Dissertation

The SnoopyConcept: Leveraging Recommendations for Knowledge Curation

Wolfgang Gassler

submitted to the Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck

in partial fulfillment of the requirements
for the degree of "Doktor der technischen Wissenschaften"

Advisor: Univ.-Prof. Dr. Günther Specht

Innsbruck, August 11, 2017

Abstract

In recent years, online knowledge curation on the internet has been lifted to a new level—online mass-collaboration. Knowledge bases such as Wikipedia and Wikidata are curated by huge communities and produce a vast amount of knowledge. Therefore, the prevailing challenge is to maintain a homogeneous structure in those knowledge bases to ensure efficient search capabilities, allow automated reasoning, and provide semantically linkable knowledge for the Linked Open Data cloud. In this thesis we propose the SnoopyConcept which leverages recommender systems to support the user to homogenize knowledge already during the insertion process. The recommendations furthermore aim at increasing the quality and quantity of stored knowledge in collaborative information systems. In addition, we implement the universal SnoopyConcept in several domains and systems to evaluate and assess the recommender system based approach.

Acknowledgements

Eva, Sarah, and Robert, thanks for 10 incredible years; we shared almost everything. We enjoyed the good moments, and handled the bad. Robert, you were not only my flat mate, you also challenged me every day, by sharing your deep technical knowledge. Thanks for all the fun weekends in the office working on world-shaking algorithms and software. And thanks for being such a good friend all the time. Sarah, you introduced me to the beautiful world of spontaneity, taught me how to be happy without being in control of everything, and pushed me out of my comfort zone to make bold steps. I never would have been at this point without you; thank you a lot for everything. Eva, you have accompanied me through the world of academia for more than 15 years. We've worked on the "Weltfrieden", survived rainy days, and have been riding the waves. Thanks for listening, supporting, and coaching me everywhere and anytime, and of course, being my best friend.

Thanks to all the DBIS members: Niko, Doris, Rainer, Gabi, Sylvia, Peter, Michael, Matthias, and Lukas. Special thanks goes to: Michi, Martin, Seppi, Skifahrlehrer Schorsch, Hansi, and Dominic for all the fun we had together. Thanks to my supporting friends: Herbert, Ingä, Stefanie, Jakob, —Zugi—, and Niki who never got tired of nudging me about my thesis — your relentless persistence paid off.

I also want to thank my advisor Günther Specht for giving me the opportunity of writing this thesis, all the resources, and his support the whole time, even after I left DBIS after eight instructive years.

A special "thank you" goes to my whole family, especially my parents who have never forced me to do anything and have always supported me, no matter how ridiculous my idea was.

Thanks to all the amazing members and students at the Institute of Computer Science Innsbruck who taught me a lot, even outside the world of science.

This work was only possible due to the huge amount of available Open Source tools. Only Open Source tools were used to build and evaluate the SnoopyConcept and write all related publications and this thesis.

Thanks a lot to all the Open Source contributors.

This work was partially funded by the Austrian Research Promotion Agency (FFG) and the Nachwuchsförderung of the University of Innsbruck.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht. Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister-/Master-/Diplomarbeit/Dissertation eingereicht.

Datum	Wolfgang Gassler

The beautiful thing about learning is nobody can take it away from you.

B.B. King, 1997

Contents

Abstract I

Acknowledgements III

1 Introduction 1
- 1.1 Aims and Research Questions . . . 3
- 1.2 Contributions and Published Work . . . 3
- 1.3 Thesis Outline . . . 7

2 Background & Related Work 9
- 2.1 Information Representation & Structure . . . 9
  - 2.1.1 Strictly Structured Information Systems . . . 10
  - 2.1.2 Unstructured Information Systems—Wikis . . . 10
  - 2.1.3 Semi-structured Data . . . 11
- 2.2 Wiki Systems and Knowledge Curation . . . 12
  - 2.2.1 Wikipedia and its Structural Limitations . . . 13
  - 2.2.2 Wiki Enhancements . . . 14
  - 2.2.3 Semantics, Wikidata & Knowledge Bases . . . 16
- 2.3 Recommender Systems . . . 20
  - 2.3.1 Introduction to Recommender Systems . . . 21
  - 2.3.2 Classification . . . 22
  - 2.3.3 Collaborative Filtering . . . 22
  - 2.3.4 Content-based Recommender Systems . . . 26
  - 2.3.5 Knowledge-based Recommender Systems . . . 27
  - 2.3.6 Context-aware Recommender Systems . . . 28
  - 2.3.7 Hybrid Recommender Systems . . . 29
  - 2.3.8 Summary . . . 30

2.4 Recommender Systems in Information Systems 30
2.4.1 Synonyms and Search Capabilities 31
2.4.2 Guided Refinement and Enrichment 33
2.4.3 Guidance during the Insertion 37
2.5 Database Models 39
2.5.1 Relational Database Systems 39
2.5.2 NoSQL: Document-oriented Stores 40
2.5.3 NoSQL: Graph Stores 42
2.6 Summary 44

3 The SnoopyConcept 45
3.1 Key Idea: Incorporate the User 45
3.2 Data Model 49
3.3 Recommendations 50
3.3.1 Recommending Structure 50
3.3.2 Avoiding Synonyms by Recommendations 52
3.3.3 Semantic Refinement and Value Recommendations . . . 53
3.3.4 Validation and Recommendations of Data Types 54
3.3.5 Ranking 54
3.4 Recommendation Algorithm 55
3.4.1 Basic Recommendation Algorithm 56
3.4.2 Ranking Algorithms 61
3.4.3 Context-Sensitive Optimization 63
3.4.4 Refinement Recommendations 65
3.5 Personalizing the SnoopyConcept 67
3.5.1 User Modeling 67
3.5.2 Algorithm Extension by User Modeling 68
3.6 Tackling the Cold-Start Problem 71
3.7 Summary 76

4 SnoopyConcept Storage Models and Recommendation Implementation 77
4.1 Relational Database Systems 78
4.1.1 RDF Storage Models 78
4.1.2 RDF Storage Index Structures 80
4.1.3 SnoopyConcept Relational Storage Model 82
4.1.4 Recommendation Computation 84
4.1.5 Rule Based Recommendation 85
4.2 NoSQL: Document-oriented Stores 88
4.2.1 Storage Model 88
4.2.2 Recommendation computation 89
4.2.3 Distribution and Sharding 91
4.3 NoSQL: Graph Stores 93
4.3.1 Storage Model 94

4.3.2 Graph-based Rule Generation . . . 95
4.3.3 Graph-based Recommendation Algorithm . . . 96
4.4 Storage Model Evaluation . . . 98
4.5 Storage Models Summary . . . 102

5 SnoopyConcept Showcases 105
5.1 First Reference Implementation: SnoopyDB . . . 105
5.2 SnoopyTagging . . . 106
5.2.1 Structured Tags . . . 107
5.2.2 Recommendations based on the SnoopyConcept . . . 107
5.3 hash5 . . . 109
5.3.1 Main Concepts . . . 109
5.3.2 Recommendations based on the SnoopyConcept . . . 111
5.4 Wikidata . . . 113
5.4.1 Wikidata Property Suggestor . . . 114
5.4.2 Incorporating the SnoopyConcept Algorithm . . . 115
5.5 Summary . . . 116

6 Evaluation 117
6.1 Recommendation Algorithms . . . 117
6.1.1 Offline Evaluation . . . 118
6.1.2 Wikidata Property Suggestor Evaluation . . . 129
6.1.3 Online Evaluation/User Experiment . . . 134
6.2 User Modeling Evaluation . . . 137
6.3 Evaluation Summary . . . 140

7 Conclusion 141

Bibliography 145

CHAPTER 1

Introduction

Scientia potentia est[1] has emphasized the importance of knowledge for centuries. Especially since the beginning of the information age [37], the economy has been based on information computerization and thus, the access to knowledge has become crucial. Information systems or knowledge bases are used to store knowledge and provide users access to knowledge. In the past, information systems were solely used by specialists and usually were maintained by a small group of people. This has drastically changed with the rise of the internet and the web 2.0 movement which has lifted curation of knowledge to a new level—online mass-collaboration.

In mass-collaborative information system, such as Wikipedia[2], thousands of people with different backgrounds collaborate to curate knowledge. Those platforms usually do not restrict the user to use a predefined structure to store and structure information. The shortcoming of this structure-less paradigm is its limited search capability. Consider a complex query such as "Which Austrian cities have more than 10.000 inhabitants and have a female mayor who has a doctoral degree?". It is not possible to answer such a query through full-text search which is provided by most wiki-systems. Weikum *et al.* [171]

[1] Knowledge is power, https://en.wikipedia.org/w/index.php?title=Scientia_potentia_est&oldid=784909874 (revision 10 June 2017)

[2] http://www.wikipedia.org, accessed 2017-07-17

observed that modern information systems have to be able to support both structured and unstructured data to combine the advantages of both worlds and to be able to answer such questions. One possible solution is the usage of semi-structured information systems (*cf.* Section 2.1.3) which allows for defining a structure but do not require a predefined schema. For example tagging systems, fact based knowledge bases, document-oriented databases or XML files provide the possibility to store any arbitrary data in a structured way without adhering to a predefined schema.

The flexibility of the previously described schemaless semi-structured storage in combination with collaborative data curation leads to a massive problem. Every single user has her own view of structuring knowledge and information and uses her own terminology. Furnas *et al.* [59] already showed in the 80s that two people would spontaneously choose the same word for an object with a probability of less than 20%. This suggests that collaboratively built knowledge based on the semi-structured model shows a very high proliferation of structures, schemata and vocabulary. The resulting heterogeneous schema impedes the search facilities as a common schema is essential to answer complex structured queries. E.g., a user who searches for *numberOfStudents* cannot find information which was stored using the properties *students*, *numberStudents* or *num_students*. Therefore, especially in collaborative knowledge systems, the creation of a common schema without restricting the domain, type or amount of information is desired. Wikipedia is fighting such a heterogeneity by introducing collaboratively created templates and for the community an extended supervision by the committed community. The task of creating structure in Wikipedia is very demanding task for the community as shown by Boulain *et al.* [26]. The authors analyzed the edits in Wikipedia and identified that only 35% of all edits within Wikipedia are related to content, whereas all other edits aim at enhancing the structure within the Wikipedia knowledge base.

In this thesis, we propose a self-learning system which leverages recommender systems to guide the user to a homogeneous schema without restricting her in her way of structuring information. The recommendations are based on already stored information in the system and aim to prevent synonyms or different structures which would impede the search capabilities.

Another severe problem in collaboratively built knowledge bases is the barrier for new users to insert information to knowledge bases. In Wikipedia, most of the content is created by a very small group of users [94]. Furthermore, articles have to conform to many policies and other regulations which increase the barrier for contributions by newcomers [133, 162]. But not only in public knowledge bases, moreover and especially in enterprise knowledge bases or wikis, the poor adoption rate of content constitutes a major problem. Besides social group phenomena, especially the high costs for users to contribute and

manage wikis (e.g. wiki syntax, complex user interfaces, etc.) is the main reason for a bad contribution rate [97].

Therefore, we introduce the SnoopyConcept in this thesis which aims at incorporating the user already during the insertion process. An intuitive and guided process facilitated by recommendations, supports the user to curate semi-structured knowledge with a common and normalized schema. The goal of this assistance is to lower the entry barrier and increase the amount of stored knowledge in the information system.

1.1 Aims and Research Questions

In the introduction we mentioned several challenges in the area of collaborative information systems which are tackled by the SnoopyConcept we present in this thesis. The SnoopyConcept aims at incorporating the user already during the insertion process to homogenize the structure of stored knowledge and thus, improving the search capabilities. Therefore, we propose to leverage recommender systems to recommend highly suitable structures based on already stored knowledge in the system. The overall goal of providing guidance and recommendation to the user, is to increase the quality and quantity of knowledge stored in the information system. The aims can be summarized by the following research questions which are addressed in this thesis:

- How can recommender systems empower collaborative information systems to become more structured without losing their flexibility?
- How can direct user communication during the insertion process be facilitated by recommender systems to increase the quality of information?
- How can automated user guidance by a recommendation system result in an increased quantity of information?

1.2 Contributions and Published Work

This thesis mainly covers contributions that have been made and published between 2009 and 2014.

The main contribution is the SnoopyConcept which tackles the previously described research questions by providing smart recommendations already during

the insertion process. This approach is the first work which facilitates a recommender system to prevent schema proliferation and at the same time increase the amount of knowledge which is stored to the information system. The core idea of the SnoopyConcept was firstly published in 2010 at the ACM Conference on Hypertext and Hypermedia [61]. Subsequently, we identified different types of recommendations which were implemented in a fully functional prototype called *SnoopyDB*. At the ACM Recommender System Conference 2010, the most widely known conference on Recommender Systems, a more detailed description about the underlying algorithms was published [182]. At the CTS conference 2011 (International Conference on Collaboration Technologies and Systems), the paper "The Snoopy Concept: Fighting Heterogeneity in Semistructured and Collaborative Information Systems by using Recommendations" [60] was nominated for the best paper award. Based on this paper, an extended version was published in the *Journal on Future Generation Computer Systems* (impact factor 1.978) [64] and an extended survey about approaches tackling structure heterogeneity was published as a book chapter with the title "Dealing with Structure Heterogeneity in Semantic Collaborative Environments" in the book "Collaboration and the Semantic Web: Social Networks, Knowledge Networks and Knowledge Resources" [183]. We furthermore developed different models to implement the storage system and recommendation engine of the SnoopyConcept which are presented and discussed in Chapter 4. In this chapter, we also propose a scaling approach to cope with very big datasets. Besides the SnoopyConcept reference implementation SnoopyDB, we evaluated the SnoopyConcept in several other domain. For this purpose, we developed several prototypes which are described in detail in Chapter 5. The SnoppyTagging approach which implemented the SnoopyConcept and furthermore, extended the algorithm by incorporating user preferences, was published at the World Wide Web Conference 2012 (WWW2012) [63]. The developed personal information system hash5 incorporates the SnoopyConcept and additionally, recommends structures, tags, and values based on the full-text in PIM entries. The recommendation of Wikipedia Infobox structures using the SnoopyConcept approach was published in 2011 at the ACM Conference on Hypertext and Hypermedia [103]. Still in 2016 the SnoopyConcept approach was relevant and we proposed algorithms based on the SnoopyConcept to recommend structures in Wikidata at Wikipedia's conference OpenSym 2016 [184]. The following list contains all my publications related to the SnoopyConcept.

- W. Gassler, E. Zangerle, M. Tschuggnall, and G. Specht. SnoopyDB: Narrowing the Gap between Structured and Unstructured Information using Recommendations. In *HT'10, Proceedings of the 21st ACM Conference on Hypertext and Hypermedia, Toronto, Ontario, Canada, June 13-16, 2010*, pages 271–272, 2010

- E. Zangerle, W. Gassler, and G. Specht. Recommending Structure in Collaborative Semistructured Information Systems. In *Proceedings of the fourth ACM Conference on Recommender Systems*, pages 261–264, Barcelona, Spain. ACM, 2010

- W. Gassler and E. Zangerle. Recommendation-Based Evolvement of Dynamic Schemata in Semistructured Information Systems. In *Proceedings of the 22nd Workshop Grundlagen von Datenbanken (GvDB 2010), Bad Helmstedt, Germany.* CEUR-WS.org, 2010

- W. Gassler, Zangerle, and G. Specht. The Snoopy Concept: Fighting Heterogeneity in Semistructured and Collaborative Information Systems by using Recommendations. In *The 2011 International Conference on Collaboration Technologies and Systems (CTS 2011)*, pages 61–68, Philadelphia, PE, 2011

- A. Larcher, E. Zangerle, W. Gassler, and G. Specht. Key Recommendations for Infoboxes in Wikipedia. Website of the 22nd ACM Conference on Hypertext and Hypermedia, 2011. Poster Presentation

- E. Zangerle and W. Gassler. Dealing with Structure Heterogeneity in Semantic Collaborative Environments. In *Collaboration and the Semantic Web: Social Networks, Knowledge Networks and Knowledge Resources.* IGI Publishers, Hershey, Pennsylvania (USA), 2012

- W. Gassler, E. Zangerle, M. Bürgler, and G. Specht. SnoopyTagging: Recommending Contextualized Tags to Increase the Quality and Quantity of Meta-information. In *Proceedings of the 21st International Conference Companion on World Wide Web*, WWW '12 Companion, pages 511–512, Lyon, France. ACM, 2012

- W. Gassler, E. Zangerle, and G. Specht. Guided Curation of Semistructured Data in Collaboratively-built Knowledge Bases. *Journal on Future Generation Computer Systems*, 31:111–119, May 2014. impact factor 1.978.

- E. Zangerle, W. Gassler, M. Pichl, S. Steinhauser, and G. Specht. An Empirical Evaluation of Property Recommender Systems for Wikidata and Collaborative Knowledge Bases. In *Proceedings of the 12th International Symposium on Open Collaboration, OpenSym 2016, Berlin, Germany, August 17-19, 2016*, 18:1–18:8. ACM, 2016

Besides the thesis related publications, also contributions have been made in the field of databases [65], main memory graph storages [22], Twitter hashtag recommendations [186, 187, 188] and music recommendations [185, 189]. The following list contains all other publications which were made throughout the course of my PhD studies:

- R. Binna, W. Gassler, E. Zangerle, D. Pacher, and G. Specht. SpiderStore: Exploiting Main Memory for Efficient RDF Graph Representation and Fast Querying. In *Proceedings of the 1st International Workshop on Semantic Data Management (SemData) at the 36th International Conference on Very Large Data Bases (VLDB 2010), Singapore*. CEUR-WS.org, 2010

- W. Gassler, E. Zangerle, and G. Specht, editors. *Proceedings of the 23rd GI-Workshop "Grundlagen von Datenbanken 2011", Obergurgl, Austria, May 31 - June 03, 2011*, volume 733 of *CEUR Workshop Proceedings*, 2011. CEUR-WS.org

- R. Binna, W. Gassler, E. Zangerle, D. Pacher, and G. Specht. SpiderStore: A Native Main Memory Approach for Graph Storage. In *Proceedings of the 23nd Workshop Grundlagen von Datenbanken (GvDB 2011), Obergurgl, Austria*. CEUR-WS.org, ISSN 1613-0073, Vol. 733, 2011

- E. Zangerle, W. Gassler, and G. Specht. Recommending #-tags in Twitter. In *Proceedings of the Workshop on Semantic Adaptive Social Web 2011 in connection with the 19th International Conference on User Modeling, Adaptation and Personalization, UMAP 2011*, pages 67–78, Gerona, Spain. CEUR-WS.org, 2011

- E. Zangerle, W. Gassler, and G. Specht. *Using Tag Recommendations to Homogenize Folksonomies in Microblogging Environments*. In *Social Informatics: Third International Conference, SocInfo 2011, Singapore, October 6-8, 2011. Proceedings*. Springer Berlin Heidelberg, 2011, pages 113–126

- E. Zangerle, W. Gassler, and G. Specht. Exploiting Twitter's Collective Knowledge for Music Recommendations. In *Proceedings, 2nd Workshop on Making Sense of Microposts (#MSM2012): Big things come in small packages, Lyon, France, 16 April 2012*, pages 14–17, 2012

- E. Zangerle, W. Gassler, and G. Specht. On the impact of text similarity functions on hashtag recommendations in microblogging environments. English. *Social Network Analysis and Mining*, 3(4):889–898, 2013

- E. Zangerle, M. Pichl, W. Gassler, and G. Specht. #nowplaying Music Dataset: Extracting Listening Behavior from Twitter. In *Proceedings of the 1st ACM International Workshop on Internet-Scale Multimedia Management*, ISMM '14, pages 21–26, Orlando, Florida, USA. ACM, June 2014

1.3 Thesis Outline

This dissertation is structured as follows. Chapter 2 features an introduction to information systems, Wikis and recommender systems in the area of information systems. The chapter also covers database models that are used throughout the course of this thesis. In Chapter 3, we present the SnoopyConcept and the key idea to use recommender systems to increase the quality and quantity of knowledge in information systems. Chapter 4 presents the implementation of the SnoopyConcept with regard to used storage models and optimizations of the proposed recommendation algorithms. In Chapter 5, we describe our reference implementation of the SnoopyConcept and present several other showcases which incorporate the SnoopyConcept in different domains. Chapter 6 presents and discusses the results of the conducted offline and online experiments which evaluate the SnoopyConcept algorithms. Chapter 7 concludes this thesis.

CHAPTER 2

Background & Related Work

The SnoopyConcept aims at leveraging recommender systems to support collaborative knowledge curation. Therefore, we focus in this chapter on foundations and related work in the area of collaborative knowledge curation, more specifically wiki systems in Section 2.2. Furthermore, we introduce recommender systems in Section 2.3 and their application in the field of information systems and knowledge bases in Section 2.4. The last section is dedicated to database models that are used as an underlying basis for all presented SnoopyConcept storage approaches and concepts.

2.1 Information Representation & Structure

Knowledge is always structured but can be represented in different ways. We distinguish between three main classes of information systems depending on the knowledge representation and the strictness of structure in the system. In the following sections the three classes are described in detail.

2.1.1 Strictly Structured Information Systems

Strictly structured information systems are based on the relational model and have been used since the invent of the digital information processing. They provide a fixed data schema which is developed by an administrator or developer in advance. The end user has to adapt the information and knowledge to the predefined schema to store any data to the system. A common example is that of classical bank applications which provide the possibility to store bank transactions. Each transaction has to adhere to the predefined schema. Additional concepts such as normal forms or integrity constraints which are very common in the world of relational data further restrict the data. These restrictions and constraints are the key features of relational databases and strictly structured information systems as the quality of the stored data is very high and verified by additionally defined constraints which are guaranteed by the database system. Due these properties the relational model is best suited for applications which are used in a fixed domain. As most modern business applications are limited to one single domain, the relational model can be used to store all data in very sufficient way. Modern relational databases are able to handle hundreds of thousands transactions per second [93] where each entry is strictly structured and adhere to a predefined schema. If an application needs more flexibility regarding the schema, as not all data points adhere to the same structure, the relational model run up against its limit and other models are needed to process heterogeneous structured data in an efficient way. Those models are discussed in the two following sections.

2.1.2 Unstructured Information Systems—Wikis

With the advent of the Web 2.0 which has transformed the content creation from the author to the website users, new storage models have been introduced. The key feature of these models are the handling of loose structured or non structured information and knowledge. One of the most successful concept are the concept of Wikis which allow to store any type or structure of knowledge. Most wikis realize this feature by just offering a simple text field to insert any textual information to a page. Additional features such as interlinking or infoboxes provide the possibility to the user to enter further structural information to the simple text. The most known example is Wikipedia which have become the world largest knowledge base [110] by using the flexible paradigm. An overview about Wikis, their features, used technologies and their drawbacks can be found in Section 2.2. A combination of the strictly structured approach and the unstructured wiki approach is discussed in the following section.

2.1.3 Semi-structured Data

The *semi-structured data model* incorporates both the paradigm of structured and unstructured storage. The need for such a new storage paradigm already arose in the 90s [147, 32]. Back then it became clear that it would be increasingly important to be able to store mostly unstructured data while at the same time providing efficient and structured querying facilities. With the advent of the World Wide Web, which currently forms the largest unstructured knowledge base, it became obvious that such data cannot be fitted into a predefined schema in order to be able to query it. The application of traditional retrieval and extraction techniques to query such unstructured data reached unsatisfactory results as the formulation of structured and precise queries (e.g. "Which Austrian cities have more than 10.000 inhabitants and have a female mayor who has a doctoral degree?") was not possible due to the lack of structure. Thus, the semi-structured data model combines both the structured and the unstructured data model and provides a highly flexible way of storing data as it supports the storage of information in a structured way without the need of specifying a predefined schema.

This approach has also been incorporated by several wiki systems. For example, the Semantic Mediawiki [100] allows to specify semantic information in a semi-structured way (*cf.* Section 2.2). Also Wikipedia has introduced a new project in 2012 which is called Wikidata (*cf.* Section 2.2.3) and offers the insertion of semantic data which are computer-processable in contrast to the fulltext-format of classical articles on Wikipedia.

Throughout the last two decades, various models for semi-structured data have been developed, like e.g. [4, 38]. Currently, the most popular example of the semi-structured data model is RDF (Resource Description Framework, W3C recommendation[1]) [95]. RDF basically models knowledge as triples consisting of a subject, a predicate and an object. The subject (also called resource) is described by multiple pairs of predicates and according objects. The resource is uniquely identified by a URI (Uniform Resource Identifier[2]). Important facts about the University of Innsbruck within a knowledge base can be stored using triples as e.g. in Listing 2.1.

Listing 2.1: Triples

```
1 <http://dbpedia.org/../University_Innsbruck><numberOfStudents><26626>
2 <http://dbpedia.org/../University_Innsbruck><established><1669>
```

[1] http://www.w3.org/RDF/, accessed 2017-07-17

[2] http://www.w3.org/TR/uri-clarification/, accessed 2017-07-17

These predicate-object pairs—in combination with the article URI itself (the resource, in this case University_Innsbruck)—constitute triples. The subjects, predicates and objects are not restricted in any way and can therefore hold any information while at the same time providing structure due to the triple concept as all objects are given context by the according predicates. The triples are machine-readable and processable, thus provide the base for structured access and complex structured queries. In order to query such semi-structured RDF knowledge bases, the standard query language is SPARQL (SPARQL Protocol and RDF Query Language) [135].

As RDF is very flexible and is able to link to even external resources by specifying an external URI as the object of a triple, the interlinking between knowledge bases has become very popular. Sir Tim Berners-Lee has coined the term Linked Open Data (LOD) [23] for such linked and semi-structured data (*cf.* Section 2.2.3). Further information about knowledge bases can also be found in Section 2.2.3.

It is important to note that within semi-structured systems, users can arbitrarily choose the predicate used for storing information. This fact is very beneficial as it provides a huge amount of flexibility to the users of the system while at the same time—due to the predicate-object format—still features a certain amount of structure. This fact is crucial in online, mass-collaboration information systems, as there are thousands of different users who come from different social levels, backgrounds and edit information of different domains and contexts. Along with this essential feature of flexibility several challenges, such as preserving a homogeneous structure, arise.

To tackle those challenges related to the maintenance of a heterogeneous schema, we propose in the SnoopyConcept to leverage recommender systems to guide the user already during the insertion process to a homogeneous structure. More details about the approach can be found in Section 3.1.

2.2 Wiki Systems and Knowledge Curation

Wiki systems are widely used to facilitate mass-collaborative knowledge curation in public projects such as Wikipedia [110] but also in closed corporate environments [111]. Therefore, many research projects have been dealing with wikis, the behaviour of their users, and extensions and improvements in the ecosystem of Wikipedia. In this section we discuss wiki approaches to improve the knowledge curation but also drawbacks and limitations of wiki systems which can me mapped to the SnoopyConcept as they describe universal problems in the area of knowledge curation.

2.2.1 Wikipedia and its Structural Limitations

In context of the SnoopyConcept especially the topics of semi-structured and machine readable data in Wikipedia and its quality and quantity are most relevant. The core of Wikipedia to create and maintain simple plain text documents does not support any structured or machine readable information at all. The information retrieval in Wikipedia by users is limited to a full-text search and the usage of links. Especially the interlinking of articles is very important to structure and categorize content and knowledge and enhance search capabilities. Already in 2005, Krötzsch *et al.* [101] pointed out the limitations of the link and category system in Wikipedia[3] which was realized on top of classical page oriented system of Mediawiki. Many link pages which just contain a list of articles, for example a list of all cities in Austria were created and have to be maintained and updated manually. To create a more restricted view e.g., of all capital cities in Austria a new page has to be created and maintained. This example is relatively static as cities do not change that often. When considering a more dynamic example, such as a list of all Open Source Wiki Software, the changing frequency increases dramatically. To keep all link pages and categories up to date by using a manual approach is very error prone and time intensive.

Due to these huge amount of manual tasks since the beginning of Wikipedia bots[4] have been running to fulfill easy tasks automatically and support the committed community of Wikipedia. Currently there about 2,000 bots [5] which take care of Wikipedia's content. Although they are limited to simple tasks (e.g. inform authors about syntax errors) as semantic reasoning is a difficult task especially when dealing with Wikipedia's plain text content which provides almost no structure at all.

The only available structure in Wikipedia is the linking system which has been heavily exploited by many research projects. Very popular examples are dictionaries and thesauri that are based on the interlinking system and disambiguation pages of Wikipedia [159, 117]. In Section 2.2.3 we present more approaches which heavily rely on the knowledge stored in Wikipedia.

[3] https://en.wikipedia.org/w/index.php?title=Wikipedia:Categories,_lists,_and_navigation_templates&oldid=710181660, accessed 2017-07-17

[4] https://en.wikipedia.org/w/index.php?title=Wikipedia:Bots&oldid=705099747, accessed 2017-07-17

[5] https://en.wikipedia.org/w/index.php?title=Wikipedia:Bots&oldid=705099747, accessed 2017-07-17

In the next section shortcomings of the first wiki systems respectively Wikipedia are discussed and improvements in the Wiki ecosystem in research but also corporate environment are shown.

2.2.2 Wiki Enhancements

Since the introduction of the first wiki systems that have only provided basic edit functionality, many improvements and additional features have been introduced to the world of collaborative editing in Wikis. Especially off the beaten track of Wikipedia which has been showed a very inert ability to innovate [154], many new innovations have been introduced and developed in the enormous ecosystem of wiki systems. Two major shortcomings of classical wiki systems that are relevant in the context of this thesis, the complicated insertion process to insert information and the way of finding information in wiki systems.

Enhancing the Editing Process

Most wiki systems provide a wiki specific markup language to create and edit content and knowledge. The edit process itself is done in a simple plain text field which contains the markup language code. Especially at the beginning wikis were used by more technical or experienced users who were able to understand and use a markup language. From a technical perspective, markup editors are easy to implement but constitute a barrier for less experienced users. In general, the contribution to a wiki systems prerequisites knowledge in many areas. The user has to be aware of all rules and the overall process of creating new knowledge. Furthermore, the user has to be able to cope with all technical barriers such as the markup language or other structural enrichment like infoboxes or linking articles. Halfaker *et al.* showed in 2013 in a study [67] that the severe drop out rate of Wikipedia contributors of 30% between 2006 and 2011 could led back to the barriers that have to overcome to contribute to Wikipedia. Besides rules, regulation and the missing knowledge about the contribution process the Wikimedia foundation also mentioned the lack of a simple visual editor [144, 145]. To improve the user experience many wiki systems have introduced WYSIWYG (what you see is what you get) editors which transparently hide the underlying markup language. The Wikimedia foundation growth team[6] that was initiated to increase the number of editors again introduced a peer group system to support editors[7] and a new visual

[6] https://www.mediawiki.org/wiki/Growth, accessed 2017-07-17

[7] https://meta.wikimedia.org/wiki/Research:Teahouse, accessed 2017-07-17

editor[8] which was initially launched 2012 for testing and has been used as the default editor in Wikipedia since 2015. Pfisterer *et al.* conducted a user study in [129] to evaluate how the semantic features of the Semantic MediaWiki can be promoted to unfamiliar users and in general, how the user interaction process can be optimized. Based on the results they authors proposed and also implemented improvements of the user interface, such as the direct editing of annotations without editing the source code which lowers the barrier to add annotations dramatically.

Due to those findings, the SnoopyDB reference implementation of the SnoopyConcept presented in Section 5.1 puts a strong emphasis on the usability of the editor and its recommendations. For example, the SnoopyDB prototype aims at identifying the type of the inserted value and suggests units or semantic enhancements. This leads to an increase of the quality of inserted knowledge as described in Section 3.3.3) in more detail. Furthermore, the SnoopyConcept recommendation points to missing information in the system and thus, encourage the user to insert more information to the system.

Enhancing the Navigation and Search

As classical wikis are based on the hypermedia principle which features navigation by links, categories or tree structures are not provided by default. Especially wikis in the enterprise environment such as Confluence[9] have added the support to navigate using a tree structure. Trees are well known as they are used on websites, in content management systems or file browsers and therefore, are naturally understood by many users. Furthermore, content can be classified by using the tree structure and attached to departments or other internal structures of an enterprise.

Zubiaga *et al.* [191] identified three main navigation patterns in Wikipedia. Besides the simple keyword based approach there are two major patterns. The category-driven navigation describes the browsing by using the Wikipedia's taxonomy which is limited as Wikipedia's category system is manually maintained and not fully complete. A more successful approach is to use the link system within Wikipedia. The high number of links is one of the main characteristic of Wikipedia and therefore, well suited for a link-driven navigation. Nevertheless if a link is missing the user cannot be sure, if the information is not linked or not present in the system. Therefore, Zubiaga *et al.* proposed a social tagging system to improve the search capabilities. Tags can be used for searching but also filtering, e.g., gathering documents containing

[8] https://en.wikipedia.org/wiki/VisualEditor, accessed 2017-07-17

[9] https://www.atlassian.com/software/confluence, accessed 2017-07-17

a tag but excluding another one. Sweetwiki [31] aimed for an improvement of the user experience by providing easy to use editors and add semantic features to improve the search experience. Besides social tagging mechanisms, and a SPARQL interface the wiki consists of an ontology builder to maintain the taxonomy that can be used to further structure the content in Sweetwiki. WikSAR [15] represented a wiki as an interactive graph that can be used to understand complex structures. Furthermore the query language could be used to generate collections which are updated automatically based on the query. As an underlying technology the semantic standard RDF and SPARQL was used.

The presented optimizations and approaches show that the gap between plain text knowledge and structured knowledge is crucial as it directly influences the navigation and search capability. The SnoopyConcept addresses this issue by proposing semantic enhancements, e.g., specifying semantic links to other subjects. Those links can be used to navigate in the knowledge base but also increase the search capabilities. The following sections describes more approaches that also try to bridge this gap and built computer processable knowledge.

2.2.3 Semantics, Wikidata & Knowledge Bases

One of the most successful meta projects which are based on the knowledge of Wikipedia is DBpedia [14]. DBpedia parses all textual information of Wikipedia articles, extracts semantic information and provides the knowledge of Wikipedia in a computer processable format. One of the most import sources in Wikipedia pages are infoboxes which are tabular aggregations of the most important facts of a Wikipedia article. Figure 2.1 shows the article "Innsbruck" and its Infobox.

In contrast to fulltext, these facts have a clear semantic as they are inserted as key-value pairs. DBpedia extracts these pairs from the complex markup language, applies cleaning filters and finally converts the extracted knowledge to a computer processable format such as RDF. Furthermore, DBpedia incorporates all links, categories and other structures which are used to classify and structure Wikipedia articles. The DBpedia release 2016-04[10] consists of 9.5 billion triples which were extracted from Wikipedia and interlinked with many other datasets in the LOD cloud. DBpedia is one of the most important sources in the LOD cloud and is often shown as the central repository which

[10] http://wiki.dbpedia.org/dbpedia-version-2016-04, accessed 2017-07-17

Innsbruck (German: ['ɪnsbʁʊk], local pronunciation: ['ɪnʃprʊk]) is the capital city of Tyrol in western Austria. It is located in the Inn valley, at its junction with the Wipp valley, which provides access to the Brenner Pass some 30 km (18.6 mi) to the south.

Located in the broad valley between high mountains, the so-called North Chain in the Karwendel Alps (Hafelekarspitze, 2,334 metres or 7,657 feet) to the north, and the Patscherkofel (2,246 m or 7,369 ft) and Serles (2,718 m or 8,917 ft) to the south.

Innsbruck is an internationally renowned winter sports centre, and hosted the 1964 and 1976 Winter Olympics as well as the 1984 and 1988 Winter Paralympics. Innsbruck also hosted the first Winter Youth Olympics in 2012. The name translates as "Inn bridge".[3]

Contents [hide]

1 History
2 Boroughs and statistical divisions
3 Population
4 Climate
5 Main sights
  5.1 Buildings and monuments
  5.2 Museums
  5.3 Churches

Innsbruck

Coordinates: 47°16′N 11°23′E

Country	Austria
State	Tyrol
District	Statutory city
Government	
• **Mayor**	Christine Oppitz-Plörer
Area	
• **Total**	104.91 km² (40.51 sq mi)
Elevation	574 m (1,883 ft)
Population (1 January 2016)[1]	
• **Total**	130,894
• **Density**	1,200/km² (3,200/sq mi)
Time zone	CET (UTC+1)
• **Summer (DST)**	CEST (UTC+2)
Postal codes	6010–6080
Area code	0512
Vehicle registration	I
Website	innsbruck.at

Figure 2.1: Infobox in Wikipedia article about Innsbruck

used as a linkage base for all other datasets[11]. The DBpedia dataset was also used in our offline evaluation presented in Section 6.1.1.

