added between the user and the data, representing an ownership association.

- **delete** ($r$: Data resource): remove existing data resource $r$, if the user owns $r$. 
• linkToData(r: Data resource, a: label): establish a link to existing data resource r, with label a. Example: “like” on Facebook. Typically, users cannot create an ownership link to data owned by another user\(^3\).

• unlinkToData(r: Data resource, a: label): remove a link to existing data resource r (e.g., “unlike”).

• linkToUser(u: User): establish a social link to user u. In undirected social graphs, establishing links typically requires a two step process, where the link is requested, then accepted.

• unlinkToUser(u: User): remove existing social link to user u.

• query(q: Query): Submit query q to the system. The system will return the available data matching q, or a subset thereof\(^4\). The set of possible queries is limited by the user interface.

An important difference with the Web of Data, is that the end users of the service are both data providers and consumers. This information can be used to infer the users’ preferences and personalize the data resources displayed to the user (e.g., as a result of queries), according to some notion of relevance (see section 2.6.2).

It is also worth mentioning that in addition to the users represented in the model, most OSN are managed by a centralized authority: the service provider. This authority constitutes an additional actor that has access to additional functions: a delete function over any data, without any pre-conditions of data ownership, and a function to remove undesirable users as well.

\(^3\)There are a few exceptions; in particular, source code forges (e.g. Github) often allow code to be “cloned” from one account to another, and the new copy is under full control of the user who cloned it. Technically, content could be copied and “re-posted” (often in violation of OSN terms and conditions), but this is irrelevant if the system does not recognize the two copies as the “same”. Note that on twitter, “tweeting” is an ownership association, whereas “re-tweeting” is not: the originator of a tweet can delete it, and every re-tweet will be deleted as well.

\(^4\)The user interface may only display the top-k results, or some results may be left out due to access control policies, for example.
2.3.3 Queries

Consuming the data in an online social network is typically done by following social links and browsing individual users’ accounts. It is often also facilitated by a publish-subscribe mechanism (often implicit), where a “feed” displays new data posted to the user’s friends’ accounts.

Querying the social data (i.e. the social network information) using expressive queries is rarely directly possible for end users; Facebook’s recent graph search interface is one of a few exceptions.

2.4 Collaborative Repositories

In collaborative data management systems, the notion of “sharing” is different from the previous classes of systems: the users have a priori equal permissions to the data, which is therefore shared as a single physical object might be shared. In contrast, in the previous classes of systems there is a notion of data ownership, and in general only the data owner has write permission over the data; the notion of sharing simply means that other users have a read permission.

Collaborative repositories include wikis and content management systems (CMS), that are primarily designed for text and hypertext documents, but also other wiki-like systems designed for media files (e.g. the Wikimedia Commons repository), or structured data (e.g. Wikidata [1], Freebase [40], or the semantic wikis Sweetwiki [41] and Ontowiki [42]) and version control systems, which are mostly used for source code.

2.4.1 System Model

The collaboration model implemented by such systems is based on the idea that all the users have a priori equal access to all of the data. There is no notion of individual
data ownership, and no explicit social links between the users, which suggests the system model illustrated in figure 2.4.

We represent the data as a graph, since a number of collaborative systems manage structured or semi-structured data (e.g., Wikidata [1], Sweetwiki [41]).

Different data models are possible, and we simply consider that in each data model, the data can be decomposed into a set of atomic resources (e.g., nodes and edges, or else tuples) that the users can add, modify or remove.

2.4.2 Functionality

The different systems listed above introduce a data management layer in between the users and the repository itself, that is supposed to resolve conflicts between the participants’ contributions. Conceptually, it functions as if the users did not modify the underlying data directly, but input their modification preferences into a function, which then took all of these preferences and output a decision. An interesting proposal for exactly this functionality is the PeerCollaboration system [43], which proposes that the participating peers should vote on every change to the shared material.

In version control systems for example, there is generally no deletion, which prevents users from accidentally deleting (or overwriting) important data. Modern wikis
also often include version history and complex procedures to resolve conflicts [44].

In order to represent the functionality offered to the users in a generic way, we replace the basic publish and remove functions by pairs of functions request— and oppose— publication/deletion, that allow each user to express a preference about whether a data item is published or removed.

We obtain the following list of functions:

- requestPublication \( \langle r: \text{Data resource} \rangle \): request the publication of a data resource \( r \) to the set of shared data.
- opposePublication \( \langle r: \text{Data resource} \rangle \): oppose the publication of a data resource \( r \) to the set of shared data.
- requestDeletion \( \langle r: \text{Data resource} \rangle \): request deletion of existing data resource \( r \).
- opposeDeletion \( \langle r: \text{Data resource} \rangle \): oppose the deletion of existing data resource \( r \).
- query\( \langle q: \text{Query} \rangle \): Submit query \( q \) to the system. The system will return the available data matching \( q \). As collaborative repositories are typically centralized, their data is hosted in a database, with no particular restrictions on these queries.

### 2.5 Peer-to-peer Data Sharing Systems

The fourth class of interest is that of peer-to-peer (P2P) data sharing systems. Originally, P2P systems were defined in contrast to the client-server paradigm of the Web: the peers have symmetric roles, and are both providers and consumers of data. However, since the so-called “Web 2.0” paradigm shift [45], this is now also true at
a functional level for a large number of Web sites, in particular OSN, as discussed above. However, in P2P systems, users have administrative control over their local repository, and are not expected to be constantly available.

Another distinction is the importance of data transfers (downloads): the main data sharing activity consists in copying the data from one peer to another, thus creating multiple instances of the data resources, each one under control of one participant. This modifies the availability of the data.

Traditional P2P file-sharing networks, such as the eMule and Gnutella networks, are open networks (anybody can join), where users make media files available on their computers, and exchange these files over direct connections. A particular network (e.g. Gnutella) is only defined by a protocol, and any client that can operate this protocol can participate.

Peer Data Management Systems (PDMS), such as Piazza [21] or Orchestra [46] can be described as an extension of this concept to structured data. The peers in a PDMS are interconnected in an arbitrary graph, and host full-fledged database management systems. PDMS are typically designed for scenarios where the peers are not individual users but larger organizations.

2.5.1 System Model

As in the previous cases, data in a P2P data management system can be associated with its physical sources. Unlike the Web of Data, a P2P data exchange system allows several participating nodes to own a copy of a given resource, and the two copies are typically identifiable as the same. For example, a media file can be shared by several participants of a P2P file-sharing system. This implies that a data item can be owned by multiple sources.

The peers in the network are related by physical connections, that could be represented as a graph, however, in most cases the connections do not represent any social
relationships (e.g. trust). In addition, these connections are often under control of an algorithm, and not of the participants themselves. PDMS connections are typically only data schema mappings, and have little social significance. Therefore, we do not represent the users as nodes in a social graph, as in OSN.

In cases where the relationships do have a social significance, the more appropriate system model is the (DS)$^2$ model, presented in section 2.7.2.

The shared data can be structured; this is not the case in traditional file-sharing networks, but it is in PDMS. We therefore represent a P2P data sharing system as a graph of data resources, each associated with one or several users, and no explicit social relationships between the users, as shown in figure 2.5. For simplicity, we represent the graph of data as resources connected by links, as in the case of the Web of Data. However, while the resources and links could be interpreted according to the Web of Data standard data model and protocols (RDF and HTTP), here we do not commit to any particular data model. The links are intended to represent the fact that data resources published by one peer may be related to data published by others, in a way accessible to queries.

![Peer-to-peer data sharing system](image)

Figure 2.5: Peer-to-peer data sharing system. The users are not organized in a socially meaningful graph, but they have associations with the data that they share. The data is structured and may form a graph.
2.5.2 Functionality

The functionality of a P2P Data Sharing System includes the following functions:

- **publish** \((r: \text{Data resource})\): add a data resource to the set of shared data. The data is associated with the user by an association “owns”.

- **delete** \((r: \text{Data resource})\): remove existing data resource \(r\), if the user owns \(r\).

- **copy** \((r: \text{Data resource})\): Add an “owns” association between the user and an existing data resource \(r\) owned by one or several other users. In practice, a copy of the \(r\) is transferred across the network: this is not represented in our model.

- **query** \((q: \text{Query})\): Submit query \(q\) to the system. The system will return the available data matching \(q\), or a subset thereof. In this case the results may be limited by partitions in the network (implying that the query may not reach every peer).

As in OSN, the peers are both data providers and consumers, and have access to all of the functions.

2.5.3 Queries

Traditional file-sharing networks supported keyword queries over file names. Later systems, including PDMS, use more structured data models and support database query languages.

Query processing techniques were initially developed for networks of relational databases (e.g. Mariposa [47]), then adapted for other data models such as RDF (e.g. Edutella [48]). The main difference with the Web of Data approach is that the peers in a P2P network cooperate to distribute queries in the network, instead of the client having to individually query each data source.
More recent research on PDMS also assumes that the databases use different schemas, and propose techniques to integrate this data, i.e. algorithms to rewrite queries as they are distributed in the network [23, 24].

Under the assumption of heterogeneous schemas, the functionality of *copying* data also requires transforming it between schemas, a problem known as the data exchange problem [49].

## 2.6 Comparison

We now discuss the properties of these different classes of systems, in relation to problems that arise from conflicting interests between their users.

In a data sharing system, the conflicting interests of users can materialize in two general ways: problems can be due either to technical costs or to the nature of the shared data.

### 2.6.1 Technical Costs

The technical costs of using the system, such as storage and bandwidth usage, may cause conflicts of interests when one user can impose costs on others. This is primarily determined by the system architecture.

In centralized systems, the bulk of the cost is borne by a single organization that runs the system and hosts all of the data. These costs can be valued financially: for example, the operating costs of the Wikimedia Foundation (the non-profit organization that runs Wikipedia, Wikidata, and other lesser known projects) were around $US40 million for the fiscal year 2013-2014 [50]. These costs place significant risk on data sharing systems, which may be shut down if the host organization cannot meet these costs. Several well known services were shutdown in the past years, in particular services run by Google (Google Base, Google Knols, and soon Freebase [51]).
In P2P networks, the cost of storing the data (and of serving requests for this data) is shared between the participants, since each participant stores part of the data. If the peers store their own data, and are primarily interested in retrieving data from others then the cost of each peer’s activity is primarily borne by the others, which creates an incentive structure where “free-riding” (downloading data from others while contributing none) is the dominant strategy [52, 53].

In structured networks, the shared data is allocated to the different peers through an algorithm [54], and the cost is shared more or less equally between the participants. The heavy users may then comparatively benefit more than less active participants.

Another issue that occurs in the case of distributed storage is that users must relinquish control of their data and allow it to be stored by other participants, which they do not necessarily trust, and which may be located in a different jurisdiction, creating possible legal risks.

### 2.6.2 Conflicts over Shared Data

The second main type of conflict is over the data that is shared in the system. We can illustrate potential conflicts with two example users Alice and Bob. We consider the following cases of conflicting interests between these users:

- Alice can delete data that is useful to Bob
- Alice can publish data that is not useful to Bob.

One type of conflict that illustrates both cases is the problem of “edit wars” [55] in Wikipedia: users disagree on facts (e.g. whether “psychometrics” should be described as “pseudoscience”\(^5\)), or on minor style issues (e.g. whether “the” in the band name...

---

\(^5\) a lengthy argument (along with a resolution of the conflict) is visible on the Wikipedia talk page http://en.wikipedia.org/wiki/Talk:List_of_topics_characterized_as_pseudoscience
“The Beatles” should be systematically capitalized or not [56]), and repeatedly edit and counter-edit the text to impose their views.

In this case, the nature of the application makes it difficult for both sides to be represented: the data that Alice and Bob would like to share are mutually exclusive.

An example of the second case is the problem of “pollution and poisoning” in P2P file-sharing networks [57]: these terms refer to users introducing corrupt files or files with misleading meta-data in file-sharing networks (voluntarily or not). If Alice shares an MP3-formatted music file supposedly (according to its name and meta-data) containing a song by the Beatles, but actually containing white noise, Bob may find this by querying for “the Beatles”, and waste time and bandwidth downloading it, which arguably produces a negative payoff to Bob, in game-theoretic terms.

We now discuss how different properties of data sharing systems determine the outcome of such conflicts.

We first consider the case where some user Alice wants to delete data that Bob is interested in.

Decentralized Control vs. Distributed Control

The outcome of this conflict will depend on the control model of the data sharing system: here we can distinguish distributed control from decentralized control, based on definitions by Khare [58].

Khare defines the notion of decentralization in the context of distributed systems, where different components of the system are under control of different agencies. In this context, for a given variable shared by these different components, the variable may be distributed, in which case its value is determined by a distributed algorithm implemented by the different components, or it may be decentralized, in which case the system may be in a correct state despite the different components not agreeing on the variable’s value. In this case, each component can autonomously determine
the variable’s value using its own decision function and the values determined by the other components. The system is then *consensus-free*.

In collaborative repositories such as wikis, there is a unique collection of shared data, which is the same for all of the users, and is the result of a complex process whereby conflicts between the different users’ contributions must be resolved. Therefore, this data is under *distributed control*, in the sense of the above definitions.

In other types of systems, such as P2P file-sharing networks or online social networks, each user has control over her own contributions, and we can consider that the shared data is under *decentralized control*.

The example conflict between Alice and Bob is then resolved as follows:

- Under distributed control, Alice and Bob must express their preference with respect to the data of interest, and participate in a conflict resolution process which will reach a decision, favourable to Alice or to Bob.

- Under decentralized control, the outcome will depend on the ownership of the data. If the data is owned by a single user (this would be most likely the case on the Web of Data) then the owner can unilaterally decide to delete or keep the data. If several users own a copy (e.g. in a P2P file-sharing network), then the data will be available as long as one of the owners keeps it.

**Data Relevance**

We now address the problem of users contributing data that is somehow *not useful* to others.

In the case of distributed control, this problem is similar to the previous case, and users must resolve their disagreement through a conflict resolution process.

In the case of decentralized control, if Bob cannot delete the data contributed by Alice, then Bob’s interest is to avoid *consuming* this data.
In the simplest case, Bob can formulate queries for which the contentious data is not a valid answer. However, when queries are solely applied to the shared data, this may not be possible (as in the previous example of MP3 files), and Bob may only be able to identify the file as unwanted after retrieving it.

In this case, Bob is arguably faced with an information retrieval (IR) problem, in the sense that queries over the data may return unwanted resources, and the problem is then to separate these from the preferred query answers. Or, in IR terminology, to distinguish relevant resources from irrelevant resources.

Different classes of systems offer different affordances in this respect, depending on their logical structure.

Most IR techniques are designed for text and seek to determine the topical relevance of documents to keywords. However, the problem of data trustworthiness, or authority, has been addressed largely by using the hyperlink structure of the Web (of documents). The PageRank [59] and HITS [60] algorithms, followed by others, considered hyperlinks as implicit “endorsement” of the link target by the link source. The graph of links can then be used to infer the most trusted pages.

These algorithms (mainly PageRank) have been extended for the Web of Data: ObjectRank [61] (and variations of this algorithm, e.g. Peer-Sensitive ObjectRank [62], ReConRank [63]) are adaptations of PageRank to structured data and RDF.

It’s worth noting that these techniques require a global view of the graph of data, which implies potentially a high cost in a distributed setting.

When the users have explicit associations with the data, additional techniques become available, which are mainly designed to capture relevance in the sense of the users’ subjective preference. Typically, recommender systems use the associations of users and data to compute the similarity of users, then, for a given user, they recommend items associated with similar users. The main underlying assumption is
that users who were interested in similar data (or objects) in the past, will continue to have similar tastes in the future [64].

When the social network is also available, it provides additional information that has been shown to improve recommender systems [65,66]. Traditional IR approaches, such as those applicable to the Web of Data, also benefit from social network information [67].

In addition, an explicit social network can be used as a trust network, in the sense that users can identify participants whom they trust to provide valuable data, and consume data only from these trusted sources [68]. In this case, the IR problem is only indirectly addressed, since trust expresses belief that an agent will act “dependably” based on this agent’s history [69,70]: this notion is therefore best at identifying “honest” users, who tend to be either consistently honest, or else attackers implementing some strategy to deceive the users, rather than users who might provide subjectively “good” data, with likely much less consistency.

In Peer Data Management Systems (PDMS), a particular instance of this problem is when the different peers may host logically inconsistent data. In this case, the peers may implement various strategies to filter the query answers in order to resolve inconsistencies. Such strategies include rejecting all answers derived from inconsistent data [25], systematically resolving inconsistencies based on local data (implicitly trusting the local peer over remote peers), or applying specific trust preferences by each peer [26]. A more flexible approach is that of Martinez [71], where users can define their own inconsistency management policy.

Finally, we note that in the case of centralized systems, Bob could also be the organization that hosts the data, which can then unilaterally delete any data that goes against its interest to host.
2.7 Conclusion

2.7.1 Summary

In data sharing systems, the main situations where participants’ interests are conflicting are (i) users bearing technical costs due to other users’ activity, and (ii) disagreements over which data is shared.

Regarding technical costs, centralized systems bear most of the operating costs but may disappear if they don’t offset these costs by some other benefits, such as donations or a commercially viable use of the shared data. Distributed systems require different levels of cooperation, but typically see the cost of each user’s activity borne by other users, which creates a disincentive to participate.

For disagreements over data, there is a general dichotomy between distributed control, where conflicts must be resolved following specific procedures, and decentralized control, where each user is in control of part of the data. Distributed control can be described as a way of curating the collection of shared data, whereas in the case of decentralized control, it is the responsibility of each user to choose which data to consume, based on contributions that may be relevant or irrelevant. In this case, the logical structure of the system greatly influences the participants’ affordances to determine the relevance of data.

Finally, we note that in centralized systems, both the distributed and the decentralized control models are limited by the central authority of the service provider, in the sense that the service provider can also unilaterally delete any data that goes against its interest to host.
(a) Web of Data.

(b) Online Social Networks.

(c) Collaborative Repositories.

(d) Peer-to-peer data sharing.

(e) Proposed System Model: Decentralized Social Data Sharing.

Figure 2.6: The Decentralized Social Data Sharing model combines properties mainly of OSN and P2P networks to manage structured data.
2.7.2 Decentralized Social Data Sharing (DS)$^2$

The models for existing classes of systems are shown in figure 2.6. We propose to combine the best properties of these models as a new general design for data sharing systems.

These desirable properties are the lack of a central authority, an explicit social network, and decentralized control over structured data. The resulting model is the (DS)$^2$ model, for Decentralized Social Data Sharing systems (fig. 2.6e). We fully develop and detail this model in the next chapter.

The (DS)$^2$ model provides a decentralized control model in a distributed system, and a complete logical structure with a graph of data, a graph of users, and associations between data and users. This makes most information retrieval techniques applicable, although some may not be cost-effective in this distributed context (e.g. recommendation algorithms).

**Examples** While this description of the model is new, a few existing systems can be considered to match this model.

**Social P2P Networks**

In a few P2P networks, known as “Friend-to-friend” networks [18, 72], the peers only establish connections with trusted acquaintances, and use cryptography to keep their communications private from outsiders. They are primarily designed for privacy. Turtle [19] is one such P2P file-sharing system, and a closely related one is Tribler [73], where users express social acquaintance links that are maintained separately from the physical P2P connections. These social links are used for recommendation and trust, and influence parameters of the BitTorrent protocol used for file transfer. This is meant to address the cooperation problem associated with efficient downloads [74], based on the idea that users will be more cooperative with their social acquaintances.
Both initially started as academic research projects. However, these file-sharing systems are not designed to manage structured data.

**PDMS**

For PDMS some studies have considered explicit trust relationships between the peers [26], which also implies a social graph of peers. However, the system is only studied with respect to the problem of providing consistent query answers from inconsistent data, and uses the trust relationships to resolve inconsistencies. Our work is concerned with a much broader view of what can be done with these connections and how they can be strategically managed.

**Decentralized Version Control Systems**

Several recent version control systems, such as Git [75] or Mercurial [76], have shifted from a “star” topology, with many participants contributing to a central repository, to an arbitrary graph, where each participant is at one node and hosts a full copy of the repository. The participants know of each others’ repositories (which we can represent as a social graph), and can “pull” commits from each other. Each participant has full control over their own repository. This way of cooperating resembles the (DS)$^2$ proposal, although there is no permanent network of connections that would allow the participants to query the global collection, for example.
Chapter 3

Decentralized Social Data Sharing
Applications: Formal Definition

3.1 Introduction

In this chapter, we precisely define the concept of a Decentralized Social Data-Sharing \((\text{DS})^2\) system, by developing the model sketched in section 2.7.2.

This model builds on a previous formalization of schema-based P2P file-sharing systems \([77,78]\), greatly simplifying it\(^1\) in order to better relate the model to alternative approaches, at the level of the systems’ logical structure.

In addition, this model provides a sound basis for the generic API presented in chapter 5 and for the economic analysis of chapter 8, since it can be easily transformed into a game-theoretic construct by considering the users’ available actions as moves in a game, and associating a utility to the system state.

This chapter is organized as follows.

We detail the conceptual model of a \((\text{DS})^2\) system in section 3.2, then its functionality in section 3.3. Finally, in section 3.4 we extend the model to explicitly model

\(^1\)Our previous model considered a single data schema, and represented the network of peers as a state machine that evolved as the peers joined and left the system, while the content of the peer databases was considered separately, and its evolution was not clear in the model.
the intermittent presence of peers in a P2P deployment.

3.2 Conceptual Model of Social Data Sharing

3.2.1 Social Network

We consider a set of users $U = \{u_0 \ldots u_n\}$, organized in a social network represented by a graph $G_S = \langle U, S \rangle$. The edges $S$ represent a form of social acquaintance that may be directed or undirected. In the directed case, each peer $u_i$ may choose to assert her acquaintance with any other peer $u_j$, which is represented by the edge $(u_i, u_j)$.

In this model, there is only one type of social connection. However, it could easily be extended to multiple social relations, using labeled edges.

3.2.2 Data Resources and Graph of Data

A characteristic of (DS)$^2$ systems is that the different participants can contribute small subsets of the data, which are handled and stored independently. We call these subsets of data data resources, without committing to any specific data model. These resources could be large items such as files, but the more interesting case to us is when the data is structured.

We model structured data as a graph of data $G_D = \langle V, E \rangle$, where the nodes have unique identifiers from a domain $\Delta_V = \{v_i\}$, and the edges are labeled with symbols from a domain $\Delta_E$. Formally, $E \subseteq V \times \Delta_E \times V$.

In order to associate specific parts of this graph of data with the users who share them, we further assume that the graph of data can be represented as the union of $k$ distinct and uniquely identified subgraphs $r_1, r_2, \ldots r_k$, which are then the data resources shared by the peers. We note the set of resources $R$. The resources are mapped to an identifier space $I$ by an injective function $F_R()$. 
A graph of data, decomposed into several resources, is illustrated in figure 3.1.

![Graph of data](image)

**Figure 3.1:** A graph of data, made up of multiple resources.

This representation can be instantiated by several specific data models.

**RDF**  In RDF, the nodes $V$ of the graph are labelled with URIs, and the edges are RDF triples, labelled with the predicate of the triple. In this case the resources shared by the peers are RDF graphs (or triples)$^2$.

**Relational or Document-Oriented Data**  Relational data can be represented as a graph where the nodes are tuples, (which can be labeled by a primary key of the tuple) and the edges are foreign keys pointing to other tuples; their labels are the foreign key attributes themselves. The resources are then the tuples, and each resource includes a node with all its outgoing edges. This model is also the one used by document-oriented databases such as couchDB [80], and the one used in our previous work [77, 78].

---

$^2$In fact, URIs identify arbitrary resources in the Semantic Web sense of this term. Our data resources are then a specific kind of information resources [79].
3.2.3 Data Sharing

Each peer shares a set of resources, and any resource can be shared by several peers. The association of users to resources is represented by a bipartite graph $G_A = (U \cup R, A)$, where the edges $A$ relate users to resources.

The association $A$ represents ownership: users own and control (a copy of) each resource that they share.

3.2.4 System Model

The different elements of this conceptual model are represented in figure 3.2.

In this example, the resources $r_i$ each contain one node and all of the outgoing edges of this node. This data model can represent relational data, where a node represents a record, and outgoing edges represent foreign key values referencing other records.

Figure 3.2: Sharing graph data in a Decentralized Social Data Sharing system.
3.3 System Evolution

The above model represents a snapshot of a data-sharing system, i.e. its state at a given time. As users join and leave the system, publish new data, the system evolves.

3.3.1 Modifying Operations

The available actions to modify the system state are listed below, and their effects are given formally in algorithm 1.

Data Management operations:

- Publish($r$: Data Resource): Publish resource $r$.

- Delete($r$: Data Resource): Remove the shared resource. More precisely, this means that the association between $u_i$ and the resource is removed, and if the resource is no longer associated to any users, then it is removed entirely.

- Copy($r$: Resource): re-publish a resource that is already published by another user. This creates a copy of the resource under control of the user $u_i$.

Modifications to the social graph:

- Join(): $u_i$ joins the graph $G_S$.

- Leave(): $u_i$ leaves the graph $G_S$. This also implies that all social connections are dropped, and that $u_i$’s data is deleted. More precisely, any associations between $u_i$ and a resource are removed.

- Connect($u$:User): $u_i$ connects to $u$.

- Disconnect($u$:User): $u_i$ disconnects$^3$ from $u$.

$^3$we could have separate functions for dropping incoming and outgoing edges, but it seems that one would drop both or none.
Algorithm 1 (DS)²: Modifying operations. The effect is given with respect to the (arbitrary) user \( u_i \) that executes the action. We use the pseudocode notation \( \leftarrow \) to denote variable assignment, and the operator \( \setminus \) for set difference.

**Function** Publish\((u_i, r)\):
\[
R \leftarrow R \cup \{r\} \\
G_D \leftarrow G_D \cup r \quad \text{//Merge } r \text{ into existing graph of data} \\
A \leftarrow A \cup \{(u_i, r)\} \quad \text{//} u_i \text{ owns } r
\]

**Function** Delete\((u_i, r)\):
\[
A \leftarrow A \setminus \{(u_i, r)\} \quad \text{//} u_i \text{ no longer owns } r \\
\text{if } \nexists u, (u, r) \in A \text{ then} \\
R \leftarrow R \setminus \{r\} \quad \text{//No users own } r: r \text{ is removed} \\
G_D \leftarrow G_D \setminus r \quad \text{//Remove subgraph } r \text{ from } G_D \\
\text{end if}
\]

**Function** Copy\((u_i, r)\):
\[
A \leftarrow A \cup \{(u_i, r)\}
\]

**Function** Join\((u_i)\):
\[
U \leftarrow U \cup \{u_i\}
\]

**Function** Leave\((u_i)\):
\[
U \leftarrow U \setminus \{u_i\} \\
\text{for all } u_j \in U \text{ do} \\
\quad \text{if } (u_i, u_j) \in S \text{ or } (u_j, u_i) \in S \text{ then} \\
\qquad \text{Disconnect}(u_i, u_j) \quad \text{//} u_i\text{'s social connections are dropped} \\
\text{end if} \\
\text{end for} \\
\text{for all } r \in R \text{ do} \\
\quad \text{if } (u_i, r) \in A \text{ then} \\
\qquad \text{Delete}(u_i, r) \quad \text{//} u_i\text{'s data is deleted} \\
\text{end if} \\
\text{end for}
\]

**Function** Connect\((u_i, u)\):
\[
A \leftarrow A \cup \{(u, u_i)\}
\]

**Function** Disconnect\((u_i, u)\):
\[
S \leftarrow S \setminus \{(u_i, u); (u, u_i)\}
\]
3.3.2 Deletion vs. Modification

It is important to note that the above model does not include an operation to modify a resource. In our model, resources have unique identifiers: modifying a resource therefore implies modifying its identifier, which means that modifying a resource is equivalent to deleting it and replacing it with another identifier. This deletion and modification can be done with the above semantics.

This guarantees consistency between different resources in a distributed deployment.

3.3.3 Queries

The system state, which we represent by the three graphs $G_D$, $G_S$, and $G_A$ can also be queried. Queries do not modify the system state.

Queries in existing data-sharing systems can be classified according to which of the above graphs they apply to. For example, most P2P data-sharing systems are intended to handle queries over the graph of data $G_D$. The graph $G_A$ contains the information necessary to process queries of recommender systems. Recommender systems are designed to answer implicit queries taking a user as input, and returning a set of resources recommended to this user. These recommendations are typically made on the basis of the similarity of users, computed from the edges of $G_A$.

Finally, $G_S$ can be used to support social queries, such as finding the social distance between users, or predicting/recommending missing edges in the graph ("people you may know") [81].

In this thesis we mainly study the classic problem of querying the graph of data $G_D$. We leave the generalization of queries to the union of $G_D$, $G_A$, and $G_S$ to future work, and discuss relevant issues in section 9.4.1.
**Trust Indicators**  However, we do include information from the graphs $G_A$ and $G_S$ for the purpose of ordering query results, similarly to “ORDER BY” clauses in SQL and other classic languages. These orderings are intended to help the users identify the most valuable data, and constitute the trust indicators of our application P2Pedia, presented in section 4.4.

**Regular Path Queries**

Generally, graph queries can be defined using the notion of graph patterns [82–84].

Fully defining this general class of queries is not that relevant here, since in this thesis we only study the subclass of Regular Path Queries (RPQ). Instead, we simply specify RPQ using the more general terminology of graph patterns (as formulated by Reutter [83]).

**Definition 1** Single-source regular path queries are of the form $Q = \langle (V_0, r, x), \{x\} \rangle$: the query pattern $Q_P = (V_0, r, x)$ is a single edge $(V_0, r, x)$ connecting a constant node identifier $V_0$ to a node variable $x$, with an edge labeled by a regular expression $r$. The node constant $x$ is the free variable of the query.

**Query Answers** Answers to the query from a graph of data $G$ are bindings of the free variable of $x$ to node labels of $G$, such that there exists a path between $V_0$ and $V_1$ that can be mapped to the regular expression $r$, i.e. the concatenation of labels along this path is in the regular language $L(r)$.

**3.4 Model Refinement for a P2P Deployment**

This model is intended to be general enough for multiple classes of systems, with different system architectures. In particular, it can describe server-based deployments\(^4\), although in this case it abstracts away the central authority that controls the server, which is problematic from an agent-oriented perspective.
as well as peer-to-peer deployments.

### 3.4.1 Churn

However, while servers are expected to be permanently available, peers in P2P networks are expected to be only intermittently available, creating a phenomenon known as churn. The above primitive functions do not distinguish between joining/leaving the network permanently and temporarily. In the latter case, it would be convenient to automatically “save” the user’s connections and data before a temporary absence, and “restore” them on the peer’s return.