Also Google introduced a new feature which relies on the knowledge of Wikipedia and is called Knowledge Graph [83]. This graph is exploited to better understand the search terms and show a summary (*cf.* Figure 2.2) of important facts regarding the inserted search terms. Considering the search term "Albert Einstein", Google shows a Box consisting of Photos of Albert Einstein, important facts such as birthday or relations to other persons e.g. Stephen Hawking. Besides Wikipedia Google's Knowledge Graph is also based on Freebase[12] which was acquired by Google in 2010. Freebase [24] stores about 1.9 billion facts (June 2017) about well-known people, places, music, books and in general things. It is publicly available and can be accessed by multiple APIs and interfaces. In contrast to DBpedia which is based on extracted Wikipedia knowledge, the content of Freebase already consists of structured fact tuples which can be edited directly. This direct usage of such a structure, reduces noise and errors which can be introduced by intermediary processing steps such as extracting and unifying knowledge.

Many people, especially around Krötzsch and Völkel [101][167], have criticized the lack of semantic features in Wikipedia since 2005 which constitutes a big shortcoming when dealing with the processing of information in Wikipedia by automated processes. Due to these shortcomings the project Semantic Mediawiki was developed [100] which provides the possibilities to add semantic

[11] Linking Open Data cloud diagram, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net, accessed 2017-07-17

[12] http://www.freebase.com, accessed 2017-07-17

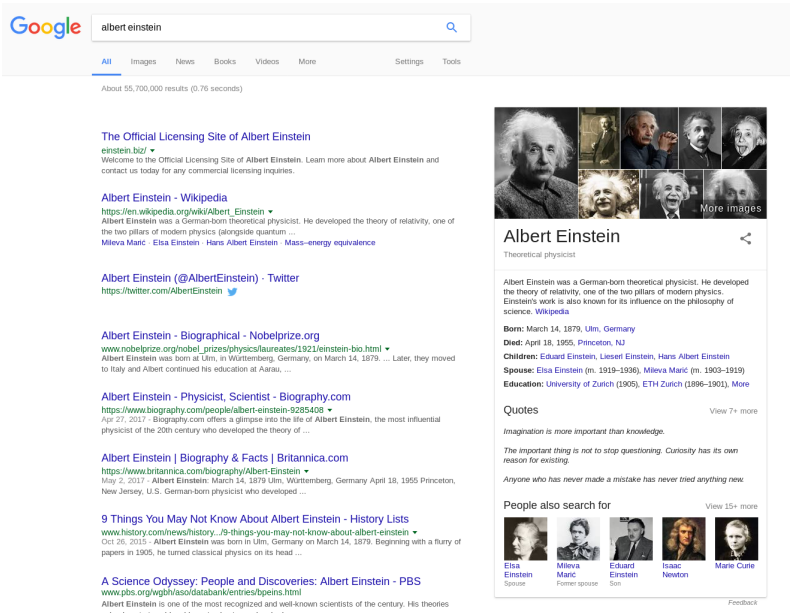


Figure 2.2: Knowledge graph summary for Albert Einstein at Google

information to the textual knowledge by using a simple markup language. The semantic information is created by annotating wiki pages. The Semantic Mediawiki introduced three different types of annotations. *Categories* to classify wiki pages, *Relations* to describe relationships between wiki pages and *Attributes* to specify information of the wiki page in a structured and semantic manner. Categories are based on the already available category system of Mediawiki and is just lifted to a semantic representation and interpretation. Relations can be seen as simple links between articles which are also already present in Mediawiki. The Semantic Mediawiki extended the power of links by specifying the type of a link. This provides the possibility to add additional semantic information to a link. Consider the example of Albert Einstein who is linked with Alfred Kleiner. One cannot see how these two people are related by just considering the simple link. If a typed relation was added in the Semantic Wiki page of Albert Einstein, such as `[[doctoral advisor::Alfred Kleiner]]`, one can simply identify the type of the relationship between Albert Einstein and hist doctoral advisor Alfred Kleiner. Attributes, the last type of annotations in the Semantic Mediawiki, allow to specify relationships to things which are not present as a wiki page. For example the birthday of Albert Einstein can be specified by adding the text snippet `[[birthday:=1879-03-14]]` to the wiki page of Albert Einstein. Furthermore attributes can be typed by specifying an additional relationship between the attribute and a type. For example the Attribute::birthday and the type:Date can be interlinked to specify the type of birthday. All this information can be used for further automatic

processing and reasoning. The Semantic Wikimedia system provides such a processing and offers querying facilities to answer queries or even create dynamic pages or parts of pages. For example on the wiki page of Alfred Kleiner all advised phd students can be listed automatically. The big advantage is that such lists do not need to be held up to date manually as they are computed dynamically. Also complex queries such as "List all scientists who were born in 1879" are possible by querying the category and birthday attribute of wiki pages. As this approach does not limit the user in the choice of attributes, the problem of proliferation of attributes (*cf.* Section 2.4.1) arises and directly impede the search capabilities. To solve this problem, the SnoopyConcept computes recommendations of suitable attributes to prevent synonyms in the vocabulary of attributes and thus, increase the search capabilities.

The community of the Semantic Mediawiki ecosystem has requested since the beginning of the semantic extension of Mediawiki to integrate the semantic annotation feature to the Wikipedia system. But it took nearly 10 years until the Wikipedia community appreciated the worth of a semantic knowledge base and in April 2012 a new project in the Wikipedia ecosystem was initiated — Wikidata[169, 170]. Wikidata aims at providing computer processable knowledge which is maintained by the Wikipedia community. It was the first new project since 2006 by the Wikipedia Foundation and forced by the German chapter of Wikimedia. The project was funded by Allen Institute for Artificial Intelligence, the Gordon and Betty Moore Foundation, and Google, Inc. and has been lead by Denny Vrandečić who was one of the founders of the Semantic Mediawiki [13]. The Wikidata project is based on the Mediawiki software to provide highest interoperability and offers the possibility to store key-value pairs in the fashion of a document-oriented database. The main goal of the database is to store general facts in a central place. Considering the fact about the birthday of a famous person which is stored in multiple Wikipedia articles in different languages. As each language is operated by its own Wikimedia association and own infrastructure the birthday is stored and maintained in different databases. Especially often adapted values, such as the population of a country, have to maintained in different languages and Wikipedia versions. This storage concept is very error prone and lead to multiple inconsistencies. Thus, Wikidata provides the possibility to store these facts in a centralized way which can be accessed and referenced by all Wikimedia projects. Additionally to the fact itself, a reference url is stored to specify the source of the information and increase the traceability. Besides the facts which are located in infoboxes also other structural or semantic knowledge is stored in Wikipedia. The whole category system or interlanguage links between articles about the same item in different languages were defined by using the markup language

[13] http://en.wikipedia.org/w/index.php?title=Wikidata&oldid=571240689, accessed 2017-07-17

inside articles. Such links, category membership information and lists have to be maintained in each language by their local community. All this information can be stored in Wikidata and reused for example to automatically generate category member lists. Currently many people and projects including their bots are striving to move already present data to Wikidata and also fill the new knowledge base [170]. Furthermore, Wikidata is not only important for all Wikipedias but also paving avenues for further external projects and researches who are now able to access millions of community proven facts. We used the Wikidata dataset to evaluate the Wikidata's property suggestor and an extension we proposed in Section 5.4.

Besides Wikidata, DBpedia and Freebase several other knowledge bases which represent common world knowledge exists. Many of them are based on knowledge harvesting (*cf.* Section 2.4.2) methods which try to construct knowledge bases automatically by crawling and analyzing websites. KnowItAll/ReVerb [55, 53], NELL [36] and Yago [76, 161] aim at extracting facts out of written sentences. The systems use predefined knowledge for seeding and identify patterns which subsequently used to find new knowledge. Most approaches present the automatic extracted facts to users who can decide if the fact is true. This manual feedback can be used to refine the extraction algorithms, judge found patterns and improve the precision of the certainty level of found facts. Besides the scientific approaches also commercial databases such as Microsoft's Concept Graph[14] or Watson's database [113] are built.

Most of the presented datasets are curated by a committed community which tries to sustain a homogeneous schema in the knowledge base. The SnoopyConcept is well suited to build such a knowledge base while relieving the community by pushing the task of homogenization to the user. This is achieved by guiding the user to a homogeneous structure already during the insertion process as explained in detail in Section 3.3. Furthermore, the SnoopyConcept recommends semantic enrichment to the user which increases the amount of links in the dataset. Especially, when incorporating the LOD cloud this interlinking is beneficial for the search capabilities.

2.3 Recommender Systems

The core of the SnoopyConcept is a recommender system which aims at guiding the user to a homogeneous schema and increasing the quality and quantity of stored knowledge in an information system. In this section we introduce the

[14] https://concept.research.microsoft.com/, accessed 2017-07-17

foundations of recommender systems and describe different recommendation approaches.

2.3.1 Introduction to Recommender Systems

As human beings we are often forced to make choices without sufficient experience in the respective area. To cope with this lack of experience we often rely on recommendations from other people by word of mouth, recommendation letters or general reviews in newspapers and guides [138].

Recommender Systems assists the user and augment the natural social interaction with the system by recommending suitable items to the user [138]. The first recommender system was implemented in 1992 [66] and introduced "collaborative filtering" which has become the de facto standard in the area of recommender systems. Since then the topic of recommender systems has grown heavily in research and industry and is nowadays present in many different areas.

These days recommender systems can be found in most of all online platforms. One of the first well known examples is the shopping platform Amazon[15] which suggests their users further suitable articles based on user profiles [107, 155].

Beside suggesting books to users within the last decades recommender systems have become an important piece of technology in many areas. Especially in areas that heavily rely on user preferences it is obvious to use recommender systems. Therefore, the recommendation of suitable music and movies to users has become a very important field in research and industry. The most prominent representative is MovieLens [71] which is a dataset consisting of movie ratings that was released in 1998 and since then has been served the basis for many research approaches. Boosted by the rise of streaming services for music and movies this field of research has been growing rapidly within the last few years [148]. This is also reflected in the amount of workshops and tracks related to music and movie recommender systems in the top conferences such as RecSys[16] or ISMIR [17]. In particular in the domain of music and movies not only user profiles are used to build recommendation algorithms but also content based features are considered. For example the style or genre of a music track or the metadata of a movie can be incorporated to improve the recommender system. Such content based recommender systems are often combined

[15] http://www.amazon.com, accessed 2017-07-17

[16] http://recsys.acm.org, accessed 2017-07-17

[17] http://www.ismir.net, accessed 2017-07-17

with the classical user based approach to improve the overall performance of the recommender. An additional dimension which has become increasingly important within the last few years is the context of recommendations. This contextual information, such as location and time can have a significant influence on recommended items. Systems that incorporate the context for the computation of recommendations are so called context aware recommender systems and are widely used [92]. Especially one field that was originally not classified as a recommender system heavily rely on the context adaption, namely search engines. They have evolved from simple search tools to powerful, context aware recommender systems that exploit user profiles and several other sources to deliver personalized search results [156]. The mentioned areas only scratch the surface of all use cases for recommender systems but already demonstrated that recommender systems are widely used in many different areas and domains. The different types of recommender systems are described in the following section.

2.3.2 Classification

Recommender Systems were classified by several authors [86, 6, 139, 119, 40] and differ on a detailed level. In this work we classify recommender systems in a similar way into the following types.

- Collaborative filtering
- Content-based recommender systems
- Knowledge-based recommender systems
- Context-aware recommender systems
- Hybrid recommender systems

In the following sections the approaches and their differences are described.

2.3.3 Collaborative Filtering

Collaborative filtering is the de facto standard in the area of recommender systems and was also used by Goldberg in 1992 [66] to implement the first recommendation system. Goldberg *et al.* introduced the term Collaborative Filtering which describes the usage of collected opinions or behaviors of ex-

isting users. Based on this gathered data the recommendation systems tries to predict which items the current/new user likes or is interested in. As this approach was the first recommender approach it has been extensively explored by the research community and nowadays, is also widely used in many areas and products.

The basic algorithm is based on a matrix with two dimensions, namely users and items. Every cell contains the information if the respective user likes or dislikes the respective item. The algorithm takes this matrix as an input and provides a ranked list of items that are interesting for a specified user. The provided list contains only items that the user has not marked as (not) interesting so far.

The recommendation task is formally defined by Adomavicius *et al.* [6] as follows:

$$\forall c \in \mathcal{C}, \quad s'_c = \max_{s \in \mathcal{S}} u(c, s) \tag{2.1}$$

Let u in Equation 2.1 be a utility function that measures the usefulness of an item s to the user c with $u : \mathcal{C} \times \mathcal{S} \rightarrow \mathcal{R}$ where $\mathcal{R}$ is a totally ordered set. For each user $c \in \mathcal{C}$ we want to find a new item $s' \in \mathcal{S}$ that maximizes the user's utility function. As the function u is not defined for every combination of s and c the aim of an recommender system is to predict the result of the function u on items that have not been rated so far. For example a movie recommendation system has to predict if a non-rated movie is "useful" for a specific user c.

The most common approach is a user based algorithm which relies on user profiles and their similarities. A user profile of user c_j is defined by all computed values of the utility function $u(c_j, s)$ of all rated items. The prediction of $u(c_j, s)$ for a non-rated item s is realized by considering all other user profiles that are similar to the user profile of user c_j. Depending on the ratings for item s by all similar users the result of $u(c_j, s)$ can be estimated. Considering the example of a user c in a movie recommender system, the recommender system tries to find other users who have a similar taste. The taste is defined by likes and dislikes of movies. If users like or dislike the same movies it is obvious that they have a similar taste. The recommender system can subsequently recommend movies that haven't been watched by the user c but already liked by all other similar users. The similarity between two users can be calculated by e.g. applying the cosine similarity or any other set based similarity measure on both user vectors that contain the user ratings of all items.

Since the beginning in 1992 the basic algorithm has been improved and extended by several approaches [87]. For example nowadays, the "usefulness" of an item is no longer defined as a boolean value ("like" or "dislike") but as a rating (e.g. between 1 and 10) and therefore, allows a more fine-grained computation of recommendations. Also the user profile can be extended [6] by e.g. adding demographic information or other meta-data about the user. These additional information can be used to improve the algorithm to find similar users as not only the ratings are considered. Another improvement to get more insights about the taste of users is to exploit external sources. For example publicly available data such as tweets can be harvested to build user-item information and find relations between items. For example, we built a music recommender which implements a collaborative filtering approach based on tweets about listened music tracks [185]. Especially to overcome the cold start problem or to cope with very small user bases the consultation of external sources, such as Twitter, is very beneficial.

Regarding the implementation of the collaborative filtering approach we can distinguish between two major approaches [6, 28], namely memory-based and model-based approaches.

Memory-Based Collaborative Filtering

Memory-based algorithms are straight forward as they consider the full matrix of all previous ratings which are created by all users. The name derives from the fact, that the matrix has to be loaded into memory to compute a recommendation. Most approaches apply nearest neighbor algorithms to find similar users. Widely used algorithms are Pearson's correlation coefficient, adjusted cosine similarity, Spearman's rank correlation coefficient or the mean squared difference measure [87]. According to Herlocker *et al.* [74], the Pearson's correlation coefficient outperforms other algorithms when considering user-based collaborative filtering.

The basic collaborative filtering approaches consider likes for all items with the same weight. In real world examples it can be seen that there are many items that are voted by most of the users. Thus, votes or ratings on such items are not very helpful to find good recommendations. Therefore, the idea is to put more weight on more controversial items that are not voted very often. Breese *et al.* [29] introduced the "inverse user frequency" and Herlocker *et al.* [74] addresses the problem by a so called "variance weighting factor" to improve the recommender accuracy.

Model-Based Collaborative Filtering

In contrast to the memory-based approach which computes recommendation by considering all available data, the model based approach uses a precomputed model for the recommendation computation [87]. This model is computed in advance in an offline step and takes all available raw data into account. The "learned" model is able to approximate results but does not need all the raw data and therefore, is more efficient in terms of memory usage than the memory-based approach. Especially when considering very large user bases with tens of millions of users a memory-based approach might be no longer feasible.

For the model creation task "dimension reduction" algorithms and "matrix factorization" methods are widely used [87, 85, 99]. These techniques derive a set of latent factors in the user-item matrix to be able to collapse the matrix into a smaller approximation. One of the first approaches was the "singular value decomposition" (SVD) approach which was introduced by Deerwester *et al.* in 1990 [49] in the area of semantic analysis. The algorithm aims at finding co-occuring terms which are highly correlated and can be reduced to one single value.

As model-based approaches are based on approximations one could obviously state that model-based approaches have a lower precision as not all data is considered which is true for many cases. Although experiments showed [87, 146] that in several use cases model-based approaches outperforms memory-based approaches which can be led back to the removal of noise which negatively impacts the accuracy.

Furthermore, within the last decades many improvements in the area of model-based collaborative filtering have been introduced [139, 98, 126, 137] and there is still a very large research community which works on further improvement in the area of model-based collaborative filtering. One striking example is the winner team [165] of the Netflix prize competition [18, 17] that created a highly tailored recommender system that outperforms the Netflix recommendation algorithm by over 10%.

Besides the optimization of the user based memory or model based approaches many algorithms also consider meta-data about items. In literature and research item based approaches are sometimes not classified as collaborative filtering approaches but build their own class of content-based recommender systems which are described in the following section.

2.3.4 Content-based Recommender Systems

Content-based recommender approaches [88] approaches take information about the item into account to compute recommendations. This is achieved by building a item profile and a user profile which is then compared to recommend suitable items. For example in the area of movie recommendation systems a genre based classification can be used. For the computation process in this simplified example we consider user profiles which consist of genre preferences to find movies that adhere to the user's preferences.

The meta-data about items can be classified into technical features and subjective or qualitative features. Technical features such as the genre or the list of actors of a movie can be easily retrieved and exploited for the computation of recommendation. In contrast, qualitative features such as quality or taste are difficult to retrieve and more difficult to interpret. Especially the subjective impression and the resulting rating about a movie is very difficult to track and understand as it is often not based on hard, technical facts that are accessible by the recommender system.

The core of a content-based recommender system is a similarity function that compares items based on defined features. Based on the calculated similarity the recommender system suggests items that are similar to the user's previously liked items. For example simple text or keywords can be used to describe items. As this problem of describing and classifying objects is a very well known problem in the domain of information retrieval many algorithms can be adapted to compute recommendations. The most widely used technique in the area of information retrieval is the vector space model which describes an items by a vector of features. In this example keywords could be used to create the vectors which consist of multiple boolean values. Every boolean entry in the vector defines if the respective keyword is present. To improve the search process which aims at finding similar vectors, furthermore, the TF-IDF weighting scheme can be applied. The weighting scheme was introduced by Salton *et al.* in 1975 [143] and is based on the frequency of occurrences of keywords. It considers how often the keyword in present in the description and in the whole dataset. The TF-IDF weighting scheme has become a de facto standard which nowadays, is used in several variations in many text based information retrieval systems. Another very popular approach is to use k-nearest neighbours (kNN) methods [19] to find similar items in the vector space model.

In the context of recommendation systems features of the vector space model can be manifold. For example the previously defined user-item matrix which is used for user-based recommendations can be reused and exploited in a content-

based recommendation approach as well. To achieve this, the transposed matrix is used and every item is represented by its rating-vector. The similarity of two items is subsequently defined as the similarity between all user ratings of these two items. However, the more common approach is to use features that can be directly extracted from the items without analyzing the user interaction. This comes along with the big advantage to be able to overcome the cold start problem as we can already recommend items right after the user liked the first item. For example when considering a music recommender system typical acoustic features and sound textures about genre, mood, timbre and tempo are automatically extracted [166, 152]. Besides the automated extraction of information about items external sources can also be exploited to feed a recommender system [51]. For example information about the genre, actors, budget, director or revenue of movies can be retrieved from an external catalog and build the basis for a movie recommender system.

In general recommender systems that solely rely on item based features are not very common as they do not use the very powerful source of user opinions. Therefore, it is obvious to combine both, the user and item based approach and benefit from both sides. These so-called hybrid recommender systems are explained in more detail in Section 2.3.7.

2.3.5 Knowledge-based Recommender Systems

Knowledge-based recommender systems [89] come into play when user-based or content-based approaches reach their limits. Consider a shopping platform for cars, real estate, horses or specialized technical equipment which are not bought very often. For such use cases, recommender systems have usually only few information about the correlation of items or the users as the systems have to deal with just one order per user in the worst case. Even a recurrent user could have changed her preferences regarding buying a new house or car due to a changed family situation. The same holds for technical equipment which outdates very quickly due to the fast pace of development. For example information about the order of a PC system five years ago is not very suitable to compute recommendation that are up to date. Also in very specialized domains such as medicine information about e.g. a medical device can be outdated very quickly.

To tackle this problem there are two common approaches used in a knowledge-based recommendation system. The first method exploits explicit recommendation rules (knowledge) which are manually created by a domain expert. The other possibility is to interact with the user to create a user profile on the fly which can subsequently used to find suitable items. Such a system could also

be considered as a search engine but still confirms with the definition of a recommender system by Burke [33]: "guide a user in a personalized way to interesting or useful objects in a large space of possible options or that produce such objects as output".

Knowledge-based recommender systems are able to cope with some shortcomings of traditional user-based or content-based approaches and furthermore, provide the big advantage that there is no cold start problem at all. However, the costs of a domain expert are usually very high and the system is very inflexible due to the fact that all rules and constraints are manually generated. Therefore, the approach comes along with high maintenance costs and thus, is not widely used.

2.3.6 Context-aware Recommender Systems

All previously described recommender approaches are dealing with only two types of entities, namely user and item. Context-aware recommender systems (CARS) [5] consider the additional dimension "context" of the user. For example time, location, mood or company of other people could completely change the preferences of a user and thus, have to be considered for the computation of recommendation. Every time the user interacts with the system, the context might have changed which directly influences the user preferences and subsequently the user profile. The context could be modeled as a continuously changing user profile. As such a modeling approach would interfere with most recommender definitions, Adomavicius *et al.* [5] model the context as an additional dimension as shown in Equation 2.2.

$$\begin{aligned} ClassicalRecommenderSystem &: User \times Item \rightarrow Rating \\ ContextAwareRS &: User \times Item \times Context \rightarrow Rating \end{aligned} \tag{2.2}$$

According to this definition the rating information is attached to a triple (user, item, context) in contrast to a classic recommender system which uses the pair (user,item) to identify a rating. The context can be defined by additional attributes in the same way as users or items can consist of further properties. As contextual information may contain a lot of different aspects respectively dimensions the contextual information could increase the complexity dramatically. Considering again the very simple example of a movie recommender system that includes contextual information about location (or distance to

theaters), time and company. These three dimensions can be modeled by the following hierarchical trees:

$GeoCoordinates \rightarrow City \rightarrow State \rightarrow Country$
$Date \rightarrow DayOfWeek \rightarrow TimeOfWeek$
$Date \rightarrow Month \rightarrow Quarter \rightarrow Year$
$Company : Girlfriend \rightarrow Friends \rightarrow NotAlone \rightarrow AnyCompany$

These already simplified dimensions build a huge multidimensional space that has to be considered in the computation of recommendations. Furthermore, the dimension *time* is very dynamic and involves hypes and a lot of exceptions such as holidays. The dimension *company* may flip preferences of a user completely. Due to these additional non-trivial dimensions context aware recommender systems are significantly more complex than traditional systems [5].

For the implementation Adomavicius *et al.* [5] distinguish between three different approaches, namely contextual pre-filtering, contextual post-filtering and contextual modeling. The two first approaches are built on top of classic recommender algorithms and apply additional filters. Based on the context the pre-filtering technique filters the dataset that is used by the recommender system. Post-filtering applies a filter on the final list of recommendations to adapt the recommendation to the respective context. Contextual modeling integrates the context consideration directly to the core function of the recommender and influences the selection, scoring, ranking or the model of the system.

Similar to the combination of tradition user-item based recommenders and contextual information also other approaches can be combined to improve the overall accuracy. These so called hybrid systems are described in the following section.

2.3.7 Hybrid Recommender Systems

The previously described approaches Collaborative Filtering, Content-based, Knowledge-based and Context-aware recommender systems are coming along with different advantages and disadvantages depending on the use case and domain. Therefore, it is obvious to combine different approaches to gain from advantages or be able to cope with disadvantages. According to [90] nowadays, most recommender systems are already hybrid solutions that combine different approaches to improve the overall accuracy. Jannach *et al.* [90] provide

an overview about hybrid recommender systems and discuss different aspects about their implementations.

2.3.8 Summary

In the presented sections about recommender systems we provided an overview about the main approaches to implement recommender systems and presented a basic classification system which can be used to classify different approaches of recommender systems. Collaborative filtering approaches analyses the user behaviour to compute recommendations, whereas content-based approaches take meta-information about items into account. Especially if such meta-information is not present or correlations between items are not known, the collaborative filtering approach is more powerful as it solely on usage data. Therefore, the availability of a sufficient amount of user data is a hard requirement for collaborative filtering approaches. To tackle the lack of meta-information or usage data knowledge based approaches incorporate specialist and their domain knowledge to build recommender system in a more manual fashion. Furthermore, extensions such as the consideration of contextual information to refine recommendations were discussed. The examples that were presented in the previous sections are mostly located in the domain of e-commerce and the personalization in this domain. As most recommender systems were developed to provide shopping suggestions it is obvious that many approaches are tailored for this domain. In the next section we discuss a very young field of research, namely the usage of recommender systems in information systems to guide users during their interaction with the system.

2.4 Recommender Systems in Information Systems

Recommender systems are very common in the domain of e-commerce, especially to realize personalization. Nowadays, recommender systems are also used in the domain of information systems to guide users during their interactions with the system. One indisputable property of mass-collaborative information systems is the homogeneity of structured knowledge as it is crucial in terms of accessibility and searchability. The challenge of maintaining a homogeneous schema which is also the main goal of the SnoopyConcept and possible solutions for the challenge are described in the following sections.

2.4.1 Synonyms and Search Capabilities

One of the most severe challenges in semi-structured information systems is the usage of synonyms—semantically equivalent but syntactically different terms. They dramatically decrease the search capabilities of the information system [54] as information of the same type is tagged with different labels. This problem occurs in information systems that do not limit the user to a predefined schema but provide the possibility to freely choose terms to store information. Consider for example two properties *inhabitants* and *population* which were introduced by two users in an information system. Both properties may be used to describe the same information—namely the number of citizens—and therefore, are semantically equivalent. On the other side, both properties are syntactically different and therefore, increase the heterogeneity of structure the information system. This heterogeneity decreases the search capabilities as the second property *inhabitants* respectively the subject that uses this property is not found if searching for property *population* and vice versa. This challenge was already described by Furnas *et al.* [59] who showed in the 80's that the chance of two humans choosing the same term for the same object is only about 20%. Especially in the area of mass-collaboration, this fact has a dramatic impact as thousands of different users from different social levels and backgrounds add terms and properties in different domains and settings to the information system.

In science, this phenomenon was studied deeply in the area of folksonomies. The term Folksonomies was introduced by Thomas Vander Wal [128] and is a portmanteau of folk and taxonomy and describes the resulting tag-cloud of a social tagging system. Tagging systems provide the user to freely annotate arbitrary resources by simple terms. The term social-tagging refers to large public systems such as BibSonomy[18] (tagging of publications), del.icio.us[19] (tagging of bookmarks) or Flickr[20] (tagging of photos) which use a collaborative tagging system to organize their community-created content. Folksonomies contain information about the usage of tags within a social tagging system in the form of triples. In particular, a triple in the form of `(tag,user,resource)` is used to store the information about the user who assigned a tag to a specific resource. The concept of social tagging was one of the main principles in the Web 2.0 area and has become a de facto standard nowadays. For researchers, such folksonomies have become a major research target as they are very large, consist of thousands of terms and are public available. Many research publications analyze the social usage and creation of folksonomies and try to find

[18] http://www.bibsonomy.org, accessed 2017-07-17

[19] http://del.icio.us, accessed 2017-07-17

[20] http://www.flickr.com, accessed 2017-07-17

trends and user behavior patterns in the area of folksonomies. The found patterns give insights into the behavior of communities or can be used as a basis for other research topics, such as recommender systems. Multiple research results [108, 109, 68, 40] show that folksonomies follow a long-tail distribution which means that few tags are used very often and a big amount of all unique tags are used on a very rare basis.

Characteristic	**Value**
Messages containing one or more hashtags	49,696,615
Hashtags usages total	65,612,803
Average number of hashtags per message	0.16
Average number of hashtags per message (within set of tweets containing at least one hashtag)	1.32
Maximum number of hashtags per message	47
Median of hashtags per message	1
Hashtags distinct	7,777,194
Hashtags occurring ≥ 5 times in total	757,832
Hashtags occurring < 5 times in total	7,135,627
Hashtags occurring < 3 times in total	6,841,523
Hashtags occurring once	5,765,835
Average number of usages per hashtag	8.43
Median number of usages per hashtag	1

Table 2.1: Hashtag distribution of crawled Twitter dataset

This long-tail distribution can be found in many collaborative created datasets. For example, it was also clearly evident in our crawled Twitter data-set which was published in [188] as 6 million of 8 million distinct hashtags appear only once (*cf.* Figure 2.3).

Only 700,000 hashtags occur more than 4 times as shown in Table 2.1. This distribution is strongly connected to the usage of synonymous terms as every user has a different linguistic competence and uses a different thought pattern. Furthermore, especially in this example of Twitter, which limits messages on their platform to 140 characters, abbreviations and other short forms increase the probability of additional synonyms. For example our dataset, which was crawled in summer of 2012, consists of many tweets regarding the Tour de France (a world-famous bicycle race in France) which were annotated by #tdf, #tdf12, #tdf2012 #tourdefrance, etc. All hashtags have the same meaning but feature a different syntactically structure and therefore, limit the search capabilities if only one hashtag is used to search for tweets.

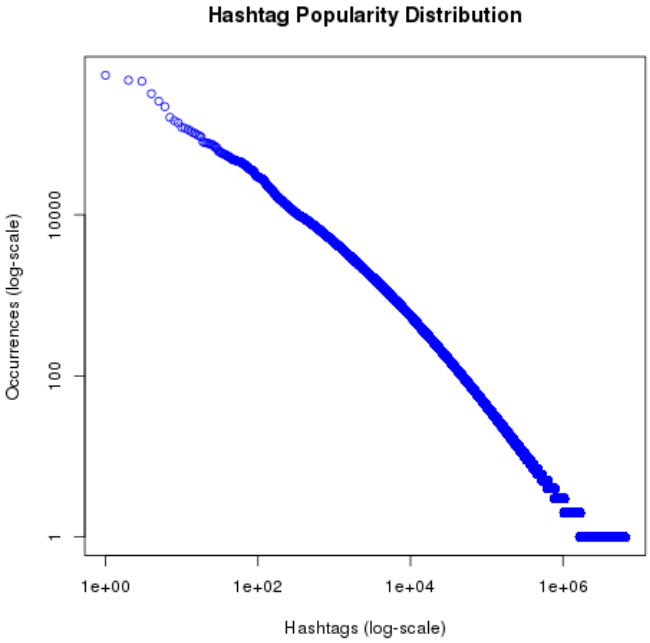


Figure 2.3: Hashtag distribution of crawled Twitter dataset

Due to the problem of the long-tail distribution of folksonomies, recommender systems have evolved in this area to cope with this problem of synonyms. The avoidance of synonyms is also one of the key features of the SnoopyConcept (*cf.* Section 3.3). In general, the approaches which aims at homogenization knowledge can been categorized into two main classes. The first class consists of traditional approaches that deal with already stored knowledge, that is aligned by other users or administrators at a later point in time. The second class of approaches aims at aligning the knowledge already during the insertion process. Both types are sketched in Figure 2.4 and discussed in more detail in the following sections.

2.4.2 Guided Refinement and Enrichment

The main idea of the following approaches is the collaborative refinement of already existent semi-structured data. The possible refinements, alignments and extensions of already stored knowledge are mostly computed based on external sources. Such sources can be other structured or semi-structured knowledge bases. But even the web—the biggest available knowledge base—can be exploited to gather new refinements of already stored knowledge. Especially the exploitation of the web is very challenging due to its size, the unstructured

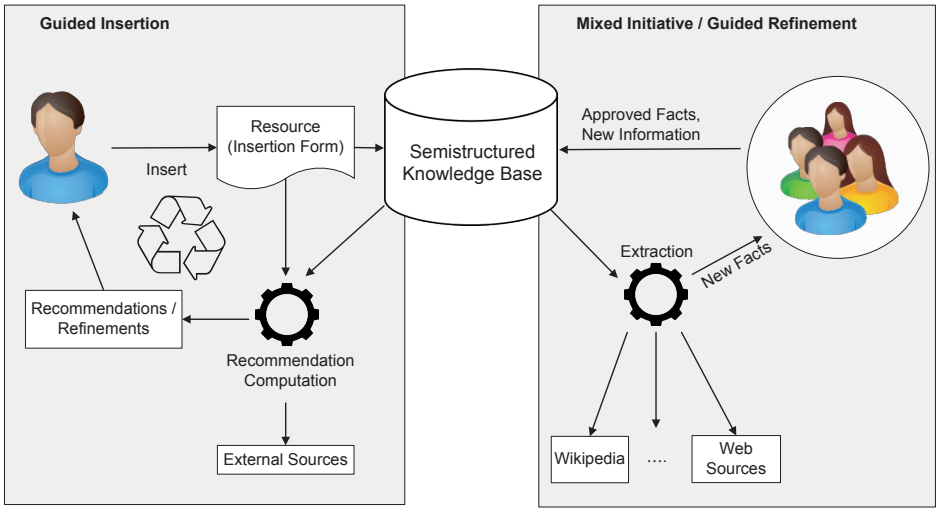


Figure 2.4: Two main approaches to improve heterogeneity

format and the associated scaling and performance issues. Approaches which use external sources to compute refinements and subsequently rely on a collaborative review of the detected refinements (sketched in Figure 2.4 (right box)) are described in the following sections.

Alignment & Refinement

As already carved out in the introduction of this section, collaboratively curated data may also be refined and aligned *after* the insertion of data. This is a crucial step in order to provide efficient querying facilities. Such an alignment mostly aims at homogenizing the set of predicates to a common schema.

During the last years, various approaches for aligning and matching RDF data have been developed. Hausenblas and Halb use a manual approach [73] that enables users to collaboratively interlink resources within RDF data. Beside such a manual alignment, also automatic alignment methods have been developed. The method proposed by Horrocks *et al.* [78] is based on common naming schemata, namely the comparison of properties within the datasets. Further approaches only rely on string-matching, e.g. for the DBpedia Lookup

service[21]. Such alignment services may also be based on context information of the entities which have to be aligned, e.g. based on the geographic coordinates, data types, etc. The Silk project [168] aims at combining these different alignment techniques to interlink RDF datasets.

Knowledge Harvesting & Information Extraction

The process of information extraction or knowledge harvesting aims at scanning the unstructured text of arbitrary documents and extracting structured knowledge to extend or build large knowledge bases. The retrieval of facts from natural language sources, such as unstructured web documents, is mostly based on NLP-techniques. Natural Language Processing (NLP) [91] is basically concerned with how a computer system can understand natural language in order to gather the sense of a sentence. By doing so, computers are able to summarize texts, detect entities and to extract certain facts from a text.

Also YAGO and DBpedia (*cf.* Section 2.2.3) extract information from the plain text of articles although, the extraction method is very limited as it is based on fixed patterns which are focused on a very small amount of the available information in an article—either infoboxes or category information. Therefore, the following approaches extend the simple pattern matching to more complex matching algorithms or NLP-techniques, which analyze the grammatical structure of sentences.

For an automated creation of large knowledge bases, the manual creation of patterns is not feasible. Therefore, ground truth data or other knowledge bases are used to create patterns which are learned and refined automatically. Suchanek *et al.* [160] introduced SOFIE which uses the knowledge base YAGO to find new patterns. This is accomplished by searching already known information (trusted facts) in natural language documents. Consider the known fact that Einstein was born in Ulm which is stored in a triple `<Einstein> <bornIn> <Ulm>`. If many documents contain the sentence "Albert Einstein was born in Ulm" the pattern "X was born in Y" can be derived for finding values for the property *bornIn*. After this step, the pattern can be used to harvest new *bornIn*-information of already known persons (X) and cities (Y) in arbitrary natural language documents. Nakashole *et al.* [122] showed that such an approach is also feasible for the web in terms of scalability.

Wu *et al.* [178] introduced an approach to find missing values of infoboxes in the natural language text of Wikipedia articles as a part of their "Intelligence in Wikipedia" project. The extraction process is accomplished by the Kylin

[21] http://lookup.dbpedia.org/, accessed 2017-07-17

extractor. This extraction process is started by a preprocessing step which is responsible for the creation of a training set for the extractor. This is done by retrieving the most popular attributes from pages which make use of the same infobox template. For each of these attributes, standard NLP techniques are used to label a matching sentence which contains the attribute's value. The extraction of facts for Wikipedia infoboxes is done by learned extractors based on conditional random fields.

All described approaches result in new semi-structured facts which feature a varying confidence value as they were created using an automated process without any user interaction. Therefore, an additional review process is needed to dismiss invalid or wrong facts and confirm true facts. The SnoopyConcept guides the user by recommending semantic refinements already during the insertion (*cf.* Section 3.3.3) and therefore, does not require a review process at a later point in time. Different approaches for the review process are described in the following section.

Review Process / Mixed-Initiative

Knowledge bases, such as YAGO or DBpedia described in the section above, provide reliable knowledge with a very high confidence as they are based on very limited patterns which are optimized for a single source like Wikipedia infoboxes. To go beyond this scope, new approaches introduce more flexible patterns and extractors such as Kylin or SOFIE described above. However, this flexibility implies higher error rates and lower confidence which have to be tackled. This can be realized by automatic reasoning or by a collaborative review system which exploits the human resources and knowledge of a large community. Automatic reasoning is able to dismiss many wrong facts by using logical rules [122, 160]. E.g., it is not possible that a doctoral advisor is more than 100 years older than the PhD student. The precision (correct found facts) of reasoning enhanced systems can be up to 98% [122, 160] if the underlying knowledge base is comprehensive and the searched relation is clear and without ambiguity. Especially semantic meanings and ambiguities decrease precision dramatically. E.g., the ambiguity of city names in the United States for a *bornIn* relation is very misleading (e.g. there are over 40 different cities called "Washington" in the USA).

Especially the disambiguation of such homonyms or the review of more complex relations requires collaborative involvement of human users. Such approaches lead to a higher number of correct facts [76, 178, 180]. Approaches combining the human and machine intelligence are called mixed-initiative approaches [79]. They basically try to exploit the advantages of both the user and automatic information extraction processes as the extracted chunks of infor-

mation are verified by human users before this information is finally published. The "Intelligence in Wikipedia" [173] project uses the extraction framework described in the section above to compute new infobox entries. Subsequently, these candidates are shown to the users of the system who are then able to decide whether the candidate infobox entry was successfully and correctly extracted from the text. This is of crucial importance especially in the case of ambiguities which cannot be resolved by automated processes and can only be resolved by users. This mixed-initiative approach features the advantage that automatically extracted information are still reviewed by humans and therefore (i) verified information is published as it is subsequently added to the according infobox and (ii) the training sets of the information extraction system are refined and reviewed and as such, the extraction process as a whole is improved. It was also shown that the acceptance of such an system is very high, as the tasks that are have to be fulfilled by the users are small and easy to accomplish. The users only have to decide whether an extracted fact is correct or not. By doing so, normal users (not just active contributors to Wikipedia) are encouraged to contribute to the mixed-initiative approach.

All described approaches combine the intelligence of both humans and machines, as the algorithms can filter extensive data sources by mining algorithms in the first step. Subsequently the human user can review the small amount of recommended knowledge and import the knowledge by accepting the recommendation or resolve ambiguities and other errors. Furthermore, the recommendations and guidance mechanisms help to lower the barrier to contribute to a system and encourage the user to increase the quality and quantity of information with the knowledge base. This approach is also used in the Snoopy Concept but is already applied during the insertion process when the user—usually a specialist in her domain—is still present and open to refine her inserted content. More details about the guidance during the insertion can be found in *cf.* Chapter 3) and the following section.

2.4.3 Guidance during the Insertion

In the previous section all approaches are dealing with already stored knowledge. Considering the fact that in most cases the user who inserts data to the knowledge base is a specialist in the field of the edited content, it is worth to incorporate the user into the complex task of alignment and improvement of the content. Consider Wikipedia as an example, the interface to insert new information to Wikipedia consists of only one input field. All features to structure, link or format the content is realized by using a very complex wiki-syntax which has to be known by the user by heart. There is hardly any support or wizard functionality which would support the user during her task to provide

new knowledge to the information system. To the best of our knowledge the SnoopyConcept presented in Chapter 3 was the first approach which aims at using this powerful resource of knowledge [61, 60].

A very similar approach named "property suggestor" was introduced in 2014 in the Wikidata platform (*cf.* Section 2.2.3) to assist the user in entering information by providing suggestions for novel properties. This tool which is discussed in more detail in Section 5.4 is based on the Predicate Suggestion approach by Abedjan and Naumann which aims at enriching RDF datasets by a set of rule mining approaches [3, 2]. The approach uses FP-growth [69] to find patterns and distinguishes between subject mining, predicate mining, and object mining. Subject mining aims at finding similar subjects and rules like *George Washington* $\rightarrow$ *Lyndon B. Johnson* by analyzing the used predicates. If subjects share the same predicates it is more likely that the describe similar things, e.g., persons or presidents. Predicate mining uses this information of predicates which occur together on subjects, to generate rules such as $\{associatedBand,\ instrument\} \rightarrow associatedMusicalArtist$. Such rules identify co-occurrence patterns which can be subsequently used for recommendations. The last type of mining patterns analyses the co-occurrence of objects and construct rules like *Buenos Aires* $\rightarrow$ *Argentina* if they are frequently stored in the same subject. For the recommendation process proposed in [3], the predicate and object mining is taken into account. The computation of the predicate computing recommendations, the approach uses a two-dimensional predicate to predicate matrix including the association rule confidence values which are finally added up to generate a ranking score. This ranking strategy is equal to a SnoopyConcept ranking strategy described in Section 3.4.3 which would only consider the confidence value for ranking. The authors showed that the approach was able to reconstruct randomly removed predicates in a DBpedia dataset with a precision up to 64% at 5 recommendations. In comparison to the SnoopyConcept the approach proposes a very basic ranking algorithm. We extended the proposed algorithm by applying SnoopyConcept ranking strategies (*cf.* Section 3.4 and 3.5) and conducted an evaluation of the extended algorithm [184]. In Section 6.1.2 the evaluation results prove that the enrichment by context, as proposed in the SnoopyConcept, significantly increases the accuracy of the recommendations. More details about the approach and its improvement can be found in Section 5.4.

The concept of incorporating the user already during the insertion process is still a very unexplored research area and not widely used in information systems. Nevertheless, the presented work and the fact that Wikidata introduced the "property suggestor" are strong indicators that the user has become more important during the insertion process and thus, is incorporated to increase the quality and quantity of knowledge stored in a semi-structured information system.

2.5 Database Models

The presented SnoopyConcept and its recommendation algorithms are independent of the underlying storage technology. We proposed three models in Section 5 to implement the SnoopyConcept. In this section we introduce the underlying database models, namely, relational databases, document-oriented databases and graph databases.

2.5.1 Relational Database Systems

Relational Database Systems are based on the Relational Model which was introduced by Codd in 1970 [42]. Despite the old age of this model it is still the most popular database model and has a long tradition in the database ecosystem. In the DB-Engines database model ranking which can be seen in Figure 2.5 the Relational Model has still an popularity score of over 80%. The score is calculated by measuring mentions on web-pages, entries in discussion forums, number of job offers and mentions in professional social networks such as LinkedIn[22]. As this measurements might have a bias towards modern new movements which are more present on the internet in contrast to older mature systems it is even more outstanding that this model reaches such a high popularity score.

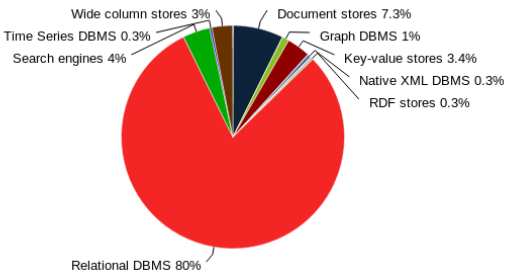


Figure 2.5: Ranking scores per database model, popularity measures based on internet information retrieval methods, Source: DB-Engines.com, Jun 2017

The Relational Model stores information in mathematical relations which are connected by relationships. Every relation conforms to a predefined schema.

[22] http://www.linkedin.com, accessed 2017-07-17

Popular representatives are Oracle[23], DB2[24], Microsoft SQL-Server[25] and MySQL[26]. The Relational Model is strongly connected with the Structured Query Language (SQL). The quasi-standard to query information stored in a Relational Database System is also used to create schemata and data. The declarative language is very flexible and allows to create filtered, aggregated and converted views of the stored data in the database system. It is easy to learn and thus, is also very popular with non-developers such as management or controlling staff. SQL can be seen as figurehead of relational systems and therefore, is also used in the name of the biggest counter-movement of relational systems—NoSQL—which is described in the following section.

2.5.2 NoSQL: Document-oriented Stores

In contrast to the Relational Model the term NoSQL does not describe a model but stands for a big movement which has been increased dramatically within the last years. The term which is an abbreviation for "Not only SQL" is used for many different models and systems which want to break the traditional monolithic database design of relational database systems. Stonebraker *et al.* argued already in 2007 [157] that the era of "one size fits all" databases is over and more specialized data storage engines which are tailored for specific use cases will evolve. Especially NoSQL databases try to sacrifice unnecessary features to get other advantages in return. For example, the softening of consistency constraints simplifies the distribution and thus, enables scalability related features.

One of the most stated model behind the NoSQL movement is the so called Key-Value Store. It is a database system which stores arbitrary values under a corresponding unique key which can be compared to a primary key in the relational model. The type of the value is not specified in detail and can feature any binary format such as an integer, string or an image. Document-oriented Stores are extended Key-Value Stores which store documents which are identified by unique keys. Documents are values which feature a structure. A document is similar to a tuple in the relational model which is as well identified by a unique key. The big difference is that the structure of a document is not specified in advance. A document consists of any arbitrary

[23] https://www.oracle.com/database, accessed 2017-07-17

[24] http://www.ibm.com/db2, accessed 2017-07-17

[25] https://www.microsoft.com/sql-server, accessed 2017-07-17

[26] http://www.mysql.org, accessed 2017-07-17

fields which can be further structured to build complex structured hierarchies within a document.

Widely used representatives of document-oriented stores are for example MongoDB[27], CouchDB[28] and the full text search focused system ElasticSearch[29]. One big difference to traditional databases such as Oracle or DB2 is the simplified handling and maintenance of NoSQL databases. The NoSQL databases are easy to install, run out of the box and do not need any additional tweaking or tuning for most use cases. This simplification was also one reason for the great success of these alternative databases and is also mentioned as "no knobs" by Stonebraker *et al.* [157]. Furthermore, most of the NoSQL databases and all three named document-oriented databases do not comply to all ACID properties. They soften the ACID properties and guarantee only eventual consistency which is often classified as providing BASE (Basically Available, Soft state, Eventual consistency) [134]. This is one key factor to enable good scalability which is provided by most of the document-oriented stores. As consistency would result in an synchronous update mechanism of all replicas in the cluster and thus, a high latency with all writes, the only way to achieve low latency writes in distributed systems is to soften the consistency guarantee. Furthermore, document-oriented systems hold all data of one document physically together. Relational systems store all information about one "document" normalized and distributed over several tables. This approach reduces the amount of needed disk space but increase the complexity when retrieving data in a distributed environment due to the fact that the data has to be fetched and merged from different tables using a distributed join. Distributed joins are very inefficient as a lot of data has to be send over the network [50]. Most document-oriented stores do not support joins at all related information is stored together in one document.

These two big differences to relational systems—the document-oriented model and eventual consistency—provide the basis for distributed high-performance systems and enable a very important feature, scalability. Especially in the field of web oriented systems, scalability is often more important that consistency. Most of these systems do not deal with critical data such as bank account data but have high demand on scalability due to the nature of web growth. For such web projects which expect a very fast growth it can be seen that "one size fits all" no longer holds and more specialized databases are needed.

[27] http://www.mongodb.com, accessed 2017-07-17

[28] couchdb.apache.org, accessed 2017-07-17

[29] http://www.elasticsearch.com, accessed 2017-07-17

Another class of NoSQL systems that is described in the next section is "graph based stores" which offers the support of relations to connect information.

2.5.3 NoSQL: Graph Stores

Graph stores are often mentioned as a subclass of the new NoSQL movement. New graph stores can be classified like this, nevertheless, databases based on a graph model were already introduced in the 1960s. One of the first databases was IDS developed by Charles Bachman [16] who received the Turing Award 1973. The goal of the IDS system was to provide a unified storage system for all systems at General Electric. The outcome was IDS which used a network model to store data. In contrast to the hierarchical model which is more restrictive the network model allows to connect objects by arbitrary relationships—and thus, forms a graph. Due to limitations regarding the query facilities and the strong coupling between physical storage and processing of data the relational model took over in the late 1970s [42]. In the 1990s object oriented databases—another type of graph databases—had evolved as the object oriented paradigm had become more popular [44, 105, 12]. Also more sophisticated graph based model such as the Hypergraph and the Hypernode model [106, 131] where introduced. It allows nested graphs to encapsulate data and is still used in new graph based database systems. Nevertheless, object oriented systems did not become generally accepted and the relational model has been mainly used until now. However, within the last decade the graph model has become more important again due to the nature of mass collaborative created content and network based data in the internet. Especially social media and linked data stimulate the development of graph based databases and revive the very traditional data model. Nowadays, most modern graph databases implement a variation of the Hypernode model or use the RDF model which are both described in the following section.

Hypernode/Property Graph Model

The basic mathematical model of a graph is defined as follows: an ordered pair $G = (V, E)$ comprising a set of nodes/vertices $V = \{1, 2, \ldots, n\}$, a set of edges $E = \{(u, v) \in \{V \times V\}\}$. Consequently, nodes can be connected by more than one edge provided that different edges are used. Moreover the definition also allows recursive edges between a single node.

A Hypergraph is a generalization of the basic graph in which an edge can consist of an arbitrary number of nodes. Formally it is defined as a pair $H = (V, E)$ where V is a set of nodes/vertices and E is a set of non-empty

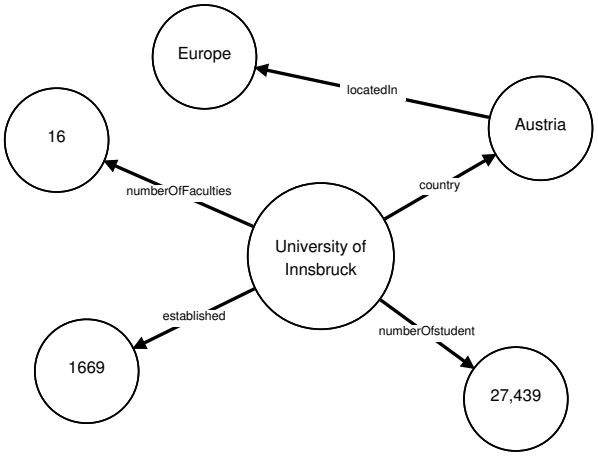


Figure 2.6: Example of a RDF subject University of Innsbruck

subsets of E called hyperedges. An extension of this definition is the definition of Hypernodes [131] which defines that every node can consist of a nested subgraph. A related graph model is the property graph model which is used for example in Neo4j[30]. It is similar to the Hypernode model as it allows additional information on nodes and edges in the graph. In the property graph model additional information is represented as key-value pairs which can be attached to nodes and edges.

RDF Graph Model

The resource description framework (RDF) [104] enables to store information about a certain subject as property-value pairs, e.g. the number of students at a certain university could be stored as *numberOfStudents: 20.000*. One such fact is called Triple as it consists of (subject, predicate, object). Subjects and predicates are usually specified by using URIs to uniquely identify them. The object can be a literal or a URI to link to another resource. By applying this simple structure every arbitrary graph structure can be represented. Due to its simplicity and flexibility it has become the de facto standard to represent knowledge.

Information about the University of Innsbruck could be structured using RDF as shown in Table 2.2 or respectively in Figure 2.6. Considering the node Austria, another important feature of graphs and RDF is shown. The node which is used as an object connected via the property country, contains further

[30] https://neo4j.com/developer/graph-database/#property-graph, accessed 2017-07-17

connections and therefore, acts as a subject as well. Thus, nodes can be reused and be connected in any arbitrary way.

University of Innsbruck	
country	Austria
numberOfStudents	27,439
numberOfFaculties	16
established	1669

Table 2.2: Example of a subject

By using such triples to represent information, all information stored is machine-readable and therefore can e.g. further be used for automatic reasoning tasks and complex structured search facilities. To query RDF data the query language SPARQL [135] has become widely used and is supported by many graph databases. RDF is also used in the Snoopy Concept which is explained in more detail in the following chapter.

2.6 Summary

In this chapter we discussed and introduced related work and technologies used throughout the course of this thesis. The first section consists of an introduction to knowledge representation and the semi-structured format which is used by the SnoopyConcept. Challenges in the area of mass-collaborative curation of knowledge using the example of wiki systems and knowledge bases are discussed in the second section. Especially the goals of the SnoopyConcept, a simplified editing process, the navigation in the dataset, and the maintenance of a homogeneous structure are covered. As the SnoopyConcept leverages recommender systems to tackle those challenges, we subsequently, introduce recommender systems in the fourth section. The very young field of research of recommender systems in the area of information systems is presented in the fifth section. The last section is dedicated to database models that can be used as an underlying basis for all presented SnoopyConcept storage approaches and concepts. In the following chapter we present the SnoopyConcept and its algorithms.

CHAPTER 3

The SnoopyConcept

There are many approaches which aim at increasing the quality and quantity of knowledge in an information systems. The majority of these approaches enhance the stored knowledge after it was entered into the information system. This is a complex task especially when dealing with semantic enhancement. In this section, we introduce the SnoopyConcept which aims at increasing the quality and quantity of knowledge by incorporating the user who inserts data to the information system. The concept exploits the knowledge of the user and tries to "snoop" as much information as possible. This incorporation of the user already at an early stage increases the quantity of information while decreases the heterogeneity of structure to increase the search capabilities of the information system. The foundation and the benefits of the SnoopyConcept are discussed in the following sections.

3.1 Key Idea: Incorporate the User

Considering a domain expert who inserts new knowledge to an information system, it is highly important to facilitate direct communication with her already during the insertion process to exploit her expertise in the respective domain. Take the example of a user who inserts information about the iPhone mobile phone into a semi-structured information system as shown in Table 3.1.

All three entered entries are not complete and require further refinement to be understood by a computer. The first problem arises with the term "handy" which is only used by German speaking people and is used as a synonym for "mobile phone". The second line "weight" does not contain any information about the unit, such as grams, ounces, kilos, etc. The last property about the manufacturer is specified as "apple".

iPhone	
type	handy
weight	135
manufacturer	apple

Table 3.1: Example of a set of information about the iPhone

The task of completing all information or enhancing the information to more distinguishable facts which could lead to the enriched entry shown in Table 3.2 is very trivial for a human being. The common inference rules that 135 kilos or ounces is to heavy for a mobile phone, the manufacturer cannot be a fruit and the knowledge that "handy" is a mobile phone are obvious for human beings. These obvious rules are very trivial although a computer system which is able to solve these problems would be considered as very smart (*cf.* [113]). If we consider a more complex example, e.g. by inserting information about the recently discovered 14th moon of Pluto, the task becomes even harder for human beings and terribly difficult for computer systems.

iPhone	
type	mobile phone
weight	135g
manufacturer	Apple Inc.

Table 3.2: Guided version of the iPhone entry

The SnoopyConcept aims at solving these problems by identifying missing semantic information at an early stage and consulting the person who probably has the most extensive knowledge about the entry which is currently inserted — the user herself. The following three simple questions asked by the information system may lead to a completed and enriched entry which can be used for further automatic processing by computer systems:

- Many other people used `type:mobile phone`... are you sure about the term `handy`?

- Other users used `gram`, `ounces` or `tons` for their property `weight`. Which unit do you want to use?

- We found two other entries in the system: `Apple Inc.` and `Apple (Fruit)`. Do you want to link the property `manufacturer` to one of them?

There is no automatic process which is able to offer the same or better performance than the user itself when dealing with semantic issues such as the meaning of the entered value `Apple`. Thus, the key idea of the SnoopyConcept is to exploit the knowledge of the collaborating users already during the insertion process—just a few clicks for the user but a really time-consuming task for the community (*cf.* Section 2.2) and a even more difficult task for a computer system.

Another goal of the SnoopyConcept is to support the users in the creation of a common, homogeneous structure. The task of maintaining a homogeneous structure in an information system is very demanding. Boulain *et al.* [26] showed that only 35% of all edits within Wikipedia are related to content, whereas all other edits aim at enhancing the structure within the Wikipedia knowledge base. Additionally, Wu and Weld [179] showed that infoboxes which adhere to predefined templates are nevertheless divergent and noisy. The SnoopyConcept aims at coping with this problem by incorporating the user into the structure alignment process. This is realized by suggesting highly suitable structures to the user during the insertion process. Consider again the example of inserting information about Kerberos, a moon of Pluto which was discovered 2011. Assume that the user already inserted the information that Kerberos is a satellite of Pluto as shown in Table 3.3.

Kerberos	
Satellite of	Pluto

Table 3.3: User just started to insert information about the moon Kerberos

The entered triple is already sufficient to compute other highly suitable attributes which can be suggested to the user. For example the properties “Discovered by”, “Discovery date”, “Named after”, “Orbital period” or “Apparent magnitude” are recommended to the user as shown in Table 3.4. The user is not tempted to introduce new properties or structured to describe the moon as she has only to choose suitable properties from the recommendation list. If the user does not retrieve any recommendations she could introduce synonymous properties, such as “Discoverer” or“Discovered on”, which would introduce

new structures that decrease the homogeneity and thus, impede the search capabilities.

Kerberos	
Satellite of	Pluto
Recommendations for Kerberos	
Discovered by	Example: Douglas Adams
Discovery date	Example: February 18, 1930
Orbital period	Example: 90465 days
Named after	Example: Fritz Important
Apparent magnitude	Example: 26.1 +/− 0.3
Known satellites	Example: 5

Table 3.4: Recommended properties based on Table 3.3

The user has to analyze and recognize all recommendations and has to decide if the suggestion is useful in the current context. This recognition is sped up by example values as they help to illustrate the context and type of the property. Consider the example of the property "Known satellites" for e.g., describing planets. If the user chooses to insert information about the satellites she may insert the total amount of satellites but may also insert a comma separated list of names of the satellites which would break a common structure. Therefore, the example value "5" is necessary to help the user to understand the meaning and type of the property "Known satellites" as fast as possible and prevent type-conflicts in the system.

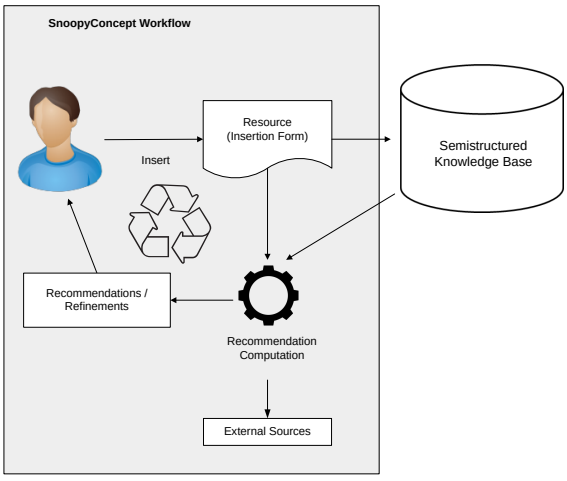


Figure 3.1: SnoopyConcept workflow of curating knowledge

Summarized, the SnoopyConcept enables the user to insert information as simple and efficient as possible. More precisely, the SnoopyConcept exploits

the laziness of the user who retrieves suitable recommendations and therefore, does not need to invest in additional considerations about the structure. By using this user-centric insertion approach, which is also sketched in Figure 3.1, the following benefits can be achieved:

- Avoid proliferation of structures (*cf.* Section 3.3.1)
- Avoid synonyms in the system (*cf.* Section 3.3.2)
- Semantic refinement by resolving homonyms (*cf.* Section 3.3.3)
- Exploit user's extensive and valuable knowledge (*cf.* Section 3.3)
- Increase the quantity of information contained in the system (*cf.* Section 3.3.1)
- Increase the quality of information in the system (*cf.* Section 3.3)

All recommendations aim at supporting the user, exploiting the knowledge of the user and therefore "snooping" as much information as possible. Thus, the quantity and quality of stored information is increased. The underlying measures and approaches of the SnoopyConcept enabling these benefits are discussed in the following section. The algorithms are explained in detail in Section 3.4.

3.2 Data Model

The SnoopyConcept is based on the semi-structured data model. Essentially, the SnoopyConcept proposes to model information and knowledge as subject-property-value triples which is based on the concept of RDF [104]. This format enables users to store information about a certain subject as property-value pairs, e.g., the number of students at a certain university can be stored as *numberOfStudents: 20.000*. Storing information about a certain subject (also called resource in RDF or article in the context of wikis), e.g. the University of Innsbruck, could be structured as shown in Table 3.5.

By using such triples to represent information, all information stored is machine-readable and therefore can e.g., further be used for automatic reasoning tasks and complex structured search facilities. Furthermore, as the data

University of Innsbruck	
country	Austria
numberOfStudents	27,439
numberOfFaculties	16
established	1669

Table 3.5: Example of a subject

model is compatible to the RDF model, the data can be exported as valid RDF at any time and ensure high interoperability with other systems.

3.3 Recommendations

The key enabler for the described benefits within a knowledge repository based on the SnoopyConcept is a recommender system [138]. Essentially, a recommender system analyses all information stored within the system to subsequently provide its users with useful recommendations. Traditionally, recommender systems are used in online shops where clients are pointed to further products (*cf.* Section 2.3).

In the context of the SnoopyConcept, the recommender system suggests suitable structures the user might want to use (*cf.* Table 3.4 in Section 3.1). Also, not only properties (structures) are recommended to the user. The SnoopyConcept also proposes to recommend values, links, types, input formats and other refinements to the user. These recommendation types and their benefits are described in detail in the next sections.

3.3.1 Recommending Structure

The term of structure recommendations refers to the recommendation of additional properties during the insertion process and is the most important feature of the SnoopyConcept as it significantly contributes to a common and homogeneous schema within the system. We propose an additional ranking mechanism to ensure that more popular or more suitable properties are preferred in the list of recommendations in order to efficiently use the limited visual space in a user interface and cope with the limited cognition of the user. The ranking algorithm is described in detail in Section 3.3.5.

These recommendations are computed on the fly and are based on the just entered properties by the user and all already used properties in the knowledge base. Any additional specified property or accepted property recommendation during the insertion process results in the recomputation and refinement of all structure recommendations for the current subject.

In contrast to classical fixed-schema information systems, the common schemata in a SnoopyConcept enabled system are very dynamic, as they are based on all stored subjects and therefore, are influenced by every newly stored subject and its properties. Every newly stored property is automatically taken into the set of possible property recommendations and can influence further recommendations to other users who want to enter information about a similar subject. The similarity of subjects is solely defined by the properties used on these subjects, as subjects do not feature an explicit, predefined type or schema (e.g., the type "University" or "Building"). The type of a subject is rather solely defined by its properties and hence, the type or category system within Snoopy is neither predefined nor rigid. Instead, the implicit, continuously changing types within Snoopy are dynamically computed based on the information/properties stored about a subject. Such an approach is also widely used for classification or object identification in the domain of schemaless data such as Linked Open Data [125]. In this domain of schemaless data, the analysis of structures is used to identify and reconciliate objects or find similar objects of the same type.

Due to this described flexibility and the fact that users are free to modify the recommended structure, the system cannot guarantee a completely unified and aligned schema. Nevertheless, the user is guided to a common schema without restricting her in her way of structuring information or extending existing schemata. Therefore, in the best case, the SnoopyConcept does not require any schema matching [153] after the insertion of data. The alignment is done implicitly during the insertion process by the user with her extensive knowledge. The user is always more powerful than any automated alignment algorithm as the algorithm do not have all the information that is available to the user (e.g. interlinked information that is stored in the human brain). Furthermore, in the case of multiple semantically similar properties which are all used within the system, the community decides which one is more appropriate by using the according property and hence, 'voting' for the property. This way, a property becomes more popular, gets recommended and hence, used.

Furthermore, the recommendation of structure increases the quantity of information as the recommendations indicate "missing" bits of information and ease the input of new information. In the mentioned example of the domain of universities, the system could recommend the property *rector*. By pro-

viding such additional property recommendations, the user is encouraged to enter more information than she originally intended to insert and the valuable knowledge of the user is once more exploited. Such information gathered by the "snooping"-process would be lost without recommendations and cannot be completed by any machine afterwards and therefore enhance the information system dramatically.

All details about the recommendation algorithm can be found in Section 3.4.

3.3.2 Avoiding Synonyms by Recommendations

The usage of synonyms is severe challenge in semi-structured information systems as they are able to dramatically decrease the search capabilities of the information system (*cf.* Section 2.4.1). The SnoopyConcept tackles this issue by applying the following recommendation procedure. During the specification of further content, the user is supported by an intelligent auto-completion feature. The system suggests suitable properties to the user which have already been used within the information system. Consider a user who started entering the property *number*. The system subsequently suggests all previously used properties in the system which are related to the term *number*. In this case, the system would suggest *numberOfStudents* and *numberOfFaculties*. In most cases the user accepts such a recommended property if it is suitable in the respective context. In this example, the user would implicitly be prevented to insert a new synonymous property, such as *numberFaculties*. A more severe challenge in information systems lies in coping with syntactically different synonyms as it cannot be solved by string-based matching approaches. Consider the example of a user entering *Faculty*, a string based matching approach would not recommend the already entered property *Academic Staff*. By using a thesaurus, *Faculty* can be matched with the semantically equivalent, already existent property *Academic Staff* which can then be suggested to the user. This approach is heavily dependent on the quality of the thesaurus. Especially in very broad information systems or frequently changing domains, it is very difficult to find a appropriate thesauri. Novel developments that extract thesauri from public community maintained sources such as Wikipedia [118, 123, 80] may help to cope with this problem. Regardless of the used technologies, the list of auto completion recommendations may be very long, e.g, there are over 400 properties in the DBpedia dataset that contain the substring *number*. To increase the accuracy of recommendations, the set of auto completion suggestions can be filtered and ranked by applying the current context of the subject similar to the approach of structure recommendations explained in the previous Section 3.3.1. For example if the property *rector* was already inserted to the subject, the properties *numberOfStudents* and *num-*

berOfFaculties are more appropriate than the property *numberOfMoons* which can be achieved by analyzing similar subjects that contain the property *rector*. The procedure can be implemented analogous to the approach in Section 3.3.1 which is described in detail in Section 3.4.

3.3.3 Semantic Refinement and Value Recommendations

The guidance of the user is not limited to properties, also possible values can be suggested. This mechanism features the advantage of providing the user with values in an already aligned form, which prevents the user from entering synonyms of already existing values. This can be achieved by using the same mechanisms used for the recommendation of properties previously described.

In contrast to properties that are present in a common dictionary, values often contain proper names or strings which cannot be found in a dictionary. To prevent typos and misspellings in this context, the recommendation of already present values is crucial as the usage of a dictionary can solve this problem only partially.

Furthermore, not only the value itself can be recommended, but the system can additionally suggest semantic links between values and other subjects. This measure copes with the common challenge of preserving semantic "correctness" of homonyms. If the user e.g., specifies the city of *Freiburg* as a twin city of *Innsbruck*, it is not clear whether the user refers to *Freiburg* in Germany or *Freiburg* in Switzerland. Within the SnoopyConcept, this problem is resolved by recommending possible semantic links from the entered value *Freiburg* to already existent, semantically equivalent subjects (e.g., Freiburg, Germany and Freiburg, Switzerland). The user is then able to specify the meaning just by accepting the appropriate recommendation and therefore, add more contextual information to the entry by creating a semantic link to the respective subject. As the user has extensive knowledge about the content to be inserted, she can provide more semantic information than any automated extraction process can do afterwards. Even if the user is not aware of ambiguities (e.g. the second city *Freiburg*) she is pointed to the ambiguity-problem and should be able to make a correct decision by already present knowledge or by considering external sources. Using these measures, homonyms are further semantically enhanced by humans, which leads to a high confidence of semantic data in the system.

3.3.4 Validation and Recommendations of Data Types

The SnoopyConcept also proposes to extend the analysis of values in the information system by a validation process, which includes determining a data type for each newly entered value, e.g., the value for the property *numberOfStudents* is asserted to be an integer value. Vice versa, if a property is added that already exists, has a data type assigned due to the usage of this data type by the majority of other property instances, the user is prompted to enter values according to this data type. As already explained in the previous sections, displaying example values indicates the correct or often used data type and avoids incorrect data types at an early stage. The detection of data types, especially in the case of numeric types, is also very important regarding search capabilities as it is crucial for queries based on numeric evaluation. E.g., the query "List all universities having more than 10,000 students" is only possible if the value of the property *numberOfStudents* is stored as a numeric value. Furthermore, also other data types like e.g. date, time, HTML, file, image, audio or video are possible and lead to special behavior in the user-interface according to the data type (e.g., date picker, calendar views, content-based image search, mp3 metadata search, etc.). Additionally, the syntactic correctness is validated according to the respective data type (e.g., correct date format).

All these measures enable the user to enter information fast and efficiently by just accepting recommendations while at the same time additional, semantically equivalent properties and values are avoided. Furthermore, the amount of unified information, the confidence and the amount of semantic data in the system are increased.

3.3.5 Ranking

Due to the fact that both the cognition of the user and the space available for displaying the recommendations are limited, the size of the set of recommended properties is restricted. In most cases a set of 5–10 recommendations is most appropriate which also corresponds to the capacity of short-term memory [115] and furthermore, can be perceived very quickly in a user interface. The problem of choice overload has also been addressed by Bollen *et al.* [25] who state that top-5 recommendations are easy to choose from by the user. The goal of the ranking algorithm is to push the most suitable recommendations out of the (probably large) set of recommendation candidates to the limited list of suggestions which are actually shown to the user. To achieve this task, the ranking algorithm has to take multiple impact factors into account

which influence the suitability of each recommendation candidate in the given context.

The most important influence factor is the confidence of the recommended item, which defines how suitable the recommended property is in the current context based on data already stored in the system. The context may consists of multiple dimensions, such as the already inserted information of the subject by the user, the user herself respectively the user's profile, and the collected data about the user.

Also the usage behavior of the property itself has to be considered by the algorithm. For example if only the popularity of a property is taken into account, novel properties will not be recommended due to the rare usage rate in the information system (*cf.* Section 2.4.1). Therefore, to overcome the "chicken or the egg dilemma" the SnoopyConcept proposes to randomly recommend novel properties to increase the probability to be chosen by the user. The concept of serendipity is well know, for example Bono [47] proposed the "lateral thinking" to avoid selective and sequential thinking, but accept accidental aspects, that seem not to have relevance or simply are not sought for. Within the last years, serendipity has become also more important in recommender systems to escape the filter bubble [163] and discover new elements [190, 82].

This usage behavior can be further analyzed and a potential popularity-rise of properties may indicate the importance of a property. Consider the example of an information system about cars. 20 years ago the information about the fuel economy and CO_2 emissions of motor vehicles were not very common. Nowadays, such information is very common and even required by law. For example, the rise in popularity a property about the CO_2 emissions which could become very important and popular after a new legislation, is potentially recognized by an intelligent information system and can be used in recommendations at an early stage to point user to this new very popular but missing piece of information.

The detailed ranking algorithm of the SnoopyConcept can be found in Section 3.4.2. In the following section, the technical implementation of all recommendation algorithms of the SnoopyConcept is described in detail.