A meaningful representation of the system state would then be not only the current graph of peers in the network, but also the temporarily unavailable peers, and the temporarily unavailable data, as illustrated in figure 3.3.

![Figure 3.3: System model with online peers and offline peers, available and unavailable data.](image)

In terms of functionality, this would require two additional functions, “go offline”, and “go online” which would be exposed to the API, while the system would automatically manage the social connections between online peers. The “connect” and
“disconnect” exposed to the user would then have the semantics of establishing or undoing long-term social connections.

Our reference implementation of the above conceptual model and functionality, presented in chapter 5, includes these P2P-specific functions.

3.4.2 The Problem of Query Processing

In a (DS)$^2$, we would like to apply these queries to a distributed collection of shared data, and the properties of its distribution make query processing a difficult problem.

RPQ essentially amount to traversing the graph of data, i.e. following edges from node to node. From a given node found inside one data resource, one must therefore find the data resources containing the adjacent nodes. Due to the peers’ autonomy in keeping the data they choose, these data resources could be anywhere (or nowhere) in the network.

Therefore, the available techniques to locate them are essentially peer-to-peer search protocols. One of our key research problems is therefore to process expressive queries using such protocols. We address this problem in chapter 6.

3.4.3 Query Semantics

The “default” semantics of queries are that any data of $G_D$ that matches the query is an answer.

However, if some of the peers are unavailable (due to churn) and/or the network is not connected, a query will not be applied to the full graph $G_D$, but only to the subset of $G_D$ available to the querying user.

Under the basic query semantics, this can be seen as simply incomplete answers.

By analogy with Semantic Web approaches [85,86], we can also define the scope of the query as the reachable data, in which case some algorithms may provide complete
3.4.4 Strategic Behaviour

Where this becomes interesting from a multi-agent perspective is that under these semantics, the participants of a (DS)$^2$ can then modify the set of answers to a query simply by modifying their social network connections, which in turn modifies the set of available data, to which the query will be applied.

In the context of conflicts related to data relevance (discussed in section 2.6.2), a user may want to exclude some irrelevant data from query answers. One way would be to express this preference in the query itself (which may not be possible), but in a context where the cost of accessing the data is high to begin with, it may also be valuable to the users to strategically manage their connections in order to have access to less data, but primarily to relevant data. We investigate this in chapter 8.

3.5 Conclusion

The (DS)$^2$ design is based on existing data sharing systems, and is intended to inherit and combine their valuable properties.

The decentralized control over the data ensures that users have a high level of control in choosing which resources they share, and the social network can facilitate their selective access to the data shared by other users that they trust.

In chapter 4 we illustrate this model by applying it to the design of a wiki system, and in chapter 5 we present a generic platform that can be used as a P2P infrastructure for implementing other (DS)$^2$ applications. Together with the definition presented here, these provide a blueprint to implement (DS)$^2$ systems.

Then, chapters 6, 7 and 8 evaluate this model as follows:

- chapter 6 evaluates the cost of processing RPQ in this setting,
• chapter 7 evaluates the decentralized control model through its application to a wiki,

• chapter 8 addresses the problem of cooperation among self-interested agents in a more general context, and demonstrates the importance of the users’ control over the social network.
Chapter 4

P2Pedia, a \((DS)^2\) Wiki

As an illustration of the \((DS)^2\) model, we now show how it can be applied to the design of a new type of wiki system. We describe P2Pedia, a wiki system designed by replacing the traditional collaboration model of wiki systems with the \((DS)^2\) data sharing principle.

After explaining the motivation for such a system, we show how the concepts and use cases of a traditional wiki can be adapted to the \((DS)^2\) model. We illustrate the functionality of P2Pedia with a simple scenario, and introduce several useful classes of queries, including trust indicators that use information from graphs \(G_A\) and \(G_S\) to rank query answers.

The rest of this chapter is organized as follows. Section 4.1 motivates P2Pedia, and section 4.2 discusses relevant related work. The functionality of P2Pedia is presented in section 4.3, and its trust indicators in section 4.4. Finally, we draw some conclusions in section 4.5.

4.1 Motivation

Traditional wikis, such as Wikipedia, are data-sharing applications that can be classified as collaborative repositories, according to our classification of chapter 2.
As an encyclopedia, Wikipedia provides an intuitive illustration of the distributed control principle implemented by collaborative repositories (discussed in section 2.6.2): the pages in Wikipedia are articles defining the concept identified by the page title. They can be represented by shared variables, where the page title is the variable name, and the page content is the value.

The contributors’ opinions can be represented by “preferred values” for the variables, and their interaction in editing pages, constrained by Wikipedia’s policies and procedures, is a distributed algorithm that produces a single value for the variable. The supposed “consensus” represented by Wikipedia content can be seen as the result of executing this algorithm.

In many cases, the execution of this algorithm involves edit wars [55], where disagreeing contributors repeatedly edit and counter-edit a page to impose their views, until an administrator finally intervenes to settle the dispute, based on an informal poll of other users’ opinions.

Conflict resolution procedures are process losses [44], considered a necessary cost in online collaborative work.

In the design of P2Pedia, we challenge the need for a consensus at all. Conflicts and edit wars are due to the collaboration model that requires a single article on each topic in Wikipedia, reflecting an idealized “neutral point of view”. Why not, instead, embrace the diversity of viewpoints, and allow them to co-exist?

As a solution, a wiki based on the principle of decentralized control would allow “multiple, distinct, simultaneously valid” articles about each topic\(^1\). While some may consider that the wiki concept is fundamentally associated with collaboration in the traditional sense (i.e. distributed control), we see it rather as the principle of creating and editing web pages through a browser, that can be largely reduced to a simple set

---

\(^1\)Khare [58] describes decentralized variables as variables that can have “multiple, distinct, simultaneously valid values”.

of use cases – creating, deleting, editing, and browsing pages. The basic expectations of these use cases, although defined in a distributed control model, are compatible with the decentralized control model.

4.2 Related Work

4.2.1 Decentralized Content Management

With the advent of decentralized version control systems (DVCS) such as Git, Mercurial, or Bazaar, it was a natural step to adapt such version control systems to serve as a back-end for wikis. While such systems have not received much academic attention, several implementations are available: among others we can cite gitit\(^2\), git-wiki\(^3\), Olelo\(^4\) (derived from the previous), and gollum\(^5\), the wiki engine used in the code forge Github.

Decentralized approaches to version control assume multiple repositories and define operations to exchange content between these repositories, i.e. cloning, forking, and propagating changes. To some extent, they provide a decentralized control model.

However, in the DVCS-based wikis listed above, the decentralized control principle is not directly applied to the normal wiki use cases: rather, users edit the wiki through traditional use cases, and a separate interface (presumably available only to administrators) supports DVCS-specific operations, such as a cloning or forking the wiki content.

\(^2\)http://gitit.net
\(^3\)http://atonie.org/2008/02/git-wiki
\(^4\)http://www.gitwiki.org
\(^5\)https://github.com/github/gollum
4.2.2 Decentralized Wikis

One semantic wiki, DSMW [87], includes functionalities relevant to decentralized control management. This semantic wiki implements a multi-synchronous collaboration model [88], closely related to the DVCS idea. Participating peers host a full wiki, and subscribe to feeds of updates from other peers. The feeds are then automatically merged with an algorithm that ensures CCI consistency (Causality, Consistency and Intention Preservation [89]).

However, the consistency guarantees restrict the possibilities of divergence. The multi-synchronous workflow is intended to lead to eventual consistency, and although a participant can choose not to integrate a set of changes from another participant, this will prevent it from integrating any changes made later on, virtually disconnecting the two versions. Our approach is much more flexible, encouraging divergence while allowing exchange of content at any time between any participants, regardless of their history.

More closely related to our work is the Smallest Federated Wiki (SFW) project\(^6\), spearheaded by Ward Cunningham, the creator of the original wiki.

In Cunningham’s view [90], wikis should be interoperable and users should be able to copy content from one to another, in the same spirit as one can “fork” a software project using a DVCS. In this view (as in our own), consistency between different versions of a page is neither a goal nor even necessarily desirable.

In SFW, each participant has a wiki server. The network of servers is maintained automatically by the system (the users cannot directly manage their social network). Users of SFW can explicitly navigate the different nodes of the federated wiki. By following wikilinks, the user navigates between the pages of her own wiki, and when the link points to a page that she doesn’t have, the system automatically selects a

\(^{6}\text{https://github.com/WardCunningham/Smallest-Federated-Wiki. Note that we describe this project based on its implementation and available documentation, as it has, so far, not been discussed in any academic publication.}\)
version from a neighbour, and directs the user to view this version. The user can then “fork” it, which copies it to her local server. We note that there is no impediment to each server being used on its own, as a traditional wiki.

Cap [28] has proposed a similar idea, although this paper does not refer to any implemented system. This proposal suggests complex procedures where the users would cooperate as in a traditional wiki, then fork (split an article into “variants”) in the case of conflicts. Relevant challenges includes malicious users attempting to control multiple variants, or using misleading metadata regarding the point of view that they defend. Potential solutions, involving complex access control policies and distributed infrastructures, are discussed.

In comparison, P2Pedia and SFW have the advantage of relying on a simple control model, and be testable in an implemented system.

4.2.3 Wikipedia Language Editions

Many have observed that the different language editions of Wikipedia already allow a diversity of viewpoints, and several elaborate interfaces have been proposed in order to allow the readers to perceive these diverse viewpoints [91,92].

Although such systems are limited to querying the system, they bring up interesting approaches to compare the alternative versions of wiki pages and the viewpoints that they express.

4.3 Functionality of P2Pedia

4.3.1 Overview

P2Pedia is a data-sharing application where users share wiki pages in a social network. The users can publish new pages, which they then share with other users, in the sense
of a (DS)² application: other users have *read-only* permission on this page, but may copy it: they then have full control over the new copy.

Many pages sharing the same title (and thus describing the same topic) can co-exist, representing multiple viewpoints over the topic. When a user owns a copy of a page, no other user can delete it: they can simply copy it and edit their copy, thus creating a competing version. As a result, the version history has the structure of a tree, as shown in figure 4.1.

The users can query the system, find pages by page title, keyword, or queries on the version history. Search results are annotated based on implicit queries of the graphs $G_A$ and $G_S$: the users can see which other users own copies of each page, and such information is translated to indicators such as the *popularity* of a page, which may help users find pages with relevant content.

(a) Versioning in a centralized Wiki: v3 (b) Versioning in P2Pedia: v4, v5 and v6 are “leaf” versions in the versioning tree.

Figure 4.1: The different versioning processes available in a centralized and decentralized collaboration model produce different versioning hierarchy structures. Each figure shows an example hierarchy of versions for a single page.
4.3.2 Example Scenario

We illustrate the use of P2Pedia by a simple scenario, and we show the result in figure 4.2.

Four users $u_1, \ldots, u_4$ are using P2Pedia. They have formed a social network where $u_1$ is friends with $u_2$ and $u_3$ (they have bidirectional relationships), $u_3$ and $u_4$ consider $u_2$ a friend (who has not reciprocated).

$u_1$ first creates the page with the title “Sandbox”. The page id is “A”\(^7\) and it does not have a parent, since it was the initial revision. This page is shown below in JSON syntax:

\[
A: \{
"id": "A",
"title": "Sandbox",
"content": "this is a document",
"parent": \{
\}
\}
\]

The user $u_2$ then copies the page, and edits it to create a new revision, with the identifier “B”. Page “B” is a new revision of the “Sandbox” page, and references “A” as its parent revision:

\[
B: \{
"id": "B",
"title": "Sandbox",
"content": "this is not a pipe",
"parent": "A"
\}
\]

Similarly, the user $u_3$ copies page “A” and edits it independently to produce revision “C”, and finally user $u_4$ copies and edits revision “B”, producing revision “D”. (We omit the contents of these two last revisions).

\(^7\)Real pages have longer identifiers generated by hash functions, to guarantee their unicity; we use simple ones here for clarity
The result state of P2Pedia (illustrated in figure 4.2), with the users and pages involved in this scenario, can be represented as an instantiation of the (DS)$^2$ conceptual model, i.e. the graphs $G_D$, $G_A$, and $G_S$.

![Figure 4.2: Instantiation of the (DS)$^2$ conceptual model, for an example scenario.](image)

The users of P2Pedia are organized in a social network $G_S$. The resources $R$ that they share are wiki pages, and each user owns (and shares) copies of specific pages, as captured by the graph $G_A$.

The pages’ version history is recorded as links between related page versions, thus producing the graph structure $G_D$: the nodes of $G_D$ are the pages (described by two key-value pairs with keys “title” and “content”), and a directed edge labeled “parent version” from a page $p_2$ to a page $p_1$ indicates that $p_2$ was produced by editing $p_1$.

### 4.3.3 Use Cases

The use cases of P2Pedia are shown in the use case diagram of figure 4.3.

The use cases shaded in grey are the traditional use cases of a wiki, and must be redefined following the (DS)$^2$ principles. P2Pedia provides additional use cases to manage the social network: “friend” and “unfriend”. 
Figure 4.3: Use Cases of P2Pedia

Create, Delete, Copy These use cases can be mapped directly to the basic functionality of a (DS)² infrastructure. The use case Create Page, Delete Page, and Copy Page are implemented respectively by the basic functions publish, delete, and copy, described in section 3.3, where each time the resource being passed as argument is a wiki page.

Edit Page The use case Edit Page is more complex, and happens in three steps:

1. If the user does not own the page, then the page is copied (therefore, the “copy page” use case extends the edit page “use case”).

2. An editing form is presented to the user with the text of the existing page.

3. Once the user has entered changes and presses “save”, a new page is created: it contains the text in the editing form, and contains a “parent” link to the
old version of the page (therefore, the “edit page” use case includes the “create page” use case).

It is important to note that in this use case, as in all of the other ones, a page being edited by the user is not necessarily the most recent revision of its version history. The different versions of a page, however they may be related in terms of versioning, are all available to be edited. This implies that the version history of a page is a tree (Figure 4.1b), not a linear sequence as in traditional wikis (Figure 4.1a).

**Follow Wikilink**  In traditional wikis, wikilinks are internal hyperlinks defined by their label: the link points to the (single) page bearing that title. In P2Pedia, the wikilink conceptually points to the entire set of wiki pages sharing this title. The collection of pages is therefore searched for pages with the given title. The Use Case “search page” is therefore included (see below for a description).

**Search for Page**  P2Pedia offers an interface where the user can search for pages according to different criteria. The collection of pages can be searched by keyword, by page title, or based on the graph structure of the versioning links. For example, a user could want to find all of the “ancestor” pages (preceeding versions), or “sibling” pages (pages derived from a common parent). Such graph queries can be answered by traversing the graph of versions. These queries can be expressed as *Regular Path Queries* (see below). For any type of query, the results are a list of available pages presented to the user. The user may choose a page for viewing, as described by the “Choose Page” Use Case below (which therefore extends the search use case).

**Choose Page**  This use case consists in the user selecting a page to be viewed, among the list of search results. If the user does not own a copy, then the page will first be copied. By default, the pages that constitute the search results are organized
as in the graph $G_D$, which is a forest (i.e. a set of trees). The search results are ranked (see the included Use Case “rank search results” below), using one of the available trust indicators.

**Rank Search Results**  This use case consists in sorting the results of a search according to a trust indicator. Trust indicators are intended to be correlated with relevance: the available indicators and their underlying assumptions are discussed in section 4.4.

**Friend, Unfriend**  These use cases can be mapped directly to the social network management functions *connect* and *disconnect*.

### 4.3.4 Regular Path Queries

The semantics of versioning links (in P2Pedia but also in general) makes it interesting to traverse them in a recursive manner: in order to see the history of a page, the relevant query is an “ancestor” query, the transitive closure of the “parent” relation. This query can therefore be expressed using the regular expression $(parent)^*$.  

Using the data shown in figure 4.3, an “ancestor” query from page $D$, the answers to this query are page revisions $A$ and $B$.

Another example is the “sibling” query discussed above. In order to answer such a query, it is necessary to traverse links in both directions. This can be done with an extension to RPQ called the “inverse” operator. We denote it with a $-1$ superscript, the regular expression for this query is thus: $(parent)^{-1}(parent)$.

Applying the query to page $B$, the answers are the page revisions $B$ and $C$.

We detail these definitions and address the problem of processing these queries in chapter 6.

---

8under the simplest definition, where we cannot express the condition that two documents are different, $B$ is its own sibling.
4.4 Trust Indicators in P2Pedia

Trust indicators are extrinsic criteria that are used to evaluate the relevance of pages. They take the form of numerical scores computed by queries over the graphs $G_A$, that indicates which users own which pages, and the social graph $G_S$.

4.4.1 Motivation and Fundamental Assumptions

In traditional wikis, the content of a wiki is subject to the collaborative interaction of many users, that produces a kind of “editorial process”. While we have argued that this is not necessarily a good editorial process, the lack of a process transfers the burden of evaluating content to the reader. Different pages on the same topic, while a priori “equally valid”, may include a lot of redundancy and irrelevant content. If these pages become numerous, then it becomes impractical for the users to read them all and truly evaluate them.

Our intuition is that the collective behavior of many participants can produce a “wisdom of the crowd” effect [93], and lead to emergent properties in the distribution of wiki pages. For example, we expect some contributions to be agreed upon by a large proportion of the users, and others to be “popular” only within specific social circles: such aspects can be captured by specific trust indicators.

These indicators, that apply to users or to pages, are not an absolute assessment relying on a central authority, rather they are relative, and depend on local criteria such as the querying user’s social network and the pages that this user owns. In addition, they are simply indicators, that users can combine in order to make a personal assessment of other users and pages.

In the $(DS)^2$ model, these trust indicators are queries of the graphs $G_S$ and $G_A$. These graphs can provide meaningful indicators provided the following assumptions hold: (i) users own copies of pages that they consider to be relevant, (ii) social
relationships convey trust, and (iii) the similarity of users is meaningful\(^9\).

These assumptions rely on what we intuitively consider to be “normal” behavior. However, it is an open problem to determine whether the system, and (DS)\(^2\) data-sharing systems in general, provide any incentives or disincentives for users to behave this way. We partly address this question in chapter 8.

### 4.4.2 Social Networking Trust Indicators

The social network \(G_S\) can be used to determine subjective trust indicators associated with users. For a considered user \(u_i\) (viewing the search results), the following indicators will be attached to another user \(u_j\) (that may have a copy of a given page in the search results):

- **User Popularity**: The number of users that have connected to \(u_j\), i.e. the indegree of node \(u_j\) in \(G_S\).

- **Network Distance**: The distance (length of the shortest path) from \(u_i\) to \(u_j\) in the social graph \(G_S\). Assuming that users trust their neighbours and that trust has some level of transitivity, the network distance can be a good trust indicator.

### 4.4.3 Similarity

The “similarity” of two users \(u_i\) and \(u_j\) is measured by the Jaccard coefficient\(^{10}\) of the sets of pages they own. The set of pages owned by a user \(u\), which we note below

\[
pages(u)\]

is the set of neighbouring nodes of \(u\) in the graph \(G_A\).

\[
\text{trust}_{\text{sim}}(u_i, u_j) = \frac{|\{pages(u_i) \cap pages(u_j)\}|}{|\{pages(u_i) \cup pages(u_j)\}|} \tag{4.1}
\]

\(^9\)this is the assumption that supports recommender systems: past similarity of users, as measured by what pages they own, is predictive of future similarity; also see section 2.6.2

\(^{10}\)also known as “Tanimoto coefficient”
Again, this indicator can be associated with a user, and will intuitively give high scores to users with similar tastes.

4.4.4 Page Indicators

Ranking search results requires indicators that apply to pages, whereas all of the above indicators apply to users.

The simplest indicator that can be associated to a page is its popularity: this assumes that many users choosing to own a copy of a particular page is an indication of quality. Formally, the page popularity of a page $p$ is the indegree of $p$ in the graph $G_A$.

Another strategy is to score a page $p$ according to the trust scores of the users who own a copy of this page, using one of the previous indicators. Currently, the P2Pedia interface allows the user scores to be combined using the following simple functions: mean, maximum, minimum and sum. Creating new functions (e.g. weighted popularity) would be very simple.

4.5 Conclusion

Ward Cunningham, who invented the original wiki, considered that its centralized nature was a mistake [90]. Although centralized tools are the norm for computer-supported collaborative work, his intuition of a decentralized approach is interesting.

We have independently designed P2Pedia, a wiki with a decentralized collaboration model and an explicit social network. Early experiments have shown the need for a way of ranking and filtering pages, using extrinsic criteria. Useful criteria are provided by the associations of users with pages, and the social network.

Our intuition suggests that these indicators generalize to other (DS)$^2$ applications. However, an important problem is to validate their usefulness, and the effectiveness of
P2Pedia as a collaboration tool. This is addressed mainly by a user study presented in chapter 7. Since this user study was conducted in a somewhat controlled setting (tutorials with a set workflow, in a classroom environment), it does not fully address the question of how truly independent and self-interested agents might have used the system. Chapter 8 explores this question more generally for (DS)$^2$ applications.

Just as the “wiki-specific” functionality of P2Pedia can be described in terms of the underlying conceptual model, this functionality can be implemented in a thin “application layer” on top of a generic infrastructure that implements the (DS)$^2$ functionality. This generic architecture is presented in the next chapter.
Chapter 5

A P2P Platform for social data-sharing applications

5.1 Introduction

P2Pedia, presented in the previous chapter, is one of many applications that can be designed according to the (DS)$^2$ principle.

Most of the functionality in P2Pedia is handled directly by the generic (DS)$^2$ functions, which suggests that a good system architecture would be a layered architecture where the wiki-specific functionality is a thin “application layer” implemented on top of a generic and reusable infrastructure, implementing the (DS)$^2$ model and functionality.

In a centralized deployment, this infrastructure could largely be realized by a single graph database, storing the graphs $G_S$, $G_A$ and $G_D$, along with a relatively simple access control module.

As discussed in chapter 2, the physical infrastructure is to some extent orthogonal to the system functionality as defined by the (DS)$^2$ model. However, the (DS)$^2$ conceptual model in a centralized deployment is not complete without an additional agent, the service provider, looming in the background, as shown in our model of
OSN of section 2.3. The service provider is an authority that may arbitrarily delete data, ban users, or shutdown the entire system.

For this reason, we have rather focused on peer-to-peer deployments, where the users are in full control of their data. The different components of the (DS)$^2$ conceptual model map well to a P2P deployment: the users can store their own data, and connect to their social acquaintances, as in friend-to-friend (F2F) networks. However, existing F2F networks are simple file-sharing applications, that cannot handle structured data, and do not expose a programmable interface to extend their functionality.

However, a F2F protocol is certainly a useful building block to build a (DS)$^2$ infrastructure. If we break down the expected functionality of a (DS)$^2$ platform into such building blocks, we obtain a layered architecture, including a F2F protocol as a base network layer, a database as a base storage layer, and the functionality of the (DS)$^2$ model layered on top of these, up to the application layer.

This chapter presents our proposal of such an architecture, and a reference implementation, that exposes the (DS)$^2$ functionality through an API following the principles of the REST architectural style (REpresentational State Transfer [94]).

REST was designed primarily for networked applications on the web, connecting clients such as browsers to remote application servers. In our case, the server side of the interface is the generic (DS)$^2$ platform, and the clients are (DS)$^2$ applications that make use of this data-sharing functionality through its API.

The networked interface strongly decouples the application layer from the data-sharing functionality, and allows the two components to be implemented in different languages and to be deployed in very flexible ways, either closely integrated or on different machines running different OS.

We first discuss some related work, i.e. other frameworks for P2P applications exchanging structured data, in section 5.2.

In section 5.3 we provide some background about the REST architectural style.
In section 5.4 we present the overall platform architecture, showing how the different functional components are organized, then we detail each of these components in a systematic way, in sections 5.5 through 5.9. For each component, we give its implementation and specify the corresponding part of the REST API.

5.2 Related Work: Frameworks for Data Sharing Applications

Our proposal can be compared to a number of other frameworks for data sharing applications that extend P2P file-sharing technologies for structured data.

The following systems do not exactly implement the full (DS)$^2$ system model, as they lack a proper social network, but with each of these frameworks, applications can be built making use of an underlying data-sharing network.

SWAP

SWAP [95] was an EU project to share data in a peer-to-peer network, where the shared data was described using reference ontologies for the domain of interest. One application was Bibster [96], an application to share bibliographic data. The users can express queries using two ontologies, one of computer science research topics, and one of the bibliographic domain (describing the concepts of author, editor, different types of publications, etc.), in order to locate Bibtex records which they can then download. The domain ontology allows the users to easily locate data relevant to particular topics defined within this ontology. However, this framework does not provide any control over the P2P connections, or any other form of social networking.
**DBin**

DBin [97] is designed for RDF data, and presented as a framework for P2P Semantic Web Applications. DBin is based on the OSGi framework\(^1\). Advanced users can build OSGi/DBin applications called “brainlets”, and the users of each brainlet form a community that exchanges RDF data of some domain of interest. The brainlet code provides a user interface through which users can interact with the data, using built-in queries or query templates. Queries in the P2P network are used to locate RDF data related to specific URIs, and more complex queries (SPARQL) are executed only on local data. DBin addresses the problem of trust by including a digital signature which can be associated to a minimal RDF graph fragment, suggesting that those who sign a fragment “endorse” it. The act of sharing is independent from the act of signing. Peers can use trust policies (e.g. blacklisting specific people and ignoring their contributions...) when importing data.

**U-P2P**

U-P2P [77, 78, 98, 99] was designed at Carleton University, developed for several research projects, and is a foundation of our current work. U-P2P is a framework for sharing XML data, and also uses a notion of *community*, i.e. groups of users sharing data of some domain of interest. Each community is defined by a simple schema, representing a concept. Within a community, documents encoded in XML conform to the schema and represent instances of the concept. For each community, templates are used to render the shared documents in a web browser, and javascript can be used to further extend the functionality of the community. Users can be part of multiple communities, and can also publish the community definition in the network, in the “community of communities”, a kind of bootstrap community that all users are part of. A benefit of this approach is that it allows applications, implemented as

\(^1\)The framework underlying the Eclipse IDE
communities, to be discovered and downloaded just like other data.

The platform presented here goes much farther in decoupling application-specific functionality from generic data-sharing functionality, thus supporting a much less constrained class of applications. However, this implies that applications cannot be discovered and seamlessly “installed” through the P2P network, as was the case with U-P2P. In addition, we have modified the P2P protocol to give the users control over their connections, thus producing a “friend-to-friend” network protocol, and introduced messages to implement social trust indicators (discussed in chapter 4), which would have been irrelevant without the social network.

5.3 Background: REST

REST was proposed by R. Fielding as an architectural style for web applications [94]. One of the key ideas is that a web application should expose a set of named resources, that clients can access and manipulate through representations (hence the name REpresentational State Transfer). A client can make a request to inquire the current state of a resource, in which case the server returns a representation of the resource, or can modify the state of a resource, by sending a representation of its target state.

In our case, the API is designed to expose the accessible elements of the (DS)² logical structure, namely the graphs $G_S$, $G_A$ and $G_D$, as well as the progress and status of potentially slow operations such as queries and file transfers.

The main constraints of REST are the following [94]:

- client-server: the architecture style differentiates client and server components, the difference being that a client initiates communication by making a request, whereas a server listens for connections and responds to requests in order to supply access to its services.
• stateless server: the server maintains the state of the resources it exposes, and the client is responsible for maintaining the state of the session.

• self-descriptive messages: the messages (requests and responses) should contain all the information necessary to process them. This constraint also supports the statelessness of client-server communication: as messages contain all the information to process them, the server doesn’t need to maintain a context to understand them.

REST traditionally makes use of HTTP communication, and the verbs GET, PUT, POST, and DELETE. Their general use is outlined in the following:

• GET is used to retrieve a representation of the addressed resource.

• PUT is used to replace the existing resource by the one passed in the request.

• POST is used to change the state of a resource. POST can also be used to create new resources, by adding them to a “collection” resource.

• DELETE is used to delete resources (either elements in collections or entire collections).

In the following sections, we detail the different functional components of the platform, with their implementation, and the resources they expose to the API.

These resources are named using HTTP URIs, and their representations are expressed in XML. For simplicity, we will omit here the generic part of the URIs (that address the platform) and only indicate the part that distinguishes the different resources relevant to a given application.
5.4 Layered Architecture

We now detail the architecture for our P2P data sharing platform, decomposed into functional layers, as illustrated in figure 5.1. We briefly explain the role of each functional layer, before detailing our current implementation of each layer in the rest of this chapter.

![Layered Architectural Model for (DS)² applications.](image)

**Base Network Layer**  The base network layer represents the interconnection of online peers. It is an abstraction of the underlying physical network, which offers primitives to manage P2P connections and route messages in this network.

**Base Storage Layer**  The base storage layer is alongside the base network layer. It is an abstraction of a storage device such as a database, and its data model must store data resources. It exposes primitives to store and retrieve local resources, and execute simple search queries over the local collection of resources.

**Social Networking Layer**  The Social Networking Layer exposes the social graph $G_S$. It automates the management of P2P connections with available peers, and exposes functions to modify and query $G_S$. 
Data Sharing Layer  The data-sharing layer builds on the underlying storage and social networking layers, to expose the global set of data resources and the peers that are associated with them, i.e. the graphs $G_S$ and $G_A$. It exposes functions to retrieve available data resources regardless of their location, and to retrieve social information (e.g. distance, popularity) about the peers that provide these resources.

Structured Data Management Layer  The structured data management layer manages complex queries over the graph of data $G_D$, based on the set of data resources $R$ exposed by the underlying data sharing layer. This layer is responsible for coordinating the execution of complex queries using the underlying layers to access the distributed collection of data. The available query language and the query processing algorithm are detailed in chapter 6.

Application Layer  Finally, above the structured data management layer, a variety of applications can be built in the top-most “application” layer of our model. These applications can access all of the API functions. The main application example presented in this thesis proposal is P2Pedia, detailed in the previous chapter.

The following sections now detail our reference implementation of these functional layers, and the corresponding REST API that is exposed to applications at the top-most level.

5.5  Base P2P Layer

5.5.1  Layer Implementation

Our peer-to-peer network uses an adapted version of the original (v. 0.4) Gnutella protocol, which we have called Stracciatella. Peers are identified by 16-byte identifiers intended to be globally unique. A peer identifier is generated the first time that the
peer software is run, and is reused in future sessions\textsuperscript{2}.

Stracciatella uses most of the basic messages of Gnutella, but is adapted for social sharing: the main difference is that Stracciatella uses directed connections, and does not automatically add new connections to arbitrary peers.

Instead, the connections are delegated to the social networking layer that is logically above it.

\textbf{5.5.2 Resources}

The resources exposed to the application layer are the local peer’s active connections (i.e. edges adjacent to this peer in the graph of online peers), and a list of other peers observed in the network.

Individual peers can also be addressed by their unique identifier. The lists of peers are populated automatically and as such these resources cannot be directly modified via the API. The API is therefore a query-only API, allowing the user to discover new peers outside of the peer’s social network, and to know which of the peer’s neighbours are currently online.

The active connections can be indirectly modified by modifying the social network (see section 5.6).

The available resources and actions are summarized below.

\textsuperscript{2}this is the intended behavior. There is clearly a security weakness here as identities are cheap \cite{100}. However such security issues are beyond the scope of this thesis.
5.6 Social Networking Layer

5.6.1 Implementation

The social network is managed through the concept of a “friend list”: by declaring a peer as a “friend”, a peer establishes an outgoing edge in the social network graph $G_S$.

At the underlying network level, the local peer will attempt to establish a connection to each of its online “friend” peers whenever it goes online, then periodically polling all the non-connected friends for the duration of the session.

5.6.2 Resources

The actions available to the peer are to modify the social network by adding a peer to the friendlist, or removing a peer from this list.

Adding a peer is done by sending a “POST” request to the resource `peers/friendlist`, with a description of the peer. Removing a peer is done by sending a “DELETE” request, addressed to a URI designating the peer within the context of the friend list: `peers/friendlist/[guid]` The entire list can also be cleared by addressing a DELETE request to the URI `peers/friendlist`.

The available resources and actions are summarized below.

<table>
<thead>
<tr>
<th>URI</th>
<th>Resource</th>
<th>Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/peers/connections/</code></td>
<td>the list (collection) of active connections</td>
<td>GET</td>
</tr>
<tr>
<td><code>/peers/hostcache/</code></td>
<td>the hostcache: list of known peers</td>
<td>GET</td>
</tr>
<tr>
<td><code>/peers/[guid]</code></td>
<td>the peer with id [guid]</td>
<td>GET</td>
</tr>
<tr>
<td>URI</td>
<td>Resource</td>
<td>Verbs</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>/peers/friendlist/</td>
<td>the list of friend peers</td>
<td>GET, POST, DELETE</td>
</tr>
<tr>
<td>/peers/friendlist/[guid]</td>
<td>the entry [guid] in the friendlist</td>
<td>GET, DELETE</td>
</tr>
</tbody>
</table>

5.7 Base Storage Layer

The base storage layer allows the peer to store data resources. Conceptually, all resources stored locally are associated to the peer in the graph $G_A$.

5.7.1 Implementation

The data resources stored by the peers are documents with a unique identifier. A document can be seen as a set of key-value pairs, as in the NOSQL “document database” model (e.g. couchDB), or, equivalently, as a tuple (or record) in the relational model. Each document has a unique identifier generated using the MD5 hash function applied to the document content, and as such documents are immutable with respect to their identifier. The documents are serialized in XML and stored in an XML database.

The query language that can be used in search requests (local queries) is based on predicates over the key-value pairs. A query includes a key and a boolean expression: if the document contains any value associated with the given key and matching the boolean expression, then the document is considered to match the query.

For example, a simple search on page titles in P2Pedia can be expressed by the predicate `title~="Sand"`, where the predicate `~=` represents substring matching.

The structured data layer builds onto this model by allowing values referencing
other documents (in the key-value pairs). This creates an acyclic graph of documents, as presented in previous work [78] and summarized again in section 5.9.

### 5.7.2 Resources

The documents (data resources) stored by each application (several applications may be deployed over the same platform) are grouped in a list, that can be addressed by the URI `/data/`.

Documents can be added to the collection by a POST request to that URI, containing the document. The document identifier is returned as a response and can then subsequently be used to retrieve the document, via the URI `/data/[docid]/`.

The available resources and actions are summarized below.

<table>
<thead>
<tr>
<th>URI</th>
<th>Resource</th>
<th>Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/data/</code></td>
<td>the list of documents stored locally</td>
<td>GET, POST,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DELETE</td>
</tr>
<tr>
<td><code>/data/[docid]</code></td>
<td>the document [docid] in the friendlist</td>
<td>GET,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DELETE</td>
</tr>
</tbody>
</table>

### 5.8 Data-Sharing Layer

The data sharing layer exposes the available data through a global search function, where search results are associated with peers and social trust indicators about these peers.

Remote resources can also be retrieved, which copies them to the local peer’s storage.
5.8.1 Implementation

The search protocol implemented by our platform is based on the early Gnutella protocol, adapted for a directed network graph. Searches from the querying peer are sent to all the peer’s outgoing neighbours, and propagated following the directed edges of the social network, up to a limited depth. Duplicate messages are dropped, ensuring that searches are propagated at most once across each network connection. Responses are routed back asynchronously along the path of the queries.

Search response messages also include information about the responding peers, thus providing information about the graph $G_S$, which is not directly visible beyond the local peer’s immediate neighbourhood.

Resources are retrieved by downloads over direct HTTP connections, and two strategies are implemented to get around firewalls and NATs (Network Address Translators). The first strategy is the use of “PUSH” messages as in Gnutella, whereby a peer that cannot initiate a download connection can request the target peer to initiate the connection instead. If both peers are behind firewalls, a third peer reachable from both can be used as a relay. These different methods can be attempted for each peer offering the content.

5.8.2 REST resources

Search results arrive asynchronously, and in the protocol, no termination signal indicates that the results are complete (i.e. that no more will arrive). Similarly, downloads are operations that may require several attempts and can take a long time. For these reasons, it is impractical to implement search and download requests through synchronous GET requests.

Instead, we have used an alternative approach based on the concept of search and download sessions, which are exposed as REST resources.
The application can initiate a search session or a download, using a POST request. The response returns an identifier for the session, and it can later be terminated using a DELETE request applied to a URI constructed with this identifier.

Multiple search and download sessions may exist in parallel. For each search session, a list of results is available, which can be polled at any time using a GET request to a URI constructed from the search session identifier.

For downloads, one session is associated to each ongoing download, and its status can also be polled. The status of a download reflects which methods have been attempted with the different peers, the progress of the transfer, or a failure.

These resources and actions are summarized below.

<table>
<thead>
<tr>
<th>URI</th>
<th>Resource</th>
<th>Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/search/</code></td>
<td>the list of current search sessions</td>
<td>GET, POST, DELETE</td>
</tr>
<tr>
<td><code>/search/[searchid]</code></td>
<td>the search session with identifier [searchid]</td>
<td>GET, DELETE</td>
</tr>
<tr>
<td><code>/search/[searchid]/results</code></td>
<td>the results of the search session with identifier [searchid]</td>
<td>GET</td>
</tr>
<tr>
<td><code>/transfer/</code></td>
<td>the list of current download sessions</td>
<td>GET, POST, DELETE</td>
</tr>
<tr>
<td><code>/transfer/[transferid]</code></td>
<td>the download session with identifier [transferid]</td>
<td>GET, DELETE</td>
</tr>
</tbody>
</table>
5.9 Structured Data Management Layer

In our layered architecture, the base storage layer is a general storage medium that can accommodate a variety of structured data models, where data items of a nominal granularity are stored with an associated unique identifier. This simple model allows the data distribution to be managed independently of the structured data model and complex queries, that may access data from multiple sources.

5.9.1 Data Model

Our structured data model, previously described in [78], simply extends the basic “document-oriented” data model with references between documents: the key-value pairs in a document may include values referencing other documents.

As a result, the graph of data $G_D$ is defined as follows. Each node in the graph maps to one document (data resource). The node is identified by the document $id$, and is annotated with the key-value pairs in the document. Key-value pairs that reference other documents represent outgoing edges from this node, where the key is the edge label, and the value identifies the neighbour node.

As an example, we refer the reader to the example P2Pedia pages shown in section 4.3.2.

5.9.2 Modifying the Graph of Data

We note that the graph of data cannot be directly modified (e.g. adding or removing an edge). Instead, the graph of data must be indirectly modified by manipulating the data resources (see section 5.4).
5.9.3 Queries

The graph of data $G_D$ can be queried by a class of graph queries called Regular Path Queries (RPQs), detailed in chapter 6.

As RPQ are monotonic, the query answers will be obtained incrementally during the processing of the query (i.e. additional processing will not remove any of the previously obtained results), and as for simple searches RPQ can be managed as sessions.

5.9.4 REST Resources

The corresponding resources and actions are summarized below.

<table>
<thead>
<tr>
<th>URI</th>
<th>Resource</th>
<th>Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>/rpq/</td>
<td>the list of current query sessions</td>
<td>GET, POST, DELETE</td>
</tr>
<tr>
<td>/rpq/[queryid]</td>
<td>the query session with identifier [queryid]</td>
<td>GET, DELETE</td>
</tr>
<tr>
<td>/rpq/[queryid]/results</td>
<td>the results of the query session with identifier [queryid]</td>
<td>GET</td>
</tr>
</tbody>
</table>

5.10 Conclusion

We have presented a P2P data-sharing platform that can support multiple (DS)$^2$ applications, providing access to the basic data sharing functionality of a (DS)$^2$ through a REST interface.

This platform builds on a previous P2P framework to share graph data. This
framework defined an application as a data schema and customized stylesheets to render this data in a web browser. The REST/HTTP interface proposed here can support a much wider range of applications, built using arbitrary languages and even running on different machines.

The constraints defining the REST architectural style induce a number of desirable properties for a networked client-server application, such as reliability, visibility, separation of concerns between components, and evolvability of components [94]. At a less technical level, the use of REST principles, well known in the IT community, can make the platform and API easier to understand for potential adopters.
Chapter 6

Processing Regular Path Queries in a $(DS)^2$

6.1 Introduction

As the example of P2Pedia shows, building useful applications on a $(DS)^2$ infrastructure – or a data sharing infrastructure in general – requires applying expressive queries to the shared data.

While distributed architectures have several valuable properties (no single point of failure, shared cost, etc.), processing complex queries in a distributed setting is much more challenging than in a centralized system.

When the distribution of data can be controlled, query execution algorithms can be optimized to make use of the known location of data (e.g. key values may be mapped to network locations). However, the $(DS)^2$ approach allows the participants full autonomy in choosing which data they store locally, and as a result the shared data can be arbitrarily distributed in the network, making many existing query processing techniques inapplicable.

In this chapter, we study the problem of processing Regular Path Queries (RPQ), an important class of graph queries, in the $(DS)^2$ setting.
Regular Path Queries (RPQs) were first introduced as part of the G query language for graph databases [84,101], and gained particular interest as a way of querying the distributed graph formed by the World Wide Web pages and hyperlinks [102–104]. More recently, RPQ were included in version 1.1 of the SPARQL query language (“property paths”) for the Semantic Web, prompting renewed interest [105,106].

Useful examples of RPQ occur in our P2Pedia application: several examples are given in section 4.3.4.

In the Semantic Web context, much research has gone towards centralized query processing [105,106], as part of a “data warehousing” approach, where all of the available data is collected in advance from the known sources, then stored and pre-processed for local query execution. While such approaches allow fast query answers, they require massive resources to collect and store the data, which is only possible for large organizations (e.g. search engines). In addition, it can be difficult to keep up with data sources that change or add new data frequently, meaning query answers may be out of date.

Distributed algorithms for RPQs have been presented for the Web [103,107] and for distributed graph databases [108]. However, these algorithms rely on the key assumption that the data is *localized*, in the sense that each node in the graph of data is found in a single, known location, and the edges pointing to a node indicate its location. This is the case for example with the graph of Web pages and hyperlinks: each page is a node, and edges are hyperlinks, which indicate the location of the target node (along with its outgoing edges). In this context, a path can be traversed simply by following hyperlinks, dereferencing the HTTP URIs in each link.

This notion, illustrated in figure 6.1, is central to our study, as it largely determines the applicability and cost of various query processing strategies.

In fact, even when data can be described as localized, a difficulty occurs if we want to use the “inverse” operator, which extends RPQ to include paths traversing
edges in the "reverse" direction [82,109]. In the Web context for example, traversing a hyperlink in the reverse direction would require finding all the pages that link to a given page, and even Google may not be able to achieve this feat.

In other words, for Regular Path Queries with Inverse on distributed data, data can be considered localized only if the distribution and placement of data is controlled in a centralized way, for example by a shared algorithm. For example, in the DHT-based RDF repository RDFPeers [110], each RDF statement is replicated in three locations, corresponding to the hashes of the subject, predicate, and object resources. This way, any statement related to a given resource is guaranteed to be found at the location responsible for this resource.

As we have mentioned above, most existing distributed algorithms for RPQ rely on localized data, and are thus not directly applicable to the (DS)$^2$ setting. In this chapter, we discuss how they might be adapted, and evaluate their cost. We show that
for all existing algorithms, in the worst case the amount of network traffic generated during the query execution is greater than that required to collect all of the data locally.

Although this message cost can be described as linear in the data size, in practice it is unmanageable. However, while for each query processing strategy there exist “worst case” datasets and queries, in practice we can expect the majority of queries to be more selective.

Ideally, we would like a query processing strategy that performs well on “real-life” cases, and we would also like to determine conditions to choose between the different strategies, and identify a priori the problematic, worst-case queries.

We present a query processing strategy that has a low cost for selective queries, and propose a technique to estimate the “selectivity” of queries from a sample of the data. We evaluate our techniques using a real dataset from the biomedical domain and a set of meaningful queries.

The rest of this chapter is organized as follows. In section 6.2 we provide some background definitions and algorithms for RPQ in general. In section 6.3 we evaluate existing query processing techniques for the (DS)$^2$ setting. We then present an alternative technique in section 6.4, with a parallel version of the algorithm that is well-suited to run on top of asynchronous peer-to-peer network protocols. In section 6.5, we evaluate and compare two query execution strategies against the biomedical data, and determine conditions to choose between the strategies. We present our query cost evaluation techniques in section 6.6. Finally, we illustrate the use of all these techniques in a brief case study (section 6.7), and draw some conclusions in section 6.8.
6.2 Definitions and Algorithms for RPQs

RPQ were briefly introduced in section 3.3.3 and illustrated with two examples in section 4.3.4. We provide more complete definitions here, with several variations: single- and multi-source queries, regular path queries with inverse.

6.2.1 Notations

All queries are applied to an edge-labelled directed graph $G_D = \langle V, E \rangle$, where $V = \{v_0, v_1, \ldots, v_N\}$ are nodes and $E \subseteq V \times \Delta \times V$ are edges labelled from a set of labels $\Delta = \{\delta_i\}$.

A path in $G_D$ from a node $v_0$ to a node $v_k$ is a sequence of adjacent edges $(v_0, \delta_1, v_1), (v_1, \delta_2, v_2), \ldots, (v_{k-1}, \delta_k, v_k)$ starting at node $v_0$ and ending at node $v_k$.

The notation $v_0 \xrightarrow{w} v_k$ indicates that there exists a path from $v_0$ to $v_k$ such that the sequence of edge labels $\delta_1, \delta_2, \ldots, \delta_k$ along this path forms a word $w$. If $r$ is a regular expression, we note $L(r)$ the regular language defined by $r$.

6.2.2 Definitions

Definition 2 (Multi-source Query) A multi-source query $Q_r$ is defined by a regular expression $r$ over $\Delta$. When $Q_r$ is applied to the graph $G_D$, the answers to $Q_r$ are defined as follows:

$$\text{Ans}(Q_r, G_D) = \{(v_i, v_j) \in V \times V | v_i \xrightarrow{w} v_j, w \in L(r)\}$$

Definition 3 (Single-source Query) A single-source query $Q_{r,v_0}$, applicable to the graph $G_D$, is defined by a regular expression $r$ over $\Delta$, and a distinguished node $v_0$ of
The answers to $Q_r$ are defined as follows:

$$\text{Ans}(Q_{r,v_0}, G_D) = \{ v_j \in V | v_0 \xrightarrow{w} v_j, w \in L(r) \}$$

### 6.2.3 Regular Path Query With Inverse

The notion of a Regular Path Query can be extended to the notion of a Regular Path Query with Inverse (RPQI [109], also sometimes called 2RPQ [82]), when directed edges can be traversed in both directions (see the example of the P2Pedia sibling query in section 4.3.4). Such queries can be particularly useful in directed acyclic graphs, since for any pair of nodes related by a path $v_1 \xrightarrow{w} v_2$, there is no path from $v_2$ back to $v_1$. Using the “inverse” operator allows an extended notion of a path to exist between $v_2$ and $v_1$.

Conceptually, adding the inverse operator amounts to duplicating the edges in the graph, adding, for each existing edge, a parallel edge pointed in the opposite direction. The additional edges semantically represent the inverse relation of those represented in the original graph. We construct the labels of the new edges in a systematic way, by appending a “$-1$” superscript to the existing labels.

Formally, we consider the extended alphabet $\Delta'$ defined by extending $\Delta$ with an additional symbol $\delta^{-1}$ for each symbol $\delta$ in $\Delta$.

For a given graph of data $G_D$, we construct the extended graph $G'_D = \langle V, E' \rangle$, where $E'$ is defined as:

$$E' = E \cup \{(v_j, \delta^{-1}, v_i) | (v_i, \delta, v_j) \in E\}$$

RPQI applied to $G_D$ are then simply RPQs on $G'_D$.

**Definition 4 (Multi-source RPQI)** A multi-source RPQI query $QI_r$, applicable to the graph $G_D$, is defined by a regular expression $r$ over $\Delta'$. When the query is
applied to the graph $G_D$, the answers to $Q_{I_r}$ are the answers to the RPQ $Q_r$ applied to $G'_D$.

$$\text{Ans}(Q_{I_r}, G_D) = \text{Ans}(Q_r, G'_D)$$

Single-source RPQI are defined in a similar way.

### 6.2.4 Example

As an example, we consider the graph of data shown in figure 6.2, and the following example queries, applied to this graph of data:

- $Q_1 = (1, a^*bb)$ is a single-source query that requests all nodes that can be reached from node 1 by a path matching the regular expression $a^*bb$. An automaton for this regular expression is shown in figure 6.3. The answers to the query are nodes 5 (path 1-4-5, $bb$) and 8 (path 1-2-6-9-3-8, $aaabb$). For node 8, there exist an infinite number of paths yielding this result, including any number of traversals of the cycle 2-6-9-2.

- $Q_2 = (ac(a|b))$ is a multi-source query that requests all pairs of nodes related by a path matching the regular expression $ac(a|b)$. The answers to this query are the pairs of nodes $(1, 5), (9, 5)$ (both paths labelled $aca$), $(1, 8), (9, 8), (2, 7)$ (paths labelled $acb$).

- $Q_{I_3} = (1, a^*b^{-1})$, is a single-source RPQI, where $b^{-1}$ denotes following an edge labelled $b$ in the opposite direction. Answers to $Q_{I_3}$ are nodes 4 and 7 (paths 1-2-5-4 and 1-2-6-7). Again, the cycle 2-6-9-2 can be added to both of these paths.
6.2.5 Basic RPQ Algorithm

The main algorithm to answer such queries (detailed below) was sketched in a 1989 paper by Mendelzon and Wood [111].

We detail their general algorithm, which is the basis of most approaches (as we will show later on). We will refer to this algorithm as the product automaton algorithm, or PAA:

1. build a finite automaton $A_1$ associated with the regular expression $r$. The initial state of $A_1$ is $q_0$, the accepting states are $\{q_f\}$.

2. consider the graph of data as an automaton $A_2$ (nodes $\rightarrow$ states, edges $\rightarrow$ transitions), and compute the cross-product of the automata $A_p = A_1 \times A_2$.

3. • **(single-source RPQ):** the initial node in the graph is set ($N_0$): search $A_p$ from the initial state ($q_0, N_0$) to find all reachable accepting states ($q_f, N_j$). All nodes ($N_j$) are answers to the single-source query.

   • **(multi-source RPQ):** we are looking for all pairs of nodes related by the regular path: search $A_p$ from all initial states ($q_0, N_i$) to find all reachable accepting states ($q_f, N_j$). All pairs of nodes ($N_i, N_j$) are answers to the multi-source query.
For step 3, any graph search algorithm can be used, such as breadth-first or depth-first.

### 6.2.6 Application to RPQI

Since RPQI query answering can be reduced to answering RPQ on another graph, the same algorithm can be used. However, the graph $G'_D$ is an artifact of the definition and method, and is not materialized as a database. Traversing an inverse edge amounts to traversing an existing edge in the opposite direction, and depending on the data model, this may not be possible.

In a Web setting, this is prohibitively costly, because it means finding the hyperlinks pointing to a given page, which could be anywhere. Discovering all such links would require crawling the entire Web. On the other hand, distributed graph databases can be designed to efficiently support edge traversal in both directions, which is required for graph query languages such as Gremlin\(^1\).

### 6.2.7 Complexity

For database query languages, the traditional parameters considered in a complexity study are the data size and the query size [112]. The cost of executing a fixed query, as a function of the data size is the data complexity, whereas if both the query and database are considered variable then the cost is the combined complexity in Vardi’s taxonomy [112].

The size of the data graph has two parameters, the number of nodes $|V|$ and the number of edges $|E|$.

For a regular path query, the query size is the number of characters and operators in the regular expression [113], which we will note $m$.

---

\(^1\)http://gremlin.tinkpop.com
Based on these parameters, the cost of the PAA algorithm is the cost of building the query automaton, plus the cost of building and searching the product automaton. In practice, only a subset of the product automaton will be searched and built on the fly. However, in the worst case the full product automaton is reachable and must be built and searched.

A non-deterministic query automaton can be built in $O(m)$ time and has $O(m)$ states [113]. The product automaton will have $O(|V|.m)$ states and $O(|E|.m)$ transitions.

The complexity of a graph search (BFS or DFS) over this automaton is therefore $O(|E|.m + |V|.m)$. The total combined complexity of the PAA algorithm is therefore $O((|E| + |V|).m)$. We note that a variation of RPQ where the paths should be regular simple paths (paths where nodes are not repeated) is NP-complete [111].

### 6.2.8 Optimizations

To our knowledge, two approaches for optimizing the PAA algorithm have been proposed.

The first one, introduced by Fernandez et al. in [107], is based on graph schemas, i.e. partial knowledge of the graph structure. A graph schema can indicate that some paths, although they match a prefix of the regular expression, will not lead to a solution. These paths can then be pruned from the search space, and thus reduce the cost of the search.

The second one, by Koschmieder [114,115], is based on knowledge of the frequency of different labels in the graph of data, and reduces the subgraph explored in the query.

If the regular expression contains a label known to be rare, the regular expression is split into smaller queries around the occurrences of the rare label, which are then used as “waypoints” for the query execution.

For example, for our example query $Q_2$ using the regular expression $ac(a|b)$, the
label \( c \) is rarer than \( a \) or \( b \), only occurring three times in the entire graph (whereas \( a \) and \( b \) occur 6 times each). The optimal query execution would then start from all edges labelled \( c \), which are the edges 4-3, 2-3, and 6-8, then search backwards for \( a \) (i.e., \( a' \), using the notation we introduced for RPQI) and forwards for \( a|b \). This produces only 3 starting points, as opposed to 6 if the search starts from every \( a \) edge, and 12 if the search started from all candidates for the last edge, which may be labelled \( a \) or \( b \).

This approach assumes that edges can be efficiently traversed in both directions (as discussed for RPQI), and is also mainly efficient for multi-source queries: for single-source queries, the starting point occurs only once, and therefore constitutes the best rare waypoint to begin the traversal from.

For both of these optimizations, it is worth noting that the basic PAA algorithm is still used. In the first case, the optimization helps prune the search of the product automaton \( A_p \), while in the second case the main query is split into smaller subqueries that are also executed using the PAA.

We can therefore consider the PAA algorithm to be the fundamental basis of RPQ processing, and in the rest of this paper we will focus on adapting this algorithm to the distributed setting, without further consideration for these optimizations.

### 6.3 RPQ Processing on Distributed Data

#### 6.3.1 Overview

Research towards querying large-scale, distributed graph data has been motivated by the emergence of the World Wide Web – a distributed graph of pages and hyperlinks – and more recently of the Semantic Web – a distributed graph of RDF resources and properties. We briefly present the different approaches that have been proposed in
these different contexts.

In the Semantic Web context, much research has gone towards “data warehousing” approaches, where all known available data is collected in advance and pre-processed to optimize local query processing. In this approach, the research on RPQ query processing focuses on optimizing the local data storage and indexing [106, 116].

An advantage of this approach is that local query processing is fast, and applicable to any query language, only requiring a basic data retrieval service from the data source sites (as opposed to complex query processing capabilities). However, this approach is only possible for organizations with large storage capabilities, such as Semantic Web search engines ([117, 118]), and has the additional disadvantage that query results are necessarily out of date, since the data was collected sometime in the past. Finally, if the querying agent is not the organization managing the data, then it has little or no control over which data is used to answer the query, unless the stored data is augmented with provenance meta-data. In an open system of autonomous participants, provenance and trust considerations may be important.

When collecting all the available data is considered too costly (e.g. assuming the dataset to be infinite in size, as proposed by [102]), an alternative approach is to identify and retrieve only the data needed for each query. Based on this idea, Mendelzon et al. [104] propose a “locality” metric for queries, to estimate the number of sites that need to be queried (and hence the amount of data that will be retrieved) to answer a query. Several studies on the processing of SPARQL queries [119,120], are based on the same idea and propose techniques to discover and exploit new relevant data sources during the query processing. These studies do not explicitly address RPQs, but could be extended for this purpose.

This general strategy, while necessarily slower than the previous, will have a lower message cost (i.e. less data will need to be transferred over the network), in the best cases. However, in bad cases the full dataset may still be needed, or it may
be impossible to identify a smaller subset that could be sufficient. In particular, the techniques proposed to determine “data locality” in [104] rely on a distinction between “local” and “outgoing” edges in the query language itself, and on the underlying assumption that the data is localized, i.e. each Web page is available from a single, known location.

In addition to the centralized approach, two distributed query processing algorithms have been proposed. The first one, based on the idea of “query shipping”, was proposed for single-source queries on the Web [102, 103] and extended for richer queries (multi-source, with path weight minimization) on a distributed graph database [108]. In this approach, the query is recursively decomposed into subqueries, which are shipped to the sites where the relevant data is located. Again, it is assumed that the data is localized. In addition, in bad cases the query could be shipped back and forth between sites an arbitrary number of times (i.e. causing more message traffic than collecting the full data graph to one location).

As a solution to this problem, Suciu [121] has proposed a query decomposition approach where the query is decomposed into a set of subqueries to be executed at each site, such that their answers, once collected at the querying site, can be used to reconstruct the final query answers. This technique only requires one message exchange with each site, and could thus be more efficient than the query shipping approach. However, this approach relies on each site knowing which graph nodes are “incoming”, i.e. have edges elsewhere pointing towards them, and is even more constraining than the “localized data” assumption. Presumably, the sites could exchange and maintain this information, for example using a gossiping algorithm. However, the cost of maintaining this information must be factored into the cost of the query processing, and is unclear.

Finally, we note that these two distributed processing approaches share two important disadvantages: they require complex query processing capabilities at each
site, and although their cost may be low in the better cases, there is no clear way to identify bad cases, or to limit the cost in these bad cases.

Ultimately, none of these techniques were designed for a setting where the data is arbitrarily distributed (“non localized”). Aside from “data warehousing” approaches (which only require the data to be accessible), it is unclear whether these techniques can be adapted, and what their cost might be in this setting.

In this section, we analyze the different algorithms summarized above, in order to answer these questions. We review the centralized approach of [104], which we will call strategy S1, and the two main distributed algorithms, the “query shipping” approach of [102,103,108] (strategy S2) and the query decomposition approach of [121] (strategy S3). For each approach, we describe the algorithm and discuss its applicability to the (DS)$^2$ setting, highlighting the problematic cases. We compare the costs of the queries in section 6.3.4.

![Figure 6.4: A graph of data arbitrarily distributed and replicated between 3 sites.](image)

### 6.3.2 Example Database and Query

As an example, we consider the graph of data shown in figure 6.2, and the single-source query $Q_1$, also introduced previously:
\[ Q_1 = (1, a^* bb) \]

We give an example non-localized distribution of this data in figure 6.4.

Finally, note that we assume the sites to be accessible to one another, and to communicate via broadcast (one-to-all) or unicast (one-to-one) messages, as in a peer-to-peer network.

6.3.3 Query Processing Strategies for the \((DS)^2\) setting

**Strategy S1: collect relevant data locally**

The simplest strategy to process a query is to first retrieve all the necessary data, then execute the query locally. This strategy can be straightforwardly applied to the \((DS)^2\) setting.

However, the main challenge, which is to avoid retrieving the entire graph of data, or at least be able to detect cases where this may happen, cannot be tackled the same way as when the data is localized. This is the purpose of the “locality” metric introduced in [104], which largely relies on a distinction between local and non-local edges in the query language itself. This distinction is justified in the Web context, where it is trivial to know whether the target of a hyperlink is local or remote, and where local hyperlinks have arguably different semantics and purposes from outgoing links to other sites.

In the \((DS)^2\) context, the distinction is somewhat meaningless, and it appears difficult to assess the locality of a query: traversing a single edge could lead to a node replicated on every site.

It may be possible, instead, to estimate a form of query selectiveness, based on the labels that appear in the query. If the regular expression in the query doesn’t contain any wild-cards, only the edges with labels appearing in the query need to be
retrieved. If the data is uniformly distributed across the peers, then the selectiveness of a set of labels can be estimated on a sample of the data.

In our example using the query $Q_1$, applying this strategy would consist in first retrieving all of the edges labelled $a$ or $b$ (the symbols present in the regular expression $a^*bb$ of the query), rebuilding the resulting graph locally and applying the PAA algorithm.

In cases where every possible edge label appears in the query, or if the query contains a wild-card symbol (".") , this approach will result in the full graph of data being retrieved.

**Strategy S2: Shipping the query**

A second strategy for query execution is a distributed algorithm based on the idea of “query shipping”.

In this strategy, detailed in [103], during the graph traversal the query is shipped to the nodes hosting the edges being traversed.

We note that while the technique presented in [103] is designed for single-source queries, a similar idea is used in [108] for an extension to multi-source RPQ for weighted labelled graphs, where the path with minimal weight is also computed. We describe here the basic technique for single-source queries.

For our example, the query execution would proceed as follows:

1. site 1 would process the query locally, traversing the local sub-graph up to any solutions, or to edges leading to external nodes. Node 5 would be found as a solution (path 1-4-5, labelled $bb$) and the outgoing edge 2-6 would be detected.