3.4 Recommendation Algorithm

In this section we discuss the algorithms that are used to recommend suitable structures to the user as previously described. The basic algorithm of the SnoopyConcept is based on an association rule approach [7, 8], which was

adapted for the mining of relations between properties. Association rules are used to compute so called frequent item sets which were originally used in the area of business intelligence to find items which are frequently bought together. Formally, to compute a frequent itemset we define the set of all items in our database $I = \{i_1, i_2, ..., i_n\}$ and the set of all transactions $T = \{T_1, T_2, ..., T_m\}$. An arbitrary transaction $T_k \subseteq I$ contains a subset of all items in I and represents a shopping basket of a customer. Based on this transaction database, the goal is to calculate association rules which are implications $X \rightarrow Y$, where X and Y are subsets of I, and X and Y do not have any items in common ($X \cap Y = \emptyset$). If a transaction confirms to a rule $X \rightarrow Y$, X and Y are contained in the transaction T - respectively are found together in one shopping basket and therefore, are bought together. Consider the example rule $diapers, nuts \rightarrow beer$ which indicates that a customer who has diapers and nuts in his basket is going to take beer as well (there are different opinions about the truthfulness of this example in reality [132]).

Finding truthful association rules is a very demanding task, as the number of potential rules corresponds to the number of all permutations of articles in all transactions/baskets. All potential rules have to be checked for their accuracy thus, are valid in as many transactions as possible. Most business cases define a minimum support of rules to decrease the amount of useful rules and reduce the rule space which has to be analyzed.

The approach based on association rules can be easily mapped to the recommendation problem of the SnoopyConcept and is described in the following section.

3.4.1 Basic Recommendation Algorithm

The basic recommendation algorithm of the SnoopyConcept is deduced from the concept of association rules as described in the previous section and is defined as follows. Considering the item set $I = \{p_1, p_2, ..., p_n\}$ consists of all properties occurring in the Snoopy system and a transaction $s_j \subseteq I$ comprises all properties occurring together within the subject s_j. The set of all subjects forms the transaction database $S = \{s_1, s_2, ..., s_m\}$. Based on this transaction database, the goal is to calculate association rules which are implications $X \rightarrow Y$, where X is a property and Y is another property which co-occurs with X on the same subject. Due to performance reasons (*cf.* Chapter 5), the head X and tail Y of all rules consist of exactly one property in contrast to the original set-based definition of association rules [7]. This constraint reduces the complexity, the amount of permutations and therefore, allows a simplified and fast computation of rules.

Algorithm 1 is based on the described association rule approach above and shows the generation of the set $\mathcal{R}$ consisting of all rules in the form of (p_a, p_b, c) with the head property p_a, the tail property p_b and an additional counter c which indicates the number of subjects which support the respective rule. If a user e.g. specified the properties `name`, `location` and `numberOfStudents` on the same subject, the pairs (*name*, *location*), (*name*, *numberOfStudents*) and (*location*, *numberOfStudents*) are formed. If the same property is specified multiple times in the same subject—for example when specifying lists—the property is only considered once respectively, the support value is not influenced by multiple occurrences of the same property in one subject. Due to performance issues regarding the search of rules, every rule is stored directed and therefore, is stored in the system twice in different order. Thus, the ruleset $\mathcal{R}$ contains the rule (p_a, p_b, c) and the reversed rule (p_b, p_a, c).

```
   Input: set S of all subjects
   Output: set R of rules
 1 R ← ∅
 2 foreach s_j ∈ S do
 3 |   foreach p_k ∈ s_j do
 4 |   |   foreach p_l ∈ s_j do
 5 |   |   |   if (p_k ≠ p_l) then
 6 |   |   |   |   f ← false
 7 |   |   |   |   foreach (p_a, p_b, c) ∈ R do
 8 |   |   |   |   |   if (p_a = p_k ∧ p_b = p_l) then
 9 |   |   |   |   |   |   R ← R \ {(p_a, p_b, c)}
10 |   |   |   |   |   |   R ← R ∪ {(p_a, p_b, c + 1)}
11 |   |   |   |   |   |   f ← true
12 |   |   |   |   |   end
13 |   |   |   |   end
14 |   |   |   |   if (f = false) then
15 |   |   |   |   |   R ← R ∪ {(p_k, p_l, 1)}
16 |   |   |   |   end
17 |   |   |   end
18 |   |   end
19 |   end
20 end
21 return R
```

Algorithm 1: Computation of recommendation rules

The goal of the basic recommendation algorithm of the SnoopyConcept is to compute further suitable properties for an arbitrary subject which already contains some properties based on frequently used patterns. Classical associa-

tion rule based mining algorithms such as the Apriori [7, 8] or FP-growth [69] algorithm try to find the most used patterns and do not consider rarely used rules or rules which show a very low support. Such an approach is optimized for the domain of business intelligence and data warehousing but is not suitable for the SnoopyConcept as recommendations of structures have to be present also for infrequently maintained subjects or a very specialized topic which consists of only few subjects. Consider the information of moons of Pluto stored in Wikipedia. Only a few articles respectively subjects are stored in the system which contain information about this topic. Even in this low-frequent-changing area a recommender system has to be able to suggest suitable structures. Due to performance reasons most classical association rule based mining algorithms ask for support and confidence thresholds and therefore are not able to consider this low-frequent-changing area.

Another very important property of a recommender system is the accuracy. In contrast to the classical field of application of association rule based mining algorithms such as retail, the accuracy of online recommender systems is directly related to the dynamics of a recommender system. The online recommender system has to reflect a user's recent activity to keep a very high accuracy of recommendations [181, 46]. Consider the video sharing platform YouTube and the user who is watching videos. It is crucial that the recommender systems take the recent activity of the user into account [46]. This can be achieved by adapting the user's profile by adding watched videos. This interference is taking place in user profiles only and does not change any recommendation/association rules on the fly. Therefore, the real time adaption of the recommender system is achieved by small and fast changes on the user profiles. The complex and very expensive task of updating association rules can be realized in the background or regularly every night by analyzing e.g., all log data [46].

There are however other use cases which place higher demands on the update performance of the recommender system. Consider a photo tagging platform (*cf.* SnoopyTagging in Section 5.2) and a user who tags photos in the system. If the user introduces a new tag, the user will expect that the recommender systems considered her new tag and will recommend the respective tag when tagging the next photo. It is imperative that the recommender systems executes updates in real time and considers new tags immediately in this use case. Thus, to be up to date, the recommender system has to adapt not only the user profiles but also has to consider the behavior of all users and update the association rules by incorporating the behavior in real time or near real time.

Especially when considering mass-collaboration systems, the real-time computation of association rules cannot be executed within reasonable time as

the computation of all rules in a very large system is very expensive. An alternative to the recomputation of all rules (full-update) is the incremental computation of rules based on updates in the system. Depending on the real-time demands of the use case, the incremental adaptions of rules can be computed asynchronously and thus, can be executed whenever computational resource power is available. For example, YouTube analyzes log files to adapt their recommender systems [46] regularly. When considering the presented SnoopyConcept approach, an incremental update mechanism of the recommender system can exploit the simple form of the used association rule format which consists of one property in the head part and one property in the tail part of a rule. Algorithm 2 shows an incremental update algorithm based on the presented rule approach.

Input: set $\mathcal{R}$ of rules, properties $\mathcal{P}_{S_i}$ of edited subject S_i, updated property p_u, update type $type_u$ (removed or added)
Output: updated set $\mathcal{R}_{new}$ of rules

$\mathcal{R}_{new} \leftarrow \mathcal{R}$
foreach $p_k \in \mathcal{P}_{S_i}$ **do**
  $found \leftarrow 0$
  foreach $(head, tail, c) \in \mathcal{R}_{new}$ **do**
    if $(head = p_k \wedge tail = p_u \vee head = p_u \wedge tail = p_k)$ **then**
      $found \leftarrow 1$
      $\mathcal{R}_{new} = \mathcal{R}_{new} \setminus (head, tail, c)$
      if $type_u = removed \wedge c \neq 1$ **then**
        $\mathcal{R}_{new} = \mathcal{R}_{new} \cup \{(head, tail, c-1)\}$
      end
      if $type_u = added$ **then**
        $\mathcal{R}_{new} = \mathcal{R}_{new} \cup \{(head, tail, c+1)\}$
      end
    end
  end
  if $type_u = added \wedge found = 0$ **then**
    $\mathcal{R}_{new} = \mathcal{R}_{new} \cup \{(p_k, p_u, 1), (p_u, p_k, 1)\}$
  end
end
return $\mathcal{R}_{new}$

Algorithm 2: Incremental computation of recommendation rules based on updates in a SnoopyConcept based system

To implement the incremental update two different cases have to be considered—namely the addition of a new property which was not already used in the subject and the removal of a property that is no longer used in the subject. The decision which procedure has to be executed can be made locally by only

taking the properties of the edited subject into account. Also the next step, the increase or decrease command (line 9 or 12) can be generated locally. These increase and decrease commands can be stored to be processed at a later point in time. All stored commands can be processed in a batch-like job regularly when the system is not under heavy load. It is also possible to group all update commands by the rule and thus, decrease the number of update statements on the large rule set. Due to this incremental update algorithm which can also be executed asynchronously, the recommender system can be realized as a near real time system. This real time behavior is very important as the accuracy of recommender systems is strongly connected with the acceptance of recommender systems by the users. Users explore recommendations to test how the recommender systems behave and evaluate if the recommendations are "useful" for them [75]. Therefore, it is very important to build trust by incorporating the user's behavior and demonstrate to the user that the recommender system adapts to the user's needs [41].

The design of the basic algorithm of the SnoopyConcept which is described in the following part tries to consider all previously discussed properties of recommender systems and provides highest flexibility regarding real time adaptions and computability to satisfy the requirements of real-time scenarios as described above.

Algorithm 3 shows the basic step of the basic SnoopyConcept recommendation algorithm to compute the recommendation candidates. The input for this algorithm consists of the previously computed set of all rules $\mathcal{R}$ (see Algorithm 1) and an arbitrary subject S_i. The set of properties belonging to a certain subject S_i are denoted as $\mathcal{P}_{S_i}$. The final set $\mathcal{C}$ will contain all suitable properties (recommendation candidates) for the subject S_i.

For each property p_i occurring on the input subject S_i, all rule-triples (p_a, p_b, c) where p_i is the head of the rule, are detected. Rules containing a tail which is already present in S_i are not considered. All found rules are subsequently compressed (*cf.* second foreach-loop in Algorithm 3) to pairs which represent the recommendation candidates. Every recommendation candidate pair holds a suitable property which was extracted from the tails of all found rules and an additional value c which indicates the suitability of the respective property. This value c is defined as the sum of all count values of all rules which pointed to the respective property. This number indicates the probability that the property is suitable in the current context of the subject S_i and its already present properties $\mathcal{P}_{S_i}$. The higher the value c the higher the probability that the recommendation candidate is appropriate in the current context.

After having detected this (probably large) set of recommendation candidates, the most important and therefore the most useful recommendations for the

Input: $\mathcal{P}_{S_i}, \mathcal{R}$
Output: set $\mathcal{C}$ of all recommendation candidates for S_i

$\mathcal{C} \leftarrow \varnothing$
$\mathcal{T} \leftarrow \varnothing$
foreach $p_i \in \mathcal{P}_{S_i}$ **do**
  foreach $(p_a, p_b, c) \in \mathcal{R}$ **do**
    if $(p_a = p_i \wedge p_b \notin \mathcal{P}_{S_i})$ **then**
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{(p_a, p_b, c)\}$
    end
  end
end
foreach $(p_a, p_b, c_i) \in \mathcal{T}$ **do**
  if $(\exists (p_x, c) \in \mathcal{C},\ where\ p_x = p_b)$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \setminus \{(p_x, c)\}$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(p_x, c + c_i)\}$
  else
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(p_b, c_i)\}$
  end
end
return $\mathcal{C}$

Algorithm 3: Computation of recommendation candidates

user have to be extracted. This is done by ranking algorithms which are explained in the following part.

3.4.2 Ranking Algorithms

The set of of recommendation candidates which are computed as described in Section 3.4.1 is probably very large. Due to the fact that both the cognition of the user and the space available for displaying the recommendations is limited, the size of the set of recommended properties is restricted. To show the top-n suitable properties an efficient ranking algorithm has to be applied to the large set of recommendation candidates. The naive ranking is performed by sorting the pairs by value c (see Algorithm 3) which indicates the suitability of the property. The value is defined as the sum of all support values of all rules which point to the respective property.

The value c is an absolute number which is based on the number of subjects which confirm to the respective rule. Therefore, rules with a high support value are ranked very high, even if there would be more suitable rules with a

rule	count
inhabitants → mayor	100
elevation → mayor	100
name → mayor	100
name → website	50,000

Table 3.6: Example of rules and corresponding count values

property	global count
website	100,000
mayor	1,000
inhabitants	800
elevation	500

Table 3.7: Global occurrences of properties

New York City	
name	New York City
inhabitants	21,001
elevation	33 feet

Table 3.8: Example subject about New York

lower support. Consider the example in Table 3.6 which lists some example rules and corresponding support values (number of subjects which confirm to the respective rule). Table 3.7 lists properties and their total number of occurrences in the system. If these example rules and an example subject as listed in Table 3.8 are input to the recommender system, the recommendation candidates and the according c-values are:

$$\{(mayor, 300), (website, 50000)\}$$

After applying the ranking algorithm, the property `website` is ranked higher than the property `mayor` as the single rule pointing to website has a very high c-value. This would indicate that `website` is more appropriate in the given context. When using this approach, strong but often trivial rules which confirm to a large set of subjects (e.g. property `name` appears on every subject) are hard to "beat" by other more appropriate rules. For example, `mayor` might be the better recommendation in this context as it is not that obvious as `name` or `website`. This problem can be led back to the ordering which is based on the total sum of appearances and neglects the context of the recommendation—the subject itself. Especially, when showing only a limited amount of recommendations (e.g., ten properties due to user interface constraints) those trivial rules have an huge impact on the accuracy of recommendation as they would always occupy the first ten ranked positions. To cope with this problem, the algorithm has to be adapted to use a relative score which takes into account

the number of rules which feature a property candidate as described in the following section.

3.4.3 Context-Sensitive Optimization

Recommendations are always computed in a specific context and should be as accurate as possible in the respective context. In the scope of the SnoopyConcept one dimension of the context is the subject itself and its structure. As the previously presented Algorithm 3 heavily relies on the global popularity of co-occurring properties (confidence) the incorporation of the context further augments and emphasizes the importance of rule suitability for the respective subject and weakens the influence of globally popular rules. This is achieved by extending the Algorithm 3 to provide two additional scores which can be used for a more sophisticated ranking strategy. Thus, the set of recommendation candidates consists of triples in the form of:

$$(p, c_{context}, c_{confidence})$$

The score $c_{context}$ of property p indicates the number of rules which feature property p. $c_{confidence}$ is similar to the already present count score of Algorithm 3 but takes the total amount of global occurrences of the respective property into account and thus, mitigates very strong rules.

Equations 3.1 and 3.2 show the computation of the score $c_{context}$ of a given property $p_{candidate}$ of the set of recommendation candidates. For each property p_i which was already stored in subject S_i (subject which is currently edited by the user) we check if there exists a rule with the head p_i and the tail $p_{candidate}$ and count all occurrences of matching rules.

$$compute\_context(p_{candidate}) = \sum_{p_i \in \mathcal{P}_{S_i}} feature(p_i, p_{candidate}) \tag{3.1}$$

$$feature(r_{head}, r_{tail}) = \begin{cases} 1 & \text{if} \quad r_{head} \rightarrow r_{tail} \in \mathcal{R} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Equation 3.4 shows the computation of $c_{confidence}$ of a given property $p_{candidate}$. For each property p_i which was already stored in subject S_i, we retrieve the count value of rule $p_i \rightarrow p_{candidate}$ divided by the global count value of p_i. The resulting sum specifies the confidence of the property candidate $p_{candidate}$.

$$compute\_confidence(p_{candidate}) = \sum_{p_i \in \mathcal{P}_{S_i}} confidence(p_i, p_{candidate}) \quad (3.3)$$

$$confidence(r_{head}, r_{tail}) = \frac{count_{r_{head} \rightarrow r_{tail}}}{count_{r_{head}}} \quad (3.4)$$

Subsequently, the computation of recommendation candidates as shown in Algorithm 3 can be extended by incorporating $c_{confidence}$ and $c_{context}$ as presented in Algorithm 4. The new function `getGlobalPopularity(`p_i`)` returns the number of global occurrences of a given property p_i. For example, considering Table 3.7 the function call `getGlobalPopularity(website)` would return 100,000.

Input: $\mathcal{P}_{S_i}, \mathcal{R}$
Output: set $\mathcal{C}$ of all recommendation candidates for S_i

$\mathcal{C} \leftarrow \varnothing$
foreach $p_i \in \mathcal{P}_{S_i}$ **do**
  foreach $(p_a, p_b, c) \in \mathcal{R}$ **do**
    if $(p_a = p_i \wedge p_b \notin \mathcal{P}_{S_i})$ **then**
      $c_{popularity} \leftarrow$ `getGlobalPopularity(`p_i`)`
      if $\exists (p_x, c_{context}, c_{confidence}) \in \mathcal{C},\ where\ p_x = p_b$ **then**
        $\mathcal{C} \leftarrow \mathcal{C} \setminus \{(p_x, c_{context}, c_{confidence})\}$
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{(p_x, c_{context} + 1, c_{confidence} + \frac{c}{c_{popularity}})\}$
      else
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{(p_x, 1, \frac{c}{c_{popularity}})\}$
      end
    end
  end
end
return $\mathcal{C}$

Algorithm 4: Computation of recommendation candidates by considering context and confidence

Consider again the example subject in Table 3.8 and the available rules in Table 3.6 on page 62. When applying the adapted Algorithm 4 on the example, the resulting set of property candidates are:

$$\{(mayor, 3, \frac{100}{800} + \frac{100}{500} + \frac{100}{1000}), (website, 1, \frac{50000}{100000})\} \equiv \{(mayor, 3, 0.425), (website, 1, 0.5)\}$$

Using the previous Algorithm 3 the property `website` with a score of 50,000 always "beats" the property `mayor` with a score of 300. In contrast the new algorithm balances the confidence scoring to a very similar score of 0.425 and 0.5 and furthermore indicates a strong suitability for the property `mayor` by its context score of 3.

Both values $c_{context}$ and $c_{confidence}$ are very valuable and should be incorporated by the ranking algorithm. This can be achieved by using either a simple ordering by two attributes, or a weighted and normalized combination. The simple ordering sorts all entries based on the first attribute, e.g., $c_{context}$ and takes the second attribute, e.g., $c_{confidence}$ into account if two entries feature the same $c_{context}$. The normalized and weighted ranking normalizes both values to the range between 0 and 1 (feature scaling [10]) and subsequently, applies a weighting function as follows:

$$total\_score = \alpha \cdot c_{context} + (1 - \alpha) \cdot c_{confidence} \text{ with } \alpha \in [0, 1]$$

The weighting factor α which can be defined in the range between 0 and 1 indicates if the context score or the confidence score has the higher influence on the final *score*. For example a weighting factor of 0.5 indicates a balanced weighting between both scoring values. The final ranking is subsequently computed by using the combined *score* value.

Our evaluation results in Section 17 showed for the used dataset that the normalized and weighted ranking is not able to perform better than the simple ordering by two attributes. Thus, the overhead by normalizing and weighting can be omitted which results in a performance gain of more than 30% (*cf.* Section 4.4).

3.4.4 Refinement Recommendations

The previously introduced algorithms aim at recommending structure respectively properties. In this section we describe approaches to compute refinement

recommendations which incorporate the user already during the insertion process to increase the quality of data before it is stored to the information system as proposed in Sections 3.3.3 and 3.3.4.

Refinement recommendations guide the user to refine values and therefore, increase the quality of the inserted values. The first type of recommendation guide the user to specify the type of a value. For this, we apply pattern matching using predefined regular expressions for e.g., integers, strings, and URLs to automatically define the type of the value in our reference implementation SnoopyDB described in Section 5.1. Furthermore, we provide manually curated unit patterns to recognize e.g., weight or volume units. Matched types and units are suggested to the user who can validate, approve, or adapt the proposal. Predefined conversion formulas automatically convert typed values internally to a unified SI base unit [84]. For example, SnoopyDB unifies all weight values internally to kilogram. This unified typed value is subsequently used for search and thus, increases the search capabilities.

The second type of recommendation aims at reusing already present values or subjects. To achieve this, an auto-completion mechanism already proposes entries during the user is typing a new value. The recommendations are computed by applying a full-text search using e.g., n-grams [39] on all values and subjects in the information system and subsequently applying a ranking according to the amount of usage of all matched items. Additionally, we use a thesaurus [114] to improve the auto-completion by searching for syntactically different but semantically similar terms. Furthermore, a dictionary based spell-checker is used to suggest corrections to the user to avoid typos or misspellings. Both measurements aim at preventing the introduction of additional synonyms.

If those full-text based recommendations contain matching subjects, the recommendations can be used to propose semantic links which resolve ambiguities as described in Sections 3.3.3. The system suggests all matching subjects and proposes to choose one to set a semantic link to the respective subject. By accepting a proposed subject, the system creates a semantic relationship between the value/object and the respective subject which enhances the plaintext value entry by specifying its meaning.

All these recommendations incorporate the user and aim at increasing the quality of stored information. Our user experiments presented in Section 6.1.3 show that those refinement recommendations are accepted by the user and result in a higher quality of information stored in the system.

3.5 Personalizing the SnoopyConcept

The basic SnoopyConcept algorithm pursues a "wisdom of the crowd" approach which does not incorporate the behavior of a current active user in the system. Thus, the computation of recommendations is based on the collective knowledge only and does not consider the habits of a single user. This approach is well suited for community-based knowledge bases such as Wikipedia due to the democratic community-based decision process. In Wikipedia, the community regulates the suggested structure which should be used to create articles or respectively store knowledge to the information system. Other systems and use cases allow their users more freedom when inserting data into the information system. Consider a large tagging system, which provides the possibility to categorize objects by using tags. It is obvious that the tag vocabulary should be kept as small as possible to reduce synonyms and minimize the long tail distribution of tags in the information system. This can be achieved by encouraging the user to reuse already present tags in the information system. To accomplish this task, a recommendation system which uses the basic Snoopy algorithm can be used. However, in such a scenario the personal preferences of a very committed user is not considered. If the user for example is tagging photos which were shot in Innsbruck, the user always uses the tag City:Innsbruck. If the algorithm is only based on the collective knowledge, a tag City:NewYork would be recommended as it is more often used than the tag City:Innsbruck, even if the less used tag is more appropriate in the context of the user who is interested in photos about Innsbruck. This observation has triggered the incorporation of an additional context, namely the context of the user's preferences. Especially for recurring users the extension of the Snoopy algorithm by the context of the user's preferences is very important and increase the acceptance rate of the recommendation system. Therefore, we discuss an extension of the basic Snoopy algorithm by incorporating a user modeling approach in this section.

3.5.1 User Modeling

User Modeling is a traditional research area [96, 57, 30] which aims at building and refining user profiles which are subsequently used to adapt an information system to the users. The challenge of modern information systems was summed up by Gerhard Fischer [57] as follows: "The challenge in an information-rich world is not only to make information available to people at any time, at any place, and in any form, but specially to say the "right" thing at the "right" time in the "right" way.". This challenge is tackled by so-called

user-adaptive systems which use user modeling strategies to filter or adapt their information based on experience with their users.

The very mature research area of user modeling can be tracked back to Allen, Cohen, Perrault and Rich in 1979 [96, 140, 43, 127]. Since then, many developed user modeling techniques were used and adapted by the recommender system community due to the very similar goal of user modeling and recommmender systems. These adaptions can also be mapped to the challenges in the Snoopy Concept and is discussed in the following section.

3.5.2 Algorithm Extension by User Modeling

Due to the described limitation of a recommendation approach which is only based on global knowledge the algorithm was extended by a user modeling approach. We introduced the extension of the algorithm in [63] (see also Section 5.2) which takes the user's context and behavior into account. This is realized by incorporating the user's co-occurrence space to personalize all types of recommendations. The user's co-occurrence space is a subset of the global co-occurrence space and consists of all subjects which are related to the user. The decision whether an arbitrary subject is related or non-related to a specific user is dependent on the use-case of the information system. For example, subjects may be related to their authors, their creators or users who starred, liked or somehow declared interest on the subject. In the SnoopyTagging (see Section 5.2) prototype, subjects that were created by the specific user were defined as "related" to the user. The generalized extension of the SnoopyConcept algorithm (see Algorithm 3 in Section 3.4) to incorporate the user's co-occurrence space is shown in Algorithm 5. In contrast to the basic algorithm we now distinguish between two sets, namely *global* which incorporates all subjects of the information system and *user* which incorporates only subjects which are related to the specific user. Therefore, we have to compute two candidate sets as shown in the extended Algorithm 5. The first set $\mathcal{C}_{global}$ is computed by applying Algorithm 4 and contains all recommendation candidates which are based on the global rule set denoted by $\mathcal{R}_{global}$. In the second step the user's specific rule set $\mathcal{R}_{user}$ which only consists of rules that are related to the respective user u is used for the computation of the candidate set $\mathcal{C}_{user}$. Furthermore, the function `getGlobalPopularity(`p_i`)` is replaced by the personalized function `getUserSpecificPopularity(`p_i`,` u`)`. The return value of this function defines how often a given property p_i was used by a specified user u. The computation of $c_{context}$ and $c_{confidence}$, which is used in both candidate sets, corresponds to the already described procedure of Algorithm 4 which incorporates the context of the current edited subject.

Input: $\mathcal{P}_{S_i}, \mathcal{R}_{global}, \mathcal{R}_{user}, u$
Output: set $\mathcal{C}_{global}$ and $\mathcal{C}_{user}$ of all recommendation candidates for S_i and user u

$\mathcal{C}_{global} \leftarrow \varnothing$
$\mathcal{C}_{user} \leftarrow \varnothing$
foreach $p_i \in \mathcal{P}_{S_i}$ **do**
  foreach $(p_a, p_b, c) \in \mathcal{R}_{global}$ **do**
    if $(p_a = p_i \wedge p_b \notin \mathcal{P}_{S_i})$ **then**
      $c_{popularity} \leftarrow$ `getGlobalPopularity`(p_i)
      if $\exists (p_x, c_{context}, c_{confidence}) \in \mathcal{C}_{global}$, *where* $p_x = p_b$ **then**
        $\mathcal{C}_{global} \leftarrow \mathcal{C}_{global} \setminus \{(p_x, c_{context}, c_{confidence})\}$
        $\mathcal{C}_{global} \leftarrow \mathcal{C}_{global} \cup \{(p_x, c_{context} + 1, c_{confidence} + \frac{c}{c_{popularity}})\}$
      else
        $\mathcal{C}_{global} \leftarrow \mathcal{C}_{global} \cup \{(p_x, 1, \frac{c}{c_{popularity}})\}$
      end
    end
  end
  foreach $(p_a, p_b, c) \in \mathcal{R}_{user}$ **do**
    if $(p_a = p_i \wedge p_b \notin \mathcal{P}_{S_i})$ **then**
      $c_{popularity} \leftarrow$ `getUserSpecificPopularity`$(p_i,\ u)$
      if $\exists (p_x, c_{context}, c_{confidence}) \in \mathcal{C}_{user}$, *where* $p_x = p_b$ **then**
        $\mathcal{C}_{user} \leftarrow \mathcal{C}_{user} \setminus \{(p_x, c_{context}, c_{confidence})\}$
        $\mathcal{C}_{user} \leftarrow \mathcal{C}_{user} \cup \{(p_x, c_{context} + 1, c_{confidence} + \frac{c}{c_{popularity}})\}$
      else
        $\mathcal{C}_{user} \leftarrow \mathcal{C}_{user} \cup \{(p_x, 1, \frac{c}{c_{popularity}})\}$
      end
    end
  end
end
return $\mathcal{C}_{global}, \mathcal{C}_{user}$

Algorithm 5: Computation of recommendation candidates by considering user and global context based on Algorithm 4

For the final ranking algorithm, we have to prepare both candidate sets $\mathcal{C}_{global}$ and $\mathcal{C}_{user}$ as shown in Algorithm 6. This preparation step consists of the normalization (feature scaling) of $c_{context}$ and $c_{confidence}$ in the corresponding candidate set to the range between 0 and 1. This normalization of all involved values to this range provides the possibility to easily combine these values in further computations and ranking algorithms.

Input: $\mathcal{C}$
Output: $\mathcal{C}'$
$c_{confidence_{min}} \leftarrow undefined$
$c_{confidence_{max}} \leftarrow undefined$
$c_{context_{min}} \leftarrow undefined$
$c_{context_{max}} \leftarrow undefined$
foreach $(p, c_{context}, c_{confidence}) \in \mathcal{C}$ **do**
  $c_{confidence_{min}} \leftarrow min(c_{confidence_{min}}, c_{confidence})$
  $c_{confidence_{max}} \leftarrow max(c_{confidence_{max}}, c_{confidence})$
  $c_{context_{min}} \leftarrow min(c_{context_{min}}, c_{context})$
  $c_{context_{max}} \leftarrow max(c_{context_{max}}, c_{context})$
end
foreach $(p, c_{context}, c_{confidence}) \in \mathcal{C}$ **do**
  $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{(p, \frac{c_{context} - c_{context_{min}}}{c_{context_{max}} - c_{context_{min}}}, \frac{c_{confidence} - c_{confidence_{min}}}{c_{confidence_{max}} - c_{confidence_{min}}}\}$
end
return $\mathcal{C}'$

Algorithm 6: Normalization of context and confidence scores

As Algorithm 5, which computes candidates based on the global and user's scope returns two sets of potential candidates and corresponding context and confidence scores, we have to adapt the ranking algorithm to incorporate both sets and compute the best suitable recommendations. We propose to use a hybrid scoring function as follows:

$$total\_score = \gamma \cdot score_{global} + (1 - \gamma) \cdot score_{user} \text{ with } \gamma \in [0, 1]$$

The weighting factor γ defines the weight of the final score between the user-specific and global set of recommendations as shown in Algorithm 7. The additional weighting factors α and β regulate the balance between the context and confidence score in the respective scope (global or user) according to the ranking described in Section 3.4.3.

Detailed information about the behavior of the ranking regarding the three weighting factors α, β and γ can be found in Section 6.2. In our experiments a value of $\gamma = 0.1$ turned out to be beneficial. Thus, the user's scope was much more important than the global scope which is strongly dependent on the use case of the information system but shows that the incorporation of the user scope can improve the quality of recommendation drastically and can outperform the default Snoopy algorithm. The quality of global recommendations

Input: $\mathcal{C}_{global}$, $\mathcal{C}_{user}$, α, β, γ
Output: set $\mathcal{C}$ of all recommendation candidates for S_i

$\mathcal{C} \leftarrow \emptyset$
foreach $(p_g, c_{context_g}, c_{confidence_g}) \in \mathcal{C}_{global}$ **do**
  $\gamma' \leftarrow \gamma$
  $score_{global} \leftarrow \alpha \cdot c_{context_g} + (1 - \alpha) \cdot c_{confidence_g}$
  if $\exists (p_u, c_{context_u}, c_{confidence_u}) \in C_{user}$, *where* $p_u = p_g$ **then**
    $score_{user} \leftarrow \beta \cdot c_{context_u} + (1 - \beta) \cdot c_{confidence_u}$
  else
    $\gamma' \leftarrow 1$
  end
  $total\_score \leftarrow \gamma' \cdot score_{global} + (1 - \gamma') \cdot score_{user}$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{(p_g, total\_score)\}$
end
return $\mathcal{C}$;

Algorithm 7: Hybrid ranking of candidate computation based on global and user context

are heavily dependent on the data that was already stored to the system. Especially at the beginning when the information system is almost empty recommendation can be very inaccurate. This problem is called Cold-Start problem and described in the following section.

3.6 Tackling the Cold-Start Problem

The cold-start problem is a typical challenge of recommender systems [149], especially when using collaborative filtering approaches. Consider a movie rental system which recommends suitable movies to a user. A recommendation algorithm of such a system could be based on movie ratings and user profiles. Such a scenario is a typical use case of collaborative filtering approaches which compute recommendations by finding similar user profiles and incorporating user reviews of movies. These recommendation algorithms presume a sufficient amount of user ratings of movies and extensive user profiles. If we consider a movie which has not been rated or rented yet, the recommendation algorithm cannot recommend this movie as no information about this movie is present. This issue can be solved by introducing a "new movies" section or using content-based and meta-data (e.g. genres) based recommendation (*cf.* [99]). This behavior is realized by so-called Switching Hybrid Recommender Systems [90] which switch the recommendation method based on the expected quality of the recommendations. Therefore, the recommender

system can switch to a fallback method if insufficient data are present to compute suitable recommendations. The more severe problem is the computation of recommendations for a new user who just registered. The system does not have any profile information of this new user after the registration and therefore, cannot suggest any suitable movies to the user.

This cold-start problem is also present when recommending structures to the user. Consider an information system which implements the SnoopyConcept and computes structural recommendation during the insertion process. Four possible cold-start types can be identified:

- Empty information system
- New (empty) subject
- New user
- New properties

The first type "empty information system" occurs when the SnoopyConcept based system is just installed and it does not contain any data yet. In this state, the recommendation algorithm has no information at all and is not able to give any recommendation to the user. Therefore, the system is very vulnerable during this state as all information which are inserted to the system during this phase will influence the recommendations in the future. First users are therefore very powerful as they may bias the information system. Consider a simply typo in a property which was added by the first user. Due to the cold-start problem, the system does not have sufficient data to cope with this typo and accepts the information as ground truth. Recommendations to all further users will contain the incorrectly spelled property. If all users just accept recommendations and do not recognize the typo, the property will become mature and it will be more often used in recommendations. Especially typos can be multiplied over years as they are simply overlooked by many users. The same problem holds for a property which was used in an improper context. Such biased recommendations cannot be handled by the system automatically. Thus, it is very crucial that in this phase the inserted information is well structured as this inserted structure is continuously suggested to all users. Typos can be avoided by spell checkers but the avoidance of improper semantic usage is very hard and requests a more sophisticated approach. To tackle this problem, two approaches are imaginable. The obvious approach is a manual approach which is done by a very committed community or administrators who have an extensive domain knowledge. They can fill the system with a suitable ground-truth dataset which represents an optimal structure

of common subjects and lowers the influence of new users. This approach is only crowned with success if the domain of the information in the information system is limited. Considering a use case such as Wikipedia, an introduction of a suitable set of properties is very difficult as all possibilities have to be covered. In such a extensive use case scenario, the administrative staff can try to fix wrongly used properties as early as possible. These tasks can be supported by a spell checker or the usage of ontologies to try to locate problems and point the administrative staff to them to take a closer look. This manual approach of initially filled ground truth data was also used in the evaluation (*cf.* Section 6.1.3) to overcome the cold start problem.

The second possible approach is an automated approach which is based on an external source which can be exploited by the recommendation algorithm. For example Wikipedia, DBpedia or already existing enterprise-information-systems within a company can be exploited to recommend possible properties or structures if the system is still empty and is not able to compute suitable recommendation based on the data in the information system. The prerequisite for the usage of such pre-existing systems is that the systems exhibit a common structure which is forced by a fixed predefined schema or maintained by an administrator or by a community, e.g. the community of Wikipedia (*cf.* Section 2.2). One could also fill the system by automated extracted information from external systems such as Mail2Wiki [70] which extracts information from emails. Due to the fact that the recommendations heavily base on the initial data the quality of recommendations are directly related to the quality of the automated extraction process. Therefore, manually well curated data are more suitable for tackling the cold start problem than automatically generated data.

A combination of the manual and the automated approach was chosen to realize an importer for the SnoopyDB [130]. This importer allows an administrator to import arbitrary JSON-APIs which are very common and provided by many online platforms. Such an importer can also be used to overcome the "empty information system" problem as the system can be automatically filled with a suitable ground-truth set which is imported from a JSON-based API. The semi-automated process of the importer allows the user to define a mapping between the schema of the JSON-data and the final schema of the SnoopyDB. Figure 3.2 shows a screenshot of the main step of the mapping process. The process is realized as an iterative process which complies with the insertion process in SnoopyDB. Thus, recommendations for the mapping are computed and suggested to the user who is defining the mapping. The raw attribute information (name and value) of the JSON-data is used a basis for the recommendation computation. Based on the name of an attribute suitable synonyms or other keys which are already present in the system are suggested to the user. Continuously, the user may choose suitable attributes and define

a mapping between the original attribute in the JSON-object and the key in SnoopyDB. In the example in Figure 3.2 the RottenTomatoes JSON API [1] is used to initially import movies. The screenshot shows already defined mappings by the user that are indicated by the arrows. Based on these mappings additional suitable Properties are recommend which can be easily added by choosing a field of the original JSON data that is present in the drop down field. An additional preview of entries in the JSON list (bottom right) supports the user to find the correct mapping. After the successful definition of a mapping scheme all subjects of the JSON-API can be imported automatically. If there are any uncertainties, the importer can ask the user to clarify data (e.g. spelling issues or possibilities to link values to other subjects to improve the semantic information in the system). The prototype of the SnoopyConcept based importer was published in [130].

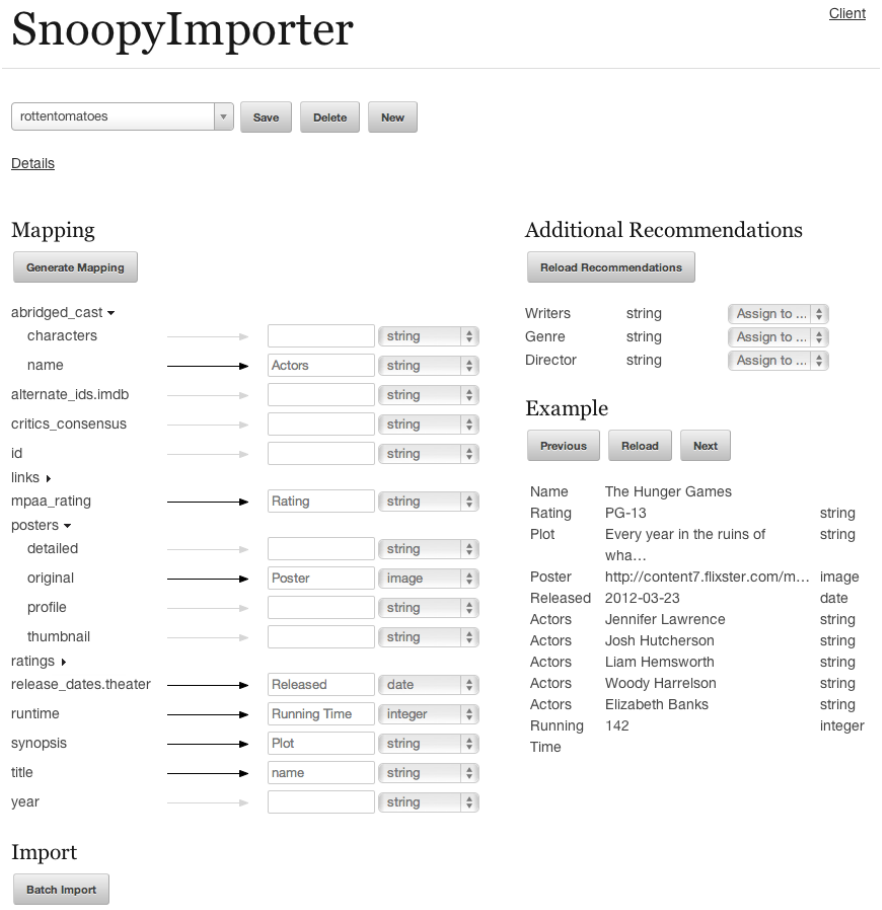


Figure 3.2: SnoopyImporter web interface: importing Rotten Tomatoes

[1] http://www.rottentomatoes.com, accessed 2017-07-17

The second type of cold-start problems occurs when the users create a new subject which does not contain any properties yet. In this case, the system cannot compute any recommendations based on the currently entered information. The SnoopyConcept proposes a switching hybrid recommender to tackle this problem and use a fallback which recommend properties that are often used in the system. For example a very popular property `name` or `website` which is used by most of all subjects can be recommended to the user. If the user who inserts information is well known by the information system the recommendation engine can be extended by incorporating the user's context and behavior based on previously entered subjects. Thus, the computation of popular properties is applied on the user's scope and the recommendations contain properties which are often used by the user. Consider a user who already inserted many photos about the location Innsbruck. When creating a new subject without any properties, the snoopyfied recommendation engine recommends the property and its value `location:Innsbruck` to the user as the probability is very high that the user inputs another image about Innsbruck.

When incorporating the user's scope, the third cold-start problem "new user" emerges as new users who do not have any profiles which could be used in the recommendation computation. As the SnoopyConcept proposes a hybrid recommender system this cold-start problem can be tackled by a switching approach which switch to a fallback method to compute recommendations. In this case, the fallback is to compute recommendations based on the global scope without performing any refinements based on user profiles. An improvement of this fallback handling can be the exploitation of vague user profiles that for example are based on the location, retrieved by analyzing the user's IP address or other vague information that can be retrieved due to domain knowledge (e.g., the referrer). The fallback approach based on global computations corresponds to the basic Snoopy algorithm without any User Modeling extensions.

The last topic regarding the cold-start problem is concerned with the insertion of unknown properties. Considering a subject which only consists of properties which are unknown to the information system. This situation is very similar to the previously described type of empty subjects as the recommendation is not able to recommend any properties based on the current inserted information. The same approach of recommending popular properties based on the global scope or the personalized user's scope is possible. Furthermore a thesaurus approach can be applied to try to understand the unknown properties. This can be realized by using an external thesaurus and finding synonyms or related words of the current inserted properties. All found words which are semantically equivalent or related to the inserted properties can then be used as a basis for the default recommendation strategy to find suitable recommendations.

Summing up the above, the optimal strategy to cope with the cold-start problem is to use a mixed-initiative approach which involves a committed community supported by automated tools to cope with the large amount of data (*cf.* Section 2.4.2). Tools can analyze big data very fast and automatically and can point the administrative staff to potential problems. Especially complex semantic problems can only be detected by automated tools and have to be solved by humans who are able to understand the complex semantic context.

3.7 Summary

In this chapter we presented the SnoopyConcept and the key idea to incorporate the user already during the insertion process to increase the quality and quantity of knowledge in the information system by guiding and encouraging the user to insert homogeneous structured data. Besides the underlying triple based data model the recommendation algorithm was explained. Furthermore, the extension of the algorithm by considering user modeling was described. Last, approaches to tackle the cold start problem in information systems applying the SnoopyConcept were discussed. In the subsequent chapter we present different data models which can be used to implement the SnoopyConcept. Furthermore, implications and details about the implementation of the recommendation algorithms based on underlying data model are discussed.

CHAPTER 4

SnoopyConcept Storage Models and Recommendation Implementation

In this chapter we discuss different implementation aspects of the SnoopyConcept and its algorithms. The implementation of the SnoopyConcept can be split into two major challenges.

The first challenge is to provide a fast storage and retrieval system for triples which hold predicate-object information about subjects. As the SnoopyConcept (*cf.* Section 3.2) is similar to the generic RDF model (*cf.* Section 2.1.3, all RDF based databases and approaches may be used to implement the basic storage system for Snoopy triples.

The second and most important challenge of the SnoopyConcept is the implementation of a high-performance recommender system which is able to compute suitable recommendations (*cf.* Section 3.4) within milliseconds. This performance provides a real time experience to the user and increases the user's perceived accuracy [136].

To tackle those challenges, we propose three different implementation approaches based on *Relational Database Systems*, *Document-oriented Stores*, and *Graph Stores*.

For every database model we discuss a suitable storage model and the computation of recommendations. We cover performance and scalability considerations which are crucial when dealing with large datasets which are common in mass-collaboration systems. The last section consists of a performance evaluation of all proposed storage models.

4.1 Relational Database Systems

The Relational Model (see Section 2.5.1) is one of the most used database models and therefore, has been used as a basic layer in many RDF databases. Therefore, it was the first choice to implement the SnoopyConcept based on the relational model. In the following sections we describe common RDF storage approaches and how we adapt them to develop the relational storage model for the SnoopyConcept.

4.1.1 RDF Storage Models

These so-called Triple Stores based on the relational model have been realized using different mapping strategies. The main two classes of mapping strategies are *schema-oblivious* and *schema-aware* representations [164]. The schema-aware mapping makes use of a present RDF schema (RDF/S) which defines the structure of RDF data. As each subject adheres to a predefined class in the schema, each class can directly be mapped to an according relation (table) in the relational model. Each type/property is mapped to an attribute (column) in the according relation. Thus, one tuple (row) in a relation represents one resource (subject) in the RDF representation. The schema-oblivious approach is also feasible when no RDF-schema is present and the structure of the RDF data is not known in advance which is the case for the SnoopyConcept. Each triple in the RDF data—subject-predicate-object—is mapped to one tuple in one big relation. This relation consists of three columns which represent subject, predicate and object. The differences between the mapping approaches mainly show in terms of retrieval performance. The schema-aware approach is able to retrieve one resource by fetching only one row. When using the schema-oblivious approach one has to fetch multiple rows to retrieve all properties of one single resource. On the other side, the schema-aware approach might lead to many NULL-values as especially in collaborative curated knowledge bases many properties are used rarely (*cf.* Section 2.4.1). To cope with this sparsity problem hybrid solutions were introduced which map frequently used properties to corresponding property-columns and store all remaining sparse properties that do not occur frequently by using the schema-

oblivious approach. To realize such an approach, the RDF data has to be analyzed which dramatically increase the complexity of the approach. This is one major reason why hybrid systems have not become widespread. Furthermore, the number of tables can be extremely high due to the potentially large amount of classes in collaborative information systems. Especially traditional relational database management system are not able to handle thousands of tables which restricts the approach when using classical RDBMS.

One approach to handle such a large set of relations and the problem of many NULL-values was proposed by Abadi *et al.* in [1]. The approach makes use of column oriented database system which are able to handle large amounts of tables and columns. They propose to map classes to tables and one property to one column. Column oriented database systems do not store data table or row-wise but store each column as a separated data structure. This storage model allows to apply compression algorithms column-wise and compact columns more efficiently. Considering the problem of many NULL values in one column due to the fixed mapping of all properties to columns, such an approach is able to compact all NULL-values by e.g., a simple run-length encoding [158]. In contrast to classical relational database systems, column oriented database system are designed without any limits regarding the number of tables or columns in one table. This feature is also crucial for such a proposed mapping, as it is usual that RDF data contains large classes with hundreds of properties which results in hundreds of columns in one table in the relational model. The big advantage of the vertical approach [1] lies in the reduced complexity as one fixed mapping can be used, while at the same time, solving the problem of efficiently storing data and their resulting NULL values.

In general, due to the high complexity of schema-aware and hybrid approaches, the schema-oblivious approach has emerged and is now the default approach when considering fast and modern Triple Stores based on the Relational Model. The schema-oblivious approach can be realized with different underlying storage models, such as row or column oriented storage models. When using a row-based approach, typically the three-column layout of a Triple Store is used. The performance gain by mapping one property to one line can be caught up by optimizations such as adding well-chosen index structures. Such indices are very important due to the nature of the relational representation. Especially the usage of a schema-oblivious mapping approach results into many self-joins as all information are stored in one big table. Consider the example of a query "List all cities which have a female mayor and print the name of the city and the mayor" which is executed on one big triple table which holds all information. The transformed query to SQL can be found in Listing 4.1 and consists of four self joins to be able to retrieve all needed facts. The relation `t1` (line 3) selects all entries which indicate that the subject is of type city. In the next step the connection between the city and the mayor is searched (line 5).

Listing 4.1: Trivial query transformed to SQL results in four self joins

```
1 SELECT t5.object as CityName, t4.object as MayorName
2 FROM triples t1, triples t2, triples t3, triples t4, triples t5
3 WHERE t1.predicate = 'type' AND t1.object = 'City'
4 AND t1.subject = t2.subject
5 AND t2.predicate = 'mayor' AND t2.object = t3.subject
6 AND t3.subject = t4.subject
7 AND t3.predicate = 'gender' AND t3.object = 'female'
8 AND t4.predicate = 'name'
9 AND t5.subject = t1.subject AND t5.predicate = 'name'
```

Relation `t3` contains the gender of all found mayors of `t2` and is continuously filtered to all female mayors by the selection in the where clause (line 7). The join between `t3` and `t4` results in the set of all names (`t4`) of all found female mayors (`t3`). In the last step the same approach is used to fetch the names of all found cities which are contained in `t5`. Thus, the objects of `t4` and `t5` which contain names of the mayors and the cities are projected respectively printed in the final result set. Due to this heavy usage of self joins, indices are needed to execute such queries in an acceptable amount of time when using the schema-oblivious approach. Those index structures are discussed in the following section.

4.1.2 RDF Storage Index Structures

To access the triple based storage as described in the previous section as fast as possible, additional index structures are needed. Hart *et al.* proposed in [72] to create multiple indices to cover all access patterns of queries and speed up the execution of queries. They used a quadruple based RDF storage system which besides the subject-property-object triple concept also takes the context (namespace/prefix) into account and therefore, stores all RDF information chunks as quads in four columns. To speed-up all possible queries, where any of subject, property, object or context is either defined or a variable, they introduced $2^4 = 16$ access patterns. To cover all access patterns, 16 indices are needed in a naive approach. Hart *et al.* also showed in [72] that all 16 patterns can be covered by only six indices when using B+-tree index structures. The resulting indices are `spoc, poc, ocs, csp, cp` and `os` where each letter denotes the corresponding column in the original table. A very similar approach is proposed by Wood *et al.* in [177] who proposed to store quads in six different orders to cover all possible queries. However, both approaches do not consider all possible permutations ($4! = 24$ when using quads) when taking into account the cyclic order of the indices. For example there is no suitable index when querying all subjects of a given property, e.g. all cities where the property mayor is specified, as we would need an index with the combination `ps` to solve this query. This problem is tackled by Weiss *et*

al. in the Hexastore approach [172]. They argue that most other RDF stores prioritize subjects or properties and therefore, prerequisite specified property or subject values in queries to use indices. Such patterns which do not specify properties are very common in more complex queries, e.g. finding incoming edges/properties of a specified object, but are not addressed by most RDF stores. In [172], the concept of Hexastores is introduced which pays equal attention to all RDF items and does not privilege any item. This goal is reached by materializing all possible orders of the `spo` triple which results to six indices ($3! = 6$) `spo, sop, pos, pso, osp` and ops. Table 4.1 shows that six indices are able to cover all 15 possible patterns which are possible in a query.

Pattern → Index Hexastore
`s??` → `spo` or `sop`
`sp?` → `spo`
`spo` → `*`
`so?` → `sop`
`sop` → `*`
`p??` → `pos` or `pso`
`ps?` → `pso`
`pso` → `*`
`po?` → `pos`
`pos` → `*`
`o??` → `osp` or `ops`
`op?` → `ops`
`ops` → `*`
`os?` → `osp`
`osp` → `*`

Table 4.1: Coverage of Hexastore's indices

Weiss *et al.* use vector based data structures in their Hexastore approach [172]. Thus, a vector with all according properties is attached to every subject when considering the `spo` index. Furthermore, every property points to another vector which consists of according objects of the spo index. The advantage of storing all information in separated vectors is the possibility to reuse vectors. For example the object vectors attached to `sp` of the `spo`-index can be reused in the `pso`-index as the objects are defined by `sp` and `ps` independently of the order of subject and property. This approach reduces the amount of required disk space and allow to perform fast join algorithms due to the sorted layout of the vector based approach. We use a very similar storage model in SnoopyDB, our reference implementation of the SnoopyConcept, which is described in more detail in the following section.

4.1.3 SnoopyConcept Relational Storage Model

We build our approach on the classical relational database model using a schema-oblivious mapping. Thus, each RDF triple is stored in a triple table consisting of three columns, namely subject, property, and object. Due to performance reasons and to save disk storage we use a dictionary approach to store only numeric values in the triple table and store the string representation or URIs of subject, property, and objects in corresponding lookup tables. Furthermore, the lookup tables can be used to store statistics about the respective values which are used in the computation of recommendations, e.g., global count of a property.

Subjects

sub_id	**sub_text**	**sub_count**
1	http://dbpedia.org/resource/Innsbruck	4
2	http://dbpedia.org/resource/Austria	1

Properties

prop_id	**prop_text**	**prop_count**
1	http://dbpedia.org/ontology/country	1
2	http://www.w3.org/2000/01/rdf-schema#label	3
3	http://xmlns.com/foaf/0.1/homepage	1

Objects

obj_id	**obj_text**	**obj_count**
1	http://dbpedia.org/resource/Austria	1
2	"Innsbruck"@en	1
3	"Innsbruck"@de	1
4	http://innsbruck.at/	1
5	"Austria"@de	1

Table 4.2: Dictionary/lookup tables

Table 4.2 and 4.3 contains an excerpt of the DBpedia entry of the city Innsbruck[1] mapped to the SnoopyConcept storage model. The shown table structure is simplified as the SnoopyConcept reference implementation furthermore stores information about the type, language and other statistical data. The raw data of the subject Innsbruck is stored in the triple format as shown in Table 4.3. The corresponding string and URI representations are stored in

[1] http://dbpedia.org/page/Innsbruck, accessed 2017-07-17

dictionary tables as shown in Table 4.2. The numeric ids of all string representations are generated during the import process. When querying the data, the used strings in the query have to be mapped to the numeric representation. The same holds for the result set which has to be mapped back to a string representation. The dictionary approach has a huge impact on performance which is mainly based on the reduced disk storage and the handling of numeric values. As every string is only stored once, the size of the main triple table can be reduced dramatically. This storage reduction results in a faster processing of all triples and a reduced size of all index structures on the main triple table. Especially when dealing with a lot of self-joins, the database system works more efficiently with numeric values as integer based indexes are faster and therefore, speed up the lookups of join algorithms. The overhead of mapping and lookups is therefore negligible as the compute-heavy operations, such as joins, are speeded-up.

sub_id	**prop_id**	**obj_id**
1	1	1
1	2	2
1	2	3
1	3	4
2	2	5

Table 4.3: Triple table which contains the raw information

For the definition of indexes on the main triple table, we identified in the first step all possible patters of queries regarding defined columns (subject, property or object) in the selection part and the queried columns in the projection part which are part of the result set. Consider again the query about cities that contain the information about the mayor a typical pattern would be `(?,mayor,?)` to identify all subjects that have a specified property `mayor`. As the order of the defined columns does not influence the choice of the index we were able to identify 13 possible query patterns which are shown in Table 4.4.

The first three columns in Table 4.4 define which columns are defined (✓) in a corresponding query-pattern. The question mark indicates which columns are queried and therefore, projected as the result of the query-pattern. The fourth column indicates the column order which is needed in a suitable index to answer the query-pattern. The last column shows all indices of the SnoopyDB that are suitable to answer the corresponding query-pattern.

As the information about the object column in the `spo` and `pso` index is redundant, we store the full `spo` index and reduce the `pso` index to `ps`. This procedure is accordingly applied to the other indices which results in six in-

S	P	O	Needed information (ordered)	Index SnoopyDB
✓	?		`sp`	`spo`
✓		?	`so`	`sop`
✓	?	?	`spo` or `sop`	`spo` or `sop`
	✓	?	`po`	`pos`
?	✓		`ps`	`ps`
?	✓	?	`pso` or `pos`	`pos`
	?	✓	`op`	`op`
?		✓	`os`	`osp`
?	?	✓	`osp` or `ops`	`osp`
✓	✓	?	`spo` or `pso`	`spo`
?	✓	✓	`pos` or `ops`	`pos`
✓	?	✓	`sop` or `osp`	`sop` or `osp`
✓	✓	✓	*	`sop`,`spo`,`pos` or `osp`

Table 4.4: Coverage of SnoopyDB's indices

dices, four trivalent indices and two bivalent indices, namely `spo, sop, pos, osp, op,` and `ps`. As shown in Table 4.4, we are able to answer all possible query patterns by these six indices while at the same time reducing the amount of needed disk space to about 87% of six triple-indices. In contrast to the Hexastore approach [172], our approach does not need its own storage layout and implementation. The presented index approach solely relies on default index structures which are provided in all classical relational database models. This huge advantage simplifies the implementation of the SnoopyConcept drastically as it can be developed on top of any relational database system.

4.1.4 Recommendation Computation

For the recommendation part of the SnoopyConcept the relational model offers two possible approaches to compute suitable recommendations. The first approach computes recommendation on the fly. This means that based on the currently edited subject, the system has to find suitable properties. If we consider a simple triple store as described above, we have to perform a query as

shown in Listing 4.2 to get a ranked list of all suitable properties. The query searches for subjects that contain properties that are present in the currently edited subject and thus, show a similarity to the currently edited subject. The set of properties that are present in similar subjects but not present in the currently edited subjects (guaranteed by the `NOT IN` statement) forms the set of suitable properties. This candidate set needs to be ranked to recommend the most suitable properties in the current context. In the query in Listing 4.2, properties are ranked by the number of votes (*cf.* Section 3.4.3) they get by other properties, thus, the higher the number of properties in `t1` that point to the respective property in `t2` the higher the rank in the final set of recommendations.

Listing 4.2: On-the-fly computation of recommendations

```
1 SELECT t2.property FROM triple t1, triple t2 WHERE
2 t1.property IN ('prop1','prop2','prop3') AND
3 t2.property NOT IN ('prop1','prop2','prop3')
4 AND t1.subject = t2.subject
5 GROUP BY t2.property
6 ORDER BY count(*) DESC
```

Although the query contains only one join and a grouping, in practice these two operations are very expensive as the join generates a extensive intermediate result set which is continuously grouped by the property. These two operations are inefficient on large datasets and therefore, too slow for any user-satisfying real-time recommendation computation.

4.1.5 Rule Based Recommendation

Due to the previously explained performance considerations of on-the-fly rule generation, the second and recommended approach to compute the set of recommendations involves the usage of precomputed association rules (see Section 3.4). The realization using such rules in the relational model is straight forward and can be realized by a two-column table which represent the head and tail of one rule. The generation of rules is shown in Listing 4.3 and loops over all properties in the system and computes all corresponding rules where the current properties denotes the head of the rule. The result is a very large table containing all rules including duplicates. To compact the rule table and provide a possibility to perform a simple ranking as described above, an additional count may introduced. By grouping all rules by head and tail and store the count value, as shown in Listing 4.4, the quality of the recommendations can be increased by introducing a simple ranking based on the count values which is also described in detail in Section 3.4.

After generating all rules, the recommendation computation can be performed by a very simple and therefore fast query on the rule set as shown in Listing 4.5. As the set of rules is already compacted and no join is needed, the SQL query does not need to handle large intermediate join-results and only needs to perform a grouping operation on a manageable set of suitable properties. Moreover, there is no need for additional index structures, as the primary key of head and tail is sufficient for all computations.

Listing 4.3: Precompute association rules

```
1 FOREACH (properties as curr_prop)
2   INSERT INTO rule (head,tail) (
3     SELECT t1.prop, t2.prop
4     FROM triple t1, triple t2
5     WHERE t1.subject = t2.subject
6       AND t1.prop = curr_prop
7       AND t1.prop <> t2.prop
8     GROUP BY t1.subject,t1.prop,t2.prop
9   );
10 END FOREACH;
```

Listing 4.4: Compacting rules

```
1 INSERT INTO rulecount (head,tail,c)
2   (SELECT head, tail, count(*)
3    FROM rule GROUP BY head,tail);
```

Listing 4.5: Compute top-10 recommendations based on precomputed rules

```
1 SELECT tail as recprop FROM rulecount
2 WHERE head IN ('prop1','prop2','prop3',...)
3   AND tail NOT IN ('prop1','prop2','prop3',...)
4 GROUP BY tail
5 ORDER BY sum(c)
6 DESC LIMIT 10;
```

The recommendation algorithm can be improved by taking the confidence and context values into account (*cf.* Section 3.4.3). The SQL based query of the optimized ranking can be found in Listing 4.6. The context value `count(*)` reflects the number of rules which are pointing to the respecting property. The confidence value `sum(c/prop_count_dist)` denotes the strength of the rules in proportion to the number of subjects that contain the respective property at least once.

Listing 4.6: Recommendation computation considering confidence

```
1 SELECT tail as recprop, count(*),
2 sum(c/prop_count_dist) FROM rulecount
3 JOIN dict_prop ON (head=prop_id)
4 WHERE head IN ('prop1','prop2','prop3',...)
5   AND tail NOT IN ('prop1','prop2','prop3',...)
6 GROUP BY tail
7 ORDER BY count(*) DESC , sum(c/prop_count_dist) DESC
```

```
8 LIMIT10;
```

The normalized (feature scaling) and weighted combination between context and confidence as described in Section 3.4.3 can be translated to SQL as shown in Listing 4.7.

Listing 4.7: Hybrid recommendation computation incl. weighting

```
1  SELECT recprop,
2    IFNULL((context-mincontext)/(maxcontext-mincontext), 0)
3      as contextnorm,
4    IFNULL((confidence-minconfidence)/(maxconfidence-minconfidence), 0)
5      as confidencenorm
6  FROM
7    (SELECT min(context) as mincontext,
8            max(context) as maxcontext,
9            min(confidence) as minconfidence,
10           max(confidence) as maxconfidence
11    FROM
12      (SELECT tail as recprop,
13              count(*) as context,
14              sum(c/prop_count) as confidence
15       FROM rulecount JOIN dict_prop ON (head=prop_id)
16       WHERE head IN ('prop1','prop2','prop3',...)
17         AND tail NOT IN ('prop1','prop2','prop3',...)
18       GROUP BY tail) as rectemp
19       ) as maxmin,
20
21       (SELECT tail as recprop,
22               count(*) as context,
23               sum(c/prop_count) as confidence
24        FROM rulecount JOIN dict_prop ON (head=prop_id)
25        WHERE head IN ('prop1','prop2','prop3',...)
26          AND tail NOT IN ('prop1','prop2','prop3',...)
27        GROUP BY tail
28        ORDER BY context DESC , confidence DESC
29        ) as recommendations
30
31 ORDER BY (contextnorm * @alpha) + ((1-@alpha)*confidencenorm) DESC
32 LIMIT 10;
```

The implementation based on the relational model turned out to be very flexible as it can be used on top of every relational database system but at the same time, computes recommendation very fast. In our evaluation the recommendations were computed in 3-20ms depending on the amount of initial properties. More details regarding the performance of this approach can be found in Section 4.4.

4.2 NoSQL: Document-oriented Stores

In contrast to the previously described Relational Model, most NoSQL databases (see also Section 2.5.2) do not provide any support for join operations. Especially in document-oriented stores documents are not normalized or split to multiple chunks and thus, do not need any join operations to reassemble the original document. Such a non-normalized storage model results in the redundant storage of data in exchange of an increased performance due to the lack of reassembly-costs and easier distribution to multiple nodes in a cluster environment (*cf.* Section 2.5.2).

4.2.1 Storage Model

One obvious model to store RDF data in a Document-oriented Store is to map all properties of one subject to one document. Each document is referenced by its key, in the use case of RDF the name or URI of the subject, which can be used to retrieve all properties of one subject by execute one single-point query. Listing 4.8 shows the subject "University of Innsbruck" stored in a document in MongoDB. MongoDB is a popular NoSQL database which we use for the implementation of the SnoopyConcept.

Listing 4.8: Document in MongoDB about the University of Innsbruck

```
1 {
2   "_id" : "University of Innsbruck",
3   "country" : "Austria",
4   "numberOfStudents" : 27439,
5   "numberOfFaculties" : 16
6   "established" : "1669"
7 }
```

When only considering properties of one subject, the query can be executed very fast. Although, due to the lack of join operations, more complex queries which have to resolve relationships between subjects cannot be realized by one single query. Consider again the example of a query "List all cities which have a female mayor and print the name of the city and the mayor". The first task of finding all cities which include the information about the mayor can be easily realized by one query. In the next step, the documents of all found mayors which include the information about the gender of the mayor have to be retrieved by an additional query in the system.

The first query is realized by performing a full-table scan as document-oriented stores do only provide fast access via the primary key. If the primary key is not given in the query, all data has to be analyzed. Depending on the

underlying database system and its features, it is possible to increase the query execution performance by creating additional indices which allows to specify non-primary-key fields in the search query. In document-oriented stores indices are created on properties (similar to column based indices in classical relational database systems). The index is realized as a secondary index which points to all documents that use the respective property. Furthermore, depending on the implementation, the corresponding value of the property can also be stored in the index. To speed up all possible queries, a new index would be needed for each used property.

When using an index based approach to speed up queries the same problems arise, as already discussed in the previous Section 4.1. When trying to cover all used properties in the document-oriented store by indices, for each property an index has to be created. Properties are not known in advance and creating an index for each new properties would lead to a unpredictably large amount of indices which would result in a large overhead. For example to store DBpedia the database would have to manage over 32,000 indices to cover all properties in the DBpedia dataset.

4.2.2 Recommendation computation

Due to the limitations regarding indices described in the previous section a live computation of recommendation based on the basic data is not possible in an acceptable amount of time. Therefore, a precomputation approach similar to the previously described approach in 4.1 has to be applied to compute recommendations efficiently.

The rule based approach explained in the previous section has to be adapted to the document-oriented model. The access in document-oriented databases is optimized to retrieve single documents . Thus, one single document has to contain as much information as possible. Storing only one rule per document would lead to a big amount of documents which have to be fetched to compute one single recommendation. Therefore, multiple rules have to be combined and stored in one document.

The recommendation computation explained in Section 3.4 fetches rules based on their heads. Thus, the rule grouping can be realized by the head as shown in Listing 4.9. The id/primary key of the document denotes the head of all rules that are stored in the document. The tails and the corresponding count-values are stored in the document as well. This storage model allows to fetch only a few documents to get all needed rules for computing one recommendation.

The number of documents that have to be fetched corresponds to the number of distinct properties that are already present on the current edited subject.

Listing 4.9: Representation of grouped rules in one MongoDB document

```
{
  "_id" : "name",
  "tails" : [ {
      "property": "mayor",
      "count": 100
    }, {
      "property": "website",
      "count": 50000
    }
  ]
}
```

After the retrieval of all required documents respectively all suitable rules, all rules have to be grouped and merged to the set of recommendation candidates. Subsequently, a ranking has to be applied to get the final set of recommendations.

For the realization of the recommendation computation, the aggregation framework of MongoDB (available since Version 2.2) is used as it provides a fast alternative to long running map/reduce queries and therefore, is best suited for the real-time computation of recommendations. It provides a pipelining interface to apply multiple aggregation and matching tasks in an ordered sequence. An example of a pipline in MongoDB which represents the basic recommendation computation of Listing 4.5 can be found in Listing 4.10.

Listing 4.10: MongoDB aggregation pipeline to compute recommendations

```
db.rules.aggregate( [
  { $match : { $in : ['prop1','prop2','prop3',...] },
  { $unwind : "$tails" },
  { $group :
    { _id : "$tails.property" , number : { $sum : "$tails.count" } }
  },
  { $match : { $nin : ['prop1','prop2','prop3',...] },
  { $sort: { number: -1 } },
  { $limit : 10 },
] )
```

Another big advantage of most NoSQL databases is the scalability they offer. Especially when dealing with big datasets this could be a big benefit which is explained in more detail in the following section.

4.2.3 Distribution and Sharding

The processing of large datasets which are created by mass-collaboration on a web-scale can be very demanding regarding the throughput of data. Especially the computation of up-to-date recommendations in an online collaboration system which is constantly changing is very expensive as all changes have to be considered to be able to react on trends and new information. If one single node is no longer able to handle the traffic and computation, a distribution of computation and data storage is needed. In this section, we describe how the SnoopyConcept can be used in a sharded and distributed environment.

Most of the state-of-the-art NoSQL databases provide such scalability features out of the box which is also a substantial reason for their success. MongoDB provides an automated distribution feature that distributes and replicates the stored data to several nodes while taking care of replication and redundancy. But not only the data distribution is done automatically, also the computation is distributed to speed up the execution. For example, a query using the MongoDB aggregation framework is automatically distributed in a sharded MongoDB cluster. Furthermore, MongoDB provides the possibility to use the MapReduce model to distribute computation. MapReduce was developed by Google [48] for complex computations or big data use cases that cannot be processed or stored on one single node.

Listing 4.11: MapReduce algorithm to count words [48]

```
1  map(String key, String value):
2    // key: document name
3    // value: document contents
4    foreach word w in value:
5      EmitIntermediate(w,"1");
6
7  reduce(String key, Iterator values):
8    // key: a word
9    // values: a list of counts
10   int result = 0;
11   foreach v in values:
12     result += ParseInt(v);
13   Emit(AsString(result));
```

The MapReduce paradigm can be led back to the functions map and reduce that are widely used in several functional programming languages such as Lisp. In data processing many operations can be split to a map function which is applied on all entries and outputs key-value pairs as a result. The key-value pairs subsequently are grouped by the key and sent to a a reduce function to gather the final result. Due to the principles of functional programming the map and reduce function itself do not have any global dependencies and can be executed independently. Those prerequisites provide the possibility to

execute all functions in parallel and distribute the execution to different nodes. To explain the MapReduce concept the example of the wordcount problem is commonly used. In Listing 4.11 the pseudo-code of a map and reduce function to count words in documents is shown.

The map function in the shown code example emits all occurrences of all words with a count of "1". The reduce function receives all key-value pairs and groups them by the received key. The value is added up to a final sum, the count value of every used word. Due to the independent scope of every function the execution of them can be distributed over many nodes. For example considering 1000 documents which are distributed over ten nodes. Every node can process 100 documents independently and execute the map function. Even on one node, documents can be processed in parallel on e.g. multiple cores. The final key/value pairs can be send to reduce functions on a separate node. To speed up the process we can also build a hierarchical reduce system which already builds a local sum on every node and finally sends over the sums to a centralized reduce function. Furthermore, the independence of all functions provides the possibility to easily adapt the process to an incremental approach by storing the states of all reduce functions. All those advantages, the flexibility and the simplicity of this model have lifted the MapReduce approach to a de facto standard of distributed data processing which is nowadays, implemented by many systems.

Listing 4.12: Map and Reduce function in MongoDB to create rules

```
map: function() {
  for (var head in this) {
    if (head !== '_id') {
      for (var tail in this) {
        if (tail !== '_id' && tail !== head) {
          emit(head,tail);
        }
      }
    }
  }
}

reduce: function(key, values) {
  var tailmap = {};
  for (var i=0; i < values.length; i++) {
    if (!tailmap[values[i]]) {
      tailmap[values[i]] = 1;
    } else {
      tailmap[values[i]] += 1;
    }
  }
  return {tails: tailmap};
}
```

Therefore, we decided to propose an implementation of the Snoopy algorithm using MapReduce, as it can be distributed without any additional changes.

In the following section, we discuss how the rule generation described in Section 3.4.1 can be mapped to a scalable MapReduce based approach.

The emit function collects all properties which are used on a specific subject and computes all permutations of properties to create the corresponding rules of one subject. The emit function sends one rule consisting of one head and tail property. The reduce-function collects all rules, groups them based on their heads and tails and counts the occurrences of the rules. After this reduce-procedure, the final set contains triples in the form of head→tail and a corresponding count value.

As described in the previous section, rules with the same head are stored in the same document in MongoDB. To map the reduced information to this schema, the reduce-function furthermore, groups all rules information based on the head of the rule. The MapReduce functions are shown in Listing 4.12. The generated rules can subsequently be used to compute the set of recommendations as shown in Listing 4.10.

The presented approach using MongoDB does not only represent an implementation based on a document-oriented store, but furthermore, demonstrates a generic distributed approach using MapReduce. This distributed approach is scalable, can be easily adapt to an incremental processing, and improves the performance when dealing with very large datasets.

4.3 NoSQL: Graph Stores

The SnoopyConcept uses a RDF based model to store information which can be represented as a graph. Therefore, it is obvious to also consider the usage of a graph based database to store the raw data. In this section, we discuss the implementation using a graph database. More information about graph database models can be found in Section 2.5.3. For the graph-based implementation we use the property graph model [141] which is supported by one of the leading graph database storage engines Neo4j[2]. Property graphs are attributed, labeled, directed multi-graphs which can be defined as follows:

- A set of vertices, where
  - each vertex has a unique identifier.

[2] http://www.neo4j.com, accessed 2017-07-17

  - each vertex has a set of outgoing edges.
  - each vertex has a set of incoming edges.
  - each vertex has a collection of properties defined by a map from key to value.

- A set of edges, where
  - each edge has a unique identifier.
  - each edge has an outgoing tail vertex.
  - each edge has an incoming head vertex.
  - each edge has a label that denotes the type of relationship between its two vertices.
  - each edge has a collection of properties defined by a map from key to value.