2. In a setting where the data is localized, the query would then be shipped directly to the site hosting node 6. In the $(DS)^2$ setting, site 1 would have to broadcast a query to locate any instances of node 6.
3. In this case site 3 would respond, and site 1 would then ship the query to site 3. As node 6 would be reached after traversing a path of length 2 labelled $aa$, the subquery shipped to site 3 would be $(6, a^*bb)$: this regular expression describes valid suffixes of the currently traversed path $aa$, which will produce paths matching the initial regular expression.

4. Site 3 would then process the subquery and again reach an outgoing edge: edge 6-9. In this case, node 9 is found at sites 1 and 2. Shipping the query to both sites could lead to multiplication of the processing, so a more reasonable solution would be to randomly select one of the sites (e.g. site 1) to compute the rest of the query. The subquery being shipped would be $(9, a^*bb)$.

5. At site 1 the outgoing edge 2-6 would again be found, as well as the outgoing edge 9-3. Calculating the appropriate suffixes as above produces subqueries $(2, a^*bb)$ and $(3, b)$. The first of these subqueries has already been encountered, and can this time be dropped. The second subquery can be processed either by site 2 or by site 3. The final result 8 will then be obtained and returned as an answer to the query.

In this process, the query is shipped each time an edge of the graph must be traversed between two nodes found on different sites. In terms of message cost, the worst case would be the situation where no pairs of adjacent nodes in the graph were located on the same site. This would mean that in the third step of the PAA algorithm, each edge of the product automaton traversed will cause a broadcast to locate the target node, and a subquery being shipped between sites. As the product automaton is larger than the graph of data, this means that this technique may cause much more more message traffic than the message cost of collecting the full graph of data locally (the worst case of strategy S1).

We note that this technique was originally developed for the Web (the graph of
pages and hyperlinks), where not only is data localized, but it is also stored in such a way that each node is stored along with its outgoing edges. In the “good” cases, most of the edges being followed are local. However, if the data is stored following a data model such as RDF, then a node is only an identifier, and the edges adjacent to a given node can be located anywhere, together or separately. This implies that after traversing an edge, possible successors may be found anywhere: and a broadcast search is required every time.

It is also worth noting that with the Web data model, supporting the “inverse” operator will also cause additional searches, since incoming edges to a node can be located anywhere. This is presumably the reason why the inverse operator is not considered in [103].

**Strategy S3: query decomposition**

As strategy S2 may cause a large amount of traffic in shipping queries back and forth between sites, a better solution may be the query decomposition technique proposed by Suciu [121], which allows a query to be answered by exchanging only one pair of messages (subquery-answer) between the querying site and each other site.

The idea of this technique is to anticipate the query shipping by enumerating all the possible “suffix” subqueries (as explained for strategy S2), and applying the subqueries to all “incoming nodes”, i.e. nodes that have incoming edges from other sites. The results of all these subqueries can then be collected at the querying site to reconstruct the sub-graph traversed in the query execution.

The process is as follows:

1. The initial query is sent to each site.

2. At each site, the valid suffixes are computed, and the PAA is executed from the start node of the initial query (using only the initial regular expression), and
from each “incoming node” (using the suffixes), until either an answer to the query is found, or an outgoing edge is found.

3. The results are then returned to the requesting site, and reassembled.

For our example query $a^*bb$, the valid suffixes are $a^*bb$ (following any sequence of $a$-labelled edges), $b$, or $\epsilon$. This decomposition can be done using the query automaton (shown in figure 6.3), and taking each legal state (i.e., each state that may lead to an accepting state) as an initial state: the resulting automaton corresponds to a valid suffix.

In our example data distribution, the “incoming nodes” at site 1 are nodes 2, 5, and 9. The initial query would therefore be processed from node 1, and each subquery from “incoming nodes” 2, 5, and 9, producing the following results:

1. $bb \rightarrow 5$

2. $aa \rightarrow 6$

3. $b \rightarrow \emptyset$ (no results from this subquery)

4. $\epsilon \rightarrow 2$

5. $a^*bb \rightarrow \emptyset$ (no results from this subquery)

6. $b \rightarrow \emptyset$ (no results from this subquery)

7. $\epsilon \rightarrow 5$

8. $aa \rightarrow 6$ (subquery $a^*bb$)

9. $b \rightarrow 3$ (no results from this subquery)

10. $\epsilon \rightarrow 9$
Similar processing would happen at site 2 (incoming nodes 3, 9, 7) and site 3 (incoming nodes 2, 3, and 6). Among the results, site 2 would obtain $9 \rightarrow 8$ and site 3 would obtain $6 \rightarrow 9$. By combining these two results with the result $1 \rightarrow 6$ obtained at site 1, the querying site can infer the path $1 \rightarrow aabb \rightarrow 8$.

The main problem with this technique is that it requires each site to know which of its nodes have remote incoming edges, and to compare the technique fairly with the others, the cost of obtaining this information (and maintaining it up-to-date) must be factored into the query processing cost.

In addition, as nodes may be replicated in many places, the steps to obtain intermediate results are repeated in various places, and the amount of data returned to the querying node may be significant.

The worst case here would be as in strategy S2, the situation where pairs of adjacent nodes are found at different sites. This would mean that every node would be an “incoming node”. The cost of distributing this information would be high: every node would broadcast information about its outgoing edges, and if every edge is outgoing, then this amounts to the message cost of collecting the full graph of data at every site.

### 6.3.4 Complexity Comparison

In this section, we attempt to measure and compare the asymptotic complexities of the above distributed query processing algorithms.

#### Complexity Metrics and Parameters

Traditional cost metrics for distributed query processing are message complexity and response time. When the distribution is used to parallelize and speed up the processing, the response time is the more relevant metric; in this case however, the distributed
processing of S2 and S3 is used only to reduce the communication overhead of collecting all the data in a centralized location (S1). Therefore it is more relevant here to focus on the message cost. Ultimately, the communication overhead will also strongly affect the final response time.

As for centralized query processing, the size of the data graph (parameters $|E|$ and $|V|$) and the size of the query ($m$) are important complexity parameters.

In the $(DS)^2$ setting, the characteristics of the network (mainly its size) are also essential cost factors.

This network can be represented by a graph of $N_p$ nodes with $N_c$ edges. As we have mentioned previously, each data resource (atomic piece of data) may be available in zero to many arbitrary locations. However, we can consider that the data replication rate $k$ is a cost parameter: this means that on average, each data resource is found in $K$ locations, where $K = k.N_p$. (to distinguish these, we will call $K$ the replication factor, and $k$ the replication rate.)

We note that for asymptotic cost estimations, we will consider that $K = O(N_p)$.

**Communication Primitives**

In order to process a query, the peers (sites) can communicate in one of two ways: *unicast* or *broadcast* messages. Broadcasts are messages from one site to all the other sites, that are distributed using a peer-to-peer protocol. Unicasts are point-to-point messages between specific sites, and we consider their cost to be constant (independent of network size).

In the absence of network topology optimizations, and under the minimal assumptions of a reliable asynchronous network [122] the message cost of broadcasting a fixed-size message is $O(N_c)$. The time complexity of a broadcast is $O(N_p)$. We note that if multiple peers are broadcasting messages at the same time, it becomes difficult to account for network congestion.
An acknowledgement can be returned by all the peers using a “convergecast” protocol, whereby the query originator will know that all the peers have received the query; the message and time complexity are (asymptotically) unchanged [122].

Responses to the broadcast query can be sent as unicast messages.

As the centralized query processing algorithm (the PAA) is of linear complexity, the network communication will be the performance bottleneck in a distributed query execution. We therefore focus our comparison on the communication costs involved in the query execution.

Summary

These costs can be summarized as the amount of data being broadcast, and the amount of data being unicast.

We first detail the cost for each strategy, then compare them in table 6.1 (in section 6.4.3).

**S1:** In strategy S1, the data is retrieved by a single broadcast, and the size of this message is proportional to the size of the query ($O(m)$). In the worst case, the full graph of data is returned. As all the peers storing data will respond, this means this amount of data is multiplied by the replication factor $K$. With $K = k.N_p$, the amount of data being sent by unicast is $O(k.N_p.(|E| + |V|))$.

**S2:** In strategy S2, in the worst case the PAA algorithm is executed with one broadcast message and a number of unicast (response) messages at every step of the product automaton search. The product automaton has $O(m.|V|)$ nodes and $O(m.|E|)$ edges, therefore searching this graph requires $O(m.(|E| + |V|))$ operations. The size of the broadcast messages sent at each step is bounded, and the total amount of data being broadcast is therefore $O(m.(|E| + |V|))$. At each step the unicast messages are peers
responding to the broadcast. Altogether over the full query execution, messages are returned for all the nodes in the graph, multiplied by their replication factor $K$. The amount of data transmitted by unicast is therefore $O(m.k.N_p.(|E| + |V|)^2)$.

**S3:** In the worst case, every edge at every site may be an outgoing edge. This information must be broadcast, for sites to identify their “incoming” nodes, and therefore the message cost of this step is $O(k.N_p.|E|)$. The query execution itself can be efficient, with only one broadcast (of the initial query) and one response per site. However, in the worst case, which is that of each edge being “outgoing”, the intermediate results returned by the query could amount to the full graph of data, replicated $K$ times. The amount of data to be broadcast is therefore $O(k.N_p.|E| + m)$ and the amount of data sent by unicast is $O(k.N_p.(|E| + |V|))$.

**6.3.5 Conclusion**

The distributed query execution strategies were designed for the case where data is localized, and are poorly suited to the $(DS)^2$ setting. In addition, they assume query facilities at each site, which is problematic when the different sites are under independent control: there are no guarantees of cooperation.

When a simple retrieval service is assumed, a wrong assumption simply means that the data at this site is not available. When the sites are expected to cooperate and process part of the query, the failure to cooperate can compromise a large part of the query.

The study of worst-case complexity paints the somewhat naive strategy S1 in a very favourable light. However, its worst case still involves transferring the entire graph of data over the network, which is problematic. In addition, this worst case appears quite likely to happen: with our simple selection method, the mere presence of a wild-card in the query is enough to cause it.
In any case, it appears that with every strategy, we have an inescapable worst case where computing a query actually requires the full graph of data, and therefore the message cost can reach at least $\Theta(|E| + |V|)$.

With very large datasets, even this baseline complexity seems unmanageable. This motivates a search for alternative strategies to reduce the costs: can we anticipate very costly query processing by analyzing the query? Can we cap the cost by interrupting the processing once a limit has been reached?

For the second question, it seems clear that the answer is no, and this is a weakness of a distributed processing approach (and of S1): once the query is sent out, the querying agent has little visibility or control over the processing.

In the next section, we introduce an algorithm that shares with S1 the advantage that it does not require complex processing capabilities from other sites. In addition, our algorithm can be interrupted once a maximum number of messages has been exchanged. In the worst case, it is more costly than strategy S1. However, we anticipate that in practice, for “average” cases it performs better. We will evaluate this experimentally in section 6.5.

### 6.4 Asynchronous Iterative Centralized Query Processing

The strategy that we propose for this problem is the *iterative centralized processing* strategy, and we will simply call it strategy S4, following our study of the other three main strategies S1, S2 and S3. The idea of this strategy is to first construct the query automaton, then construct the product automaton (as in the PAA) while collecting the graph of data “on the fly” from the distributed data collection, accessing it as if it were the local disk storage.
6.4.1 Sequential Algorithm

We first consider the case where the data access API is synchronous, and can be represented by the following functions:

- \( \text{getOutNeighbours}(v_0 : \text{node}, a : \text{label}) : \text{nodeset} \): given a node \( v_0 \) and an edge label \( a \), this function returns the set of nodes \( \{ v_i \} \) in the graph such that there exists a directed labelled edge \( (v_0, a, v_i) \)

- \( \text{getInNeighbours}(v_0 : \text{node}, a : \text{label}) : \text{nodeset} \): this function enables RPQI by allowing edges to be traversed in the reverse direction: symmetrically to the previous, given a node \( v_0 \) and an edge label \( a \), returns the set of nodes \( \{ v_i \} \) in the graph such that there exists a directed labelled edge \( (v_i, a, V) \).

Algorithm 2 lists the sequential steps of the algorithm. First, the query automaton \( A_x \) is constructed from the regular expression \( x \) using Thompson’s algorithm [123]. We do not detail Thompson’s algorithm here; it produces an automaton for which we use the following notations: \( A_x = \langle Q_x, \Delta', \delta_x, q_0, F_x \rangle \) is the NFA that recognizes the regular language corresponding to \( x \):

- \( Q_x = \{ qx_0, \ldots, qx_n \} \) is the set of states;
- \( \Delta' \) is the input alphabet (see section 6.2.3);
- \( \delta_x \) is the state transition table
- \( q_0 \) is the initial state;
- \( F_x \) is the set of accepting states.

The product automaton is then constructed and searched using a depth-first search. We note that the graph search choice (depth-first) is an arbitrary choice, which will become irrelevant in the parallel version of the algorithm presented next.
Algorithm 2 Sequential query answering algorithm.

1: Function RPQI($v_0, x$):
2: $A_x \leftarrow$ Thompson($x$) //Construct $A_x$
3: $ans \leftarrow \emptyset$
4: $visited \leftarrow \{v_0\}$
5: stack.push([$v_0$, $q_0$]) //$q_0$ is the start node of $A_x$
6: // — Product automaton construction and search —
7: while true do
8: if stack.is_empty() then
9: return $ans$ //All reachable nodes have been visited
10: end if
11: [$v_c$, $q_c$] $\leftarrow$ stack.pop() // PA node to be visited
12: // — Automaton transitions —
13: for ($s, q$) in $\delta_x(q_c)$ do
14: if $s = \epsilon$ then
15: nodes $\leftarrow \{[v_c, q]\}$ //epsilon-transition
16: else
17: if $s \in \Delta$ then
18: nodes $\leftarrow$ outNeighbours($v_c$, $s$) //edge of $E$
19: else
20: nodes $\leftarrow$ inNeighbours($v_c$, $s$) //edge of $E^{-1}$
21: end if
22: end if
23: end for
24: for $v_i \in$ nodes do
25: if $v_i \notin$ visited then
26: stack.push([$v_i$, $q$])
27: visited $\leftarrow$ visited $\cup$ $v_i$
28: end if
29: end for
30: if $q_c \in F_x$ then
31: ans $\leftarrow$ ans $\cup$ nodes
32: end if
33: end while
34:
6.4.2 Example

With our example query $Q_1$, the query automaton produced by Thompson’s algorithm is shown in figure 6.5.

![Figure 6.5: NFA for regular expression $a^*bb$.](image)

Exploring the distributed graph of data used previously (see section 6.3.2) will proceed as follows.

The start node of the product automaton is $(1, q_0)$, since $(1)$ is the query start node and $q_0$ is the automaton start node. When the graph search starts, the transitions of $q_0$ are listed: $(a, q_0)$ and $(\epsilon, q_1)$. The next nodes in the product automaton are then obtained by traversing the graph of data from $(1)$ following edges with the labels of these transitions: $a$ and $\epsilon$.

Traversing the (data) edge $1 \xrightarrow{a} 2$, simultaneously with the automaton transition $q_0 \xrightarrow{a} q_0$ gives us the PA node $(2, q_0)$. This involves a call to the data access function $outNeighbours(1, a)$\(^2\), which retrieves node $(2)$ either locally or via a broadcast search. This is transparent to the algorithm.

For an $\epsilon$-transition, no edge in the graph of data needs to be traversed\(^3\), and thus we immediately obtain the PA node $(1, q_1)$.

These nodes get pushed to the stack. In the next iteration, the top node on the stack is popped (e.g. $(1, q_1)$), and the PA graph search continues. From this node, the only automaton transition is $q_1 \xrightarrow{b} q_2$, and therefore we must traverse edges labelled $b$ from node $(1)$. Querying the graph, we find the edge $1 \xrightarrow{b} 4$, leading us to the PA node $(4, q_2)$.

\(^2\)Note that for an “inverse” edge traversal, the algorithm would use $InNeighbours()$.

\(^3\)We remind the reader that $\epsilon$ represents the empty string.
In the next iteration we similarly get to \((5, q3)\), meaning graph node \((5)\) is a solution, since \(q3\) is terminal. However, there are no outgoing automaton transitions and the exploration of this branch ends. The next node on the stack is \((2, q0)\). The PA graph search continues from there, ultimately leading to the discovery of the rest of the graph and the other solution, node \((8)\).

### 6.4.3 Complexity

The complexity of the algorithm can be decomposed as follows. Thompson’s algorithm is linear in the length of the query \((O(m))\), and produces an automaton with \(O(m)\) nodes and \(O(m)\) edges. The product automaton therefore has \(O(m.|V|)\) nodes and \(O(m.|E|)\) edges, and searching this graph, depth-first or in another order, has a cost of \(O(m.(|V| + |E|))\), counting one operation to read each node or edge in the graph.

However, as the storage is not actually local, this “read” involves a broadcast search and several unicast responses. This would imply \(O(m.(|V| + |E|))\) broadcasts, and a total of \(O(m.k.Np.(|V| + |E|))\) responses sent by unicast. However, a simple optimization where the graph of data is cached locally (rather than being retrieved multiple times) reduces the number of broadcasts and unicasts to \(O(|V| + |E|)\) and \(O(k.Np.(|V| + |E|))\), respectively.

These figures can be compared with those of the previously described strategies. We add a line to table 6.1, which summarizes the message complexity analysis for the different query processing strategies.

For strategy S4, the time complexity of the algorithm can also easily be calculated, since it is sequential and has a clear critical path: the main time complexity comes from the broadcast messages, each completing in \(O(N_p)\) time (the unicast responses to each query are sent in parallel and complete in constant time). The total response time is therefore \(O(N_p.(|E| + |V|))\).
6.4.4 Parallel Algorithm

In the algorithm above, the data access operations constitute the main performance bottleneck. The problem is that in the worst case, every data resource we need is located at the far end of the network, and retrieving them takes $O(N_p)$ time. However, in practical cases we can expect the data to be distributed across the network, and to be returned well before we receive the last responses from the far end of the network. So for each broadcast, the set of responses can be provided either synchronously, as a whole after $O(N_p)$ time, or asynchronously, the first arriving quickly and the last after $O(N_p)$ time.

There is therefore an opportunity to speed up the algorithm by asynchronously making use of the early responses as they arrive, instead of blocking the algorithm during the synchronous completion of the `inNeighbours()` and `outNeighbours()` functions. This will require parallelizing the algorithm to handle the asynchronous responses.

In order to realize this optimization, we introduce an asynchronous version of the data access functions, and introduce multiple processes, interacting in a “shared memory” approach.

The synchronization is realized by a tuple space, and the different processes are implemented as workers reading and writing data to the tuple space.
**Tuple Space Workers**

The main primitive operations supported by a tuple space, first defined in the Linda coordination language [124], are the functions read(...) and write(...), defined as follows:

- **write(field\_1, field\_2, \ldots, field\_k):** output the tuple \( \langle field\_1, field\_2, \ldots, field\_k \rangle \) to the tuple space. The \( field_i \) can be CONSTANTS (we will represent constants using upper case) or variables (lower case) from previous processing.

- **read(field\_1, field\_2, \ldots, field\_k):** synchronously read any tuple matching the pattern \( \langle field\_1, field\_2, \ldots, field\_k \rangle \) from the tuple space (block until such a tuple appears)\(^4\). Here \( field_i \) can be CONSTANTS, previously set variables, or template fields, which we will note “? \( \rightarrow \) y”: a template field matches any value and saves this value to the variable \( y \), binding the value to \( y \) in subsequent processing. Other fields must be matched exactly.

The basic computing process of a tuple-space-coordinated algorithm is implemented as a “worker” that reads and writes tuples to the tuple-space.

The principle of workers is that they are defined with a list of templates to wait for, with associated processing directives. Each time a tuple matching the template appears in the tuple space, they read it and execute the corresponding directives, then return to a waiting state (a blocking read() call).

This is represented by infinite loops in the algorithms. We note that the workers are designed not to process the same tuple twice. Reading tuples requires a template that is used as input, and matches tuples present in the space.

\(^4\)The definition of read operations may vary depending on the implementation, e.g. be non-blocking, or return only the first matched tuple. The algorithm can easily be adapted to these alternative definitions.
In the following, we describe the parallel query processing algorithm as a set of workers interacting through the tuple space.

**Asynchronous Data Retrieval Functions**

First, we must define the asynchronous version of the data access API.

The asynchronous `inNeighbours()` and `outNeighbours()` functions are implemented by a worker that reads requests for graph traversals, and either retrieves the data locally, or outputs a broadcast if it is not available. As responses to the broadcast arrive (asynchronously), a separate function of the worker is triggered and the worker outputs the corresponding tuples to the tuple space. These processes are shown in algorithm 3.

**Automaton State Workers**

The algorithm is executed by $n + 2$ workers, where $n$ is the number of states in the query automaton $A_x$ (as defined in the original algorithm 2).

For each automaton state $q$, a worker is generated, that will handle the outgoing transitions from this state. Instead of using a stack (or a queue), the nodes to be explored are placed in the tuplespace, and are processed by the appropriate worker. The worker’s process is shown in algorithm 4: we now explain this process in relation to the original sequential algorithm 2.

In algorithm 2, the node $(v_c, q_c)$ taken from the stack on line 11 is now directly read from the tuple space (as a tuple $(NEIGHBOUR, v_c, q_c)$, the reason for “NEIGHBOUR” will become apparent soon) by the worker associated with state $q_c$. This worker will first identify the nodes reachable through epsilon-transitions (line 15 in algorithm 2), and output them to the tuple-space, where they are immediately available to be processed by another worker. The worker then uses the automaton transitions to generate calls to the (now asynchronous) functions `InNeighbours()`
Algorithm 3 Query answering workers: graph traversal primitives

**Procedure** OutNeighbours()

while true do
    read (OUTNEIGHBOURS, ? → v, ? → s, ? → qi) //Template
    getNodes(edgepattern=[v, s, ?], ref=qi) //Generic data access function
end while

**Procedure** InNeighbours()

while true do
    read (INNEIGHBOURS, ? → v, ? → s, ? → qi)
    getNodes(edgepattern=[?, s, v], ref=qi) //Generic data access function
end while

**Procedure** getNodes(edgepattern, ref)

if data_is_local(edgepattern) then
    nodes ← local_graphsearch (edgepattern)) //Synchronous local retrieval
    for vk ∈ nodes do
        write (NEIGHBOUR, vk, ref)
    end for
else
    broadcast_search(edgepattern, ref) //broadcast search
end if

**Procedure** SearchResponse(v, ref) //network response
write (NEIGHBOUR, v, ref)
and OutNeighbours(). These calls (lines 18 and 20 in algorithm 2), are now tuples \((IN(OUT)NEIGHBOURS, v, s, q)\) directly output to the tuple space. Referencing \(q\) (the target state of the transition) in the tuple ensures that this reference will be associated with the responses to this request, which will be output as \((v_i, q)\) tuples to the tuple space. These tuples will then asynchronously trigger the appropriate processing by the worker associated with the automaton state \(q\), bringing us back to line 11 in algorithm 2 (with a new PA node).

**Algorithm 4** Query answering workers: automaton state worker

```plaintext
Procedure for automaton state \(q\)
while true do
    read \((NEIGHBOUR, ? \rightarrow v, q)\)
    if isTerminalState\(q\) then
        write \((RPQRESULT, v)\) // Terminal State found, return asynchronously
    end if
    for \((s, q_i)\) in \(\delta_x(q)\) do
        if \(s = \epsilon\) then
            write \((NEIGHBOUR, v, q_i)\) // epsilon-transition
        else
            if \(s \in L\) then
                write \((OUTNEIGHBOURS, v, s, q_i)\) // edge of \(E\)
            else
                write \((INNEIGHBOURS, v, s, q_i)\) // edge of \(E^{-1}\)
            end if
        end if
    end for
end while
```

An additional worker initializes the query and extracts the final responses: its procedures are given in algorithm 5. This worker can be seen as the “real” main query processing algorithm, since it must generate the query automaton (using Thompson’s algorithm), and each of the automaton state workers themselves. Once all of these workers are in place, it simply initializes the processing by introducing the start node in the tuple space.
Its other function is to retrieve the final answers to the query: for this it listens for the final answers, and passes them to the appropriate user interface.

Algorithm 5 Query answering workers: RPQ initialization and response collection

Procedure Initialize($x, v_0$)
$A_x \leftarrow \text{Thompson}(x)$
for $q$ in $Q_x$ do
    //Create and initialize workers implementing algorithm 4
    Generate_Worker($q$)
end for
write ($\text{NEIGHBOUR}, v, q_0$)

Procedure CollectFinalQueryAnswers()
while true do
    read ($\text{RPQRESULT}, ?, \rightarrow v$)
    notify_interface($v$) //show query answers asynchronously to the user
end while

The intuitive appeal of this approach is that it only retrieves the data that is needed to execute the algorithm, as opposed to the S1 strategy, which is likely to collect large amounts of data and much more than will actually be explored by the algorithm.

However, this comes at the expense of many broadcast queries, since the data must be retrieved bit by bit, while the graph is being explored. Since in the worst case the full graph of data must be explored anyway, this appeal does not translate to visible gains in a (worst-case) complexity analysis.

We can, however, compare the amounts of data broadcast and retrieved for some “real-world” queries. Given some concrete figures, we can then analyze the trade-off between these quantities, and determine conditions that favour one or the other strategy.

This is the object of the next section.
6.5 Cost Comparison on Real-world Queries

6.5.1 Dataset and queries

In order to conduct an empirical study, we acquired a “real-world” dataset from the biomedical domain, with some meaningful queries to apply to this data. The dataset is a graph of knowledge automatically extracted from a corpus of pubmed abstracts [125]. The queries are the same ones used in [115]; the queries and the dataset were kindly provided by the authors of that study.

The graph has approximately 50,000 nodes and 340,000 edges. The nodes identify concepts such as molecules, genes, or animal species, and the edges represent relationships, such as a molecule activating a gene or the terms simply co-occurring in the same pubmed abstract.

The queries express meaningful associations between biological entities, and are expressed as multi-source regular path queries, i.e. regular expressions on the alphabet of edge labels. These regular expressions are listed in table 6.2, with the number of solution pairs to the multi-source query.

As the graph has 50,000 nodes, we can create 50,000 single-source queries from each regular expression (multi-source query). However, for many nodes the relationship expressed by the query simply does not make sense, which in practice means for most nodes there will be no adjacent edges matching the beginning of a query path, and therefore the cost of evaluating the query will be basically nil. For the queries of interest here, less than 2% of the nodes were valid starting points. The exact number of valid starting points for each query is given in the last column of table 6.2.

When evaluating the cost of queries, we will restrict our analysis to these valid starting points. Obviously, calculating the average cost when it is nil 99% of the time would produce results of little value.

In order to compare the cost of strategies S1 and S4 for these queries, one way
Table 6.2: Biological queries: regular expression, number of solutions (multi-source), number of valid start nodes

<table>
<thead>
<tr>
<th>Query</th>
<th>Multi-source solution pairs</th>
<th>Start nodes with non-zero cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1 C+ “acetylation” A+</td>
<td>1710</td>
<td>477</td>
</tr>
<tr>
<td>q2 C+ “acetylation” I+</td>
<td>20</td>
<td>477</td>
</tr>
<tr>
<td>q3 C+ “methylation” A+</td>
<td>0</td>
<td>477</td>
</tr>
<tr>
<td>q4 C+ “methylation” I+</td>
<td>0</td>
<td>477</td>
</tr>
<tr>
<td>q5 C+ “fusions” P</td>
<td>0</td>
<td>477</td>
</tr>
<tr>
<td>q6 “fusions” A+</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>q7 A+ “receptor” P</td>
<td>0</td>
<td>731</td>
</tr>
<tr>
<td>q8 I+ “receptor” P</td>
<td>0</td>
<td>366</td>
</tr>
<tr>
<td>q9 A A+</td>
<td>80905</td>
<td>711</td>
</tr>
<tr>
<td>q10 I I+</td>
<td>2118</td>
<td>354</td>
</tr>
<tr>
<td>q11 C E</td>
<td>249</td>
<td>364</td>
</tr>
<tr>
<td>q12 A+ I+</td>
<td>49638</td>
<td>711</td>
</tr>
</tbody>
</table>

Edge labels (‘|’ means disjunction):

C = \{interaction \ | \ interactions \ | \ binding \ | \ complex \ | \ interacting \ | \ complexes \ | \ interacts\}

A = \{activation \ | \ activity \ | \ production \ | \ induction \ | \ overexpression \ | \ up-regulation \ | \ induces \ | \ activates \ | \ increases\}

I = \{down-regulation \ | \ inhibits \ | \ inhibited \ | \ inhibitor \ | \ inhibition\}

E = \{expression \ | \ overexpression \ | \ regulates \ | \ up-regulation \ | \ expressing\}

P = \{dephosphorylates \ | \ dephosphorylated \ | \ dephosphorylate \ | \ dephosphorylation \ | \ phosphorylates \ | \ phosphorylated \ | \ phosphorylate \ | \ phosphorylation\}
would be to evaluate them in a distributed setting, and vary the parameters of this
distributed setting. However, for these strategies, the distribution parameters only
directly affect the cost of broadcasts or unicasts, which in turn determine the cost
of the query execution. We can therefore compute the number of broadcasts and
unicasts required for each query, then calculate the costs of each broadcast and unicast
analytically for the different values of the distribution parameters.

For this purpose, we first compute the number of unicasts and broadcasts for these
different queries.

6.5.2 Broadcast and Unicast costs

In the following analysis, we consider each symbol (edge label or node identifier)
transmitted in a query or response as the unit of cost for message traffic. The true
cost would be obtained by multiplying our values by the number of bytes per symbol,
and adding some overhead for the message headers.

Cost of S1

The processing of a query by S1 requires a single initial query (broadcast) to retrieve
all the data (a subset of $G_D$) potentially needed to process the RPQ.

This data is the set of edges whose labels are found in the query, and the length
of the initial broadcast query is therefore the number of distinct labels in the query.

The data matching the query is then returned by “point-to-point” messages (uni-
casts). The amount of data to be transferred is the number of matching edges to the
initial broadcast, multiplied by the length of each edge: we consider that an edge is
expressed as 3 symbols, two node identifiers and an edge label.

It is important to notice that for strategy S1, the cost does not depend on the
query start node, and is even the same for a single-source or a multi-source query.
Cost of S4

In strategy S4, the PAA is executed locally, accessing the remote data through broadcast searches.

During the search of the product automaton, at each node there is a broadcast search to find neighbours of the current graph node. Each time, the broadcast query indicates the current node and the labels of the potential outgoing edges, which are the symbols associated with the outgoing transitions from this automaton state (see algorithm 2, lines 13, 18, 20).