Furthermore, Neo4j provides the possibility to label vertices (nodes) and edges by tags which can be used to describe the type of nodes and edges. The expressiveness of the property model is not extended by labels as all information can also be stored by using properties. Although the modeling and querying is simplified by using labels. For example, nodes can be labeled as subject, property, or objects as shown in the presented implementation in the next sections.

4.3.1 Storage Model

The graph-based model of the Snoopy Concept combines the storage of facts and recommendation-related information in one graph. The proposed model of the SnoopyConcept is an adapted two-layer model which was introduced by Huang *et al.* in [81].

The adapted two-layer model in Figure 4.1 stores all property nodes in the upper layer and all subject nodes in the lower layer. Neo4j allows to label nodes which can be used to group or classify nodes and subsequently, can be used as a filter in queries to restrict the type of nodes. In our model, properties are labeled with the label `prop` and subjects are labeled with the label `sub`. The edges between the two layers represent usages of properties on the re-

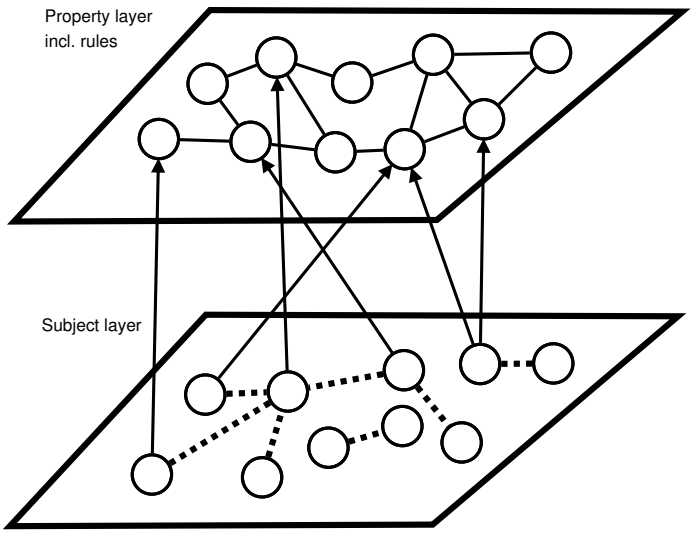


Figure 4.1: Adapted two-layer model according to Huang *et al.* [81] to store knowledge and rules

spective subjects and are modeled as relationships of the type `HAS_PROPERTY`. The value of the property is stored as an additional key value pair on the respective `HAS_PROPERTY` relationship. This data corresponds to the raw data of the knowledge graph of the SnoopyConcept. All relationships between nodes on one layer are related to recommendations and represent additional meta knowledge (*cf.* Section 4.3.2). The properties are connected by similarity edges respectively rule edges (relationship type:`USED`) with a count key-value pair which specifies the amount of co-occurrences. The dashed edges between subjects are currently not used by the SnoopyConcept but could be derived by the analysis of used properties to compute e.g., the similarity of subjects.

The resulting model is very flexible as one can directly access recommendation relevant information as well as data of the knowledge graph itself. The generation of rule edges are described in the following Section whereas the computation of recommendation based on this model is presented in Section 4.3.3.

4.3.2 Graph-based Rule Generation

One of the frequently stated advantages of graph based models is the direct access to data and the usage of relationships without consulting index structures or computing additional joins. However, the real-time computation of recommendations in a very large dataset is impossible within a reasonable timeframe. Consider the cypher query in Listing 4.13 which computes the

top-5 suitable properties of the subject `Innsbruck` based on the co-occurrence frequency. The computation of the query takes several hours and shows that a real time computation is not advisable.

Listing 4.13: Cypher query to compute co-occurrence on the fly

```
1 MATCH (start:sub)-[r1:HAS_PROPERTY]->(p1:prop)
2       <-[r2:HAS_PROPERTY]-(s:sub)-[r3:HAS_PROPERTY]->(p2:prop)
3 WHERE start.name = 'http://dbpedia.org/resource/Innsbruck'
4 WITH p2, count(p2) as frequency
5 ORDER BY frequency DESC
6 LIMIT 5
7 RETURN p2, frequency;
```

Therefore, the computation of recommendations has to be accelerated by pre-computing rules. The rule generation cannot be done by using one query in Neo4j as the intermediate results are too big and the optimizer is not able to cope with this problem. To solve this issue we split the rule computation to one computation per property which is shown in Listing 4.14.

Listing 4.14: Query to compute rules based on one given property

```
1 MATCH (p1:prop)<-[r1:HAS_PROPERTY]-(s:sub)
2 WHERE id(p1) = ${ID}
3 WITH DISTINCT s as s2,p1
4 MATCH (s2:sub)-[r2:HAS_PROPERTY]->(p2:prop)
5 WHERE id(p2) <> ${ID}
6 WITH p1,p2,count(DISTINCT s2) as freq
7 MERGE (p1)-[r:USED]-(p2)
8 ON CREATE SET r.count = freq
9 ON MATCH SET r.count = r.count + freq;
```

To compute all rules we fetch all properties and their ids in the first step. Subsequently, we execute the query shown in Listing 4.14 for every property which is given by the variable `${ID}`. The query creates a relationship called `USED` between two properties if they are used together on a subject. Furthermore, the relationship holds a counter which defines the number of subjects that contain both properties. Those relationships holds exactly the same information as all the rules presented in the relational approach in Section 4.1 and can be subsequently used to compute recommendations as described in the following section.

4.3.3 Graph-based Recommendation Algorithm

Due to the insufficient performance when computing recommendations on the fly as shown in Listing 4.13, the query presented in Listing 4.15 corresponds to the basic algorithm presented in Section 3.4.1 and is based on rules which are stored as `USED`-relationships in the graph store.

Listing 4.15: Compute recommendation based on USED-relationships

```
MATCH (head:prop)-[r:USED]-(tail:prop)
WHERE
head.name = 'http://dbpedia.org/property/areaTotalKm'
OR head.name = 'http://dbpedia.org/property/postalCode'
RETURN tail.name,count(r) as cx,sum(r.count) as cglobal
ORDER BY cx DESC, cglobal DESC LIMIT 10;

+-------------------------------------------------------------+
| tail.name                                    | cx | cglobal |
+-------------------------------------------------------------+
| "http://dbpedia.org/property/populationTotal" | 2  | 307308  |
| "http://dbpedia.org/property/populationAsOf"  | 2  | 306218  |
| "http://dbpedia.org/property/longd"           | 2  | 299958  |
| "http://dbpedia.org/property/latd"            | 2  | 299942  |
| "http://dbpedia.org/property/subdivisionType" | 2  | 292618  |
| "http://dbpedia.org/property/subdivisionName" | 2  | 287156  |
| "http://dbpedia.org/property/longew"          | 2  | 286244  |
| "http://dbpedia.org/property/latns"           | 2  | 286138  |
| "http://dbpedia.org/property/longm"           | 2  | 273188  |
| "http://dbpedia.org/property/latm"            | 2  | 273164  |
+-------------------------------------------------------------+
10 rows
290 ms
```

By defining the starting points respectively rule heads, the query fetches all connected properties or rule tails and ranks the result list as shown by the context count and global count value (*cf.* Section 3.4). The Listing consists of an example query and the corresponding result set of the query.

Listing 4.16: Graph based normalized hybrid (70:30) recommendation

```
MATCH p=(head:prop)-[r:USED]-(tail:prop)
WHERE
  head.name = 'http://dbpedia.org/property/locationCountry'
  OR head.name = 'http://dbpedia.org/property/name'
  OR head.name = 'http://dbpedia.org/property/logo'
  OR head.name = 'http://dbpedia.org/property/website'
WITH tail,count(r) as context,sum(r.count/tail.count) as confidence
WITH min(context) as mincontext, max(context) as maxcontext,
min(confidence) as minconfidence, max(confidence) as maxconfidence
MATCH (head:prop)-[r:USED]-(tail:prop)
WHERE
  head.name = 'http://dbpedia.org/property/locationCountry'
  OR head.name = 'http://dbpedia.org/property/name'
  OR head.name = 'http://dbpedia.org/property/logo'
  OR head.name = 'http://dbpedia.org/property/website'
RETURN
  tail.name,
  count(r) as context,
  sum(toFloat(r.count)/tail.count) as confidence,
  (count(r)-mincontext)/toFloat(maxcontext-mincontext) as contextnorm
      ,
  (sum(r.count/tail.count)-minconfidence)/toFloat(maxconfidence-
      minconfidence) as confidencenorm,
  ((count(r)-mincontext)/toFloat(maxcontext-mincontext) * (1 - ALPHA)
      ) +
  (ALPHA * (sum(r.count/tail.count)-minconfidence)/(maxconfidence-
      minconfidence)) as score
```

```
24 ORDER BY score DESC LIMIT 10;
25
26 +-------------------------------------------------------------------+
27 | tail                | cnxt | conf. | cnxtnorm | conf.norm | score |
28 +-------------------------------------------------------------------+
29 | patents             | 4    | 8.0   | 1.0      | 1.0       | 1.0   |
30 | trademarkedSlogans  | 4    | 8.0   | 1.0      | 1.0       | 1.0   |
31 | annualNetSales      | 4    | 8.0   | 1.0      | 1.0       | 1.0   |
32 | siteType            | 4    | 8.0   | 1.0      | 1.0       | 1.0   |
33 | developerWebsite    | 4    | 6.6   | 1.0      | 0.75      | 0.82  |
34 | ceasedTrading       | 4    | 6.0   | 1.0      | 0.75      | 0.82  |
35 | numberOfLocations   | 4    | 6.0   | 1.0      | 0.75      | 0.82  |
36 | helpline            | 4    | 5.0   | 1.0      | 0.62      | 0.73  |
37 | tradingAs           | 3    | 6.0   | 0.66     | 0.75      | 0.72  |
38 | keyDistributor      | 3    | 6.0   | 0.66     | 0.75      | 0.72  |
39 +-------------------------------------------------------------------+
40 10 rows
41 1248 ms
```

The context-sensitive algorithm which is described in Section 3.4.3 aims at finding the right balance between popular properties and most suitable properties regarding the context. In Listing 4.16 the extended algorithm (*cf.* Section 3.4) which incorporates the normalized context and confidence values by applying a weight `ALPHA` is represented as a Neo4j query (output shortened). The weighting factor `ALPHA` which can be defined in the range between 0 and 1 indicates if the context score or the confidence score has the higher influence on the final score. For example a weighting factor of 0.5 indicates a balanced weighting between both scoring values.

4.4 Storage Model Evaluation

In this Section we evaluate and discuss our previously proposed storage models in respect to performance. As the SnoopyConcept is compatible to all triple based storage systems (*cf.* Section 4) which are usually optimized for querying semi-structured data [56, 142, 120] we do not cover query facilities in this evaluation but focus on the evaluation of the previously presented computation of recommendations in the context of the SnoopyConcept. The evaluation covers the following three main steps in the process of computing recommendations:

- Import of raw data of the evaluation dataset
- Creation of rules
- Computation of recommendations

All three steps are examined for every presented storage model in this chapter. The used database systems and the according version are listed in Table 4.5. The evaluation was conducted using two Intel® Xeon® CPU E5530 @ 2.40GHz (8 cores, 16 threads in total), 192 GB main memory, RAID-5 5xSAS-HDD900 10k, and CentOS Linux release 7.3.1611. For the performance evaluation we imported a DBpedia dump which is also used for the evaluation in Chapter 6. The dataset consists of 59 million triples which results in 701 million rule instances. The detailed structure of the dataset is described in Section 6.1.1.

Storage Model	System	Version	Engine
Relational Model	MySQL	5.7.18	InnoDB
NoSQL / Document based	MongoDB	3.4.5	WiredTiger
Graph Model	Neo4j	3.2.0	Neo4j

Table 4.5: Specifications of Databases used in the Evaluation

Performance numbers regarding the data import can be found in Figure 4.3. Figure 4.2 depicts the database sizes after the import of the dataset. The import in MySQL was realized by piping a pregenerated SQL file including all insertion commands to the native client. Every insertion command triggers a stored procedure in MySQL which coordinates the import including id lookups and dictionary table management. For the MongoDB import the PHP mongodb client version 1.2.9-1 and PHP 7.1 was used. To decrease the import time, the import process used write tasks[3] consisting of 1000 bulk upsert commands each. In Neo4j the native csv import command[4] was used to import a pregenerated csv file containing all triples. The periodic commit value was set to 10,000 to increase the import performance.

After the successful import, the rule generation for the individual systems were executed. The rule generation in MySQL was implemented by using a procedure according to the approach described in Section 4.1.5. For MongoDB, a map-reduce job was created to generate all rules (*cf.* Section 4.2.3). The map-reduce approach provides the possibility to scale the rule generation process and distribute the workload to multiple nodes. In this evaluation the map-reduce job was not distributed and was executed on one single node only. The rule generation in Neo4j was realized using a bash script which sends a generation query for every property in the system as described in Section 4.3.2. The time consumption for the rule generation can be found in Figure 4.3.

[3] https://docs.mongodb.com/manual/reference/method/db.collection.bulkWrite/, accessed 2017-07-17

[4] http://neo4j.com/docs/developer-manual/current/cypher/clauses/load-csv/, accessed 2017-07-17

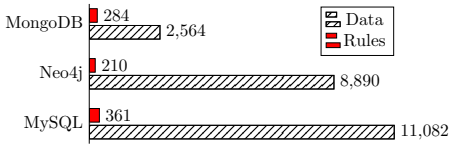


Figure 4.2: Memory (MB)

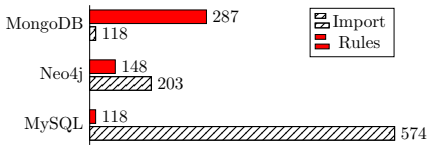


Figure 4.3: Import (minutes)

MySQL takes 574 minutes for importing the dataset and therefore, is the slowest database performing the import task. This can be led back to the manually implemented dictionary approach (*cf.* Section 4.1.3) which perform several lookups with every inserted triple, e.g., for subject, property, object, language, type, etc. MongoDB and Neo4j receive all strings without any preprocessing and therefore, perform similar for the import, i.e., 118 minutes for MongoDB and 203 minutes for Neo4j. Nevertheless, both systems also aim at compressing data but apply the compression natively and therefore, more efficiently.

Neo4j aims at reducing disk usage by applying an automated dictionary compression approach[5] similar to the dictionary approach we introduced in Section 4.1.3) for the relational model. The Neo4J compression algorithm tries to predict if it is more efficient to store a value inline or in a separated string storage by applying length based rules. For the used DBpedia dataset, the Neo4j string store consumes 2.59 GB of disk space which corresponds to the disk usage of the dictionary tables in MySQL.

MongoDB has the lowest disk space consumption of less than 3 GB which can be led back to the compression[6] that is applied by the WiredTiger storage engine. By default, WiredTiger uses block compression with the snappy compression library and prefix compression for all indexes. The imported DBpedia dataset contains mainly URIs using the same namespaces which leads to a very efficient compression.

MySQL stores the id based triple data in 3.2GB and allocates another 2.3GB for the lookup tables. To speed up queries to reason on the raw data, we proposed five additional indexes besides the primary key spo index as described in Section 4.1.3. Every additional index on the main triple table requires about 1GB which results in 5.5GB additional disk space. In total MySQL consumes 11,082 GB for all data and additional index structures. For the computation of recommendations those additional indexes are not used.

[5] https://neo4j.com/docs/operations-manual/current/performance/property-compression/, accessed 2017-07-17

[6] https://docs.mongodb.com/manual/core/wiredtiger/#compression, accessed 2017-07-17

In comparison to the raw data, the rules are very compact in all database systems. This compact size has an impact on the performance of recommendation computation as it is mainly based on rules. MySQL takes the highest amount of 361 MB for storing the rules. Nevertheless, this amount of data can be easily hold in main memory and thus, provide the required data efficiently to the recommender algorithm.

It is also important to mention, that all databases use just one CPU core for the import. For the rule generation MongoDB could be scaled by using a cluster setup of several instances on the same machine. This approach would speed up the rule generation drastically as the map reduce approach scales with the number of nodes in the cluster. MySQL and Neo4j scale on a query basis which results in a parallel execution of queries sent by multiple clients. As the import process is single threaded and thus, only one client is connected to the databases, MySQL nor Neo4j can use their parallelization strategy for the import or rule generation.

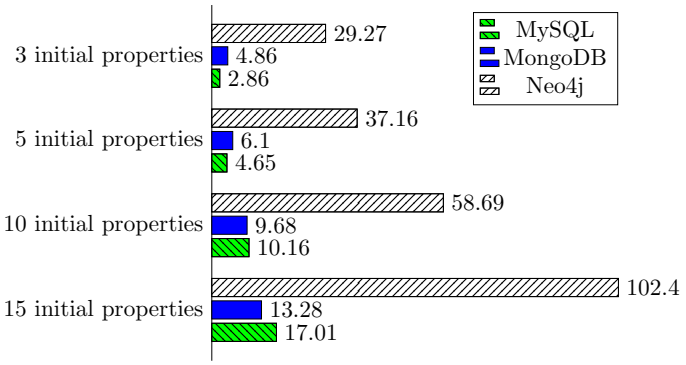


Figure 4.4: Computing top-10 recommendations (time in milliseconds)

For the computation of recommendations, Figure 4.4 shows the performance of MySQL, MongoDB, and Neo4j. The presented time is needed for the computation of top-10 recommendations for an input of 3,5,10, and 15 initial properties. The used algorithm corresponds to the algorithm presented in Section 3.4.3 which incorporates the confidence and context. For this evaluation, we used the best performing ranking strategy "context_confidence" (*cf.* Section 17) which applies an `order by context,confidence`. The evaluation was performed by sending a pregenerated set of 1,000 recommendation calls to the database systems using the native database clients. To be able to compare the systems, the random properties are pregenerated and re-used for every database system. Thus, every database receives the same starting properties and therefore, computes exactly the same recommendations. The measured time includes the creation of one connection to the local server, the transfer of 1,000 queries incl. initial properties to the server, the execution

of 1,000 queries and the transfer of the computed recommendation set back to the client. All evaluations were executed six times as the first run aims at warming the caches and five other runs were used to calculate the overall average time to compute a recommendation set.

In general, it can be seen in Figure 4.4 that every database system is able to compute recommendation below 100ms. Even when adding the network latency to the client and the time to render the user interface, the performance is sufficient to provide structure recommendations, e.g., additional properties. For recommendations which already guide the user during typing, such as auto-completion, the requirements are more strict and the latency should stay below the limit of 100ms in total for having the user feel that the system is reacting instantaneously [124, 35, 116].

MongoDB and MySQL are able to meet this criteria by providing ten recommendation in less than 18ms. The fastest computation based on three initial properties can be achieved by MySQL in less than three milliseconds. If more properties are initially present, MongoDB is able to provide recommendation faster than MySQL. This can be led back to the bigger intermediate results which slow down the JOIN performance in MySQL. The reason that Neo4j computes recommendations six times slower than Mongodb and ten times slower than MySQL is that Neo4j does not stored the rules in a clustered memory layout. Nevertheless, Neo4j is more flexible when querying the raw data as the graph-structure can be directly exploited [13].

Furthermore, we also compared the MySQL performance of the simple ordering, and the normalized and weighted ranking as presented in Section 3.4.3 for ten properties. The complex ranking takes 15.98ms on average in comparison to 10.16ms for the simple ranking, i.e., an overhead of 50% which does not lead to better results as shown in Section 17.

Due to the evaluated database performance and the presented results, we recommend to use MySQL or MongoDB when computing recommendations for real time interaction with the user. For a more holistic approach and other use cases requiring complex graph queries, Neo4j might be better suited.

4.5 Storage Models Summary

In this chapter, we presented the implementation of the SnoopyConcept using three different storage models, namely the relational model, document-oriented model and the graph model. We also discussed the efficient rule based computation of recommendations in all models and showed that the relational

model provides the best performance with our dataset in terms of recommendation computation, i.e., is able to compute top-10 recommendations in less than 3ms.

CHAPTER 5

SnoopyConcept Showcases

In this chapter, we present multiple showcases which we developed to prove that the SnoopyConcept and its algorithms can be applied to different domain to improve quality and quantity of knowledge in information systems. The presented prototypes were also used in the evaluation of the SnoopyConcept which is described in detail in Chapter 6.

5.1 First Reference Implementation: SnoopyDB

The described SnoopyConcept was implemented in a first prototype which is called "SnoopyDB". A screenshot of the SnoopyDB prototype can be seen in Figure 5.1. This figure shows the screen of a user entering information about the University of Innsbruck. The user already entered four property-value pairs about the foundation year, the founder, the number of professors and the official website of the University of Innsbruck. The three additional rows displayed in grey font mark the properties which are recommended by the system. The value fields corresponding to these properties already contain exemplary values. This way, the user can immediately recognize the type of the value, e.g. a numeric value for employees. The box on the right side of the screenshot contains further suitable properties for the current subject. These properties can easily be added to the input form by clicking on the arrow icon.

The screenshot also shows how the system automatically detects the data type of the entered information. In line 4, a link for the website of the university is created. The system automatically detected that a big percentage of the values belonging to the property website were stored as links and therefore, the system suggests the insertion of a link to the user. In this case, the user accepted this suggestion and entered the URL of the official website of the University of Innsbruck.

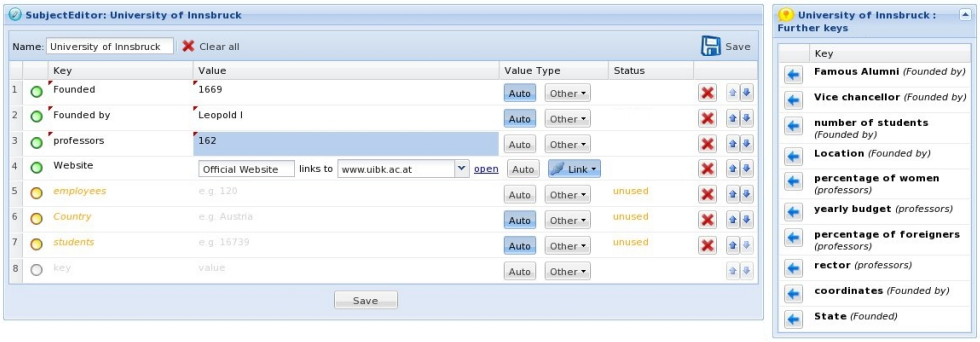


Figure 5.1: Screenshot of the SnoopyDB Prototype

SnoopyDB was the first reference implementation in 2009 and implements the algorithm (without user modeling) presented in Section 3.3. The prototype was used to perform the online evaluation which is presented in Section 6.1.3 and was published in [61].

5.2 SnoopyTagging

Another use case of the SnoopyConcept can be found in the area of *tagging*. The so-called SnoopyTagging [63, 34] prototype covers the field of collaborative online tagging which has encountered a tremendous success in the last decade. The tagging paradigm enables users to annotate online resources such as images or bookmarks with keywords aiming at creating a categorization to simplify the search and retrieval of resources. Especially in large collaborative systems, the definition of a fixed set of categories in advance is not feasible and would limit the flexibility of the system. Hence, tagging is very import to bring some kind of order to the possibly existing chaos. One major feature of tags is the simple usage as tags may be chosen freely without any restrictions. Thus, tags can be used to describe different aspects of online resources by the users. Consider the example of an image tagged with “Robert Capa”. By just considering the tag, it is not clear whether Robert Capa is pictured on the photo, whether Robert Capa is the photographer or the picture shows

Robert Capa's house. SnoopyTagging aims at solving this problem by adding contextual meta-information to tags while at the same time increasing the homogeneity of the resulting tag-vocabulary by facilitating a recommender system. As a showcase example, we implemented our approach on top of Flickr which is described in the following sections.

5.2.1 Structured Tags

The SnoopyTagging Concept is based on the SnoopyConcept and introduces Structured Tags. The SnoopyConcept based self-learning recommendation engine aims at decreasing the proliferation of tags and homogenize the tag vocabulary. The paradigm of Structured or Contextualized Tags, which was first introduced as "The Poor Man's RDF" [11], represents the basis for the SnoopyTagging system. Structured Tags consist of two parts which are divided by a delimiter sign (a colon in our case): `context:tag`. The first part of a Structured Tag defines the context of the actual tag, whereas the actual tag is specified after the delimiter sign. As most of today's tagging platforms allow colons in tags, the backward compatibility of SnoopyTagging is guaranteed. Structured Tags enable the user to provide tags with context, e.g. the tag `photographer:Robert Capa` expresses that Robert Capa is the photographer of a certain photo.
The disadvantage of freely chosen tags is the increasing latent heterogeneity of the resulting folksonomies (*cf.* Section 2.4.1). This fact is crucial in online mass-collaboration tagging systems, as there are thousands of different users from different social levels and backgrounds who add tags in different domains and settings. Consider e.g., the tags `takenBy:Robert Capa` and `photographer:Robert Capa` which are differentiated during the search process although they are semantically equivalent. We tackle this problem by applying the SnoopyConcept and creating a recommender engine which aims at providing suitable tags and contexts to homogenize the used vocabulary and avoid the use of synonyms.

5.2.2 Recommendations based on the SnoopyConcept

In order to create and maintain a homogeneous set of both contexts and tags, we make use of our developed Snoopy approach (see Chapter 3) and implemented *SnoopyTagging* (see Screenshot Figure 5.2), a first prototype based on the Flickr platform. The Snoopy approach basically aims at dealing with the problem of heterogeneous vocabularies by providing the users with suitable recommendations. SnoopyTagging includes recommendations for both the context and the actual tag of a Structured Tag. As these recommenda-

tions are computed based on all tags which have previously been entered by users, recommendations are strongly tied to the community and its vocabulary. We distinguish between two types of recommendations:

(i) Additional Structured Tags are recommended based on co-occurrence analysis (see Section 3.3) of the already entered tags and leads to the suggestion of further applicable contexts. If e.g., the tag `photographer:Robert Capa` was already specified, the contexts `camera:?` and `location:?` may be recommended to the user during the tag insertion process. Hence, this type of recommendations aims at encouraging the user to (a) use Structured Tags and (b) enter more (meta-) information.

(ii) The extended auto-completion feature recommends context and tags during the typing of the user. This intelligent feature recommends the re-usage of contexts and tags. Thus, it avoids the insertion of additional synonyms (see Section 3.3.2) and leads to a more homogeneous vocabulary. E.g., if `loc` has been entered, the key `location` and according values, e.g., `Vienna` are recommended.

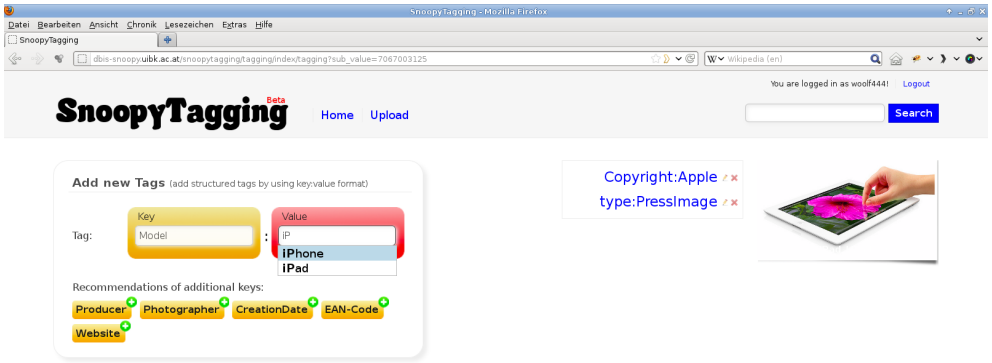


Figure 5.2: Prototype SnoopyTagging

The SnoopyTagging approach incorporates the user co-occurrence space to personalize both types of recommendations as described in Section 3.5. Once a user entered a context a, the system computes those tags co-occurring with the context a within both the set of global tags (of all users) and user-specific tags (tags used by the current user). As only the top-5 recommendations are shown in the interface, a ranking function is applied. The ranking is based on the co-occurrence rate of the already entered tags on the respective resource and the remaining tags within the datasets (ranking of the global set: $rank_{global}$, ranking of the user-specific set: $rank_{user}$). For the final set of recommendations which is based on both previously computed sets (user and global co-occurrence), we propose to use a hybrid ranking algorithm $rank = \gamma \cdot rank_{global} + (1 - \gamma) \cdot rank_{user}$ with the factor γ which defines the weight

of the final rank between the user-specific and global set of recommendations (*cf.* Section 3.5.2). In our experiments a value of $\gamma = 0.1$ turned out to be beneficial.

We published the SnoopyTagging approach in 2012 [63] including an evaluation which shows that the implemented prototype based on Flickr adds context to simple tags and homogenizes the tag vocabulary. The conducted evaluation (*cf.* Section 6.2 showed the acceptance of the concept of contexts by the users. Our experiments also showed the frequent re-usage of tags and the creation of a homogeneous tagging vocabulary due to the strong acceptance of the underlying SnoopyConcept based recommendation engine.

5.3 hash5

The previously described SnoopyDB prototype has been designed to store facts and information in a structured format while the SnoopyTagging approach aims at annotating already existing resources by meta data. The use case explained in this section combines the two approaches by providing a way to store unstructured notes and structured meta information. This use case is located in the area of personal information manager (PIM) systems [174] and was covered by a prototype called hash5 which integrates the SnoopyConcept.

5.3.1 Main Concepts

The main purpose of a PIM system is to store and manage personal information, such as notes, addresses, todos and calendar dates. hash5 provides the possibility to store textual entries and add semi-structured meta information to each entry. The meta information can be added by using a novel hashtag model which extends the classical hashtag model which is used for example by Twitter and Facebook.

The classical hashtag model was established in Twitter by its users. Twitter allows textual entries only and doesn't provide methods to categorize tweets. Due to this limitation, Twitter users introduced the idea to prefix tag or category information by the hash sign [27]. This simple concept did not need any further support by Twitter, is easy to understand by all users and provides a great flexibility as the vocabulary is not restricted. This hashtag concept has been become very popular and is used by many platforms. Even Facebook

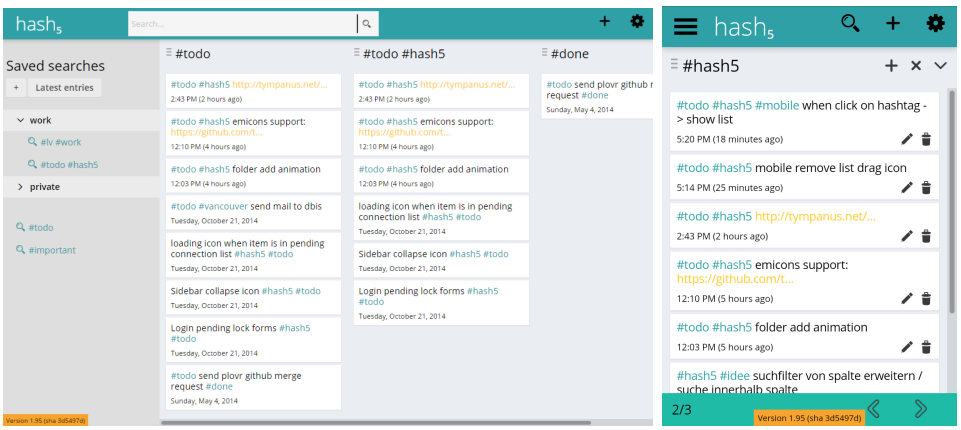


Figure 5.3: Dashboard view of hash5 (desktop/mobile screen)

supports hashtags since 2013 [1] to provide a flexible additional categorization facility of Facebook messages. hash5 has taken up this concept of hashtags and lifted the concept to a two-dimensional tagging model. The classical hashtag model allows only to specify one word respectively one dimension. It is not possible to specify a context. Consider the example of the hashtag `#2013-01-01_10am` which is attached to a textual calendar entry of a meeting. In this example it is not clear if the date corresponds to the start of the meeting, the end of the meeting, the time of the reminder or the creation timestamp of the entry itself. This ambiguity can only be resolved by adding a context which is not possible by using the classical one dimensional hashtag model. The idea of adding context to tags was already proposed in the article "The Poor Man's RDF?" by Buzz Andersen in 2005 [11] and is also used in the SnoopyTagging approach presented in Section 5.2. Andersen proposed to enhance tagging systems by introducing a split symbol which splits up the tag to a context on the left hand side of the splitting symbol and the tag value itself on the right hand side. Consider again the example, this approach would lead to the hashtag `#start:2013-01-01_10am` which indicates the start of the meeting at 10 am.

Figure 5.3 shows the responsive dashboard of hash5 which gives an overview of all notes and todos. One can already recognize the huge amount of hashtags which are used to categorize, set the status of todos or specify additional structured meta-data such as the start and end time of an appointment. Also, the columns are managed by hashtags and correspond to search queries which contain hashtags.

[1] http://www.theguardian.com/technology/2013/jun/13/facebook-to-introduce-clickable-hashtags, accessed 2017-07-17

By relying on hashtags that heavily and furthermore, introducing another dimension to tags—the context of a tag—the possible tagging-space grows tremendously which leads even in a quite small scope of private notes to proliferation of tags as described in Section 2.4.1. hash5 applies the SnoopyConcept to prevent the proliferation which is described in the following section in more detail.

5.3.2 Recommendations based on the SnoopyConcept

To simplify the usage of those structured tags and to cope with the proliferation of tags, recommendations support the user during the insertion of new chunks of information to the hash5 system. The hash5 system implements the SnoopyConcept and furthermore, provides additional recommendation mechanisms. hash5 recommends and auto-complete keys and values and incorporate the following information for the ranking and recommendation algorithm.

- Already used tags in the current entry
- Key of the current tag when recommending values
- Full-text of the current entry
- Popularity of keys and values

Besides the already known types of recommendations of the SnoopyConcept which are mainly based on the usage respectively co-occurrence of tags and are provided in hash5, the incorporation of the full-text of an entry to improve the recommendation system is very promising. Especially in the area of PIM systems which often contain similar information, the correlation of text and tags can be exploited. Consider an invite to a recurring meeting with a customer. The invite usually contains information about the company, title, agenda, attendees, signatures and information about the meeting point. This information can be used to suggest hashtags which were already used in a previous meeting with the same or similar customer. By using those recommendations, e.g., the process of classifying entries in the PIM system can be accelerated and improve the user experience. A screenshot of the edit window which also contains hashtag recommendations below the entry field is shown in the right screenshot of Figure 5.4.

The recommendation system is also integrated in the search process as shown in the left screenshot of Figure 5.4. By suggesting and auto-completing hashtags during the search process, the exploration of a potentially huge amount

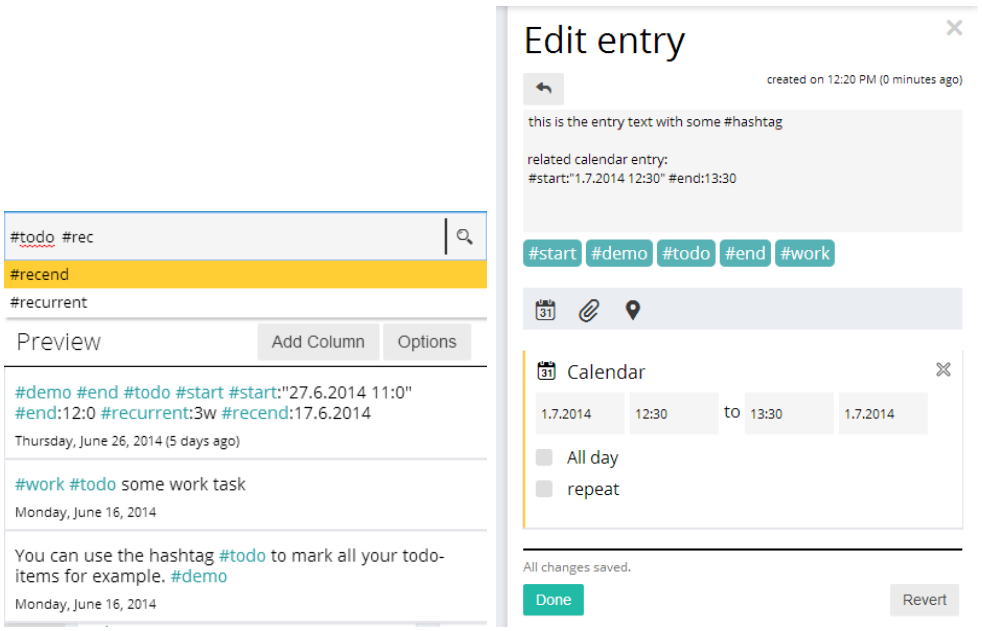


Figure 5.4: Search and edit an entry in hash5

of entries in a PIM is optimized. Every search can also be saved and shown as a dynamic column in the dashboard as shown in Figure 5.4.

The hash5 project was developed and improved by the authors of the following Bachelor theses which contain more information about the technical realization of several clients and the corresponding backend systems.

- M. Schmakeit. *Extension and Optimization of the Hash5-Server - a Hashtag-based PIM-System.* Bachelor's Thesis, University of Innsbruck, Jan. 2015. supervised by Eva Zangerle, Wolfgang Gassler, Günther Specht

- C. Esswein. *Development of a responsive webapp for a hashtag based PIM system.* Bachelor's Thesis, University of Innsbruck, Nov. 2014. supervised by Wolfgang Gassler, Eva Zangerle, Günther Specht

- C. Laqua. *Development of a responsive Android client for a hashtag based PIM system.* Bachelor's Thesis, University of Innsbruck, Nov. 2014. supervised by Eva Zangerle, Wolfgang Gassler, Günther Specht

- M. Wolf. *Extension of Information Types for an Android Client for a Hashtag-based PIM System.* Bachelor's Thesis, University of Innsbruck,

Oct. 2014. supervised by Eva Zangerle, Wolfgang Gassler, Günther Specht

- T. Fraydenegg. *Development of a light-weight HTML5 Web Client for an Information System adapting Social-Media Concepts.* Bachelor's Thesis, University of Innsbruck, June 2013. supervised by Wolfgang Gassler, Eva Zangerle, Günther Specht

- R. Bierbauer. *Development of a light-weight web client for a hashtag based PIM system.* Bachelor's Thesis, University of Innsbruck, Nov. 2013. supervised by Wolfgang Gassler, Eva Zangerle, Günther Specht

- M. Müller. *Development of a self-learning recommendation module for a hashtag based PIM system.* Bachelor's Thesis, University of Innsbruck, Nov. 2013. supervised by Wolfgang Gassler, Eva Zangerle, Günther Specht

- D. Hoppe. *Social-media concepts in personal information management systems.* Bachelor's Thesis, University of Innsbruck, June 2012. supervised by Wolfgang Gassler and Eva Zangerle

5.4 Wikidata

The Wikidata platform was introduced in 2012 and since then, has evolved to a central, consistent and structured knowledge base, which is maintained by an active community. The knowledge captured in Wikidata is semi-structured and is stored using a very similar format to the SnoopyConcept storage format. The main goal of Wikidata is to provide facts which are human and machine readable and can be used across different languages as stored facts are not tied to a single language [170]. The data provided is facilitated by an increasing number of applications with Wikipedia being the most popular one. As of May 2017, Wikidata features more than 26 million data items, which in principle "represent all the things in human knowledge, including topics, concepts, and objects"[2]. E.g., Albert Einstein or the city of New York are such items. On Wikidata, these items are described by so-called statements, where a property serves as a descriptor for a value representing the actual information. E.g., a property of the item Albert Einstein is dateOfBirth and the according value is 1879-03-14. Wikidata items and the according statements have been curated

[2] https://www.wikidata.org/wiki/Help:Items, accessed 2017-07-17

by approximately 484 million edits[3]. On average, each item on the platform is shaped by 18 edits performed by either by humans or bots.

As of 2014, 90% of all edits were made by bots as the Wikidata project strives to automate tasks and outsources these to bots [170]. However, still roughly one million edits are performed by human users every month and hence, contributions by human users play an important role. To improve this manual process of inserting new facts, the Wikidata platform supports its committed community by a so-called "property suggestor" which was introduced in 2014. This tool assists the user in entering information by providing suggestions for novel properties which are likely to be added to the current data item and is described in the following section.

5.4.1 Wikidata Property Suggestor

The Wikidata platform provides users with a property suggestor which aims at assisting users when adding new statements to a given Wikidata item. The approach taken by the Entity Suggester is based on the Predicate Suggestion approach by Abedjan and Naumann which aiming to enrich RDF datasets by a set of rule mining approaches [3, 2]. This approach is also inspired by traditional Association Rules [9]. Abedjan and Naumann claim that once RDF triples are grouped by their subject, traditional association rules can be facilitated to provide predicate suggestions to the user. Based on the previously inserted predicates for a given subject, this approach computes predicates to be suggested based on all rules that incorporate these predicates as heads of their rules. Consequently, the predicates appearing in the consequences of those rules are extracted and form the set of possibly suggested predicates. These candidate predicates are subsequently ranked by the sum of the confidence values of all rules that actually have the corresponding predicate as their consequence.

The Wikidata Property Suggestor further adds so-called "classifying" properties, which are the properties "instanceOf" and "subclassOf". The property instanceOf refers to the fact that the described item "is a specific example and a member of that class"[4], whereas the subclassOf property signifies that all instances of these items are instances of the given superclass[5]. For the recommendation approach, these properties are treated differently as for these, not only the property is used as a head of rules, but the combination of the prop-

[3] https://www.wikidata.org/wiki/Special:Statistics, accessed 2017-07-17

[4] https://www.wikidata.org/wiki/Property:P31, accessed 2017-07-17

[5] https://www.wikidata.org/wiki/Property:P279, accessed 2017-07-17

erty and each occurring object. I.e., $(instanceOf, human) \rightarrow dateOfBirth$ rules are formed and added to the set of rules. This allows to add further information to the recommendation computation process as information about the type of the given data item can be inferred and exploited (if available). Naturally, such information is valuable for the recommendation of suitable properties.

As this concept heavily rely on manual classification using an previously defined classification system we analyzed in [184] if the manual classification is beneficial and if the algorithm can be furthermore improved by applying additional elements of the SnoopyConcept. The combination of the algorithms are described in the following section.

5.4.2 Incorporating the SnoopyConcept Algorithm

The SnoopyConcept recommendation approach is also based on association rules but furthermore incorporates the context to improve the ranking of recommendations. In this case, the notion of context refers to the rules' head as these describe the context a given property is embedded in (*cf.* Section 3.4). The more such information about a property is available, the broader the foundation for a recommendation of the property. Therefore, we utilize the number of different contexts a rule is embedded in for the ranking computation. I.e., we count the number of distinct rules that actually have a given property as its consequence. For this context-based approach, we rank properties by the number of distinct rules leading to the property. I.e., the higher the number of distinct rules with a given property in the consequence, the higher the rank of the property. In case of a rank tie, we further rely on the number of total occurrences of the particular rule to resolve the tie.

We propose to create hybrid variants of the three recommender approaches presented previously as we identified one distinctive characteristic for each of the presented approaches: (i) using the sum of the rule confidences of all applicable rules for ranking in case of the algorithm by Abedjan and Naumann [3, 2] (**AN**) (ii) utilizing classifying properties to add type information in the case of the Wikidata-approach (**WD**) and (iii) the use of context–information for ranking of property candidates in case of the SnoopyConcept approach (**SN**).

We evaluated each of the approaches extended by the distinctive characteristics of the other two recommendation algorithms presented. This includes combining the SN_simple approach with classified properties as described in Section 5.4.1 (*SN_classified*), as well as adding context to perform the ranking

in this hybrid configuration (*SN_context_classified*). A second stream of hybrid configurations results from taking the Predicate Suggestion approach as described in the previous section and combining it with Snoopy's contextual ranking approach (*AN_context*) and lastly, we also evaluated the extension of the WD approach with Snoopy's context ranking approach (*WD_context*).

The evaluation was focused on the Wikidata use case and therefore, a Wikidata dataset for the evaluation was used. The evaluation results which are presented in detail in Section 6.1.2 show that in the context of Wikidata the Wikidata Entity Suggester (WD) works better than the other presented approaches. In the course of our analyses, we identify two key aspects which are essential for the quality of recommendations: incorporating classifying properties and making use of contextual information for ranking the property recommendation candidates. Combining the current Wikidata Entity Suggester approach with Snoopy's ranking strategy, which facilitates contextual information, significantly increases the performance of the current Wikidata recommender approach.

5.5 Summary

In this chapter it was shown that the SnoopyConcept can be used in various fields to improve the insertion of data. The basic idea of the SnoopyConcept is to incorporate the user during the insertion of knowledge to a traditional information system such as Wikipedia. This was shown by the SnoopyConcept reference implementation called SnoopyDB (*cf.* Section 5.1). The extension of the basic SnoopyConcept algorithm by user modeling was demonstrated by SnoopyTagging (*cf.* 5.2). Furthermore, SnoopyTagging proves that the SnoopyConcept is very flexible as it is compatible to common tagging systems and can be easily implemented and enhance classic tagging approaches. hash5 which is explained in Section 5.3 demonstrates the usage of the SnoopyConcept in a completely different field. PIM systems have other requirements but nevertheless, can benefit from the implementation of the SnoopyConcept. Since the first introduction of the SnoopyConcept in 2010 [61, 182] very similar approaches were presented by other authors. An implementation by Wikidata which conforms to the SnoopyConcept is discussed in Section 5.4 proves that the approach of guiding the user during the insertion of knowledge has emerged to mainstream.

The evaluation of all presented SnoopyConcept approaches, showcases and implementations can be found in the following Chapter 6.

CHAPTER 6

Evaluation

In this chapter all SnoopyConcept algorithms are evaluated under different aspects. In the first section we evaluate the core recommendation algorithms by conducting offline and online evaluations. Furthermore, we compare the SnoopyConcept with the Wikidata's Property Suggestor and evaluate the proposed context extensions for it. Our conducted user-centric experiments presented in Section 6.1.3 will shed more light on the user acceptance of the SnoopyConcept approach. In the last section we evaluate the personalized SnoopyConcept algorithm using the SnoopyTagging prototype to show if personalization is able to increase the accuracy of the SnoopyConcept.

6.1 Recommendation Algorithms

In this section, the recommendation algorithms which build the basis of the SnoopyConcept are evaluated. Two major test approaches were chosen to verify the recommendation algorithm and prove that the computed recommendations are suitable [151, 75]. The first evaluation section comprises an offline evaluation which evaluates the performance of the algorithms on a large mass-collaboratively created dataset which is based on Wikipedia. In the second section we compare the SnoopyConcept with the Wikidata's Property Suggestor by conducting an offline evaluation using the Wikidata dataset.

We discuss differences and assess the accuracy of our proposed context extension of the Property Suggestor described in Section 5.4. The second test approach evaluates the SnoopyConcept respectively the recommendation algorithms by conducting an online evaluation using the SnoopyDB prototype. This test incorporates real users and assesses the SnoopyConcept in a real-life environment to show if recommendations are accepted by the users and the quality and quantity of information in the system can be increased. In the last section we evaluate the personalized SnoopyConcept algorithm based on the SnoopyTagging prototype presented in Section 5.2. We determine if the personalization approach is able to increase the accuracy of recommendations and support the user during the insertion process.

6.1.1 Offline Evaluation

The idea of offline evaluations is to evaluate an algorithm or system by using a pre-existing dataset and simulate the user behavior based on this dataset. For example, in the domain of recommender systems, datasets including user ratings are often used to verify the performance of a recommendation algorithm [151, 75]. Therefore, the dataset is split into a training and a test set. The performance of an algorithm is measured by using the training set as a basis for the recommendation algorithm. The computed recommendations are subsequently compared with the test set to assess their performance.

The usefulness of offline evaluations is mainly dependent on the underlying dataset which has to be as similar as possible to the real environment in which the recommendation algorithm will be deployed. To evaluate the SnoopyConcept algorithm, a dataset of a semi-structured information system is needed. As described in Section 2.1, there are two major types of information systems. The dataset of a strictly structured information system would be useless as there is no freedom regarding the used structures in the system and therefore, is not suitable to verify an algorithm which recommends structure. On the other side, an information system that does not limit the used structures in any way is not suitable to verify the recommendations of structures as the recommendations are designed to homogenize the structures in such a non-limited information system. Furthermore, to receive significant results, the dataset must be of sufficient data size. Due to these constraints, it is difficult to find a suitable dataset which conforms to all described constraints. As to the best of our knowledge there does not exists any system which corresponds to the described SnoopyConcept and thus, there is no dataset which conforms to all described requirements above.

Dataset

Due to the above described limitations we decided to base our offline evaluation on the DBpedia dataset [14]. It is collaboratively curated, adheres to a flexible schema maintained by the community and therefore, has similar characteristics to a dataset which would have been created by the SnoopyConcept. DBpedia contains triple-based facts in the form of subject, property, value which are extracted from Wikipedia's infoboxes (*cf.* Section 2.1.2). Infoboxes contain manually aggregated information about an article and are represented in a tabular format on Wikipedia pages. The structure of such infoboxes has to adhere to predefined infobox templates which are constantly changed and adapted by the committed Wikipedia community. Despite this manual maintenance, these schemata of infoboxes are not completely homogeneous and feature a noisy collaborative style [179]. For example, synonyms are used for the same type of property and different templates are used for the same type of article. This noisy data, which has been grown with the Wikipedia system and is partly homogenized by the community, can be compared to recommendation based evolved schemata which would be present in a system which implements the SnoopyConcept and its algorithms. All these similarities to datasets of a semi-structured and guided information systems and also the very large size of the DBpedia dataset confirm the suitability for our experiments.

For the evaluation of this approach, we used DBpedia version 3.9 (released on September 17, 2013) [14] containing 4.0 million things (Wikipedia articles) and 70 million triples (infobox statements only) which were extracted from the English Wikipedia. After our cleaning (e.g., invalid characters) and import procedure, we stored 3,024,427 distinct subjects and 59,058,009 triples in our database. We randomly chose 261,808 subjects (9%) out of 3 million instances for our test set. Based on the remaining set of 2.7 million instances (50 million triples), we computed 700 million rule instances (*cf.* Algorithm 1 in Section 3.4.1). The compacted rule set of 6.5 million distinct rules provided a basis for all further recommendation computations. All details about the dataset and number of rules are summarized in Table 6.1.

Experiment Description & Metrics

In this offline evaluation, we simulate the behavior of a "snoopyfied" information system based on the underlying DBpedia dataset. To achieve this, we conducted a leave-one-out test [45] and used the training and test set as previously described. To simulate the user's behavior, we performed a reconstruction process which tries to predict previously removed properties of subjects in the test set as described below.

Property	Count
Subjects	3,024,427
Distinct properties	49,868
Distinct objects	12,888,228
Triples (total)	59,058,009
Triples in test set	5,895,613
Rule instances	701,645,970
Rules (compacted)	6,457,426

Table 6.1: Properties of the DBpedia dataset after clearing and import

Before we start the reconstruction process of a single subject, we randomly choose three facts (property-value pairs) of the subject and remove all other properties from the subject. Those removed properties serve as our ground truth. The state when only three properties of the subject are present simulates the state after the insertion of some basic properties during the insertion process of a new subject by the user. Thus, our hypothesis is that three properties are already sufficient to reconstruct a subject. Based on these three initial properties, the reconstruction process is started by executing the recommendation algorithm as described in Section 3.4. Subsequently, the resulting top-n recommended properties are analyzed and compared with the set of removed properties. If there is no matching property in the list of recommendations, the reconstruction process is stopped. If the recommendation computation returns at least one suitable property, i.e., is a member of the removed property set, the first matched property is accepted and thus, removed from the removed property set and added to the current properties of the subject. This process simulates a user who is accepting a recommended and suitable property and adding it to the subject. In this case, the recommendation was successful and therefore, the reconstruction process is continued by recomputing the recommendations. With the next recommendation computation iteration the previously added property is already taken into account as the algorithm takes as input the updated and extended property set of the subject.

This recommendation and acceptance loop is computed as long as no further properties are in the removed property set or no matching property is presented in the top-n recommendations. As subjects may contain the same property multiple times with different objects, duplicated properties on subjects are removed and considered only once. The procedure of the preprocessing step, i.e., generating the initial properties, and the reconstruction phase which consists of multiple recommendation steps are shown in Algorithm 8.

Input: set $\mathcal{S}_{test}$ of all subjects in the test set
foreach $s_j \in \mathcal{S}_{test}$ **do**
  $\mathcal{P}_{rem} \leftarrow choose\_random\_properties\,(\mathcal{P}_{s_j}, 3)$
  $\mathcal{P}_{cur} \leftarrow \mathcal{P}_{s_j} \setminus \mathcal{P}_{rem}$
  repeat
    $\mathcal{P}_{rec} \leftarrow top\_n\_rec\,(\mathcal{P}_{cur}, n)$
    $p_{match} \leftarrow \varnothing$
    foreach $p_r \in \mathcal{P}_{rec}$ **do**
      if $(p_r \in \mathcal{P}_{rem}) \wedge (p_{match} \equiv \varnothing)$ **then**
        $p_{match} \leftarrow p_{match} \cup p_r$
      end
    end
    if $(p_{match} \neq \varnothing)$ **then**
      $\mathcal{P}_{rem} \leftarrow \mathcal{P}_{rem} \setminus p_{match}$
      $\mathcal{P}_{cur} \leftarrow \mathcal{P}_{cur} \cup p_{match}$
    end
  until $(\mathcal{P}_{rem} \equiv \varnothing) \vee (p_{match} \equiv \varnothing)$;
end

Algorithm 8: Reconstruction process including preprocessing and reconstruction phase

During the reconstruction process of 261,808 subjects, multiple evaluation metrics are recorded [112]. *ReconstructionTotal* denotes to which extent an subject instance can be reconstructed using Algorithm 8. *Reconstruction* describes the percentage of the removed property set which was reconstructed. After each recommendation computation, the *Precision* and *Recall* are calculated. Precision and Recall are defined as follows:

$$precision(\mathcal{P}_{rec}) = \frac{|\mathcal{P}_{rec} \cap \mathcal{P}_{rem}|}{|\mathcal{P}_{rec}|} \qquad recall(\mathcal{P}_{rec}) = \frac{|\mathcal{P}_{rec} \cap \mathcal{P}_{rem}|}{min(|\mathcal{P}_{rem}|, n)}$$

where $\mathcal{P}_{rem}$ are the previously removed properties and $\mathcal{P}_{rec}$ is the set of top-n recommendations. Due to the fact that the number of removed properties is constantly decreased and therefore, may be smaller than the number of computed recommendations (n) the number of matching recommendations is divided by the smaller value either n or the size of $\mathcal{P}_{rem}$.

Furthermore, the *F_β-measure* which combines recall and precision values is computed. The weight β defines the weight on precision or respectively recall.

The balanced F-measure F_1 is calculated using equal weights on precision and recall and is defined as follows:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \qquad F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

As an additional measurement, we also calculated the *mean reciprocal rank* which is focused on the ranking of recommendations. The rank considers the position of correct recommendations and therefore, ranks algorithms higher that have more correct recommendations at the top positions of the list of recommendations. The rank is defined as follows:

$$MRR = \sum_{i=1}^{|correct|} \frac{1}{pos(correct_i)} \quad with \quad pos(item) = position\ of\ item$$

We conducted our evaluations using the top-n recommendations with n = 5, 10, 15, 20, and 25 on all 261,808 subjects. Due to the choice overload described in Section 3.3.5, it is not recommended to suggest more than 10 items. Nevertheless, we compute recommendations up to 25 items to examine the progression of measured metrics. The first recommendation step which is based on the remaining three properties of every subject is denoted by *step=0*. If no step is specified, the value denotes the average of all iteratively computed top-n recommendation sets regarding the respective measurement. The significance (p-value) of all results, were determined by using the Wilcoxon signed-rank test [175].

The described tests were conducted using the rule-based algorithm presented in Section 3.4.1. Furthermore, several ranking strategies shown in Table 6.2 were evaluated. The main difference of the evaluated ranking algorithms is the combination of the confidence and context value (*cf.* Section 3.4.2) which is present for every recommendation candidate. $c_{confidence}$ denotes the confidence of the underlying rules which resulted in proposing the respective candidate. $c_{context}$ represents the number of rules in the respective context (subject) that pointed to the candidate. The baseline ranking "simple" is based on the simple count score (c_{count_score}) which adds up the count values of all underlying rules.

The scores used for the ranking are listed in Table 6.2. If two scores are specified and separated by a comma, the sorting is done sequentially and the second value is only considered if the first one is equal for two candidates. The last ranking strategy *context_confidence_norm* is based on normalized values denoted by the values $c_{context'}$ and $c_{confidence'}$ and incorporates a weighting factor α which can be defined in the range between 0 and 1. This factor indicates the weight of the context score, e.g., a value of 1 would ignore the confidence score. The value of α used in the respective test run is concatenated to the ranking strategy name.

Name	**Ranking based on**
simple	c_{count_score}
confidence	$c_{confidence}$
context_simple	$c_{context}$, c_{count_score}
context_confidence	$c_{context}$, $c_{confidence}$
context_confidence_normα	$(c_{context'} \cdot \alpha) \cdot ((1-\alpha) \cdot c_{confidence'})$

Table 6.2: Ranking algorithms used in the offline evaluation experiments

Experiment Results & Discussion

This section presents the results of the evaluation metrics proposed in the previous section. Firstly, we present the results of the reconstruction evaluation of the SnoopyConcept. Secondly, we get a closer look at the ranking strategies by assessing recall, precision and the mean reciprocal rank of all ranking strategies.

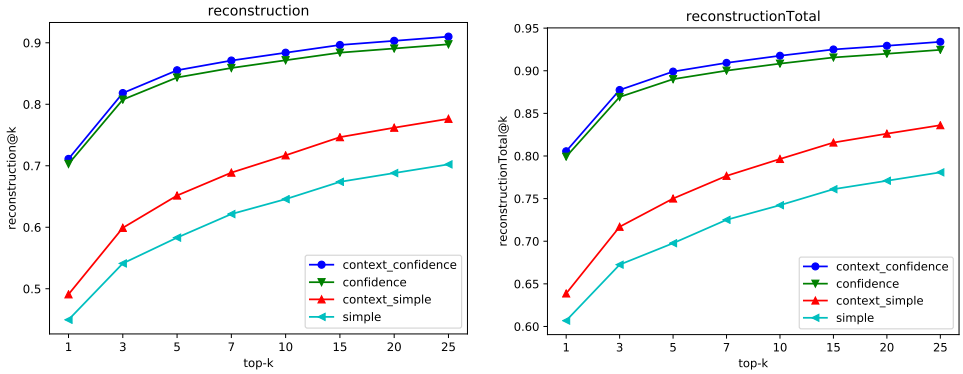


Figure 6.1: Reconstruction of removed properties / subjects

Firstly, we evaluate the presented algorithms in regards to reconstruction which is presented in Figure 6.1.

The ranking strategy "simple" is based on a simple counting value and represents the baseline in all graphs. The counting is not normalized and therefore, is mainly based on the popularity of properties. By incorporating the context, respectively the number of supporting rules, the algorithm "context_simple" is already able to achieve a 7% higher ($p < 0.001$) reconstruction rate (*cf.* Figure 6.1). The algorithm "confidence" is similar to the simple count algorithm but takes the total amount of global occurrences of the respective property into account and thus, mitigates very popular rules. Due to this fact, "confidence" is able to reconstruct 87% of all removed properties when recommending ten properties. The best performing ($p < 0.001$) algorithm "context_confidence" with a reconstruction rate of 88% was able to fully reconstruct 163,442 subjects out of 261,808 processed subjects when recommending ten properties. Overall, 253,969 subjects were reconstructed to more than 50%, which amounts to 97% of all subjects. The numbers also show that only three initial properties inserted by the user are sufficient to guide the user to a common schema by recommendations.

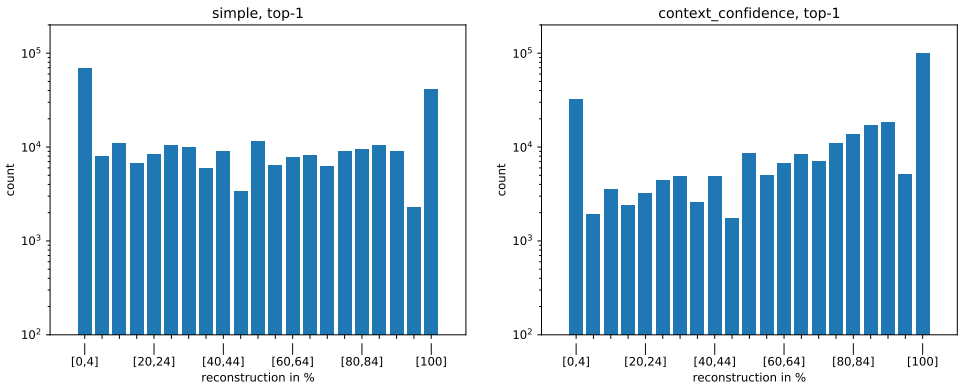


Figure 6.2: Histogram of reconstructed subjects top-1

The histograms in Figures 6.2 and 6.3 show the distribution of reconstructed subjects and the corresponding reconstruction percentage. The y-axis represents the number of subjects on a log-scale and the x-axis represents the reconstruction-buckets with a step size of 5%. The difference between the algorithm "simple" and "context_confidence" is clearly visible as the distribution of the algorithm "context_confidence" is skewed right and thus, was able to reconstruct more subjects to a higher degree. Furthermore, the histograms of "context_confidence" show that recommending just one single property (top-1) decreases the amount of reconstructed properties drastically. For example, the first bucket with a reconstruction rate of 0 to 4% contains 31,809 subjects when recommending only one item (top-1) in comparison to 5,013 subjects when recommending ten items (top-10).

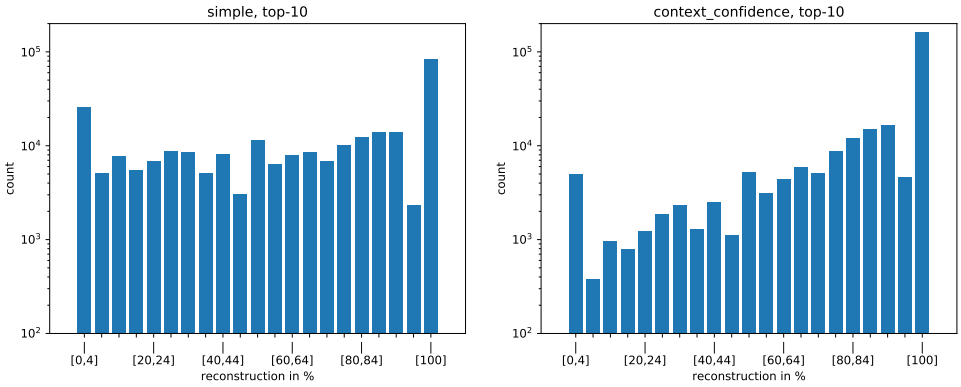


Figure 6.3: Histogram of reconstructed subjects top-10

Figure 6.4 shows the precision of all conducted evaluation runs. The algorithm "context_confidence" performs significantly better ($p < 0.001$) than all other algorithms. Even in a large and collaboratively created dataset, like the used DBpedia dataset, the algorithm features a precision of over 38% for all performed 2.6 million top-10 recommendation sets in the test run. This means that *at least three out of ten* recommendations are appropriate, the remaining seven can be inappropriate or also adequate but are not used in the respective Wikipedia infobox. The precision of the first recommendation set (step=0) as shown in Figure 6.4 is even higher and is above 61% when recommending ten properties. Thus, there are only four out of ten recommendations that are not suitable for the user when she already entered three properties. For the top-1 setting the best precision value at step-0 of 0.87 denotes that the first recommended property after entering three properties is correct and suitable with a probability of 87%.

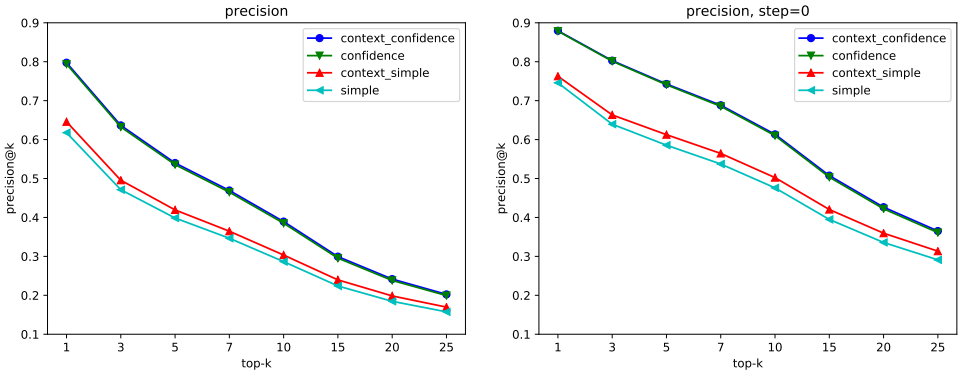


Figure 6.4: Precision of computed recommendations

The recall curves shown in Figure 6.5 present the average recall over all recommendations during the reconstruction process. Due to the definition of recall

as presented on page 121, the only possible values for recall for top-1 recommendations are 1 or 0. As long as the single recommended property matches, the reconstruction for the respective subjects continues. The recall value is 0 if the recommended property does not match and subsequently, the reconstruction process of the currently processed subject is stopped. Therefore, the recall value is higher when recommending only one or three properties. Recommending just one property (top-1) leads to a lower reconstruction rate as shown in Figure 6.1 but results in a very high recall value as the reconstruction process consists of many reconstruction steps with a recall value of 1.

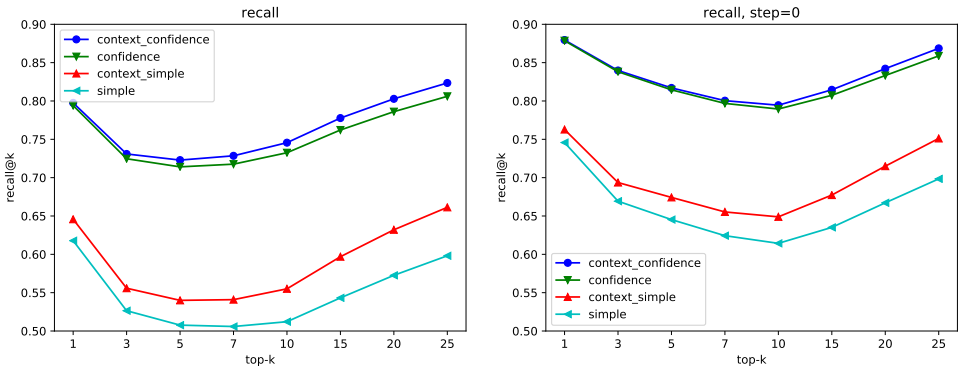


Figure 6.5: Recall of computed recommendations

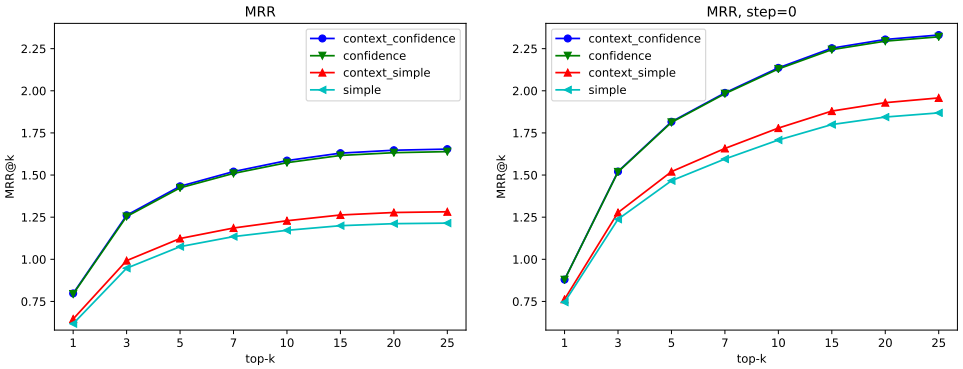


Figure 6.6: Mean reciprocal rank of computed recommendations

Figure 6.6 shows the mean reciprocal rank (MRR) for all algorithms. This measure can be very helpful when analyzing the performance of the ranking function of a given recommender system. Considering the MRR step-0 when recommending just one property, the value of 0.87 confirms the high precision value for top-1 recommendations and shows that more than eight out of ten top-1 recommendations are accurate and suitable in the respective context of a subject containing three initial properties. This high top-1 accuracy is crucial for system-features which are restricted to show only one recommendation.

For example, the input field in the subject editor can be already prefilled by a greyed-out placeholder property as shown in Figure 5.1 on page 106. Such a feature would furthermore reduce the barrier of accepting recommendations and thus, contributes to a more homogeneous vocabulary of properties.

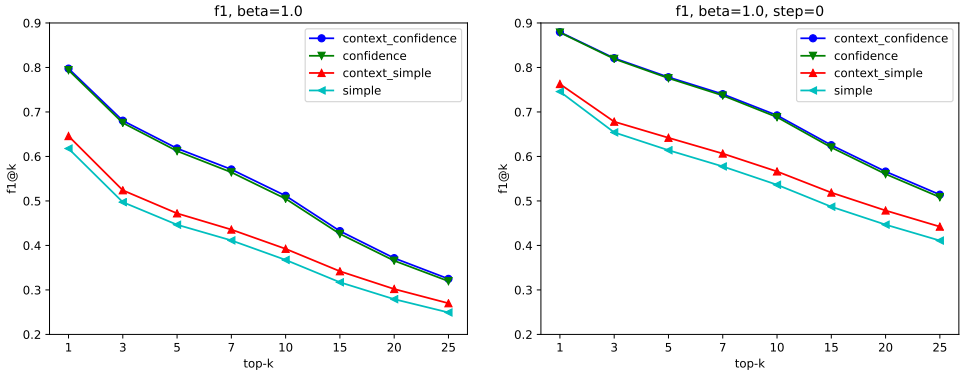


Figure 6.7: F_1-measure of computed recommendations

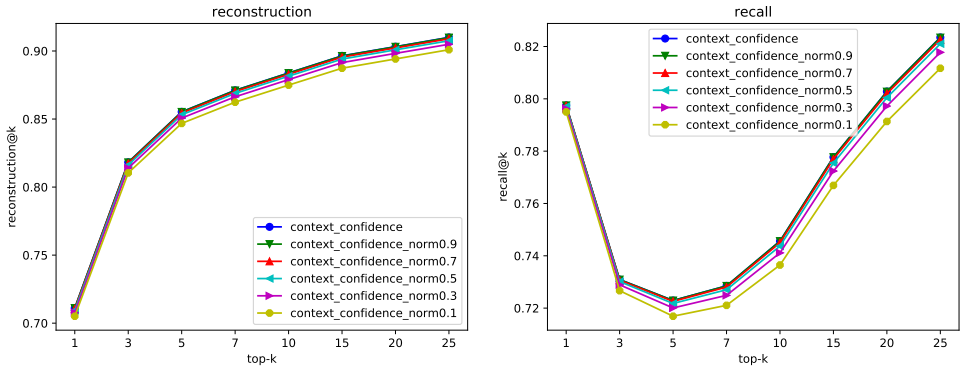


Figure 6.8: Reconstruction and recall of "context_confidence_normα"

In general, it can be seen that the curves are flattening with an increasing top-k value. This behaviour is strongly visible in the MRR graph in Figure 6.6 which shows a very flat curve with top-k values greater than 10. This behaviour is also confirmed by the sloping curve shown in the F_1-measure graph in Figure 6.7. Especially when considering a good user experience and the prevention of choice overload by too many recommendations (*cf.* Section 3.3.5 and [115, 25]), this property is crucial and proves that a limited amount of recommendations of seven to ten items is sufficient to guide the user to a common schema.

In Figure 6.8, the reconstruction and recall of the normalized version of the context-sensitive algorithm presented in Section 3.4.3 are shown. The attached number in the name of the ranking algorithm indicates the weight α of the

normalized $c_{context}$. For example, "context_confidence_norm0.9" defines that $c_{context}$ is incorporated in the final score by 90% and $c_{confidence}$ by 10%. It can be seen that a decreasing impact of the context results in a lower reconstruction and recall. As shown in Table 6.3, "context_confidence_norm0.9" and "context_confidence" are both able to reach the best recall value of 0.74. All other weights result in lower recall and recall values. Those results show that the computationally more complex ranking strategy "context_confidence_norm" which require 50% more time to compute recommendations (*cf.* Section 4.4) is not able to perform significantly better than "context_confidence". Therefore, we conclude that the weighting and normalization overhead can be omitted without trading in quality of recommendations. Thus, the algorithm "context_confidence" performs the best in terms of performance and accuracy.