The amount of data to be broadcast is the sum of the lengths of all these individual queries. The data returned as unicast messages is the set of edges in $G_D$ that match them.

We assume a simple optimization whereby two identical broadcast queries that are made at different points of the algorithm will result in a single one being processed over the network, and the second time its results are obtained from a local cache.

Unlike strategy S1, for strategy S4 each single-source query will have a different cost.

6.5.3 Results

Figures 6.6a and 6.6b summarize the values obtained for the above cost functions, with our biological queries.

For each regular expression, we compare side-by-side the cost of S1 (which is the same for all the single-source queries) and the costs of S4: we show here the mean and the maximum cost. Note that the mean is calculated only for valid starting points, i.e. it is the mean of all non-zero costs.

These figures illustrate well the trade-off between broadcasts and unicasts for the two strategies. Strategy S1 always requires a minimal amount of data to be broadcast,
Figure 6.6: Amount of data to be transferred by broadcast / unicast for the evaluation of the biological queries.
but also consistently retrieves fairly large amounts of data via unicast. Strategy S4 has typically higher broadcast costs and much lower unicast costs, and is also much more variable. We note that the queries considered here are all quite selective, in the sense that they only retrieve a small fraction of the data graph. S1 retrieves between 0.2% and 0.8% of the graph, whereas S4 retrieves less than 0.1% of the graph in almost every case.

However, due to this trade-off and the high variations in the cost of S4, it is unclear which strategy is generally preferable. In the following, we examine this trade-off in analytical terms, in relation to the parameters that determine the costs of broadcast and unicast messages.

6.5.4 Cost functions

In order to express and compare the costs of S1 and S4, we introduce the following notations:

- the number of distinct labels in a query $q$ is noted $Q_{\text{lbl}}(q)$,
- the number of matching edges, or rather the amount of data encoding these edges, is noted $D_{S1}(q, G_D)$,
- $Q_{\text{bc}}(q, G_D)$ describes the total amount of data that is broadcast with strategy S4,
- The amount of data transferred by unicast for S4, encoding the set of edges traversed by the algorithm, is noted $D_{S4}(q, G_D)$.
- The data is replicated on average $K$ times, where $K = k.N_p$.

In the above functions we have indicated the dependencies of these quantities on $q$ and $G_D$ as function arguments. In the following we leave these function arguments out in order to improve readability.
We also remind the reader that the number of messages for a broadcast in a connected network with $N_c$ edges is between $N_c$ (best case) and $2N_c$ (worst case). If the average (outgoing) node degree in the network graph is $d$, then $N_c$ can be approximated as $dN_p$. Ignoring the protocol overhead for each message, we can therefore approximate the cost of broadcasting $b$ bytes of data as $2dN_pb$.

We obtain the following cost functions:

\[
\text{cost}_{S1}(q, G_D) = 2N_c Q_{lbl} + kN_p D_{s1} \\
= 2dN_p Q_{lbl} + kN_p D_{s1} \\
= N_p (2dQ_{lbl} + kD_{s1}) \quad (6.1)
\]

\[
\text{cost}_{S4}(q, G_D) = 2N_c Q_{bc} + kN_p D_{s4} \\
= N_p (2dQ_{bc} + kD_{s4}) \quad (6.2)
\]

### 6.5.5 Query Execution Strategy Choice

Using equations 6.1 and 6.2, we can establish the following condition, determining whether $S4$ or $S1$ is preferable:

\[
2dQ_{lbl} + kD_{s1} < 2dQ_{bc} + kD_{s4} \\
\iff k(D_{s1} - D_{s4}) < 2d(Q_{bc} - Q_{lbl}) \\
\iff \frac{k}{d} < \frac{2Q_{bc} - Q_{lbl}}{D_{s1} - D_{s4}} \quad (6.3)
\]

In the following, we will use the notation:

\[
\text{discr}(q, G_D) = 2 \frac{Q_{bc} - Q_{lbl}}{D_{s1} - D_{s4}}
\]
Equation 6.3 provides a discriminating condition to choose between S1 and S4, independent of the network size. Parameters $d$ and $k$ characterize the network topology and the data distribution within this network. Higher values of $d$ (denser networks) increase the cost of broadcasts, therefore favouring strategy S1, that broadcasts less data, whereas higher values of $k$ (higher data replication rates) increase the cost of retrieving data (each data resource retrieved comes in more copies), therefore favouring S4, which only retrieves the data necessary to execute the PAA.

We also know that $k < 1 < d$, because $k > 1$ would mean that every peer has multiple copies of the full graph of data, and in a network where $d < 1$ the network graph cannot be connected\(^5\).

The sets of values for which S4 and S1 are optimal can be visualized graphically as in figure 6.7.

![Figure 6.7: Optimality of S1 and S4 depending on k, d and the query-dependent discriminating function.](image)

This gives us the following discriminating conditions:

- If $Q_{bc}(q, G_D) \leq Q_{blk}(q, G_D)$ then S4 is necessarily optimal. The trivial case is where the query starting point is not valid, and this may also happen with very long and complex queries.

- If $Q_{bc}(q, G_D) > Q_{blk}(q, G_D)$, then in the 2-dimensional space of values for $k$ and $d$ with the exception of the linear alignment of $n$ peers, where $d = \frac{n-1}{n}$.
$d$, S4 is optimal in a triangle bounded by the lines of equations $k = 1$, $d = 1$ and $\frac{k}{d} = \text{discr}(q, G_D)$.

- For any other values of $k$ and $d$ that fulfil the condition $k < 1 < d$, S1 is optimal.

- Note that if $\text{discr}(q, G_D) > 1$, then S1 is necessarily optimal, because the triangle described above does not intersect with the region where $k < 1 < d$.

For our example biomedical queries, of the 5622 single-source queries with non-zero cost\(^6\), in 42 cases S4 is necessarily optimal, and for the 5580 others, either S1 or S4 will be optimal depending on the network parameters.

The problem is that while we can immediately identify queries with zero cost, and have immediate access to $Q_{\text{lbl}}(q)$ (the number of distinct labels in the query), the other parameters are not readily accessible. Therefore, $\text{discr}(q, G_D)$ cannot be calculated without actually executing the query over the data of interest, which means that despite this analysis we still cannot tell, \textit{a priori}, which strategy to choose.

One solution would be to estimate the remaining cost parameters, which may be possible if we have access locally to a sample of the data. Such an estimate would have the added benefit that we could detect the “worst case” situations, where the full graph of data is likely to be downloaded, causing an overload of network traffic.

The next section will explore two statistical graph models that can be used to produce such cost estimations.

---

\(^6\)Altogether we could apply each of the 12 multi-source queries to 50,000 nodes, yielding 600,000 single-source queries.
6.6 Query Cost Estimation

6.6.1 Related Work

To the best of our knowledge, the only existing method to estimate query selectivity is the “locality” metric proposed in [104]. This metric mainly relies on the distinction between “local” and “outgoing” edges in the query itself to determine how many sites need to be visited to answer the query, and assumes that a bounded amount of data is stored at each site. In most query languages (and in the basic definition of RPQ) this distinction between local and outgoing edges does not exist.

Several data structures have also been proposed to “summarize” graph data, which allows a user to view the structure of the data and the existing paths to be queried, and can also be used to optimize query execution by pruning the search graph [107, 126, 127]. These techniques mainly give indications of whether paths exist, with no direct applications to cost estimation.

For XML data, some techniques to estimate the selectiveness of path queries were proposed in the context of the XPath query language [128]. These techniques were designed for tree-structured data and do not handle regular expressions. However, our approach is comparable, extended to arbitrary graphs and regular paths.

6.6.2 Estimation

Network and Distribution Parameters

These parameters will determine the cost of a single broadcast, and the multiplying factor for responses returned by multiple peers. These parameters can be obtained or estimated from simple queries. Ideally, these queries should be supported by the communication protocol.

- $N_p$, network size: can be obtained by broadcasting a “ping” query, where each
recipient simply responds with an acknowledgement.

- $N_c$, number of network connections: can be obtained by broadcasting a query requesting each peer’s number of active network connections. The sum of responses will be $2 \times N_c$.

- $k$, data replication rate: can be estimated by querying for a small number of known data resources, and counting the average number of responses. This will yield an estimate of $K$ (data replication factor), which can then be divided by $N_p$ to obtain $k$. The accuracy of this estimate depends on using a representative sample of resources. Querying for more resources incurs more costs but will improve the estimate.

### Query selectivity

We now turn to the “query selectivity” functions:

- $Q_{lb}(q)$ can be trivially obtained from the query: it is the number of distinct edge labels appearing in the regular expression.

- $D_{sl}(q, G_D)$ can be estimated by counting the label frequencies on a sample of the data, and multiplying by the total number of edges $|E|$. $|E|$ can be estimated by a broadcast query requesting a count of the distinct resources stored at each site, then dividing by the expected replication $K$.

- $Q_{bc}(q, G_D)$ and $D_{sl}(q, G_D)$ are more difficult to estimate. The message cost of executing a query is linear in the data size, but only because in the worst case it requires transferring exactly the full data graph. In terms of true cost, there is no evidence of a linear relationship between the cost on a sample of the data and the true cost, if only because it is highly unlikely that the start node will be present in the sample. Instead, we have explored the possibility
of creating statistical models of the graph of data, then evaluating the query against a larger random graph generated from this model. We have considered two models, which we describe next.

6.6.3 Statistical Graph Models

Binomial Random Graph

The first model that we investigate is based on the binomial random graph model (or Gilbert model [129]), where for any pair of nodes \( (v_1, v_2) \), there is a probability \( p \) that the edge \( (v_1, v_2) \) exists. Extending this model to labelled graphs, for any label \( a \), each edge \( (v_1, a, v_2) \) exists with a probability \( p(a) \).

The probabilities \( p(a_i) \) for the different edge labels can be estimated by frequency counts.

Based on this model, the cost of a query can be estimated by executing algorithm 2 and replacing the calls to the functions \textit{InNeighbours()} and \textit{OutNeighbours()} with a function that randomly generates edges using the binomial distribution.

Bayesian-Binomial Random Graph

The disadvantage of the above model is that it completely ignores the graph structure, in the sense that adjacent edges have independent probabilities of existing. In fact, in a real-world graph, it is likely that the presence and labels of adjacent nodes are correlated, due to the semantics of the relationships that they represent.

A more elaborate model should therefore estimate the probabilities of edges conditional to the existence (and labels) of adjacent edges. Although such a “static” model is difficult to describe, we can use a “generative” model: as above we apply algorithm 1 and replace the calls to the functions \textit{InNeighbours()} and \textit{OutNeighbours()} with a function that randomly generates edges, except this time we generate edges using
probabilities conditional to the label of the edge that brought us to this node.

### 6.6.4 Evaluation

Based on the above models, we have evaluated the quality of our cost evaluations using our biological queries.

As we have noted previously, for S4 the cost factors (the amount of data that is broadcast, and the amount of data retrieved by unicast) vary wildly. However, they are also highly correlated, and we will focus here on results for the number of edges traversed during query execution, which can be considered an (inverse) measure of the query selectivity. It seems unrealistic to hope to estimate the cost of a specific single-source query without a detailed knowledge of the graph of data, since applying the same path query to different (valid) starting points yields very different costs. However, these costs follow a specific probability distribution, which we can consider to be a good target for estimation.

For each query, we have therefore compared the real distribution (frequencies) of costs for the different start nodes, with the distribution of costs obtained for many runs of our algorithm, using each of the models. Specifically, since the graph has 50,000 nodes, we have 50,000 true costs, and we also ran 50,000 runs of the algorithms for the models. Recall that 99% of the time, the cost is nil; this was true for the models as well, within one or two percent.

We constructed the models using the full graph of data: this allows us to evaluate the technique itself, without the noise caused by imperfect sampling\(^7\). We defer to future work the evaluation of approximate models obtained using only a sample of the data.

In order to compare the probability distributions of the true costs and the models,

---

\(^7\)Ideally, a representative sample of the data should produce the same graph label frequencies, and therefore the same models. Variations due to imperfect sampling are not our main concern here.
Figure 6.8: Tail Distributions for different queries: true distribution, and estimates based on the two types of statistical graph models.

we have plotted the three tail distributions (complementary CDF) for each query: figure 6.8 shows the plots for a sample of the queries (4 queries). The others are comparable. Note the logarithmic scale on the x-axis.

Across all queries, we observe that the first model consistently underestimates costs, whereas the Bayesian model tends to overestimate them, although there are exceptions (as for query 6, shown in figure 6.8b).

As we have pointed out previously, there are no established metrics or baselines to evaluate these models, so we will limit ourselves to an informal explanation of these
resulting figures.

The reason why the simple Gilbert model underestimates the likelihood of paths is that it ignores the structure of the graph and therefore its semantics. As we have mentioned before, for a given edge label, there are only very few nodes for which the relationship expressed by this label makes sense. This could mean that at any point in a path, there is a very low probability that such a path continues. And this is the result obtained by the model: paths are very short. However, as the query (and the data) is meaningful, at any point along a path, edges will tend to lead to nodes for which the relationship does make sense, and the probability that the path continues is higher than that found by the Gilbert model.

This is precisely the point of using conditional probabilities in the Bayesian model. The Bayesian model estimates the probabilities of outgoing edges of a node conditionally to the label by which we reached this node.

For example, we could be following paths labelled $a^*bb$, where the label $b$ might be very rare, but such labels may be clustered together due to the semantics of the relationship $b$. This would mean that the probability of an edge $b$ existing between two arbitrary nodes $v_1$ and $v_2$ may be very low, but might be much higher if we know that $v_1$ has an incoming edge labelled $b$.

Where the Bayesian model falls short of perfection is that although it may produce good estimates of the number of outgoing edges from a given node, it then picks the targets of these edges at random, ignoring other structural properties of the graph such as clustering (in an undirected graph) or edge transitivity, the equivalent in a directed graph. These properties mean that in real-world graphs, if two nodes $v_1$ and $v_2$ have a common neighbour $v_3$, they are more likely to be themselves connected than would be expected by random chance. This implies that paths with a common origin will tend to merge together and explore fewer nodes (than would be expected in a random graph without those structural properties).
6.7 Example Scenario

Before concluding the chapter, as a way of illustrating the importance of the different analysis and techniques developed here, we revisit the problem of choosing a query execution strategy, with an example scenario.

We consider a biomedical researcher (we will name her Alice) involved in a data-sharing network. In this network, there are 150 researchers, with an average of 6 connections each. The data they share is the Alibaba dataset (the dataset we have been using throughout this chapter), and each data item (edge in the graph) is shared by 20% of the researchers, on average.

Alice is looking for enzymes linked to the protein known as p53, of central importance in cancer research.

Specifically, Alice would like to apply the single-source query $q = (p53, C^+\text{“acetylation”}A^+)$ to the shared dataset.

The regular expression is that of query 1 shown in table 6.2, and $C$ and $A$ refer to groups of labels given in the same table. $C$ is a list of different terms describing protein interaction, and $A$ is a list of terms describing the activation of chemical compounds (e.g. enzymes).

In this scenario, the main questions are: how should Alice evaluate the query in a way that minimizes the network traffic? How much traffic might the query generate?

Following the analysis in section 6.5.5, we use the discriminating function:

$$discr(q, G_D) = 2\frac{Q_{bc} - Q_{lbl}}{D_{s1} - D_{s4}}$$

Alice can now estimate the different components of this discriminant function:

- The network and distribution parameters $N_p$, $N_c$ and $k$ are obtained as described in section 6.6.2. Alice obtains the values 150, 450, and 0.2. In addition, she
estimates the network density $d$ to be $\frac{N_c}{N_p} = 3$.

- There are 18 distinct edge labels in this query, so $Q_{lbl} = 18$

- Alice estimates $D_{s1}$ on her local data: the labels in the query represent approximately 0.5% of the edges in the data. This means over the total graph she can expect a value of around 1800 for $D_{s1}$.

- We assume that Alice’s data produces accurate edge label frequencies, and her estimate of the distributions for $D_{s4}$ are as shown in figure 6.8a. We note that in this model, the distributions are given only for non-zero costs: Alice knows that the protein p53 is known to interact with many other proteins, so she is certain that there are edges labelled $A$ adjacent to the start node; therefore the cost will not be zero. According to the model, she estimates that there is 90% chance that $D_{s4}$ will be less than 15, and around 10% chance that it will be much higher. However, despite the very high values generated by the Bayesian model, she knows that it is bounded by the value of $D_{s1}$ (1800).

- Alice’s model for broadcast costs (not shown) provides her with a similar-looking figure, and she finds there is a 90% chance that $Q_{bc}$ will be less than 70, and a 10% chance that it will reach a higher value. Using the bounds for $D_{s4}$, Alice estimates that this higher value will be of the order of 8000.

For the higher values of $Q_{bc}$ and $D_{s4}$, the use of the discriminant function is unnecessary: it is obvious that the unicast costs of the two strategies will be comparable, whereas S4 will have a very high broadcast cost.

\footnote{There is an approximately linear relationship between $D_{s4}$ and $Q_{bc}$, we omit the details here.}
For the lower estimates of $Q_{bc}$ and $D_{s4}$, the discriminant function is:

$$
\text{discr}_{\text{low}} = \frac{2}{1800} \left( 70 - 18 \right) - 15 = 0.058
$$

and:

$$
\frac{k}{d} = \frac{0.2}{3} = 0.067
$$

Therefore, $\frac{k}{d} > \text{discr}_{\text{low}}$. This means that according to these estimates, S4 has a 90% chance of being better, whereas there is a 10% chance that S1 is better.

Alice can now make an informed choice, based on estimates that required mainly local processing, and a small number of broadcast queries (to estimate the network size and data replication rate, as discussed in section 6.6.2).

It is worth noting that even if S1 turns out to be less costly, with S4 Alice can also set an arbitrary limit to the cost, and interrupt the query if the limit is reached. This, of course, would come at the expense of completeness.

### 6.8 Conclusion

We studied the problem of evaluating regular path queries on distributed data, where data is not localized, and is accessed through broadcast or unicast messages.

Our complexity analysis shows that the worst-case message cost of all the known algorithms is equivalent to collecting the full graph of data locally (or higher), which can be prohibitively expensive. Therefore, the (somewhat naive) strategy of collecting all the data and executing the query locally appears in a favourable light.
We proposed an alternative algorithm, that has the advantages that (i) it has a reasonable cost for practical selective queries, and (ii) being centralized, it can be easily monitored and interrupted if the cost turns out to be excessive.

Our analytical and experimental comparison of these two approaches showed that the former generates most of its cost by retrieving more data than necessary, while the latter only retrieves the data that it needs, but requires more broadcasts to locate this data during its execution.

Therefore, our approach performs better on more selective queries, where the naive approach is very wasteful. However, until now there were no known techniques to estimate the selectivity of a query without executing it on the dataset of interest. In order to address this problem, we proposed a query cost estimation approach based on two classes of statistical graph models.

The above scenario shows how our techniques can help support decisions regarding query execution strategies. It is important to note that we have considered broadcasts to be queries distributed to all the data sources: however, these results generalize to situations where alternative techniques (e.g. a distributed index) can be used to locate data items. In such cases, the discriminant function has to be modified to account for the new broadcast cost, but the rest of our approach is still applicable, and with a lower broadcast cost, strategy S4 is even more attractive.

In addition, our query cost estimation techniques show promising potential to detect queries that may result in high costs: these techniques are widely applicable, and may be useful even in a centralized setting.
Chapter 7

Collaboration With P2Pedia

As an example of a (DS)$^2$ application, in chapter 4 we presented P2Pedia, a (DS)$^2$ wiki. The rationale of the P2Pedia design is that by introducing decentralized control, the need for coordination and conflict resolution, inherent to online collaboration systems [44], would be rendered meaningless, since users would be entitled to maintain their preferred version of any article, and unable to prevent others from maintaining their own.

While this addresses a central problem of existing collaborative systems, it has several drawbacks. One of them is that this design would make it difficult for users to collaborate in the traditional sense of Computer Supported Cooperative Work (CSCW), i.e. to solve a common problem and obtain a single shared work product. Secondly, an issue that we have pointed out in section 4.4 is that the users must evaluate for themselves the available content, since the collaborative “curation” process was removed.

This raises the general question of whether this decentralized cooperation model can still be useful. One of the key aspects of this question is whether P2Pedia’s trust indicators (presented in section 4.4), which are intended to help the users identify valuable content, achieve their goal.

In this chapter, we present a user study where we experimentally validate the
usefulness of P2Pedia.

In this user study, P2Pedia was used as a collaboration tool for an academic writing exercise by undergraduate students of Carleton University. The students were presented with a poorly written text, and their task was to improve its writing style. They worked with P2Pedia, alternating individual work and group work, and produced different solutions to the problem. We analyzed their work and collected feedback through an online survey.

Our analysis of their work shows that there was value in allowing multiple solutions to the problem, and indicates that two trust indicators that we provide in P2Pedia would have been useful to identify the more valuable solutions.

The rest of this chapter is organized as follows. In section 7.1, we describe our user study and its specific research questions. In section 7.2, we describe the data that we collected, and our analysis and results are detailed in section 7.3. Further insights from the users’ feedback are discussed in 7.4. Finally, we draw some conclusions in section 7.5.

7.1 Academic Writing Tutorials with P2Pedia

7.1.1 Tutorial setup and Workflow

In a classroom setting, a common practice is to have the students first work individually or in small groups on a problem, then open up the discussion to the full classroom, in order to compare and discuss the different solutions obtained by the different students or groups.

In our user study, we used P2Pedia to achieve a similar effect in the context of an academic writing tutorial, part of an undergraduate course in the Department of Law of Carleton University, on research methods for legal studies. The tutorials
were meant to expose common flaws in academic writing. Students were given a poorly written text, and asked to improve its style. This was a good example of a problem which did not have a single solution, but several possible ones. The participants worked first independently, then in small groups where they collaborated to put together their individual contributions.

The tutorials took place during the Fall and Winter semesters of 2011-2012. In each semester, there were 12 tutorial groups, of approximately 20 students each, and each group had a separate tutorial session. The tutorial sessions were held in classrooms with one computer per student, and one for the instructor, and lasted 80 minutes.

The tutorial workflow was as follows.

1. Initially, the provided text was published on the instructor’s node. Students were grouped in small groups of three to five students, so that there were four subgroups in each tutorial groups. The connections between the nodes were set up so that the students could access the work of their neighbours in the small groups.

2. Each student downloaded the initial document, then worked individually to improve its style.

3. After approximately 20 minutes, the students were asked to search the network and download the versions written by their peers from their subgroups. The students were given ten minutes to read the edits made by their peers, which could be highlighted by a “diff” functionality, as illustrated by the screenshot in Figure 7.1.

4. The students then got together by subgroups, and collaboratively created a “master” version merging their different edits. For this step they sat together,
and worked on a single computer. This second phase of editing lasted approximately 15 min.

5. The instructor then lead a group discussion with the whole classroom, of the different solutions proposed by the different subgroups.

After the tutorial sessions, the students were asked to anonymously fill out an online survey. We note that the documents analyzed here come from the first set of tutorials only, but the survey answers were collected from both sessions.

Figure 7.1: P2Pedia screenshot: “Diff” highlighting of the changes between two versions.

7.1.2 Research Questions

Our analysis of the documents aimed primarily to validate the trust indicators of P2Pedia (presented in section 4.4), and assess whether this decentralized approach to collaborative writing could benefit their learning experience.

More precisely, it addresses the following research questions:

How can the students identify valuable contributions from their peers?

In order to answer this question, we analyze the different users’ contributions and validate the underlying assumptions of P2Pedia’s trust indicators. Our findings show that document popularity and user similarity can help students identify valuable contributions. As the network topology was imposed, we could not validate our social network-based trust indicators.
Do the students benefit from having their own solutions, versus agreeing on a common solution?

In order to answer the second question, we used some elements of our data analysis, and feedback collected from the survey.

Our analysis shows that the best contributions (as measured by “popularity”) could not all be combined into a single ideal solution.

The survey had a low response rate, but the results are promising: feedback from both the students and the course instructor suggests that having access to multiple solutions was a positive aspect of the students’ learning experience in these tutorials.

### 7.2 Collected data

After the tutorials, the different versions of the documents written by the students were collected\(^1\).

A total of 241 different versions of the document were collected, 208 versions from the individual editing phase, and 35 versions from the group editing phase.

The first sentence of the document to be edited, which we will use as an example throughout this paper, was the following:

> Marriage, used to be, a long time ago, just allowed between a man and woman, but it is now defined as “the lawful union of two persons to the exclusion of all others.”

---

\(^1\)We note that students could opt-out of the study, and that this research received clearance by Carleton University’s Ethics Review Committee.
7.2.1 Atomic edits

The different versions written by the peers were analyzed using a “diff” tool, which tracks the changes between two versions of a document by listing the minimal additions and deletions to go from the first version to the second. We applied the diff tool at the granularity level of whole words, and tracked the exact location of each atomic edit with respect to the original text. For example, an atomic edit might list the words “Marriage, used to be,” as being removed at character position 0, or the word “was” being inserted at position 38 (the position refers to the position with respect to the original text; the inserted word may be only at position 10 in the new text).

This way, each document version can be represented as a set of atomic edits. As we will see, the collection of atomic edits has much more interesting statistical properties than the collection of documents: due to the limited editing time, each student only produced one or two versions of the document, and only downloaded those of her immediate neighbours to read them, as per the tutorial workflow; on the other hand, each document version contained dozens of atomic edits, and these edits were often made by many students: some appeared in over 50% of the document versions.

Our dataset of 208 versions created in the first phase produced around 5500 different atomic edits.

7.2.2 Edit inclusion

Two atomic edits, while not identical, can have some level of similarity. For example, an edit can include another edit: deleting the words “Marriage, used to be,” at position 0, includes the edit that consists of deleting only the word “Marriage” at the same position. Such inclusions were considered when counting the frequencies of edits: when counting the frequency of the edit [“Marriage” deleted at position 0], we also incremented the count when the edit [“Marriage, used to be,” deleted at position
0] occurred. The opposite is not true, of course. On average, extending the set of edits in a document with “included” edits increased the number of edits by approximately 40%.

The edit frequencies discussed below were calculated after extending the lists of edits with “included” edits.

### 7.2.3 Frequency of atomic edits

Interestingly, we observe that the frequencies of the different edits appear to follow distribution very similar to that of words in a natural language, which follow the so-called “Zipf’s law”: the frequency of the $n^{th}$ most frequent word is inversely proportional to $n$. This law is visible when term frequencies are plotted on a log-log graph.

Figure 7.2 shows the frequencies of atomic edits, plotted against their frequency rank on a log-log scale, with the line indicating the slope of a function $y \propto \frac{1}{x}$ (on a logarithmic scale). We note that there is a “cutoff” of the higher frequencies, and that the Zipf-like distribution applies for edits after the first 32 values (out of 5500).

![Edit frequency vs. frequency rank](image.png)

**Figure 7.2:** Frequency of the different atomic edits
7.2.4 Similarity of documents

As was mentioned earlier, the short duration of the tutorials and their rigid workflow mean that the distribution of articles across peers has little significance: essentially, each peer produced a single version, of which copies are found on the nodes of the peer’s immediate neighbours, in the small groups.

Thus peer similarity, here, would be quite meaningless. However, by using the finer granularity of atomic edits, we can consider each document version as a peer’s “repository” of atomic edits, and evaluate the similarity of peers based on the similarity of these repositories.

Figure 7.3 shows a plot of the Cumulative Distribution Function (CDF) of the similarities between all the document versions from the individual editing phase. Most pairs of documents have a reasonable similarity, with less than 5% of the comparisons yielding no edits in common at all, and also very few highly similar documents.

![CDFs of pairwise similarity between documents](image)

Figure 7.3: Pairwise similarity of documents: CDF. The dashed line shows the similarity of edits excluding their “included” edits (as discussed in section 7.2.2), and the continuous line shows the similarity of edits after the included edits were added.
7.3 Experimental Results

7.3.1 Edits and Document quality

We now address the following research question: is the *popularity* of the edits contained in a document an indication of the *quality* of the document?

In the specific context of these writing tutorials, the students’ goal was to improve the language of the documents, i.e. the grammar, spelling, and general writing style. Therefore, the quality of these particular texts can be evaluated in a somewhat objective way by a human expert.

In order to evaluate the correlation between popularity and quality, we automatically assigned scores to the documents, based on the popularity of their edits, and compared a sample of these scores with the evaluation of a human expert, namely the class instructor.

Our scoring function $S_1$ uses the frequencies of the edits: for each edit in a text, we assign a score equal to the popularity of this edit (i.e. the number of times it occurs across the different document versions), and sum the scores of all the edits in each version. We then ranked the documents by this score and chose ten documents corresponding to scores at the 10%, 20%, . . . , 100% percentile ranks, and submitted these documents to our expert for evaluation.

We also experimented with variations of our scoring function, which mostly produced identical or nearly-identical rankings of the papers. We include here the results of a second scoring function $S_2$, which considered only the most popular edits: each edit was assigned 1 point if it was among the top 300 popular edits (meaning it appeared in at least 10% of the documents), and 0 otherwise. The edit scores were then added up for each document, as in the scoring function $S_1$.

The results are summarized in Table 7.1, which shows the ranking of the documents according to the expert, and according to our two automatic scoring functions.
Table 7.1: Evaluation of the sample documents by the class instructor and by automatic scoring functions. The letters A, B, ..., J identify the documents of the sample.

The expert considered that several documents were of equivalent quality, and instead of assigning absolute scores to each document, he ranked them in five tiers, from the best to the worst.

In order to estimate numerically the correlation between these different evaluations, we evaluate the correlation between the rankings that they produce, using Spearman’s rank correlation coefficient\(^2\).

The correlation value of S1 with the expert correlation is 0.61, and the correlation of S2 with the expert evaluation is 0.77. We estimated the significance of these figures by estimating the correlation of the expert ranking with 1000 randomized rankings. Our true correlation coefficients were found to be within the top 5% of all the values obtained by randomization, which confirms the statistical significance of our figures with at least 95% confidence.

\(^2\)Spearman’s rank correlation coefficient is obtained by replacing each value of each random variable by the rank of that value, then applying Pearson’s correlation coefficient to the obtained “rank” variables.
We can thus conclude that the most popular edits are the most likely to make the best quality documents. This suggests that it could be valuable to use the most popular edits to make a synthetic “ideal” version. We explore this question in the next section.

### 7.3.2 Edits and multiple solutions to a problem

One important observation here is that many of the edits are incompatible, i.e., they cannot be applied to the same text. A simple example of incompatible edits would be a pair of edits where different words are inserted at the same position.

Using simple compatibility criteria, such as the one stated above, we analyzed the compatibility of the most popular edits. Among the top 100 popular edits, we found 33 pairs of incompatible edits. Among the top 300 popular edits, we found 324 pairs of incompatible edits. This is consistent with the characteristics of the problem: as the class instructor noted,

“[the] goal was not to have students create a ‘perfect’ edited version that matched some pre-established notion of what the text should look like. Rather, the goal was to challenge students to identify ways to improve the sample text. There was no single ‘right’ answer to this exercise.”

As the most popular edits were incompatible, this implies that it is not possible to make a single “ideal” document, but that several potential “best” documents could be made by combining the compatible edits together. An interesting question is then: what is the minimal number of documents that can accommodate the top $k$ most popular edits?

This problem can be expressed as a graph colouring problem, and is thus NP-complete; but it can be solved for a small value of $k$. We found that the 100 most popular edits can be accommodated in 3 different versions. The 300 most popular
edits – corresponding to all the edits that occur in at least 10% of the documents – require at least 7 different versions. We note that mathematical solutions to the problem would not necessarily lead to grammatically correct text, which simply means that the minimum number of documents with good English is likely to be higher.

In a way, this analysis provides elements to answer our main research question, regarding the value of our decentralized collaboration model. In this exercise, if the participants had been forced to choose a single solution, they would have had to give up on many of the popular (and valuable) changes that they had identified in the individual editing phase.

Again in the words of the class instructor,

“The major benefit of the P2P system for our class was that students could see their peers’ work in real time and then use the knowledge they gained from that experience to improve their own work. By using the P2P platform we were able to give students access to each other’s work, while also encouraging group collaboration. Ultimately, this could be used to allow students to improve their work through exposure to the work of their peers, while also promoting the importance of the students’ individual work.”

In addition, the feedback from the students (see section 7.4) also seems to indicate that a large proportion of the students would not have submitted the group-edited versions for evaluation, but would have rather made their own tweaks to these versions. This suggests that the students also see value in having their own solution, which is usually not possible with traditional groupware.
7.3.3 Peer similarity

We study here the following question: if two peers make similar edits on one document, are they likely to make similar edits on a different document? If this is true, then the similarity between users’ documents can be used as basis for collaborative filtering, for example, the application could recommend edits that a user is likely to incorporate. More generally, we hope – although we cannot analyze this in this setting – that this extends to documents: users that have similar collections of documents will tend to agree on their “rating” of other documents, i.e. their decision to keep those documents or not.

In order to address this question, we divide the set of edits in two parts: the edits that apply to the beginning of the text (up to a certain character position), which we will call the set $A$ of edits, and those that apply to the end of the text, the set $B$ of edits.

Then, for each pair of documents, we calculate how similar their edits were on the beginning of the text, i.e. how similar their edits from set $A$ were. Then, for those documents that have similar edits on the beginning of the text, we can estimate the similarity of the edits they made on the end of the text, i.e. the similarity of their edits from set $B$.

We note $u_{d,A}$ the set of edits from $A$ in a document $d$, and $u_{d,B}$ the set of edits from $B$ in the same document.

The similarity of $d_1$ and $d_2$ on the beginning of the text is obtained by comparing the vectors $u_{d_1,A}$ and $u_{d_2,A}$, and their similarity on the end of the text is obtained by comparing the sets $u_{d_1,B}$ and $u_{d_2,B}$. We measure the similarity between sets of edits using the jaccard coefficient (defined in section 4.4), as it is the similarity metric that P2Pedia offers to measure the similarity between peers. For all pairs of documents

\footnote{note that with a few exceptions, the documents and peers are matched one-to-one}
\((d_1, d_2)\) we can thus compute two random variables \(X_A\) and \(X_B\):

\[
X_A = jaccard(u_{d_1,A}, u_{d_2,A}) \\
X_B = jaccard(u_{d_1,B}, u_{d_2,B})
\]  

(7.1)

We can then measure the correlation of the random variables \(X_A\) and \(X_B\). As we are particularly interested in the cases where \(u_{d_1,A}\) and \(u_{d_2,A}\) are similar, we keep only the top 50% most similar pairs of documents (with respect to set \(A\)).

Calculating Pearson’s correlation coefficient for \(X_A\) and \(X_B\) gives us a value of 0.44, i.e. a high positive correlation. This correlation can also be seen graphically, by plotting one \(X_B\) as a function of \(X_A\). This plot is shown in Figure 7.4.

![Similarities on two halves of the edit set](image)

Figure 7.4: Similarity of the edits on the end of the documents, plotted as a function of the similarity of the edits on the beginning of the documents.

Again we can assess the statistical significance of this value by applying a randomization test. Here we computed the correlation of \(X_A\) with 1000 random permutations of \(X_B\), and obtained correlation values confined to the interval \([-0.02, 0.02]\). This shows that this correlation is statistically significant with very high confidence\(^4\).

\(^4\)We note that these extremely low values for the randomization test may be surprising, compared to those that could be obtained when validating the correlation of our scoring functions; these values
7.4 User perception

After the tutorial, we proposed an online poll to the students in order to collect direct feedback on the usability of the tool, and on the collaboration model. We report here the results of this survey. Unfortunately, as only a small proportion of the students took the time to answer (62 out of 330 participants in total\(^5\), i.e. 19%), we will not attempt to draw any strong conclusions from these figures. We hope to conduct more systematic user studies in the future.

7.4.1 Usability

89% of respondents found the interface easy to use (11% unsure); 82% found it easy to search and download their peers’ versions (8% disagree, 10% unsure).

7.4.2 Benefit of the activities for the students’ learning experience

79% thought that reading their peers’ work contributed to their learning experience (6% disagree, 15% unsure), 62% thought that the group editing activity contributed to their learning experience (13% disagree, 25% unsure). Overall, 71% thought the writing exercise was useful, and 90% thought that P2Pedia was an appropriate tool for the exercise.

---

\(^5\)The data analyzed in the previous section was from a single session, while the survey was proposed to students after both sessions.

are due to the fact that the variables \(X_A\) and \(X_B\) have very many values (several thousand), which makes it extremely unlikely that high correlations will occur by chance. In contrast, with only 10 values there is approximately a 5% chance of obtaining a correlation above 0.5.
7.4.3 Single vs. multiple solutions

We introduced several questions for the purpose of evaluating whether it was beneficial to allow, ultimately, multiple solutions to the problem.

74% thought that it was easy to reach an agreement (a solution that everybody agreed on) in the group editing phase (8% disagree, 18% unsure); if they had to submit the result of the tutorial for evaluation, the students would either submit the group edited version (38%), or make their own tweaks to the group-edited version (56%), and 7% would choose to write their own version entirely. Finally, regarding the solutions, 64% thought that the instructor should select the best solutions and offer them as (multiple) solutions, whereas 36% thought the instructor should either write their own or select the best one to post as a single solution.

These responses suggest that while many students see a benefit in collaboration, either by being exposed to the solutions of others, or by actively working together in a group editing activity, a large part of them would still like to submit their own, personal solution; and it seems reasonable that the students would also want to see multiple solutions provided by the instructor.

7.5 Conclusion

In this study, we demonstrated a scenario where the decentralized collaboration model was relevant and applicable. Among the factors that made the decentralized collaboration model relevant, we identified the following:

- the setting did not require finding a single, ideal solution (i.e. this was not the expected outcome by the instructor)

- the participants saw value in having their own solution, but also in having access to other good solutions
• the similarity of participants on past contributions was indicative of future similarity: in other words, similarity of contributions would be a meaningful metric for users to find like-minded peers

• the popularity of contributions (considered at a fine granularity) is indicative of value according to an objective metric

• these contributions, even the most popular ones, were not compatible, i.e. they could not be merged into a single, ideal solution

Regarding the trust indicators of P2Pedia, this study has shown that it may be necessary to break down the P2Pedia pages into smaller granularity resources, such as paragraphs or even individual edits, for relevant distributional properties to appear.

While this does not affect the properties of the underlying collaboration model, it may lead us to reconsider the P2Pedia data model, and view pages as sets of smaller resources.

Several important questions remain unanswered. The main limitation of this study is that it is limited to a very focused usage, constrained by a workflow, which means we did not study a full “natural” usage of the tool for a variety of applications, or by truly autonomous users.

In particular, the users could not manage their social network, and did not really need the trust indicators since they mainly read the solutions produced by a small number of peers.

We hope that wider deployments of P2Pedia, or of similar projects, such as Cunningham’s Smallest Federated Wiki, might allow for more in-depth studies of decentralized collaboration.

In the mean time, the next chapter seeks to address this limitation through an agent-oriented simulation of users interacting in a (DS)^2.
Chapter 8

Cooperative Behavior in a (DS)$^2$

8.1 Introduction

The writing tutorials held with P2Pedia and described in the previous chapter demonstrate the value of a decentralized collaboration model. Having access to multiple solutions to the given problem was a valuable part of the students’ learning experience.

However, these tutorials were run in a classroom setting, and the sharing was facilitated by the instructors and limited to small groups of fairly engaged students. The controlled workflow of these tutorials is at odds with one of the defining principles of the (DS)$^2$ model, namely that the participants are *autonomous*, and their actions are not constrained by any centralized control.

This raises the question of whether this collaboration model would have been successful without the controlled workflow, i.e. if the users had been truly autonomous. With truly autonomous participants, we cannot rely on them necessarily being cooperative: instead we should, at best, rely on them being rational, in the economic sense of seeking to maximize their own utility.

Based on this, was the cooperative behaviour that we observed in the tutorials rational? And beyond the specifics of that exercise, and beyond the P2Pedia application, can we establish whether the (DS)$^2$ model is conducive to cooperation, in a more
general sense? What distinguishing features of the (DS)² model affect this property?

In this chapter, we address these questions by modelling the participation in a (DS)² system as an economic *game*.

Drawing from other models of data sharing systems [52, 53, 130], we start with a very simple model that views sharing data as a *Prisoners’ Dilemma* (PD): the participants can contribute, i.e., *cooperate*, or “free-ride” by consuming the data shared by others, without contributing anything themselves. This behaviour is the strategy of *defecting*, and is the dominant strategy in the game.

Studies of the *n*-player PD game in economics and evolutionary biology [131, 132] have shown that cooperative behaviours can emerge when the players are organized in a social network. Inspired by this result, we present here a simulation of a (DS)² modelled as a PD game played in a social network, and we find that a simple connection management strategy allows the cooperative players to form a network and isolate defectors.

However, the PD game has major limitations related to its payoff model: the players receive a constant positive payoff from each other cooperative user, whereas the initial intuition behind the PD model is that the users benefit from the *data* shared by others, i.e. they derive utility from consuming data resources. The PD game does not properly model the availability and replication of data (in particular when it is shared through peer-to-peer connections), or the fact that different data resources may have different *value* to different users.

We therefore extend our game model to address these limitations, and evaluate the simple connection management strategies that were successful for the PD game. We conduct two case studies where we consider data from different domains and different models for the players’ payoffs.

We first simulate a P2P network where users share music, using datasets from the *Last.fm* music sharing site (including both a social network and the music shared by
the members of this social network\(^1\) and traces from a P2P file-sharing network\(^2\). In this experiment, the users’ payoff is the amount of data that they can access in the network, and more specifically, the total value of this data in the sense of it being music that they like.

In our second case study, we consider the scenario of users sharing ratings of local businesses (e.g. restaurants), and analyze a dataset from the online service Yelp\(^3\). In this setting, the users consume the data in aggregate form, i.e. by collecting ratings from their peers and viewing the average rating for a given business. The users’ interest is then that the result of this aggregation will predict their own experience. In other words, their payoff depends on the predictive error of the aggregate ratings with respect to their own.

In the first case, we find that churn, i.e. the users’ intermittent connections in a P2P network cause a utility loss due to the fact that they cannot access all of the data, but that this loss is much smaller than what the simple PD model would predict.

In the second case, we find that the data available to the users though a social network is more valuable than the full collection, and also better than the collection produced by Yelp’s own filter. These results hold for both Yelp’s existing social network, and a synthetic social network created by adapting our previous connections management strategy.

These results contribute towards the research objectives of this thesis in two ways. First, they shed light on the trade-off between efficiency and autonomy in the (DS)\(^2\) design, by quantifying the utility loss due to data being unavailable in a (DS)\(^2\). Secondly, they show the importance of the users’ control over the social network, which they can strategically manipulate in order to improve the relevance of the data available to them.

\(^1\)http://www.last.fm. Dataset made available for the HetRec workshop 2011 \[133\]
\(^2\)Data from the Overnet P2P network, collected in 2003 and made available by the authors of [27]
\(^3\)http://www.yelp.com. Dataset made available by the authors of [134]
The rest of this chapter is organized as follows. In section 8.2, we study the participation in a (DS)$^2$ game viewed as a prisoners’ dilemma. In section 8.3 we introduce the notion of data relevance, and illustrate the properties of a relevance function based on several examples. In section 8.4 we study the impact of churn (the problem of peers intermittently connecting to P2P networks) on data availability and the users’ utility, through the scenario of a music-sharing network. In section 8.5 we evaluate a connections management strategy for the scenario of users sharing ratings for businesses in a (DS)$^2$ network. Finally, we draw some conclusions in section 8.6.

8.2 Participation in a (DS)$^2$ as a Prisoners’ Dilemma

8.2.1 The Prisoners’ Dilemma

In the context of the P2Pedia writing tutorials, intuitively, the value to the students was in reading other people’s work, while sharing their own did not directly benefit them. In this simple view, the dominant strategy is to free-ride, i.e. to read other people’s work without contributing anything. However, if everyone makes this choice, then no data is available and everyone is worse off. This suggests that from an economic perspective, the users’ participation in these tutorials can be modelled by a Prisoners’ Dilemma game, where sharing is the cooperative behaviour.

This problem has been suggested to exist for other classes of data sharing systems, in particular P2P file-sharing systems [52, 53] and recommender systems [130, 135].

Formally, the Prisoners’ Dilemma is a symmetric 2-player game where there are two strategies, known as “cooperate” and “defect”, and the payoffs for the players are as follows: if one player cooperates and the other defects, then the defector gets a higher payoff, traditionally noted $T$ (for “temptation”), and the cooperator gets the
lesser payoff $S$ (for “sucker”). If both cooperate, they obtain the payoff $R$ (“reward”), and if both defect they obtain the payoff $P$ (“punishment”). The exact payoff values are not set in the game definition, but they must respect the ordering $T > R > P > S$, which makes defecting the dominant strategy.

This game can be extended to $n$ players: each player chooses one strategy, and the payoff for each player is calculated as the sum of all the payoffs for this player in each pair-wise interaction with another player.

### 8.2.2 Iterated PD Games and Social Networks

While the incentive structure of the PD does not lead to cooperation, it is only applicable to a “one-off” interaction, whereas participants in a data-sharing network such as a $(DS)^2$ can continually interact. A $(DS)^2$ should therefore be modelled by an iterated version of the game, where the participants play repeatedly, and remember past results. In addition, a distinguishing feature of the $(DS)^2$ model is that the users are organized in a social network, a model that has also been considered for $n$-player PD games. We briefly discuss existing studies of the PD game and its many variations.

In the iterated version of the 2-player game, Axelrod [136] has shown that adaptive strategies, such as the famous “tit-for-tat”, where a user first cooperates, then systematically mimics the opponent’s previous move, can be very successful. In other words, rational agents – agents seeking to maximize their utility over time – may end up cooperating.

The participation in a $(DS)^2$ can be modelled by different variations of the iterated PD, of which many have been studied in the fields of economics and evolutionary biology$^4$.

---

$^4$Biologists and sociologists observe that humans and animals tend to be cooperative, which seems irrational. Their interest is then to study how cooperation may have emerged through evolution.
The simplest model is that the players all repeatedly engage in a $n$-person PD game. For very small groups, adaptive strategies (e.g. “tit-for-tat”) can elicit cooperation as in the 2-player case, by punishing defectors. However, these strategies do not work well for large groups [131,137], due to the fact that the players cannot selectively retaliate without everyone ending up defecting.

A first variation is one where the players are part of a large, fixed group, that repeatedly and randomly meet within the group for 2-player PD games [132]. In this context, players may apply a strategy such as “tit-for-tat” independently to each player, or, if they witness the players’ other interactions, they can apply indirect reciprocation, where they retaliate against players whom they have seen defect with others. In this type of game, trust links may emerge between cooperative players [132]. With more complex forms of communication between the players, these trust links may be used to provide referrals [138], which then can be used to efficiently propagate trust judgements.

Another slightly different setting is a large community within which the players do not meet randomly (and choose a behavior according to their assigned partner), but rather choose which agents to interact with, based on the different agents’ reputation. The distinction is important in our context, due to our focus on the social links that may exist between users: in a (DS)$^2$, a social link provides an opportunity for future interaction. This model is well suited to represent e-commerce systems such as e-Bay, for example, and has been widely studied [139]. However, many of the strategies studied in that setting (e.g. [140]) are designed for transactions involving only two players, and difficult to apply to data sharing systems, where sharing data is potentially a one-to-many action, and better represented by strategies of the $n$-player game.

Finally, another well-studied variation is one where players may be organized in a
social network, and repeatedly play (n-player) PD games with their immediate neighbours [31, 131]. In this setting, the players may also be able to strategically modify their social connections [30, 31] to improve their utility. In this case, simulations showed cooperative behaviours generally emerging rapidly [30, 31].

This final result suggests that the users’ ability to modify their connections can be a determining factor in eliciting cooperative behaviour between the participants. This is important to us since this feature distinguishes the (DS)$^2$ model from several other classes of data-sharing systems (e.g. cooperative repositories and most P2P file-sharing networks). However, these results were obtained in simulations based on evolutionary mechanisms and not exactly matching the (DS)$^2$ functionality, making their interpretation uncertain for the (DS)$^2$ model. We have therefore implemented our own simulations based on the exact (DS)$^2$ model.

For this purpose, we define the social network PD game, and study it through simulation.

### 8.2.3 The Social Network PD Game

**Game Definition**

We define the social network PD game as follows:

- The players of the game are organized in an explicit social network, i.e. they constitute the vertices of a graph, and (undirected) edges represent social connections.

- At each turn, each player chooses a single move of the PD game (cooperate or defect).

- In addition, at each turn a player may unilaterally create an edge to another arbitrary node, remove any number of adjacent edges or leave the social network.
unchanged.

- For each player, the payoff for a round is as in a $n$-player PD game, played between each player and its neighbours: for each player, the payoff is the sum of the payoffs that this player gets from the interactions with its different neighbours.

- The game is iterated: each player can remember the results of previous rounds and modify connections accordingly.

This game is slightly simpler than the versions studied in evolutionary biology [30, 31]: the main difference is the precise mechanism of establishing and removing connections. In [31], if one player is dissatisfied with a connection, and the other is not (e.g. a cooperator would be dissatisfied with a defector, whereas the reverse would not be true), then the two players compete to remove the connection (or not): the outcome is decided probabilistically, and the probabilities depend on the players’ current score.

In our context, the players are capable of unilaterally removing connections, as per our original definition of the (DS)$^2$ model (chapter 3). We also note that players can remove any number of connections at each round; an alternative game formulation would have been to allow users to create and remove links at the same rate (1 per round). However, this would also create a competition effect between defectors trying to establish links to cooperators and cooperators trying to remove the same links, which has no useful interpretation in a real system. We therefore leave the analysis of such a game for future work.

Simulation: Strategies

We implemented this game with the Netlogo [29] simulation tool, and simulated three simple strategies.
With respect to the prisoners’ dilemma, the first strategy was to systematically cooperate, the second to systematically defect, and the third to act randomly. Then, in all strategies, the players modified their connections according to the outcome of past interactions. They created links to (random) other players, and removed links to players who had defected.

**Simulation: Results**

Using these three very simple behaviours, we observe that the cooperators quickly establish a connected network, while the defectors and random players are isolated due to their propensity to defect. In order to illustrate the evolution of this graph, we illustrate it with the average degrees of the nodes and the size of the largest connected component.

![Average Node degree, by strategy](image)

**Figure 8.1:** Social Network PD game simulation: mean node degree, by strategy: simulation with 50 nodes using each strategy.

Figure 8.1 shows the average degree of the nodes for each strategy. The defectors create new links at every round, but these links are systematically removed, due to them defecting. As a result, the nodes remain isolated.
The random players create links at each round, and these links last until one of the connected players defects. Without getting into the analytical details of these distributions, the empirical result shows us that random players (here with a cooperation probability of 0.5) have small degrees, with an average value (in this simulation run) around 0.87.

The subgraph of cooperators densifies until becoming complete: the figure shows the degree of the cooperative nodes converging towards a value near 50 (the simulation was run with 50 cooperative nodes). The random players establish links at every round, and eventually establish links to all the other cooperators. In addition, they will maintain a small number of links to random players, as discussed above.

![Image](image_url)

Figure 8.2: Social Network PD game simulation: size of largest connected component. Simulation with 100 nodes implementing each strategy.

As a result, the largest connected component of the graph is made up of all the cooperative players, plus a number of random players, namely all the random players who, since they have last defected, have established links either with cooperators, or else with other random players who themselves were linked (directly or indirectly) to cooperators. Empirically, we observe that in a simulation run with equal populations of each strategy (100 players for each), the size of the GCC hovers around 140 players.
Due to their much higher degrees, in this network the cooperators immediately acquire much higher payoffs than the players implementing other strategies (figure 8.3). As the network of cooperators becomes more dense, the cooperative players’ payoffs per round increase, then stabilize once their degrees stabilize.

8.2.4 Connected Component Payoffs

In the above game model, the players only interact with (and obtain payoff from) their immediate neighbours. However, this does not quite represent the utility of a participant in a data-sharing system: the payoff to a user should rather be proportional to the amount of data available to that user. In a data-sharing system, in particular a (DS)$^2$ system, users can share data not only with their neighbours, but with all of the peers in the connected component of the network graph.

We can revisit the payoff model of the Social Network PD game to accommodate
this principle, as follows.

Each users’ payoff should be calculated by considering each connected component in the “social network” graph as a $n$-player game, instead of only considering each player’s immediate neighbours. However, consistently with the semantics of P2P connections, users can only modify the connections with their immediate neighbours.

We modified our Netlogo model to include this scoring method. In this game, once all the cooperators formed a connected network (almost immediately), their payoffs per round became constant, reflecting the fact that they now had access to all the cooperators. At this point, there is no longer any reward in creating additional connections.

This is visible in the linear increase of scores in figures 8.4a and 8.4b. However, we also observe that the defectors suddenly obtain much higher payoffs than the cooperators (figure 8.4a).

This is due to the fact that defectors now only need a single connection with a cooperator to have access to the full network of cooperators. At each simulation round, the defectors establish a link, obtain a high payoff (the value $T$ multiplied by
the number of cooperative peers), then see their connection removed by the cooper-ator. Since they can simply re-create the link at the next round, the fact that their connections get removed is not a problem.

A realistic way for cooperators to handle this would be to “blacklist” users so that once a connection has been removed once, it can no longer be re-created.

We implemented this possibility, and modified the players’ connection management strategy to systematically blacklist users when they dropped a connection. This makes it impossible for defectors to re-establish unwanted connections. As a result, the cooperators once more became dominant (figure 8.4b).

8.2.5 Validation and Limitations

In our social network PD game, we showed that when the users can strategically rewire their connections, the PD strategy of cooperating dominates the strategy of defecting, as well as the strategy of randomly cooperating or defecting. In addition, when the players interacted with their entire connected component in the social network, it also became important to “blacklist” defectors.

More sophisticated behaviours may show opportunities to “game the system” (e.g. the whitewashing and sybil attacks [32]), but we leave this analysis to future work: for now, these results provide a good intuitive illustration of the value of allowing the users to control their connections in a social network-based n-player game.

However, it is now important to validate that this strategy is consistent with useful behaviour in a data sharing system. For this purpose, we need to validate the payoff model, which represents how the user benefits from applying this (or any other) strategy.

In the “connected component” payoff model that we used above, a cooperative player receives a constant positive payoff from every other cooperative player reachable in the social network. In a data sharing system, where users primarily derive
payoff from consuming data, this model represents the following simplifications:

- a user’s payoff is the number of data resources that this user consumes;
- every user consumes all of the available data at each game iteration;
- the data contributed by a user is available to every reachable user from the contributor in the social network graph;

Each of these simplifications can be challenged. In the next section, we discuss the implications of each one of them, and propose extensions to the PD game to address their limitations.

8.3 Data Relevance

In order to address the limitations discussed above, we need to explicitly model the data shared by the peers, its availability (in particular when some users are unavailable), and explore different ways of evaluating the payoff obtained by a user to whom this data is available.

8.3.1 Churn and Data Availability

In P2P networks, peers typically join and leave the system very frequently, a problem known as churn. This implies that the network topology changes unpredictably (and may not form a connected graph), and shared data may become unavailable as a result. In the PD game, churn is not explicitly modelled. It could easily be introduced. The effect on the game would be that at any given time, the players’ social network would be smaller, and therefore the players’ payoff would be diminished.

However, the resulting game should provide a useful testbed to compare the basic strategies outlined above with alternative strategies meant to deal with churn, e.g.
modern P2P protocols (e.g. Chord [2]). Such protocols are designed to manage the connections automatically, precisely in order to guarantee the network connectivity and the availability of resources in the face of churn.

Since these guarantees would come at the expense of the peers’ ability to choose their connections and isolate free-riders, it seems that the principle of maintaining the network manually can be at odds with the goal of maximizing the available data – and ultimately the payoffs.

However, comparing these strategies requires distinguishing the players from the data they share, a distinction abstracted away by the PD game. Without this distinction, the data allocation strategies of P2P networks, which guarantee their availability, cannot be modelled and evaluated.

8.3.2 Payoff models: the Value of Data

Once we explicitly model the shared data, we need to model the payoff that a user receives from available data. Our fundamental assumption is that a user derives utility from consuming data, which can be modelled by assigning value to each available data resource. By analogy with the Information Retrieval problem of finding useful (relevant) documents, we will use the term relevance.

We can then define a relevance function as a real-valued function that takes a user and a set of data resources as inputs, and computes the payoff that the user receives when these data resources are available.

For example, the basic payoff function of the social network PD game can be interpreted as a relevance function where all the resources have the same objective value, and for any user, the relevance of a set of \( k \) resources to this user is \( k \) times this constant value.

In some situations, such as the P2Pedia writing tutorials, it is unrealistic to assume that the users will consume all of the available data: in this example, the students
found value in reading the solutions from their small group of peers, but would not conceivably have wanted to read all two hundred available versions. In other words, the marginal return of having access to more data most likely decreases beyond a certain amount, or may even tend to zero.

On the other hand, if the available data can be aggregated then large amounts of data may still be useful. For example, in a rating system such as Yelp or MovieLens, the system will rely on large numbers of ratings to compute aggregates and produce recommendations. The quality of the recommendations computed from a specific set of data (and for a specific user) can be interpreted as a relevance function. However, ultimately there may still be a point beyond which additional data produces diminishing returns in terms of recommendation quality, and hence of the users’ payoffs [141].

In addition, a reasonable model should also see this payoff vary depending on which data resource is being consumed: for example, in the P2Pedia tutorials, all of the available versions presumably did not have the same value. Similarly, in a P2P network where users share music, the users clearly derive more utility from music that they like than from music that they dislike.

Finally, it should be noted that there are many situations where consuming data does not reasonably produce a positive payoff for a user. For example, P2P file-sharing networks are plagued by malware and corrupt data, as well as media files with misleading meta-data, known as “decoys”\(^5\).

Consuming this data (downloading, in a P2P network) arguably produces negative payoffs for the users, and the problem is that users cannot easily tell whether a file is “legitimate” until it’s downloaded. If they could tell the difference, then decoys

\(^5\)These files are introduced by the music or film industry to disrupt copyright infringement. The idea is that many files purporting to be popular music or films (but containing other content) are introduced into the network, so that it makes it harder for users to find the actual music or films they are looking for, and ultimately encourages them to get that content from legal sources [57].
(and other “bad” files) could simply be ignored in counting the available resources to calculate the payoff.

The fact that some data may produce negative payoffs is important because it means that payoffs do not necessarily monotonically increase with the amount of available data.

8.3.3 Introduction to the Case Studies

As we have discussed above, relevance functions in different domains can have different characteristics, making it difficult to model such functions in general, in an abstract game.

In the next sections, we present two case studies based on datasets from existing data sharing systems, which provide instances of relevance functions for two domains.

Last.fm Case Study

The first case study uses data from the music sharing site Last.fm. On this site, users have social connections, and share data about the music they listen to.

We can use this data to model a \((DS)^2\) where the shared data resources represent artists that the Last.fm users listen to, and we can assign value to individual resources in two ways. Either in absolute terms, as the overall popularity of the artist corresponding to this resource, or else in a subjective, user-specific way, as the number of times the considered Last.fm user listened to music by this artist.

On the real Last.fm site, the users’ social connections do not determine which data the users have access to, but for the purpose of our study, we simulate this effect: users can access the available graph of data, as defined in section 3.4. We can then represent the payoff to a user as the count of available resources, or the sum of their values, computed using one of these two methods. This relevance function is
monotonically increasing: if two sets of resources verify $S_2 \subset S_1$, then the payoff to any user is higher when $S_1$ is available than when $S_2$ is available.

Therefore, in the presence of churn, the users will necessarily experience a utility loss, since only a subset of the data will be available. In this case study, we simulate churn and quantify the resulting loss. Our analysis shows that this loss is greatly reduced compared to the basic payoff function of the PD game. In some cases, for some users there may even be no loss.

**Yelp Case Study**

Our second case study uses data from the online rating system Yelp, where users rate and review local businesses. The users can also establish social connections, although again these do not determine which reviews and ratings they have access to.

In this case, the users consume the data, i.e. the ratings, in aggregate form. When a user views a list of businesses referenced in Yelp, these are shown with their *average rating*, which the user can then use to decide which business to go to. This aggregation of ratings is valuable to a user if it reliably predicts the user’s own experience. This can be seen as a recommender system similar to those of Netflix or Amazon, and the quality of the recommendation can be measured by the predictive error of its rating predictions.

For example, if user Alice sees that some restaurant on Yelp has an average rating of “3.8 stars”, and she then decides to go to this restaurant, and has an experience there that she would rate as “1 star”, then her disappointment can be measured by the difference: $3.8 - 1 = 2.8$. If the users’ rating is underestimated (e.g. if Alice would give the restaurant 5 stars), then this is still problematic, because she might decide to go to a different restaurant instead (e.g. one accurately rated 4 stars), creating a cost of opportunity.

While most research in recommender systems has focused on reducing this error
by improving aggregation algorithms, we focus instead on improving the error with a fixed aggregation algorithm (namely a simple average) by filtering the ratings. This allows us to quantify a specific user’s payoff when a specific set of ratings is available, in a principled way. In other words, it defines a relevance function.

Unlike the previous, this relevance function is not monotonic: the full set of ratings does not necessarily yield the highest – or the lowest – predictive error. This is partly due to users disagreeing, and partly to the fact that some users may introduce fake ratings and reviews to artificially boost a particular business’s reputation. This is a well known problem on Yelp [142], where an increase of a business’s rating can have significant financial impact [143]. Yelp has introduced a controversial filter to remove these “unreliable” reviews and ratings.

It is then interesting to compare the performance of Yelp’s filter, i.e. the value of the resulting set of ratings, to the value of the ratings available through a social network. We evaluate both Yelp’s existing social network, and a network created by adapting the simple connections management strategy discussed in the context of the PD game. According to our evaluation, both social networks outperform Yelp’s filter.

8.4 Data Availability in a Friend-to-Friend Network

P2P networks where users manually manage their connections, and establish connections to trusted acquaintances are known as Friend-to-Friend (F2F) networks [18, 19, 144]. In these networks, users only establish connections to trusted acquaintances, which matches the principle of social connections in (DS)$^2$ networks.

---

6Yelp’s filter excludes over 25% of contributions. The filtering algorithm is secret, according to Yelp, to prevent users from “gaming the system”. Detractors claim that it is in fact used to “blackmail” businesses into buying advertising from Yelp, otherwise their positive ratings are filtered out. Although lawsuits have so far been unsuccessful, using this filter clearly has a cost to Yelp, at least in terms of image and of legal costs.
The efficiency limitations of such networks have been studied through simulation [144], with the conclusion that the network’s topological properties, combined with churn, would cause pairs of peers to be connected only in an intermittent way, and potentially be disconnected for many hours. In our model, communication delays are largely irrelevant, but a consequence may be that at any given time, peers would only have access to a reduced portion of the network, and consequently only a subset of the data. As we have discussed earlier, this will cause a loss of utility to the users, and is a part of the trade-off between efficiency and autonomy that characterizes the (DS)$^2$ model. We are interested in quantifying this loss.

In this section, we study experimentally the dependencies between manual connections, churn, and data availability, in a payoff model where the relevance function is increasing, as discussed in section 8.3.3.

More specifically, we explore the following questions:

- If the users of a data-sharing network connect only intermittently, and only according to their social acquaintances, what proportion of the network can a participant expect to have access to?

- What proportion of the data will the users have access to?

- If different resources have different values, what is the total value of the available data?

### 8.4.1 Approach

In order to produce a model with realistic properties, we combined a snapshot of the music sharing site Last.fm, which includes a social network and the music shared by the users, with a trace from the Overnet P2P network [27], which we use to represent these users’ intermittent connections. This dataset gives us a plausible simulation of a F2F network with churn.
We note that as an alternative to using these availability traces, we also considered using a simulator based on the discrete-event simulation formalism DEVS [145]. This formalism allows the simulation of discrete-state processes in continuous time (as opposed to cycle-based simulation), and DEVS models have the additional desirable property of being closed under composition, which provides a sound basis for complex simulations involving models from very different domains. In order to simulate P2P networks, we implemented peer models and a network model to simulate their communication through messages. However, the churn models produced by various measurement studies [146–149] are primarily aggregate models, and as such difficult to implement for individual peers, and difficult to validate. In addition, using such a simulator requires models of the exchanged messages, i.e. the users’ querying behaviour, and a corresponding way of representing the users’ utility, with and without churn. Interpreting the results would therefore have required more assumptions than our current approach, and our belief is that its validity would have therefore been more questionable.

Following this approach would have nonetheless been feasible: we published [150] a preliminary study using data from another domain: it is detailed in appendix A.

8.4.2 Datasets

The Last.fm dataset includes 1,800 users with their social network ties and a list of artists they have listened to online, with the number of plays for each artist (this can be used as a “proxy” for the user’s interest in the artist). The second dataset describes the online presence of 1,469 users of the Overnet P2P file-sharing network, recorded every 20 minutes for 2 weeks, in 2003 (i.e. 500 time points).

We took the Last.fm dataset as an example acquaintance graph, and the musical artists that these users listened to as data resources that they share. Then, we randomly selected 1469 Last.fm users and randomly mapped them to the availability
traces from the Overnet dataset.

In the resulting dataset, the social graph is connected, but at any given time only around 30% of the peers are online. Some peers are almost always online, whereas others only appear once in the 500 time points. Each peer shares approximately 50 different resources, out of a total 14,200 different ones. The popularity of these resources has a Zipf-like distribution, with few relatively popular resources and a “heavy tail” of unpopular ones. The most popular resources are shared by around 10% of the peers (around 150), with the mean popularity close to 5.

This dataset gives us a plausible simulation of a F2F network with churn. At each time we can measure the size of the graph of online peers, its connected components, and the resources available to each online peer. We can then empirically estimate probabilities, and probability distributions for random variables, by counting the frequency of different values occurring.

We can then evaluate the users’ payoffs.

### 8.4.3 Reachable Peers and Available Data

In order to evaluate the impact of churn on the users’ payoffs, we first evaluate the size of the reachable network, and the number of available resources, relative to the total number of online peers, and the total number of resources they share.

More specifically, we evaluate the distributions of the random variables $k$ and $l$, defined as follows: when a peer goes online, $k$ percent of the online peers are reachable, and $l$ percent of the set of resources shared by these peers is available.

Figure 8.5 shows the estimated cumulative distribution function (CDF) of these random variables, obtained from our simulated network.

This first result shows that for both variables, there is a sizable probability mass at zero, meaning the peers have a significant probability of finding themselves isolated when they go online, i.e. none of their direct connections are online at the same time.
Figure 8.5: Distributions of the relative sizes of the reachable network and the available resource set, for online peers.

In that situation, they do not have access to any data\(^7\).

However, if they do not find themselves isolated, then they are part of a giant connected component that comprises around 70 to 90 percent of the online peers. Correspondingly, they have access to most of the shared resources (80 to 95\%).

The existence of the giant connected component is consistent with theoretical results [151, 152]: in several classes of random graphs, including Erdős-Rényi random graphs and scale-free graphs (graphs with power-law degree distributions, such as this one), there is a threshold on the average node degree\(^8\) beyond which a significant portion of the graph (i.e. \(O(n)\) nodes) forms a single connected component. Our empirical result suggests that this is likely to be the case in a P2F network. In other words, the connectivity of the network is quite resilient to churn.

Although the nodes have a high probability of finding themselves in the giant

---

\(^7\)We ignore the data that they share themselves, since the main source of utility is assumed to be the contributions of other peers.

\(^8\)This threshold is sometimes called the percolation threshold: beyond this threshold, the graph is said to percolate, i.e. phenomena that propagate through the graph (e.g. epidemics or P2P broadcast messages) are likely to reach a large proportion of the nodes.
connected component, obviously this probability depends on the peers’ number of
customers in the underlying social network: since the peers are only online 30%
of the time, a node with only one connection will have less than 30% probability of
being in the giant connected component.\footnote{If different peers have independent probabilities of being online, then this peer will be connected
to its neighbour around 30% of the time when it is online, but this neighbour may not be itself
connected to any other peers.}

As could be expected, this probability can empirically be shown to increase
strongly for peers with more connections: peers with more than 10 friends are al-
most certainly in the giant connected component whenever they go online. This is
shown in figure 8.6.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure8_6.png}
\caption{Probability of a peer being in the giant connected component, depending
on the peers’ degree in the social network (number of friends).}
\end{figure}

The second comparison that can be made is the available data as compared with
the total amount of data shared by online and offline peers. This is an interesting
comparison, as it measures the loss of payoff due to peers only hosting their own data,
instead of redistributing it to ensure availability (as is typically done in structured
P2P networks).

The CDF of the number of available resources, as a proportion of the total, is
shown in figure 8.8 (plot “resource count”).
In this graph, we observe that peers in the giant connected component have access to around 30 to 50% of the total set of resources. This is interesting since the giant connected component represents only around 25% of the total number of peers (around 80% of the online peers, themselves making up 30% of the total). The higher proportion of available data is a consequence of data replication: a given data resource may be replicated in several peers, and will be available if any of these peers is reachable from the considered reference peer.

Therefore, resources that are more popular, in the sense of having a higher replication rate, are more likely to be available.

One interesting consequence is that if the popularity of data can be interpreted as an indication of value in some sense (e.g. as in the P2Pedia experiments), then the “best” data (in whichever sense of value is applicable), is also the most available data.

### 8.4.4 Availability and Popularity

The relationship between data availability and popularity can be evaluated empirically in our dataset. For an arbitrary peer $p$, and a resource $r$, we estimate the probability that the resource is available to $p$, depending on the resource’s replication rate (i.e. the fraction of the peers who share it). We also evaluate the dependency between this probability and the peers’ number of friends.

The results (figure 8.7) are consistent with our intuition. Data with low replication rates is only rarely available. Increasing replication makes data more likely to be available, up to a maximum probability that is reached for resources shared by approximately 15 peers.

This maximal probability reflects the fact that beyond a certain replication level, it becomes practically certain that a popular resource will be available in the giant connected component. The probability that it is available to a given peer is then the
Figure 8.7: Probability that a resource \( r \) is available to an arbitrary user \( u \), depending on the popularity (replication) of \( r \), and on the users’ number of connections in the social network.

The probability that this peer is in the giant connected component.

For example, a peer with fewer than 5 connections will be in the giant connected component with a probability around 50%, and therefore the popular data will be available to this peer with around 50% probability. For peers with many social connections, the probability that a given popular resource is available will tend to 1.

It is interesting to note that the threshold beyond which resources are almost certainly available in the GCC does not depend on the network size. Intuitively, the probability that a resource is unavailable in the GCC is the probability that all of its replicas are hosted by peers who are either offline altogether, or online but not in the GCC. Since the proportion of peers not in the GCC does not depend on the network size\(^{10}\), the probability that all of the replicas are unavailable is independent of the network size, and decreases geometrically with the number of replicas. In other words, if we interpret popularity as “proportion of peers who will host a resource”.

\(^{10}\) at least, this proportion does not increase with network size [152]
then for a given popularity, as the network size increases the number of replicas will increase, and resources with that popularity will be more likely to be available.

### 8.4.5 Assigning Value to the Available Data

We now evaluate the payoffs of a user according to different relevance functions. We consider two models: in the first, the value of data is objective (identical for all users), and is approximated by its *popularity*. In the second model, the value of data is subjective: the value of a given resource can be different for different peers. We instantiate this value using the Last.fm dataset: the subjective appreciation of a resource by a user is the number of times the user has listened to the artist represented by the resource.

As we did in section 8.4.3, we can again evaluate the payoff of a peer relative to the payoff that the peer would have received if all of the resources had been available. Figure 8.8 shows the CDF of the payoff as a percentage of total possible payoff. The three plots show the values obtained when the value is calculated (i) as the resource count (as in section 8.4.3), (ii) using a popularity-based value (the value of a resource is proportional to the number of users who share that resource), then (iii) using a subjective value (the value of each resource to a peer is measured as the number of times the corresponding Last.fm user has listened to the corresponding artist).

We observe that there is still a significant probability mass around zero: in all our payoff models, the payoff to the user is zero when no data is available. However, peers who find themselves outside the giant connected component, but not completely isolated (i.e. in a small network with 5 or 6 other peers) have markedly higher payoffs when the value of data is calculated subjectively (the corresponding plot is below the others on the left half of the figure). We can interpret this as the effect of *homophily*, the notion that social acquaintances tend to have common interests [153], and therefore the resources available from a peers’ immediate neighbourhood tend to
be more valuable than a random selection.

Then, once the users are in the giant connected component, the users’ payoffs are again much higher when the value of the data is differentiated. In the case of the popularity-based value function, this could be expected, since popular data is much more likely to be available in the giant connected component, making the available data biased towards popular data, more valuable in this model.

In the case of the subjective value function, the possible payoff values are also generally high, reflecting the fact that the users’ tastes are generally biased towards popular data (by definition of popularity), but also more spread out, reflecting the diversity of user tastes, with varying levels of agreement with the overall resource popularity.

While the distribution plots are visually informative, we can quantify the cost of data being unavailable, by comparing the expected payoffs for each value function.

- If the payoff is the count of available resources, the expected payoff for a user is around 34% of the maximum possible payoff,
• If the payoff is estimated by the popularity of the resources, then the expected payoff is around 69% of the maximum possible payoff,

• If the payoff is estimated subjectively, then the expected payoff is 75% of the maximum.

8.4.6 Conclusions

In the presence of churn, which is to be expected in P2P networks and in (DS)$^2$ systems, the connectivity of the network is difficult to maintain, and we can expect a large part of the shared data to be unavailable, which reduces the payoff of the users, since this payoff represents the data that a peer can obtain from others.

In this section, we have evaluated this payoff “loss” empirically, by constructing a plausible simulation of a F2F network evolving over a short period of time, and measuring the data available to each peer.

The results show that the loss is mainly due to two factors:

• the first factor is the lack of network connectivity, which causes peers with few social connections to find themselves frequently isolated from the rest of the network.

• the second factor is due to peers being offline altogether: the data hosted only by offline peers becomes unavailable.

Both loss factors could be suppressed entirely by using structured P2P network protocols, such as Chord [2], that guarantee the availability of data and the connectivity of the network. However, these desirable properties would come at the expense of the peers’ autonomy, and their ability to isolate free-riders.

The first loss factor can be easily mitigated: a peer that finds itself frequently
offline should simply create more social connections, and this way reduce the probability of finding itself isolated. From a purely economic perspective, if this user is cooperative then the other users should accept these connections. In practice, there is anecdotal evidence that OSN users easily accept connections from strangers [154,155].

The second loss factor is naturally mitigated in an interesting way: although not all the resources are available, the set of available resources to a given peer is biased towards popular resources in general, and in particular towards resources popular in this peers’ immediate social neighbourhood.

These results can be related back to our P2Pedia experiments, where we have shown that these properties correlate with some measure of value to the users: the popularity of edits was shown to be indicative of the professor’s evaluations of the documents obtained from those edits, meaning that if the data had been shared at the granularity of edits, the availability of data would have been correlated with its value.

Including this aspect into an economic estimate of the loss due to social connections and churn produces cost estimates\(^\text{11}\) of less than half of what we would obtain by estimating the cost from the count of shared resources.

The fact that the most available data is also the data popular in each user’s social network also has implications with respect to the connections management strategy: users have an incentive to establish their connections strategically, i.e. connect to neighbours who share valuable data. Again, our analysis of the P2Pedia data shows that this would work: since the users’ past similarity in edits was correlated with future similarity, users could have based their connections decisions on past similarity, even if their utility was to come from the data they would obtain in the future.

However, in this case study, less available data did necessarily imply a net utility\(^\text{11}\) the cost being the difference between the expected payoff (calculated in section 8.4.5) and the total possible payoff.
loss, since the relevance function was monotonically increasing, as discussed in section 8.3.3. We now explore the case where the utility function is not monotonically increasing, with the scenario of ratings being shared.

We compare the data available through a strategically constructed social network, seen as a filter, with Yelp’s own filtering mechanism.

8.5 Case Study: Cooperative Behaviour in Online Rating Systems

8.5.1 Introduction

In this case study, we investigate how the users may shape their network to have access to the most relevant data. In the social network PD game, a simple connections management strategy enabled the cooperative users to form a connected network and isolate defectors.

In an online rating system, the notions of cooperation and defection are subjective. For example, from the perspective of our example user Alice (see section 8.3.3), the users who contributed the high ratings of the restaurant she disliked (including for example some user Bob) were defecting, whereas those who contributed low ratings were cooperating. However, if some other user Charlie went to the same restaurant and had a happy experience, then the opposite could be simultaneously true. Therefore, in Alice’s perspective Bob is a defector, whereas in Charlie’s perspective Bob is a cooperator. In addition, Alice and Charlie could very well have contributed mutually useful ratings on some other restaurant, making them perceive each other as contributors.

What we are interested in discovering is the result, at a global level, of the users
applying local decisions following the simple connections management strategy discussed above, which we introduced with clearly partitioned cooperator/defector populations. In this case, since there is no “objective” labelling of cooperators/defectors, we cannot compare the utilities of these two strategies. Instead, we can let all the users apply this strategy, and compare the users’ utilities with other ways of selecting or filtering the set of ratings.

We compare this approach with the following methods of selecting ratings on Yelp:

- Baseline: all the ratings in the dataset.
- Yelp filter: the ratings that are not removed by Yelp’s filter. The average of these ratings is the value currently displayed to Yelp users.
- Yelp social network: Yelp’s internal social network is unlikely to have been built with our proposed strategy: we can consider the ratings reachable through this social network.
- Similarity-based filter: A baseline method of recommender systems is to take into account only the ratings of similar users, in a k-NN\textsuperscript{12} approach. This approach also corresponds to one of the trust indicators of P2Pedia, which makes it interesting to investigate.

**Dataset** We use a dataset of ratings from Yelp, collected and shared by the authors of [134]. This dataset includes around 280,000 ratings by 80,000 users, on 9,000 businesses listed in Boston, USA, as well as the social links between the users. For each rating, the dataset indicates whether it was flagged by Yelp’s filtering algorithm (discussed below).

\textsuperscript{12}k-NN, or k Nearest Neighbours is a prediction algorithm that takes the most similar training data points, and predicts the mean (or mode) of the associated values. The application of this method to Yelp is explained in section 8.5.5.
8.5.2 Agreement Graph

Our first question is the following: if these users apply the strategy of maintaining connections with (subjectively) cooperative users, then what are the properties of the social network that will emerge from these local decisions? One possibility would be that the strategy might allow users to form small clusters of like-minded users, with intra-cluster agreement and inter-cluster disagreement. Our analysis will show that this is not the case.

We can construct this social network using the Yelp dataset, by analogy with the results of our previous simulations for the PD game. This network is an undirected graph of users, where any two users are connected if they would consider each other as cooperators, i.e. if these users have rated businesses in common and agree on their ratings of these businesses.

We first construct the graph of users who have rated at least two businesses in common, in order to have a reasonable basis to evaluate agreement. This graph includes 25,936 users, and 1.93M edges. Of these users, 25,813 (99.5%) form a giant connected component (GCC) in the graph. The remaining 55,000 users have at most 1 rated business in common with any other user, making it difficult to evaluate their agreement with other users.

The node degrees in the graph follows an approximate power-law distribution (visible by the approximate alignment of the data points on the log-log plot of figure 8.9), showing that again, the graph has a scale-free topology, meaning its connectivity is robust to the removal of many (random) nodes and edges [156].

What is interesting is that removing the edges that represent disagreement does not break up the graph either. We considered different thresholds of agreement, between 0 (all past ratings were identical) and 1 (the average ratings difference was 1 star or less): the size of the GCC is visible in figure 8.10. Note that the threshold
Figure 8.9: Node degree distribution in the graph of agreeing and disagreeing users.

is a distance value (the mean absolute difference in ratings), instead of a similarity value, which means that lower values reflect higher agreement.

With an agreement threshold of 1, the graph is left with 1.28M edges (66% of the full graph), but still has a giant connected component comprising 23,920 users, or 92% of the total.

As discussed above, the users do not form clusters with internal agreement. Instead, we observe that most of the users have rated a small number of businesses, and for different subsets of these rated businesses, they can find a few other users whose ratings agree. The transitive closure of these local agreements forms a GCC with a large number of users, and those who are not in the GCC are mostly isolated: the second largest connected component has 11 users, and these are a separate component even in the first graph of users with common rated businesses, meaning that they were not isolated because they disagree with the rest, but because they rated a separate set of businesses. The next smallest connected components have less than 4 users.
Our conclusion is that this strategy does not allow the users to form small communities of like-minded users. Instead it appears to isolate some users who agree with nobody else (or have no ratings in common with others). As we show in the next section, this is nonetheless beneficial to the users in the connected component.

8.5.3 Evaluation of the Agreement Graph

We now evaluate this strategy with respect to the users’ utility. Since the users outside of the GCC are mainly isolated, they do not have access to any data, and for them using the agreement graph is clearly not helpful.

For the users inside the GCC of the agreement graph, is the data available in the GCC more valuable (relevant) than the full collection of ratings?

As discussed above, the relevance of a set of ratings to a user can be measured by using this data to predict the users’ own ratings, and measuring the accuracy of this prediction. Therefore, in order to evaluate this strategy, we compare the accuracy of rating prediction using the data filtered by the GCC with two baselines, namely the accuracy of predictions using the full graph of data, and the accuracy obtained using data filtered by the Yelp filter. As an accuracy metric, we use the mean absolute error (MAE) between the true rating and the rating prediction, averaged over all users.

We observe that the agreement graph, for all threshold levels, provides valuable filtering (figure 8.10). Although the absolute error reduction is quite small (note the y scale on the right), the graph shows that the error reduction is significantly\(^\text{13}\) larger with the agreement graph than with Yelp’s filter.

We note that the baseline and the performance of Yelp’s filter appear to vary depending on the (unrelated) threshold used in the agreement graph. This is simply because the performance of the rating aggregation, for all three filtering mechanisms,

\(^{13}\text{Note: we evaluated the statistical significance of this result with a paired t-test, and found it significant with a p-value } p < 0.0001.\)
Figure 8.10: Evaluation of the agreement graph as a ratings filter, against Yelp’s filter. The top plot indicates the number of users in the GCC, and its y scale is on the left axis. The other plots indicate the MAE for different filtering criteria, and their scale is on the right axis.

is evaluated on the same set of users, i.e. those in the GCC of the agreement graph, which is different for each threshold. The number of users in the set increases with the threshold, as we have discussed above, and this value is also plotted at the top of the graph, with its y scale on the left.

The fact that the error increases (for the GCC filtering, the Yelp filtering, and no filtering) with the agreement threshold can be interpreted as follows. The key is that the target users (the users on which the performance is evaluated) are inside the GCC of the agreement graph. Using the GCC as a filtering mechanism ensures that the ratings used to predict their ratings are also from the GCC: therefore, it can be expected that they agree more when the criteria to be part of the GCC are more restrictive. The performance values with no filtering and with Yelp’s filtering also increase, and our interpretation is that for lower threshold values (more restrictive), the users inside the GCC tend to have more consensual ratings, since they agree with more other users. This means that they are easier to predict, regardless of the filtering
applied. In a more statistical view, the ratings distributions are fairly well centered (unimodal), meaning that the users in the GCC when the GCC is restrictive are likely to have ratings near the mode (making them agree with others), and therefore easier to predict and resulting in lower errors. If the set of ratings were rather bimodal, with users either “loving” or “hating” the businesses, then the “lovers” and “haters” would be equally difficult to predict, at least with non-personalized prediction (aggregations) methods. In that case, we might be more likely to see different clusters of users.

As the error increases with the size of the GCC, this analysis does not lend itself to determining the best value to use as a threshold. We address this in the next section, in a more general comparison between filtering methods.

In the next sections, we evaluate the existing Yelp social network as an alternative to the agreement graph, and a similarity-based filtering approach, before comparing all these techniques and drawing conclusions.

### 8.5.4 Social Network

Yelp has its own social network. It has two types of social relationships, “friend” (an undirected connection) and “follow” (a directed connection). Our dataset includes the social graph made up of “friends” relationships. Presumably, these users did not establish their friendship links following the exact strategy that we described above. However, people tend to associate with people of similar tastes and interests, and they might have established connections with users they trust, in a strategy somewhat comparable to the “agreement-based” strategy.

Among the 80,000 users of our dataset, around 25,000 have at least one friend. They form another scale-free graph with a giant connected component of 23,538 users. Of these, only around 13,000 are also found in the GCC of the largest agreement graph, which makes the two graphs rather complementary.

14 “Follow” relationships are not public, and thus not available in our dataset.
Repeating the experiment of the agreement graph, we find again that the MAE of the rating prediction (aggregation) drops significantly for the users in the GCC of the social network, when they use the ratings available in the GCC. The exact MAE values for the baseline, Yelp filter, and social network GCC filter are respectively 0.8935, 0.9039, and 0.8882. It is interesting to note that in this case the Yelp filter performs significantly worse than the baseline dataset (paired t-test, $p < 0.0001$).

In addition, the success of these two graph-based techniques suggests an additional approach: since the nodes in the social graph’s GCC only have a limited overlap with those of the agreement graphs, it would be valuable to merge these graphs, which could potentially provide the observed improvements to more users.

Using a threshold of 1 for the agreement graph, we merge the two graphs\(^\text{15}\) and obtain a giant connected component with 34,599 nodes, for which the MAE values are: baseline 0.9091, Yelp filter 0.9194, network GCC 0.9039. Again, the network GCC provides a statistically significant improvement, with $p < 0.0001$.

### 8.5.5 Similarity-based Filtering

The success of the “agreement graph” approach suggests that the users could simply take into consideration the ratings from the $k$ most similar users, a technique first used in the collaborative filtering system Grouplens [157].

User-user $k$-NN rating predictions use the following algorithm:

To predict a user $u_0$’s rating on item $i_0$, find the $k$ most similar users $u_1, \ldots, u_k$ to $u_0$ who have also rated $i_0$, then predict the mean of their ratings.

The mean may be weighted or not, and the similarity between two users $u_i$ and $u_j$, noted $\text{sim}(u_i, u_j)$ is calculated from the average difference of their ratings across all items that they have rated.

\(^{15}\)On the nodes shared by both graphs, this “merge” does not commit to any specific choice between the edges of the social graph or of the agreement graph. This is irrelevant since we are only concerned with the GCC of the merged graph.
One difficulty is that the Yelp ratings are very sparse, meaning that in many cases (i.e. for many specific ratings) the dataset simply does not have \( k \) other ratings from users with non-zero similarity that can be used as a basis for prediction. In addition, it may also be desirable to only consider users with high similarity (above a certain threshold) and some confidence in that similarity, which requires that the similarity be computed from a minimal number of common ratings (a parameter we call hereafter \textit{minsupport}).

However, while higher values of these parameters may help with prediction accuracy, they will also “disqualify” more users, meaning they cannot be included in the \( k \) neighbours, and mechanically they also increase the number of ratings for which fewer than \( k \) neighbours are available for prediction.

When \( k \) neighbours are not available for prediction, one possibility is to use whichever number of neighbours is available, or another is to fall back to the baseline prediction (i.e. the mean of all available ratings). We chose the latter approach, as it allows us to compare different parameter values over the full set of users.

![Performance of K-NN vs. average rating](image.png)

Figure 8.11: Performance of K-NN vs. average rating. Threshold and \( k \) are varied, \textit{minsupport} = 2 fixed. If \( k \) neighbours are not available for prediction, the algorithm falls back to average rating.
Figure 8.11 shows the predictive error of the $k$-NN algorithms for various values of $k$ and $t$ (threshold). The inset shows a zoom on the performance for $k = 20$. With $t = 0.5$, the performance of $k$-NN and average rating (the baseline, shown as a green line) are essentially indistinguishable, partly because for these values, $k$-NN is applicable only for very few ratings.

In general, we observe that similarity-based filtering does not perform well, since it is generally worse or comparable to the baseline (all ratings being used).

### 8.5.6 Overall Comparison

We can now summarize our results and compare all of these different strategies.

First, the strategy of connecting to users who have agreed in the past was shown to be valuable, since it creates an agreement graph within which the users have access to more valuable data. The improvement was found for all similarity thresholds between 0 and 1.

Secondly, our analysis of Yelp’s existing social network showed that this network had a comparable size and topology to that of the agreement graph, and also provided a comparable improvement over the baseline and Yelp filter.

Third, the two previous approaches could be used in a complementary way, since many users in the social network do not find themselves in the agreement graph, and vice-versa. Merging the graphs has also shown to produce a valuable filter.

Fourth, we did not find the similarity-based filtering to be valuable, as it performed even worse than the baseline.

These different techniques were individually compared with the baseline, and on different subsets of users. We now compare them all together, on a set of users for whom all the techniques are applicable, namely the intersection of the social graph’s GCC with the most restrictive agreement graph’s GCC (with an agreement threshold of 0).
In table 8.1, we compare the different filtering techniques, and indicate for each technique what proportion of the original ratings are removed. In the case of similarity-based filtering, we leave this value blank, as its meaning in this case is unclear: when \( k \) comparable users are found, this technique makes use of very few ratings, but the ratings used are different for each evaluation. Ultimately, every rating could be used.

Table 8.1: Comparison of different ratings filtering methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Ratings removed</th>
<th>MAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>0%</td>
<td>0.79252</td>
</tr>
<tr>
<td>Similarity</td>
<td>N/A</td>
<td>0.79247</td>
</tr>
<tr>
<td>Yelp Filter</td>
<td>19%</td>
<td>0.79105</td>
</tr>
<tr>
<td>Agreement Graph:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t=1 )</td>
<td>25%</td>
<td>0.78339</td>
</tr>
<tr>
<td>( t=0.5 )</td>
<td>28%</td>
<td>0.78260</td>
</tr>
<tr>
<td>( t=0 )</td>
<td>33%</td>
<td>0.78256</td>
</tr>
<tr>
<td>Social Network</td>
<td>38%</td>
<td>0.78059</td>
</tr>
<tr>
<td>Social Net + Agreement Gr. ( t=1 )</td>
<td>21%</td>
<td>0.78203</td>
</tr>
<tr>
<td>Social Net + Agreement Gr. ( t=0 )</td>
<td>25%</td>
<td>0.78048</td>
</tr>
</tbody>
</table>

We observe that on this population, the Yelp filter outperforms the baseline. The agreement graph techniques outperform the Yelp filter (statistically significant improvement, \( p < 0.0001 \)). Among the different thresholds for the agreement graph, the difference between threshold values 0 and 0.5 are not statistically significant, but they are between 0.5 and 1. The social network performs significantly better than the agreement graph. When the two graphs are combined, performance drops slightly (i.e. the value goes up, but lower is better; the difference is statistically significant for \( p < 0.01 \)) with an agreement threshold of 1. However, as mentioned previously, with the combined graph the filter applies to more users.
8.6 Conclusions

In this chapter, we have attempted to evaluate the \((DS)^2\) model by investigating whether it is conducive to cooperative behaviour, for different interpretations of the notion of cooperation.

8.6.1 Cooperation in the PD game

The first interpretation that we considered is the game-theoretic strategy of cooperating in the classic Prisoners’ Dilemma (PD) game, which models the dilemma of contributing vs. “free-riding” in data sharing systems. When players of the PD game are organized in a social network and can rewire their social connections strategically, we found that cooperators can form a connected network and isolate defectors, confirming results obtained by evolutionary biologists. This makes cooperating a valuable strategy, which is not the case in a classic \(n\)-player PD game: this suggests that the \((DS)^2\) model’s explicit social network is key in making the \((DS)^2\) conducive to cooperative behaviour.

8.6.2 Impact of Churn

In a P2P network, we can expect the manual management of connections to be a poor way of handling churn, which may cause the network to be disconnected and resources to be unavailable. Since other participants’ contributions are the main source of utility to the users (in the original intuition of the PD model), this will result in a utility loss to the users. We simulated the effects of churn on a \((DS)^2\) using the shared data and social network from the Last.fm music sharing site, combined with availability traces from the Overnet P2P network. In our experiments, we found that despite only 30% of the network being available at any given time, most of the peers find themselves in a giant connected component and have access to most of the
shared data. In addition, as might have been expected, they are more likely to have access to the most replicated data and to the data shared by their immediate social neighbours.

If these properties are correlated with (objective or subjective) value of the resources, this implies that they will have access to generally more relevant data, further reducing the utility loss due to churn:

- if the available resources all have the same relevance, churn reduces the users’ utility by around 66%,

- if the popularity of a resource is indicative of its relevance, then the loss is only 31%,

- if the users’ usage of each resource is proportional to their relevance, then the loss is only 25%.

This implies that the value of using structured P2P networking protocols (e.g. Chord) is more in reducing communication costs than in maintaining the network connected, since most of the online peers find themselves in a giant connected component (GCC), and the isolated peers only need to establish a small number of connections to make it almost certain that they will be in the GCC as well. In addition, since the more popular resources, above a fairly low threshold, are almost guaranteed to be available in the GCC, the benefit of P2P protocols that include the allocation of resources to peers (DHT) should be evaluated by comparing the benefit of the rare resources being available with the overhead cost that these protocols cause in terms of message traffic.
8.6.3 Agreement and Cooperation

The fact that different data resources may have different values suggests another meaning of cooperative behaviour. Based on the PD premise that the cooperators are those whose contributions benefit others, we can view users with similar tastes to be subjectively cooperative in each others’ perspective, and conversely, users who disagree might see each other as defectors.

If the users’ tastes could only fall in two opposite categories $A$ and $B$, then the players managing their connections strategically would end up forming two connected components, one with all the players preferring $A$, and one with all the players preferring $B$. In each connected component, all the players would be mutually cooperating. This would intuitively be the result of playing the *Bach or Stravinsky?* economic game\(^\text{16}\).

However, in real life, our preferences when evaluated over a large number of items do not typically fall into two opposite categories. Therefore, the outcome of strategic connection cannot easily be predicted. We have evaluated it over a dataset of ratings from Yelp, and found that in fact there will be partial agreement between the users, so that if the users construct a social network by connecting with the peers they *agree* with, then most will again find themselves in a single GCC.

In this context, we evaluated the relevance of the data available to the users in a principled way, using metrics inspired by Recommender Systems literature. We found that the data available to the peers through the social network was *more relevant* than the full collection of ratings, more relevant than the data as filtered by Yelp (to remove “unreliable” ratings and reviews), and also more relevant than the data from the top-$k$ peers with the highest level of agreement (for different experimental values of $k$).

\(^{16}\)Also sometimes known as “Battle of The Sexes”: Two players with opposite preferences want to go to a concert, and must choose between music by Bach and music by Stravinsky: regardless of their choices, they are happier going together than going apart, but if they go to the Bach concert, then the player preferring Bach will get a higher payoff, and vice-versa.
This last negative result (this data was even less relevant than the full collection) is likely due to the very sparse nature of the collection of ratings.

Interestingly, the social network of Yelp users plays a comparable role, and even generally outperforms the agreement graph. Since these two graphs only have partial overlap, they could be combined to provide many users with a valuable data filtering mechanism.

In practice, Yelp could implement this filter based on the existing social network, then make “friend” suggestions to users excluded from this network, based on past agreement. Such a filter could advantageously replace Yelp’s existing filter, which is viewed with suspicion by businesses, and has caused several lawsuits.
Chapter 9

Conclusions

9.1 Introduction

In this thesis, we define and study a class of data sharing systems where the participants manage their own data and are connected in a social network. This design, which we call Decentralized Social Data-Sharing systems (DS)², is inspired from Online Social Networks (OSN) and early peer-to-peer (P2P) file-sharing systems (e.g. the Gnutella network), and seeks to combine their properties, repurposed for handling structured data.

The main motivation of this design is to require minimal cooperation between the participants, i.e. essentially interoperability and very basic query answering, which we believe is as much as can be required from independent and self-interested participants on the Web.

The main objective of this thesis was therefore to establish whether useful data sharing applications could be built with this design. For this purpose, we have:

- formally defined the (DS)² model and implemented a generic platform to build such applications,

- presented a useful (DS)² application, P2Pedia, which we evaluated through an
extensive user study,

- studied the problem of querying the shared data in a (DS)² system using expressive queries, namely regular path queries,

- evaluated the costs and benefits of participating in a (DS)² system from an economic perspective, through simulation.

This study validates our hypothesis: the (DS)² model is a valuable design for data sharing systems. More specifically, this design is well suited for situations where the participants’ interests are not aligned, as it reduces the situations where conflicts can materialize, and allows the participants to self-organize and avoid detrimental interactions.

However, this comes at a cost: the uncontrolled distribution of data makes it expensive to process expressive queries, and the limited cooperation makes it difficult to envision CSCW applications built on this model. However, our study of RPQ shows that some useful queries can still be handled at a reasonable cost, and our example of P2Pedia shows that a more “decentralized” form of cooperation than the classic CSCW approach can also be useful.

In the following sections, we first detail our main results and contributions, then discuss the implications of our findings with respect to the general problem of designing data sharing systems. Finally, we discuss the limitations of our research, and future work.

### 9.2 Contributions

#### 9.2.1 A Blueprint for (DS)² applications

Our first main research question was the following:
How can the principle of *maximizing the users’ autonomy* be applied to the design of a data sharing system?

As an answer to this question, we propose the design that we call *Decentralized Social Data Sharing*, or (DS)$^2$.

Our first main contribution is a general blueprint for (DS)$^2$ systems, including the following:

- A formal definition of the concept,
- A reference implementation, as a generic platform for (DS)$^2$ applications.

The interface of our generic platform uses the REST architectural style, which has the benefit of decoupling the platform from the application layer: this means that applications can be supported with very few constraints in terms of implementation language, operating system, or deployment device.

Our generic platform is validated by an example application: P2Pedia, a (DS)$^2$ wiki. In this case study, we applied the (DS)$^2$ design principles to the well-known concept of a wiki system: we modified the use cases of a wiki to accommodate the principle of decentralized control, and mapped these use cases to the (DS)$^2$ API, which is essentially the platform API.

### 9.2.2 Decentralized Cooperation with P2Pedia

After defining the (DS)$^2$ model, our second main question is whether this design can be applied to create *useful* applications.

We used the (DS)$^2$ model to design and implement P2Pedia, a novel distributed wiki application, and demonstrated its value through an extensive user study involving several hundred users.

Our study of P2Pedia makes the following main contributions:
• P2Pedia demonstrates an alternative collaboration model that allows the participants to express multiple points of view, instead of requiring a consensus;

• In a learning context, conducting exercises with such a tool allows the participants to be exposed to multiple solutions to a problem. The benefits of this, expressed by the course instructor and the participants themselves, are consistent with existing research in education: students learn mathematics better when they are exposed to multiple solutions to the same problem [158].

• the value of having multiple solutions was further corroborated by our analysis of the students’ contributions, which exhibited the following properties:

  – The edits made to the numerous different versions could not have been condensed into one “perfect” version: the top 100 most common edits (out of over 5000) require at least 3 different versions, and the top 300 most common edits (Each one occurring in over 10% of changes) require at least 7 different versions.

  – From the course instructors’ perspective, although there was not necessarily a unique solution to the problem, different versions had different values, and the popularity of the edits they contained were a partial indicator of this value.

  – The agreement between students was shown to be consistent, in the sense that past agreement is predictive of future agreement: this condition makes it possible to apply recommendation algorithms to the users’ contributions.

9.2.3 Efficiency of Query Answering

The third main question that we have addressed is the following:

Can any useful (classes of) queries be processed on data shared in a (DS)²?
We studied the processing of single-source Regular Path Queries (RPQ) on data shared in a (DS)², and made the following contributions:

- A study of the message complexity of existing distributed query processing algorithms for RPQ, adapted to the (DS)² setting,
- A new query processing algorithm whereby selective queries may be executed at a reasonable cost,
- A method to estimate the selectivity of queries, in order to identify potentially tractable queries.

RPQ are a class of queries for graph data that we found useful for our particular application P2Pedia, and that have also been recently incorporated in the SPARQL query language. An important use of such queries is to express queries over transitive properties: for example, a query to discover two resources related by a owl:sameAs property should in fact be expressed by the transitive closure (owl:sameAs)* query, which is a RPQ.

Existing distributed algorithms are not applicable to the (DS)² setting due to their underlying assumptions of data localization. We proposed a new iterative technique that is tractable in some cases. We evaluated our technique on real data and queries from the biomedical domain, and compared it mainly against the naïve (and often intractable) baseline of collecting all the data locally and processing the query over this local data.

We found that our technique does not consistently outperform the baseline, but we made some progress towards determining the cases where it does, and more importantly towards estimating the actual cost of processing the query. In other words, we showed that the tractability of RPQ can be evaluated on a case-by-case basis, given some statistics about the network topology, the replication of data, and partial knowledge of the graph of data.
Considering the execution cost as a random variable, we presented a technique to estimate the distribution of this variable using a statistical model of the graph of data. As no other techniques were previously available, we could not compare our estimation technique with any known baseline.

### 9.2.4 Costs and Benefits of Decentralization

Our fourth main research question was intended to evaluate the benefits of autonomy and their direct trade-off with technical efficiency: privileging efficiency (e.g. in a structured P2P network such as Chord [2]) would guarantee network connectivity and data availability, even under churn (the intermittent availability of peers), whereas allowing the users to connect through social connections and to store their own data breaks those guarantees.

What are the benefits of autonomy, and how do they trade off with the problem of data availability due to social connections and churn?

We addressed this question by simulating different models of a \((\text{DS})^2\), representing different economic models of the participants’ utility (payoff). We first simulated a Prisoners’ Dilemma game, where the users could strategically rewire their connections (as in a \((\text{DS})^2\)), and then investigated a more fine-grained model, built from real data shared in online social networks and availability traces from a P2P network.

Our key contributions from this study were the following findings:

- One of the key benefits of autonomy is the ability to establish social connections between participants

- In the Prisoners’ Dilemma (PD) model, this ability makes cooperation a rational strategy, leading to higher social welfare (overall utility of the participants). This confirms more general results obtained in the field of evolutionary biology, and shows their applicability to the case of a data sharing system.
• When real data is shared and has subjective value to the users, the economic cost due to unavailable data is greatly reduced compared to purely performance-oriented cost estimates: in our experiments, economic cost estimates were 50 to 65% lower than a basic performance-based estimate.

• If the data may have negative value, as in a dataset of ratings, then unavailable data due to social connections may have a positive impact on user utility: our experiments showed that this is the case with Yelp data.

• Our result with Yelp data implied that existing social connections in Yelp could be used as a subjective but efficient filter for ratings, outperforming Yelp’s existing (and controversial) filter. In addition, for users outside the existing social network, our results showed that past agreement on ratings can be used to make social connection recommendations, which would widen the applicability of this filter approach without affecting its performance.

These experiments explored the costs and benefits of autonomy using a wide range of economic models, and their results differ significantly from a purely performance-oriented cost evaluation, highlighting the importance of considering technical performance and economic aspects of system design together, rather than as separable concerns.

In addition, these results further contribute to validating the P2Pedia application: our user study showed the value of the decentralized collaboration model of P2Pedia in a somewhat controlled environment, and these economic simulations provide insights as to whether this value would generalize to other contexts.

The conclusions that can be drawn depend on the assumptions that we make on the users’ utility function:

\(^{1}\)In the sense of the number of users who would benefit from using it
• If we only distinguish between participation and non-participation, then the users’ utility function matches the economic model of a $n$-player Prisoners’ Dilemma (PD) game. In this case, we showed that the social network is key to eliciting cooperative behavior, implying that rational users would actively participate by sharing their work.

• If the value of the system to its users depends on the relevance of the available contributions, then we have shown that the social connections can also be used to improve the relevance of the available data. In addition, our results discussed previously show the potential value of using trust indicators to distinguish between relevant and irrelevant resources.

9.3 Implications for the Design of Data Sharing Systems

As stated above, our main conclusion is that the (DS)$^2$ model is well suited to accommodate participants whose interests are not aligned.

We can show the importance of the different design decisions by comparing the (DS)$^2$ model the alternative models obtained by choosing the alternative to each of these design decisions: centralized systems, distributed systems optimized for technical performance (e.g. DHT), and traditional cooperative repositories.

9.3.1 Centralized Systems

Centralized systems allow for much more efficient query processing and access to data. The (DS)$^2$ logical model could be fully instantiated on a centralized architecture. This would then resemble an Online Social Network.
An example is our tool SocialWiki, which we implemented\(^2\) to reproduce the P2Pedia functionality as a Moodle\(^3\) plugin. The \((\text{DS})^2\) functionality can easily be emulated in this context, with most of its benefits (further discussed below).

However, they have an “organizational” single point of failure, and the participants’ autonomy is limited by the host, which is self-interested, and whose interests can be at odds with those of the participants.

### 9.3.2 Distributed Systems

The inefficiencies of the \((\text{DS})^2\) architecture are largely due to the direct mapping between the social connections and the physical interconnections of peers.

Within distributed architectures, an alternative would be a structured network, optimized for performance. The advantage is that compared to a centralized architecture, such a system would not have a single point of failure, and the operating costs would be shared between the participants. However, optimizing their performance requires cooperation between the participants, and may compromise their ability to control who they cooperate with; for example, in terms of network management, the typical API of a structured network only offers `join()` and `leave()` functions [54]. This may lead them to find themselves in non-cooperative interactions with other participants, which has a cost. We have shown that this cost can take the form of consuming irrelevant data, and our Yelp case study shows that this cost can be measurable and significant.

An open question is whether the \((\text{DS})^2\) logical model could be implemented as a logical overlay to an (optimized) physical network, which might provide more efficient communication than an unstructured network matching the social connections. The

\(^2\)with the helpful participation of several undergraduate students, acknowledged at the beginning of this thesis

\(^3\)Moodle is a content management system for learning material, used to manage course materials by Carleton University, as well as many other universities
P2P system Tribler [73] follows such a design. The key problem is whether the security features of the system can prevent the participants of such a network from imposing unwanted interactions on peers that they are not socially connected to, and from disrupting interactions between other socially connected peers.

This may be a good approach if the participants’ interests are sufficiently aligned for them to cooperate, in the sense of implementing the prescribed algorithm.

9.3.3 Collaborative Repositories

A meaningful way of comparing the (DS)$^2$ model with a collaborative repository would be to design two closely related systems, one based on each collaboration model, then apply them to the same general task and compare the outcomes of these tasks. Without having formally conducted such an experiment, we can consider the example of wikis to support course material, a topic that has been widely published about, and can be compared with anecdotal evidence from our own experience.

Wikis are often used to create and collect learning material for courses, but their traditional collaboration model has sometimes failed to motivate the students [159]. Several studies have identified potential conflicts as a reason for users not to contribute: users were reluctant to alter the work of others [160], or did not want to see their own work altered [159, 161].

Several professors (at least two in the Systems & Computer Engineering department of Carleton University) also use wiki systems to edit their own course material, but do not open up contribution to their students. Presumably, they are concerned about their pages being vandalized.

In contrast, our Moodle plugin SocialWiki, that implements very similar functionality to P2Pedia, allowed us to share course material and open up contributions to the students. There was no risk of vandalism, as the students’ contributions could not replace any of our own, and any poor quality contributions could be allowed as
well, since they remained associated to the contributor’s account and could not be mistaken for the course instructor’s opinion.

Extending this idea, the material collected in a social wiki could then in turn be shared with instructors of similar courses in other universities. The Wikimedia Foundation project Wikiversity is a project inspired by Wikipedia, where university-level learning material is designed collaboratively, following the principles of traditional wikis.

In practice, we believe that instructors rarely make use of a single source of learning material, and often prefer to combine various textbooks, selecting material according to their personal views on the topic. A P2Pedia-based approach could be a better fit for this scenario.

9.4 Limitations and Future Work

9.4.1 Query Processing with Limited Cost

We have shown that processing expressive queries (RPQ, specifically) on the data shared in a (DS)$^2$ can be prohibitively expensive, with our technique as well as with other available algorithms.

While our technique has the advantage that it can be interrupted, which would limit the cost, this would also cause the query answers to be incomplete.

Based on the idea that not all the data has equal value (as explored primarily in chapter 8), querying the data with a bounded cost should favor the expenditure of this cost towards obtaining the most valuable (relevant) answers.

In P2Pedia, the results of queries can be ranked using several trust indicators. These are primarily useful when the relevance of a resource (e.g. a P2Pedia page) can be assessed on its own, and independently of the query path that lead to it. In
this case, relevant answers can be obtained by following paths that include irrelevant resources.

On the other hand, if the edges in the graph represent logical statements, such as the relationship `owl:sameAs` used on the Semantic Web, then relevance can be interpreted as an assessment of whether this statement is trustworthy. In this case, relevant answers to a query can only be obtained by following paths through relevant data.

In this case, the problem of retrieving the most valuable query answers can be addressed by applying trust policies at query time, rather than by ranking results. More specifically, the idea would be to iteratively process the query, but rank the intermediate results and process them in a “best first” approach, favoring the most relevant data and possibly “cutting off” some processing altogether if the intermediate data is not relevant enough.

This would an interesting direction for future work, but an obstacle towards such research is the lack of datasets on which such approaches could be validated.

### 9.4.2 Directed Social Networks

Our initial intuition in defining the (DS)² concept that the social network should not represent “friendship” (a naturally symmetric relationship) but rather a form of trust of interest, perhaps best expressed by the “follow” semantics, and therefore a directed relationship. This motivated us to modify the Gnutella peer-to-peer network protocol to handle the connections as directed.

However, when we validated the usefulness of the social network, existing work (i.e. the prisoners’ dilemma in a social network) and available datasets (Last.fm, Yelp) all used undirected social networks, and our results only hold for such undirected networks.

The consequence of directed relationships on the cooperation issues that we have
studied is an open problem. However, we do not anticipate radically different results, but rather different interpretations of the incentives and available strategies.

9.4.3 Economics of Trust Indicators

The games that we studied in chapter 8 have focused on the users’ ability to form a social network with users who were cooperative, either in the sense of contributing rather than free-riding, or in the sense of contributing (objectively or subjectively) relevant data.

Beyond this, the users’ ultimate goal is to consume relevant data, and we have shown that by strategically managing their connections, users can improve the general relevance of the available data. However, if users are able to distinguish between relevant and non-relevant data (and for example, specify this filter in a query), then it is of no importance that irrelevant data is available.

Most likely, we would expect that in a realistic application, users can partially specify relevance criteria in their queries, in the sense that a query should return predominantly relevant data, but will also return some irrelevant data. For example, in the P2Pedia writing tutorials, we could imagine the users using the data structure to query for the latest document versions, the most popular versions, or those with the most separate edits, but these are probably only correlated with the users’ personal relevance criteria, which cannot be expressed formally.

Therefore, a higher-level investigation of this problem needs to consider the available relevance criteria, including our trust indicators and drawing from domains such as Information Retrieval and Computational Trust. A user’s optimal behavior is likely to be a complex strategy involving both strategic connections to shape their network, and appropriate filtering criteria to rank and filter the available data.

However, one of the difficulties in validating trust indicators, or in general relevance ranking functions for structured data is that validation benchmarks for such
data are particularly difficult to find. This may explain why ranking metrics for structured data such as ReConRank [63] are typically only evaluated with respect to the cost of calculating them, rather than their true performance at identifying relevant content. To the best of our knowledge, the only available domain in which some kind of benchmark exists is the domain of Recommender Systems, where relevance evaluation can be framed as evaluating rating predictions, as discussed in chapter 8. However, ratings are not a very good example of structured data, as it seems difficult to apply any meaningful graph queries to a dataset of ratings.

9.5 Derived Publications and Tools

9.5.1 Conference and Journal Articles

The work presented in this thesis resulted in the following publications:


Among these papers, papers 1, 2 and 4 presented our work on P2Pedia: paper 1 presented the system, paper 2 presented the user study, and paper 4 is an overall presentation of the project, including a rewritten technical presentation, more complete results of the user study, and additional related work.

Paper 3 describes presents some preliminary research on simulating (DS)$^2$ systems (before we coined the term), briefly discussed in section 8.4 and in appendix A.

Paper 5 presents a survey of distributed wikis, which was done following our work on P2Pedia in collaboration with a team of researchers from the University of Nantes, France. While most of the contents of that paper do not appear in this thesis, they greatly shaped our understanding of (DS)$^2$ systems and related approaches to data sharing, as these different approaches underly different existing distributed wikis.

Finally, we note that the results of chapters 6 (about the processing of regular path queries) and 8 (our simulation study about cooperation in a (DS)$^2$) are being prepared for submission or re-submission.

9.5.2 Tools

The (DS)$^2$ application P2Pedia, as well as the generic platform U-P2P have been released as open source tools.

The Moodle plugin SocialWiki, which implements the logical (DS)$^2$ model but runs on a centralized server, is also open source and publicly available.
List of References


Appendix A

DEVS Simulation of \((DS)^2\)

A.1 Introduction

We briefly present here an exploratory study towards the simulation of \((DS)^2\) systems, briefly discussed in section 8.4, using DEVS \([145]\), a formalism to model and simulate systems with discrete states in continuous time, as opposed to the discrete timesteps of most simulation tools. Eventually, we found that the granularity of simulation that this methodology affords was not appropriate for the phenomena that we were attempting to investigate through simulation.

However, since P2P networks had not previously been simulated with the DEVS formalism, and since our models go further than basic network simulation by modelling graph-structured data, we believe that this work is nonetheless valuable, and may provide a starting point for future work in different directions.

We first present our DEVS model, then a proof-of-concept simulation study where we simulate peers browsing a collection of hyperlinked documents.
A.2 DEVS Model Framework

We now present the realization of this general model using the DEVS formalism. At this level of abstraction, our formal model can be represented as a framework for a coupled DEVS model.

Our framework model has the structure illustrated in Figure A.1. In the following sections we detail the components of the model, and for each component, we specify the behaviour of our default implementation.

A.2.1 Network Model

Network Graph

The NetworkGraph model keeps track of the overlay network topology. It is a state machine, and its inputs are messages from the peers that go online, offline, or attempt
to connect to other peers. The component only changes the graph state following legal transitions, and outputs messages to notify peers of successfully established connections, and of dropped connections.

Our default implementation manages this functionality, and introduces only minimal latency: successful connections are almost instantaneous.

**Physical Network**

This model transfers messages from one peer to another. It represents the network connectivity: the component accepts input messages from any peer, and outputs the same message to a specified destination peer.

Our implementation of the model delivers the messages reliably, with a random latency, following a configurable statistical distribution (independent of network congestion). For more fine-grained performance studies, this model would need to be replaced by a more complex model of the network, e.g. a packet-level simulator such as NS-2.

### A.2.2 Peer Model

**Connection Manager**

This component represents the behaviour of a peer regarding connections, i.e. the “topology” aspect of a search mechanism. The peer communicates with the Network-Graph component to establish connections to other peers, and maintains a list of its neighbours.

In our implementation of this component, the component is initially configured with a list of acquaintances. Its behaviour is then to periodically attempt to connect to each of its acquaintances. When the connections are established, the component remains idle. This behaviour creates an unstructured network based on a social
network.

**Router**

The router component represents the query routing behaviour of a peer. The component maintains the current list of neighbours of the peer (based on inputs from the NetworkGraph component), and receives input queries, both from the local peer (from the SessionManager component) and from other peers, via the network. The peer then routes the queries to its neighbours, implementing a specific search protocol.

Our implementation runs the Gnutella protocol v.0.4 [162].

**Repository**

The repository component maintains the list of documents stored locally by the peer. Its inputs are the publish and remove operations discussed in chapter 3. This model is also used to answer queries; for this purpose, it takes queries as input, and has an output for query answers.

Our default implementation of this model is based on a graph loaded on initialization of the model. This graph is a bipartite graph where query nodes are related to document nodes by edges representing the fact that the query matches a document. It then separately maintains the list of documents stored by the peers, based on the “publish” and “remove” inputs. The model responds to queries with a small, fixed latency.

**Session Manager**

The Session Manager represents the behaviour of the human user of the P2P application.

The outputs of this model are the events initiated by a human user: the peer begins a session and goes online, creates queries, or manages its local repository, by
publishing or removing documents.

We discuss our implementation of this model in more detail in section A.2.3.

In the real-time simulation, this model can be replaced by input from an external API.

**MsgIdGen**

This model is a simple convenience model that is used by all the peers to generate globally unique identifiers. Many P2P protocols, including Gnutella, make use of unique identifiers, which are typically large integers generated by random functions or hash functions. Using an external model to allocate sequential but unique IDs avoids the need for large values and the risk of collisions.

**A.2.3 A Behavioural Model for Peers**

The behaviour of the human users is an important aspect of simulating P2P networks, and for our specific research it is fundamental. Although a number of measurement studies exist, to the best of our knowledge, no model has emerged as a reference benchmark for simulating P2P networks. In this section we briefly review the measurement studies of P2P user behaviour, then describe our generic model, which is the basis of the Session Manager DEVS model described above.

**Measurement studies**

The main aspects that have been measured in P2P file-sharing networks are on one hand *churn*, which is the phenomenon of peers joining and leaving the network over time, and on the other hand the querying and downloading behaviour of peers during their active sessions.

Churn, characterized by session duration, peer inter-arrival time, and downtime,
has been analyzed in several networks, including the Gnutella, Overnet, Kad and BitTorrent networks [27, 148, 163].

Stutzbach et al. [163] show that the overall distribution of sessions lengths can be fit to Weibull distributions, and Bhagwan et al. [27] analyze the combined influences of daily activity patterns, with overall long-term peer turnover.

Klemm et al. [148] provide models for the number of queries and their inter-arrival time within peer sessions, and show significant differences between geographical regions (North America, Europe, East Asia), both in query frequencies and session durations.

The popularity of queries and content was analyzed in different networks by Gummadri et al. [164] in 2002, then by Klemm et al. [148, 149] in 2004, then more recently by Dan and Carlsson [147] (2010).

There are several major challenges in adapting these statistical models for the purpose of simulation. For one thing, the available studies do not all agree on their conclusions.

Secondly, for applications that are not about sharing music and video, there is little or no data available. We can assume certain similarities, but the exact distribution parameters are unlikely to generalize across domains. This problem was faced by Siberski et al. for the simulation of their system Edutella [165], and we are exploring entirely new applications, e.g. our P2P wiki P2Pedia.

For these reasons, we conclude that it is more important to offer a model that is generic and easy to configure, rather than one that closely matches all of the statistical distributions reported in the literature. Therefore, we have selected the behaviour that appears most general and unrelated to the application context, and made the statistical distributions configurable.
A Stateful User Model

From the studies referenced above, we found that peers had a general stateful behaviour, which follows a somewhat intuitive pattern. This pattern, illustrated in Figure A.2, can be described as follows:

1. The peer starts offline, and after some random time, goes online, and chooses between an active and a passive session;

2. - in an active session, the peer outputs a number of queries, and upon receiving responses to the queries, may choose some files to download. The downloaded files are published.
   - in an inactive session, the peer outputs no queries, but is online, and routes and responds to queries from others.

3. After a random “sleeping time”, the peer restarts the cycle.

The randomized aspects of this behaviour can be configured for each peer:

- offline time;
- probability of selecting an active session;
• in an active session: number of queries, inter-query time, after-querying time, probability of downloading files from queryhits;

• duration of an inactive session.

A.3 Random Surfers in a P2P Web

As a proof-of-concept simulation of sharing graph-structured data, we transposed the activity of “surfing the Web” (browsing a collection of hyperlinked pages) to a hypothetical (DS)^2 application, where the users access the documents through their social connections, and make the pages they download available to others.

We have simulated a very simple behaviour, the “random surfer model” used as an intuitive justification of the PageRank algorithm, by Brin and Page [59]: the “random surfer” is given a web page at random and keeps clicking on links, never hitting “back” but eventually gets bored and starts on another random page.

In a traditional Web, if the “surfers” browse the collection randomly, then the number of visits to each page should be proportional to the page’s PageRank value.

The purpose of this experiment was to compare this with the P2P Web setting.

Files in a P2P network are not permanently accessible to all the peers: the files stored by a peer are accessible only when the peer is online, and only from other peers reachable by the search protocol. The random surfer model must therefore be adapted. Our P2P random surfer keeps clicking on links, and from time to time will get bored, but will also only follow links for pages that are available in the network. If no pages are available, then the surfer searches for another random page.

In a P2P web, the visits to a page are conditional to the page’s availability, which in turn is conditioned to past visits by other surfers (who make it available). It is therefore interesting to study the effect of this circular dependency on the distribution of pages (the number of copies available), and compare it with the PageRank scores
Figure A.3: Replication of Web Pages in a P2P Web Surfing Application

of the pages.

The results of our study show a positive correlation between the number of copies of each page and their PageRank score, visible in the graph of figure A.3.

A.4 Conclusion

This study was an exploratory analysis of the idea of a “friend-to-friend” P2P network, which is the principle of our data sharing platform presented in chapter 5. This study has shown the importance of the data’s distributional properties, that directly affect their availability to the users. In a way, the offline users and network partitions act as a filter that limits the data that each user can access. This suggests that the properties of this “filter” should be considered when evaluating a user’s incentive to modify the social network or copy data.
A.4.1 Reusing the Model

The modular structure of the DEVS models makes it easy to adapt and reuse the peer model for a particular problem.

The CD++ toolkit includes most of the classic probability distributions, including all of those reported in the analytical studies referenced above. Modifying a probability distribution does not require modifying the model source code, but only a configuration file, where the name of the distribution and its parameter values must be set.

In addition, each peer in the network can be configured separately: this allows the designer of the network to take classes of peers into consideration. For example, one could create peers from different geographical regions, and assign them the different distributions reported in [148]. For large-scale networks, this configuration can be done easily with scripts.

Finally, we note that an appropriate strategy to use this model would be to configure the time-related session parameters, then to manually code more advanced logic for choosing queries and downloading files. In our case study, this allowed us to model a user “surfing” a hyperlinked collection of files.