Algorithm	Recall	Precision	MRR	ReconstTotal
simple	0.5121	0.2865	1.1720	0.7423
context_simple	0.5550	0.3034	1.2288	0.7965
confidence	0.7325	0.3852	1.5735	0.9084
context_confidence	0.7456	0.3894	1.5858	0.9176
context_confidence_norm0.1	0.7365	0.3865	1.5773	0.9108
context_confidence_norm0.3	0.7411	0.3880	1.5818	0.9137
context_confidence_norm0.5	0.7437	0.3889	1.5844	0.9157
context_confidence_norm0.7	0.7450	0.3893	1.5855	0.9168
context_confidence_norm0.9	0.7456	0.3894	1.5858	0.9175

Table 6.3: Metrics for all presented algorithms, top-10

To conclude the evaluation of all proposed recommendation algorithms, Table 6.3 lists all measured performance values in terms of recall, precision, mean reciprocal rank (MRR) and reconstruction for recommending ten properties. As can be seen, the best results are obtained by the "context_confidence" approach across all measures.

Summary & Limitations

The presented experiments evaluate the automatic recommendation of structure without any user interaction. However, the Snoopy concept provides additional recommendations while the user is typing. Consider a user who specifies the first character of a property, e.g. "A". This information can cut down the number of suitable recommendations dramatically. Furthermore, by using a thesaurus, synonyms can be matched and a more commonly used synonym can be recommended to the user. Consider a user who enters *citizens* as a property. The system recommends the usage of *population* and by accepting

this recommendation, the user contributes to a more homogeneous schema. All these user input-based recommendations heavily increase the recall and precision values but cannot be tested within the presented test scenario, as real interaction of a human user is required.

Furthermore, the presented offline evaluation is based on a collaboratively created dataset which is used as a ground truth dataset and thus, defines if a recommendation is valid and correct. We argue that such an evaluation can only serve as a baseline evaluation measure due to the restriction that only properties that have already been used on the given data item, may resolve to true positives and hence, influence the evaluation result. Therefore, we assume that the recommendation algorithm performs even better than we can prove by using this offline based approach.

The performed evaluations reveal that both scores $c_{context}$ and $c_{confidence}$ are relevant and have to be incorporated in a ranking to achieve significantly better results in terms of recall, precision, MRR and reconstruction. The scores can be sequentially combined and do not need any normalization or weighted combination. We also showed that the recommendation of seven to ten properties is sufficient and leads to reconstruction rates of up to 91% while having a precision of 39%.

6.1.2 Wikidata Property Suggestor Evaluation

The SnoopyConcept algorithm was also evaluated in the context of Wikidata and its property suggestor as described in Section 5.4. In contrast to the SnoopyConcept, the Wikidata property suggestor heavily relies on the defined class of a subject. This additional information is very beneficial for a recommender system as it allows to recommend properties which are defined for the respective class. The SnoopyConcept was designed to compute recommendation without any manually created information about the type or class. This advantage provides the possibility to also compute accurate recommendations for very specific domains as e.g., moons of Pluto (*cf.* Section 3.1).

In this section, we present the results of the comparison between the SnoopyConcept, the Wikidata Property Suggestor and a new hybrid approach which is described in Section 5.4. For the evaluation, we employed a leave-one-out test as described in Section 6.1.1 to evaluate their ability to provide suitable recommendations and reconstruct the properties of a given subject.

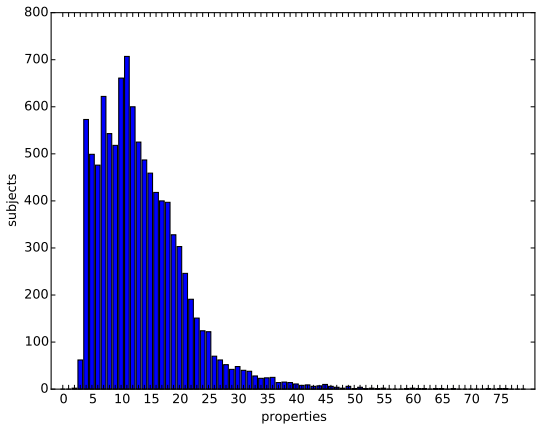


Figure 6.9: Test Set Properties per Subject Distribution

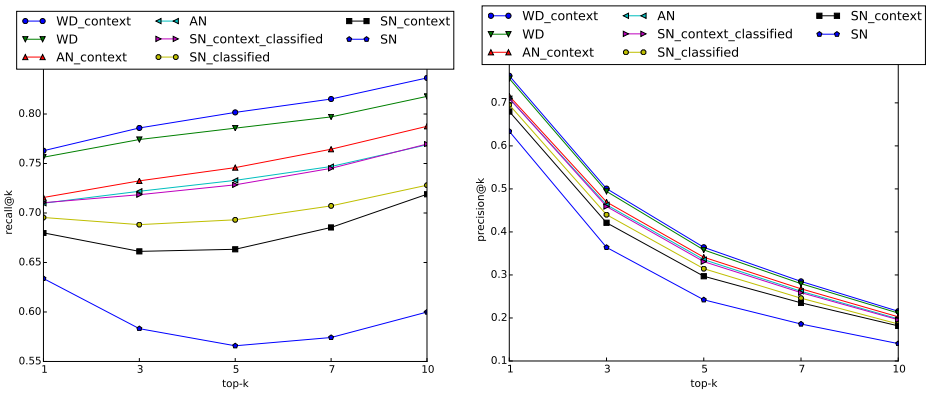
Dataset & Experiment Description

For the evaluation, we used a Wikidata dump[1], using the version of 2015-10-26. For the test set, we randomly selected 10,000 different subjects with a minimal requirement of four properties. This requirement is fulfilled by 9,254 data items that subsequently form the test set underlying the evaluation. Figure 6.9 shows the distribution of the number of properties on each item for the test set. For each of these subjects within the test set, we randomly select three properties and remove all but these three properties from the data item. We store these removed properties as these form the ground truth data for the evaluation. The resulting reduced subject serves as the first input for the recommender systems. Subsequently, the evaluation process as described in the previous Section 6.1.1 attempts to reconstruct a subject. As for the measures for evaluating the reconstruction process we rely on the traditional information retrieval measures recall, precision and F-measure, the mean reciprocal rank (MRR) [112] and the reconstruction-measure. All metric definitions and an according description can be found in Section 6.1.1.

Experiment Results & Discussion

This section presents the results of the evaluation methods proposed in the previous section in the context of Wikidata. Firstly, we present the results of the evaluation of the individual recommendation approaches in regards to recall and precision. Secondly, we get a closer look at the mean reciprocal

[1] https://www.wikidata.org/wiki/Wikidata:Database_download, accessed 2017-07-17



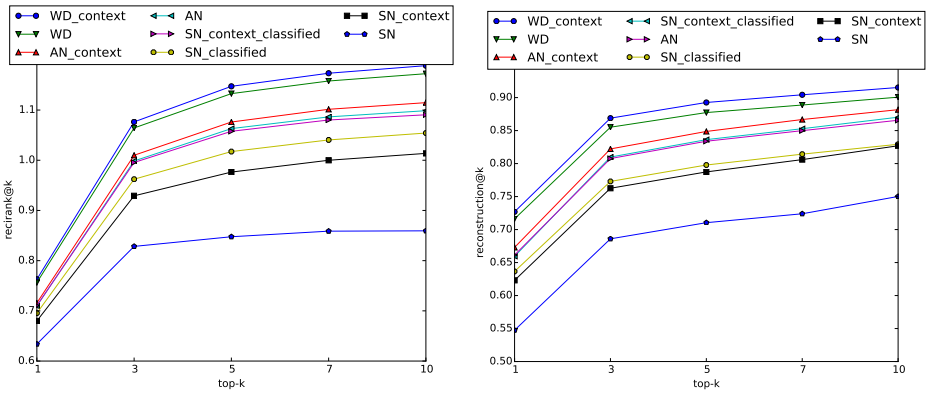
(a) Recall@k (b) Precision@k

Figure 6.10: Evaluation measures@k of all evaluated algorithms

rank and in a third step, we look at the reconstructive power of the individual algorithms.

Firstly, we evaluate the presented recommender algorithms in regards to recall and precision. Figure 6.10a shows a plot of the recall@k-measure for the presented recommenders. I.e., we evaluate the recall-measure for one, three, five, seven and ten provided recommendations and depict the different results. We observe that in terms of recall, the Wikidata recommendation approach enhanced with contextual information (WD_context as presented in Section 5.4) performs best, achieving a recall@1 of 76.29% and a recall@10 of 83.64%. The evaluations shows that this approach performs best across all numbers of recommendations given (significantly better than all other approaches; $p < 0.001$). The second best approach is the AN approach, followed by SN_context. We can also observe that the SN approach performs last, however, can be significantly improved by utilizing contextual information ($p < 0.001$).

Figure 6.10b shows the precision@k-results for the evaluated algorithms. Again, WD_context performs best, reaching a precision of 76.29%@1 and a precision of 21.55%@10, again closely followed by the AN approach. We detect significant differences in the performance of these two ($p < 0.001$) and can observe that the algorithms perform rather similar, again with SN being the worst performing algorithm. The evaluations show that using contextual information for ranking improves each of the proposed approaches significantly. We verify this finding by comparing the results of the individual algorithms once without using context for ranking and once with added context information for the ranking computation. This evaluation shows that for all of the approaches, recall as well as precision for all recommendation list



(a) Mean Reciprocal Rank (b) Reconstruction

Figure 6.11: Evaluation Measures

sizes is significantly increased when introducing context to the ranking process ($p < 0.001$). As for the contribution of classifying properties, we observe similar results. Adding special rules for classifying properties to the set of association rules significantly increases recall as well as precision of all of the approaches ($p < 0.001$).

Figure 6.11a shows the mean reciprocal rank for all algorithms. This measure can be very helpful when analyzing the performance of the ranking function of a given recommender system. Generally, we observe that the findings in this evaluation are in line with the previous results. I.e., the best performing algorithms in terms of recall and precision also perform well in terms of ranking.

The results of the evaluation of the reconstructive power of the evaluated property recommendation algorithms can be seen in Figure 6.11b. Again, the results correlate with the previous findings and show the best performance for the WD_context approach, significantly better than the WD approach ($p < 0.001$).

To conclude the evaluation of the individual recommender algorithms, we provide a detailed comparison of the best algorithm of all three proposed approaches in Table 6.4. On the Wikidata platform, the default number of recommendations provided to a user entering new information, is 7. Therefore, we list the performance@7 in Table 6.4 where we measure performance in terms of recall, precision, the F_1-measure, the mean reciprocal rank (MRR) as well as the reconstruction value. As can be seen, the best results are obtained by the WD_context approach across all measures.

Algorithm	Prec.	Recall	F_1	MRR	ReconstTotal
WD_context	0.2849	0.8152	0.4223	1.17	0.9042
WD	0.2797	0.7971	0.4141	1.16	0.8887
AN_context	0.2678	0.7644	0.3966	1.10	0.8667
AN	0.2620	0.7469	0.3879	1.09	0.8498
SN_context_classified	0.2589	0.7451	0.3843	1.08	0.8529
SN_classified	0.2460	0.7071	0.3650	1.04	0.8143
SN_context	0.2352	0.6854	0.3502	0.99	0.8060
SN	0.1859	0.5742	0.2809	0.86	0.7239

Table 6.4: Detailed Evaluation of all Algorithms@7

The performed evaluations reveal two important influence factors when it comes to recommending properties to users who currently enter information on the Wikidata platform: context (*cf.* Section 3.4.3) and classified properties. The evaluations show that incorporating these two aspects into the recommendation process can significantly enhance and improve the resulting recommendations. I.e., using classifying properties to gain further information about the type of the given data item is an important factor. Similarly, considering the context of a property, and hence, the number of distinct rules leading to its recommendation, for ranking, further adds to high-quality recommendations.

One limitation of the classifying approach we observe lies in its reduced flexibility and hence, its generalizability. We argue that information about the type or superclass of a data item may not always be available, especially when applying these concepts in a broader context. The manual choice of classifying properties seems rather inflexible and is not necessarily generalizable. Therefore, we argue that by setting up a more rigid recommender system by manually specifying classifying properties we trade in flexibility for (slightly) improved results. The evaluations show that e.g., in terms of recall@7, the difference between the WD_context and WD approaches is 1.81%. On the contrary, especially considering the fact that a majority of data items feature only a low number of properties (on average, 4.13 for the dataset underlying our evaluations), any recommender system has to face the cold-start-problem (*cf.* Section 3.6).

We showed that it is beneficial to incorporate contextual information (as proposed by the Snoopy approach) into the recommendation process of the Wikidata property suggestor. The evaluations show that not only considering the sum of confidences of the applicable rules, but also the number of rules leading

to a given recommendation candidate leads to significantly better results in terms of recall, precision, MRR and reconstruction.

6.1.3 Online Evaluation/User Experiment

The main goal of the SnoopyConcept is to incorporate the user and encourage the user to interact with the system. The evaluation of such an interactive information system cannot be conducted artificially as the interaction of users with the system cannot be simulated. Therefore, we chose to conduct a user-centric experiment based on the SnoopyDB prototype which was focused on the user-interaction regarding the acceptance of recommendations and the provided support in general [61, 60]. The goal of the evaluation was to assess the user's acceptance of the recommendation and guidance mechanisms provided by the system.

In total, 24 test users took part in the experiment. These users were recruited from different backgrounds, 2/3 of all test users were computer scientists and 1/3 of the participating users were standard computer users without any special computer knowledge or experiences with handling semi-structured data.

For the evaluation, the test users were presented with two different systems: one system was supporting the users with all recommendation and guidance features described in the previous sections (in the following referenced as *system A*). The other system was not supporting the users at all and thus, was not providing any recommendations for neither properties nor value entries (in the following referenced as *system B*). System A was bootstrapped with data created within the previous evaluation of the system [61] which contained subjects originating from two domains (cities and musicians) in order to be able to provide basic recommendations and to be able to assess how the system and the provided recommendations adapt if subjects stemming from a new, unknown domain are added.

In the course of the experiment, users were asked to fulfill the following tasks in order:

1. Insert data about an arbitrary university firstly into system B (no support provided to the user)

2. Insert data about an arbitrary subject related to the motor vehicle industry into system B

3. Insert data about an arbitrary university into system A (guidance by recommendations)

4. Insert data about an arbitrary subject related to the motor vehicle industry into system A

The subjects the users had to enter were not specified, as well as the actual information the users had to enter about a certain subject. This information was solely chosen by the test users themselves. Also, no minimum number of property-value pairs was specified and hence, the amount of information about a certain subject that was added was solely decided by the user. The only restriction was that users were only allowed to use the English language for the names of the properties. Furthermore, users were not allowed to use the English Wikipedia for seeking information as already aligned infobox properties could possibly influence the German-speaking test users and the resulting property names. During the experiments, all actions the participating test users were logged and stored in order to be able to evaluate the differences in the performance of the two systems in regards to the homogeneity of the resulting vocabulary and the acceptance of the provided recommendations. The results of the analysis of the information gathered during the user experiments are discussed in the following.

As for the acceptance of the proposed recommendations, the evaluations showed that 22% of all recommended properties were accepted by the users. This seemingly low number can be led back to the fact that the system always proposes five additional properties. Each time a user accepts a recommendation or adds new information, the recommendation list is recomputed. Hence, the total number of recommended properties is high during each edit-session. In total, 49% of all newly added property-value pairs were added by accepting a property recommendation. In 62% of all user edit sessions (including re-editing of subjects), at least one property recommendation was accepted. Furthermore, also the auto-completion feature provided as an additional guidance and support mechanism, was used frequently. 23% of all properties and 17% of all values were entered by accepting the entries proposed by the auto-completion mechanisms. A spellchecker was also implemented in the SnoopyDB prototype. The corrections proposed by the spellchecker were accepted by the participating test users in 37% of all cases. This acceptance rate is low, which can be led back to the fact that the spellchecker web service is based on information extracted from the web corpus. Hence, the spellchecker was not only suggesting corrections for simple typos but also suggested e.g. 'Formula 1' instead of 'Formula1' or 'Leopold II' instead of 'Leopold I' which were not accepted by the users. Considering only simple typos (e.g. one missing character), applicable spellchecker recommendations were accepted in 100% of all cases.

The recommendation of semantic refinements in such a user experiment is limited as only present subjects can be linked. Nevertheless, users classified 52 values in SnoopyDB (system A) as links (external or internal) and created nine internal semantic links. Thus, nine values were semantically enhanced by semantic links pointing to the correct subject.

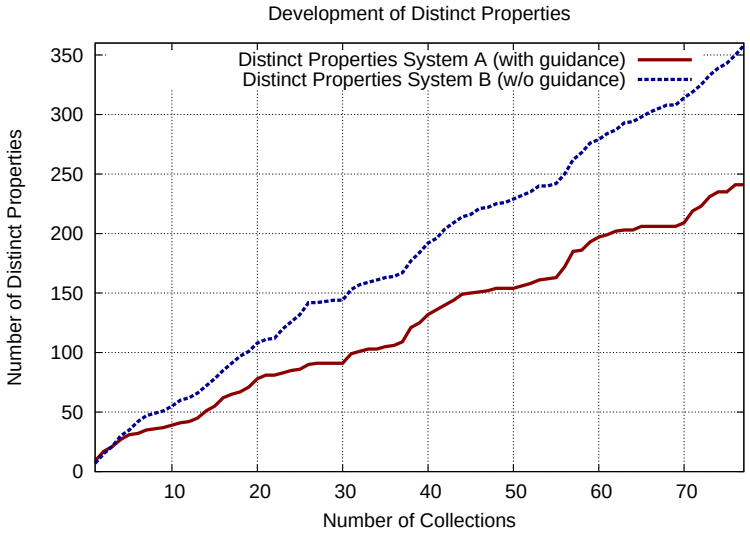


Figure 6.12: Number of distinct properties in system A/B

Figure 6.12 shows that the schema entered into the recommendation-providing system A was 33% more homogeneous in regards to the set of properties entered than without supporting the user (system B). Homogeneity within a set of properties describes how many synonymous terms were used for the description of the same subject, i.e. how many property names were directly reused and hence, no synonym was used instead. In this evaluation the resulting property vocabulary in system A is 33% smaller than the vocabulary of system B. Furthermore, object recommendations in system A encouraged the user to reusage values. 18% of all values used in system A, were created by accepting already present recommended values. In system A without recommendations, only 6.8% of all values were reused. For example, in system B no single value for the property "genre" was reused, while in system B "pop" was reused ten times and "rock" was reused seven times. Despite the reduction of properties and values, the users entered 31% more information into the system A when supported by recommendations as shown in Figure 6.13. This implies that by guiding the user during the insertion process and furthermore, enabling the user to easily add more information and also to point the user to bits of information which she still might want to enter, the total amount of useful and structured information can be increased. This value is biased by the fact that the users already dealt with subjects in the first two tasks of the evaluation

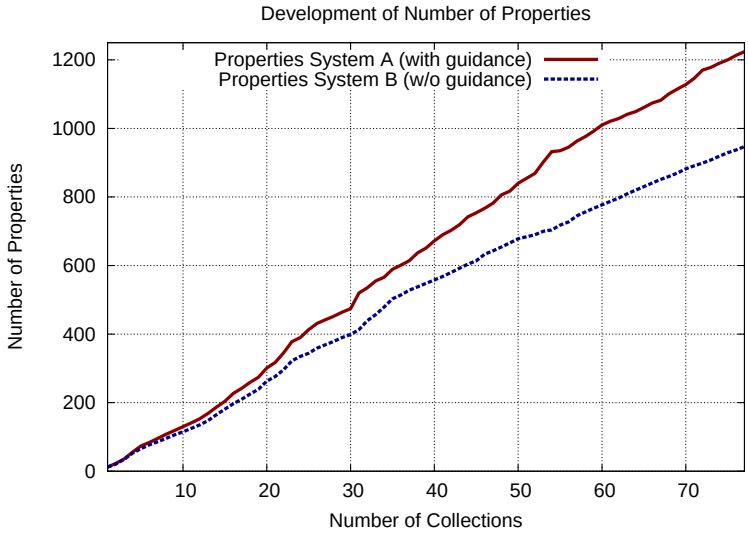


Figure 6.13: Total number of properties/triples in system A/B

(system B). However, as we wanted to simulate guidance and motivation of domain expert users, who already have extensive knowledge about the subject, such a high percentage can also be possible in real-world environments. Another important finding of the evaluations was that the introduction of the new domains did not result in a dramatic increase of newly added properties. This fact implies that most of the properties were reused.

Summarized, we showed by this user-centric experiment that recommendations of the SnoopyConcept are accepted and lead to an increase of quality and quantity of knowledge in the information system.

6.2 User Modeling Evaluation

The evaluation of the extended SnoopyConcept algorithm which also incorporates the user preferences as described in Section 3.5, was based on the SnoopyTagging prototype [34] which is presented in Section 5.2 and was officially released as a Flickr App[2] in the Flickr App Garden. The conducted user tests were also published in [34, 63]. In total, 20 voluntary test users (originating from various backgrounds and having diverging levels of computer skills) took part in this evaluation. The users were asked to use the SnoopyTagging prototype to upload and subsequently tag arbitrary photos.

[2] https://www.flickr.com/services/apps/72157625264834816/, accessed 2017-07-17

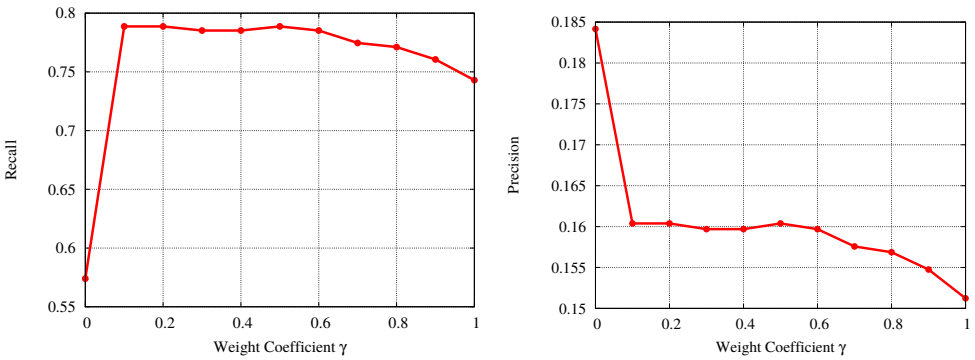


Figure 6.14: Precision/Recall@5 using different γ weight coefficients

Overall, 98% of all tags the users facilitated were structured tags, even though the insertion of simple tags was still possible. This number is a strong indicator for the acceptance of the concept of contexts. 350 recommendations (233 contexts, 117 tags) were accepted for the creation of 310 Structured Tags. Such a high percentage of Structured Tags and accepted recommendations can be attributed to the user-friendly SnoopyTagging system which allows users to easily create Structured Tags, and the acceptance of the underlying recommender system.

During the user experiments, all recommendations, user inputs and accepted recommendations by the user were logged. Based on this usage data, we evaluated the performance of the hybrid ranking function. For this purpose, we simulated all recorded user interactions and performed them again while varying the γ-values responsible for the weighting of the ranking as shown in the following definition. Subsequently, we evaluated recall and precision of the recommendations by using the dataset which was manually created by all users.

$$total\_score = \gamma \cdot score_{global} + (1 - \gamma) \cdot score_{user} \text{ with } \gamma \in [0, 1]$$

The results respectively recall and precision at five recommendation can be seen in Figure 6.14. The results show that the basic recommendation algorithm without considering any user preferences ($\gamma = 1$) already provides very good recommendations with a recall value of 74%. By increasing the incorporation of the user-specific recommendations the recall value can be increased up to 79% with $\gamma = 0.1$. The relatively low recall value of 58% when setting $\gamma = 0$ and therefore, only consider user specific recommendations can be led back to the fact that the algorithm cannot suggest any suitable tags

for unknown users. This drawback can be easily fixed by taking global recommendations into account as global recommendations are considered if there are not sufficient data about the respective user. The precision values at five recommendations also indicate that setting $\gamma \leq 0.5$ which results in a stabilized precision values around 0.16. The reason for the relatively high precision value of 18.5% with $\gamma = 0$ can be led back to the definition of precision. By considering just user-specific recommendation it is not always possible to provide five recommendations due to small tag-vocabularies of new users. Thus, less tags are recommended which results in an peaking precision value. In Figure 6.15 the precision values for recommending only the top ranked item are shown. When just recommending the top ranked item, recall and precision behaves the same which results in the same graph. The impact of the user-specific recommendations is easily noticeable by an increasing recall and precision value. The best result of about 0.37 can be achieved by setting $\gamma = 0.2$.

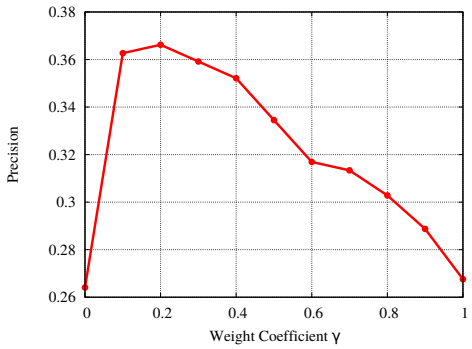


Figure 6.15: Precision@1 using different γ weight coefficients

This strong emphasis on personalized recommendations ($\gamma \leq 0.5$) can be explained by the users preferences to reuse their own tags but also by the limited amount of users and the resulting size of the folksonomy in the system. The global set of recommendations will increase with the amount of users as there are more appropriate tags to recommend. However, to the best of our knowledge there is no Web 2.0 platform which advertises Structured Tags and therefore it is not possible to observe this behaviour in a large community in a real environment.

As for the homogeneity within the tagging vocabulary resulting from the experiments, a total of 170 distinct Structured Tags were entered, where only 37 different contexts were used. This fact is remarkable considering the circumstance that users were allowed to enter photos about an arbitrary topic and shows the frequent re-usage of contexts by recommendations.

6.3 Evaluation Summary

In this chapter we presented the results of different evaluations of the SnoopyConcept and its algorithms. The offline evaluation which is based on a DBpedia dataset proved that the main recommendation algorithm is able to recommend suitable properties by simulating a user which creates new items. The usage of the DBpedia set with over 59 million triples furthermore showed that the algorithm is also able to cope with large large datasets which were created using mass-collaboration information systems. The offline reconstruction evaluation showed that up to 91% of all triples can be reconstructed by the proposed recommendation algorithm. When recommending ten properties the algorithm was able to reach a precision value of 39% and a recall value of 74%. The user experiment in Section 6.1.3 compares a naive system and a system implementing the SnoopyConcept. We showed that the snoopyfied system is able to homogenize the structures and reduce the size of the vocabulary by 33% while at the same time encouraging the user to enter more information which resulted 31% more stored knowledge in the system. Furthermore, we proved that a recommender system is accepted by the users as 66% of all created triples were based on recommended and accepted properties by the users. In Section 6.1.2 we showed that the extension of the Wikidata Property Suggestor algorithm by the context used in the SnoopyConcept algorithm led to higher accuracy. Furthermore, we showed that the usage of a manual classification system, as used in the Wikidata approach, has only a limited impact on the accuracy of structure recommendations. It improves the reconstruction rate by 2-4% but adds the requirement to manually classify every subject and thus, rigidify the system and impede the general application of a recommendation algorithm. In Section 6.2 we evaluated the personalized recommendation algorithm of the SnoopyConcept and showed that the incorporation of the personal preferences of a user can lead to a dramatically increase of accurate recommendations.

CHAPTER 7

Conclusion

During the last decades, with the enormous growth of the internet and the web 2.0 movement, collaboration has been lifted to a new level—online mass-collaboration. In this thesis, we analyzed how recommender systems can be facilitated to empower semi-structured knowledge bases to improve the curation, search and storage of huge amounts of knowledge which is created in a collaborative fashion. Thus, we identified three main research questions in Section 1.1 which are tackled by the proposed SnoopyConcept:

How can recommender systems empower collaborative information systems to become more structured without losing their flexibility?

The main idea of the SnoopyConcept is to incorporate the user already during the insertion process to prevent heterogeneous structures. Therefore, we developed a recommender system (*cf.* Section 3.3) which guides the user during the insertion process to align the knowledge to a common homogeneous schema. The recommender system is solely based on already inserted knowledge and therefore, adapts itself to new domains or structures inserted by users. We showed in Section 6.1.1 that the SnoopyConcept can be applied to a mass-collaboratively curated dataset and provide suitable recommendations to increase the homogeneity. This evaluation exposed that 40% of all recommendations were accurate and the SnoopyConcept reached a recall of up to 90%. The conducted online evaluation described in Section 6.1.3 revealed that

the SnoopyConcept recommendations were able to create a more homogeneous schema by reducing the size of the vocabulary used in the information system by 33%. Nevertheless, the user is not forced to a schema and was able to insert new domains and structured when needed.

How can direct user communication during the insertion process be facilitated by recommender systems to increase the quality of information?

The user who inserts new knowledge to an information system is usually a domain expert. Therefore, it is very import to use the opportunity of direct communication with the user already during the insertion process to exploit her expertise. Especially semantic uncertainties and type conflicts can be resolved at an early stage. The recommendation approach proposed in Section 3.3 aims at enriching entries by semantic information. We showed in our user-centric experiment presented in Section 6.1.3 that semantic enhancement recommendations are accepted and lead to more semantic meaningful information in the system, such as semantic links. Our online evaluation in Section 6.2 showed that 350 recommendations were accepted for the creation of 310 tags. In the user-centric experiment described in Section 6.1.3 49% of all newly added property-value pairs were added by accepting a recommendation. Those numbers are strong indicators for the acceptance of the recommender system and the possibility to influence or guide the user to insert more homogeneous and semantically enhanced information.

How can automated user guidance by a recommendation system result in an increased quantity of information?

By recommending suitable properties, the user is guided to re-use already present properties in the information system. As the user is a domain expert, this approach can also encourage the user to insert more information. In our online evaluation (*cf.* Section 6.1.3) we showed that this approach of pointing to missing pieces of information, encouraged the users to enter 31% more information.

In general, we developed the universal SnoopyConcept to compute accurate recommendations without limiting the fields of application.

The concept is not bound to an underlying technology which was shown by implementing the approach and its algorithms using three different data storage technologies and models (*cf.* Chapter 5). The best performing model was the relational model which was able to compute suitable recommendations in 3ms using the mass-collaboratively created DBPedia dataset. This fulfills the performance requirements of an online real-time recommender systems which is also crucial for the acceptance of a recommender system.

Furthermore, the general applicability of the SnoopyConcept was demonstrated by applying the approach to different domains (*cf.* Chapter 4), such as image tagging and personal information management systems. In those domains the recommendation approach was furthermore personalized to optimize the recommendations (*cf.* Section 3.5). In the evaluation of the personalized recommender algorithm in Section 6.2 we showed that the recall can be increased from 74% to 79% by incorporating the user specific behaviour.

Moreover, the property suggestor by Wikidata which conforms to the SnoopyConcept, was introduced 2014 and was discussed in Section 5.4. It proves that the SnoopyConcept approach of guiding the user during the insertion of knowledge which we already introduced in 2010 [61] has emerged to mainstream within the past years and emphasizes the usefulness of the approach. In Section 5.4.2 we proposed an extension of the property suggestor by incorporating the SnoopyConcept context (*cf.* Section 3.4.3). Our conducted evaluation in Section 6.1.2 proved that the context is able to furthermore improve the accuracy of the system.

In conclusion, we showed in this thesis that the SnoopyConcept is able to increasing the quantity and quality of knowledge in semi-structured information systems by leveraging recommender systems to incorporate and guide the user already during the insertion process.

Bibliography

[1] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 411–422, Vienna, Austria. VLDB Endowment, 2007.

[2] Z. Abedjan and F. Naumann. *Amending RDF Entities with New Facts*. In *The Semantic Web: ESWC 2014 Satellite Events: ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*. Springer International Publishing, Cham, 2014, pages 131–143.

[3] Z. Abedjan and F. Naumann. Improving RDF Data Through Association Rule Mining. *Datenbank-Spektrum*, 13(2):111–120, 2013.

[4] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[5] G. Adomavicius and A. Tuzhilin. *Context-Aware Recommender Systems*. In *Recommender Systems Handbook*. Springer US, Boston, MA, 2011, pages 217–253.

[6] G. Adomavicius and A. Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[7] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD International Conference on Management of data*, pages 207–216, Washington, D.C., United States. ACM, 1993.

[8] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, page 487499, 1994.

[9] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2):207–216, 1993.

[10] S. Aksoy and R. M. Haralick. Feature Normalization and Likelihood-based Similarity Measures for Image Retrieval. *Pattern Recogn. Lett.*, 22(5):563–582, Apr. 2001.

[11] B. Andersen. Meta Tags: The Poor Man's RDF? *Sci-Fi Hi-Fi Blog* *http://weblog.scifihifi.com/2005/08/05/meta-tags-the-poor-mans-rdf/*, 2005. available at http://web.archive.org version September 2, 2011. accessed 2017-07-17.

[12] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Computing Surveys*, 40(1):1:1–1:39, Feb. 2008.

[13] R. Angles, A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. Benchmarking Database Systems for Social Network Applications. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, 15:1–15:7, New York. ACM, 2013.

[14] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *DBpedia: A Nucleus for a Web of Open Data.* In *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.* Springer Berlin Heidelberg, 2007, pages 722–735.

[15] D. Aumueller and S. Auer. Towards a Semantic Wiki Experience: Desktop Integration and Interactivity in WikSAR. In *Proceedings of the 2005 International Conference on Semantic Desktop Workshop: Next Generation Information Management & Collaboration Infrastructure - Volume 175*, SDW'05, pages 212–217, Galway, Ireland. CEUR-WS.org, 2005.

[16] C. W. Bachman. The Origin of the Integrated Data Store (IDS): The First Direct-Access DBMS. *IEEE Annals of the History of Computing*, 31(4):42–54, 2009.

[17] R. M. Bell, Y. Koren, and C. Volinsky. All Together Now: A Perspective on the Netflix Prize. *CHANCE*, 23(1):24–29, 2010.

[18] J. Bennett and S. Lanning. The Netflix Prize. In *KDD Cup and Workshop in conjunction with International Conference on Knowledge Discovery and Data Mining*, 2007.

[19] N. Bhatia and Vandana. Survey of Nearest Neighbor Techniques. *Computing Research Repository (CoRR)*, abs/1007.0085, 2010.

[20] R. Bierbauer. *Development of a light-weight web client for a hashtag based PIM system*. Bachelor's Thesis, University of Innsbruck, Nov. 2013. supervised by Wolfgang Gassler, Eva Zangerle, Günther Specht.

[21] R. Binna, W. Gassler, E. Zangerle, D. Pacher, and G. Specht. SpiderStore: A Native Main Memory Approach for Graph Storage. In *Proceedings of the 23nd Workshop Grundlagen von Datenbanken (GvDB 2011), Obergurgl, Austria*. CEUR-WS.org, ISSN 1613-0073, Vol. 733, 2011.

[22] R. Binna, W. Gassler, E. Zangerle, D. Pacher, and G. Specht. SpiderStore: Exploiting Main Memory for Efficient RDF Graph Representation and Fast Querying. In *Proceedings of the 1st International Workshop on Semantic Data Management (SemData) at the 36th International Conference on Very Large Data Bases (VLDB 2010), Singapore*. CEUR-WS.org, 2010.

[23] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data-the Story so far. *International Journal on Semantic Web and Information Systems*, 4(2):1–22, 2009.

[24] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, Vancouver, Canada. ACM, 2008.

[25] D. Bollen, B. P. Knijnenburg, M. C. Willemsen, and M. Graus. Understanding choice overload in recommender systems. In *Proceedings of the fourth ACM Conference on Recommender Systems*, RecSys '10, pages 63–70, Barcelona, Spain. ACM, 2010.

[26] P. Boulain, N. Shadbolt, and N. Gibbins. Hyperstructure Maintenance Costs in Large-scale Wikis. In *Proceedings of the WWW 2008 Workshop on Social Web and Knowledge Management, Beijing, China, April 22, 2008*, volume 356 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[27] S. Boyd. Hash Tags = Twitter Groupings. *Stoweboyd.com* http://stoweboyd.com/post/39877198249/hash-tags-twitter-groupings, Aug. 2007. accessed 2017-07-17.

[28] J. S. Breese, D. Heckerman, and C. Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, Madison, Wisconsin. Morgan Kaufmann Publishers Inc., 1998.

[29] J. Breese, D. Heckerman, and C. Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52, Burlington, MA, USA. Morgan Kaufmann Publishers Inc., 1998.

[30] P. Brusilovsky. Adaptive Hypermedia. English. *User Modeling and User-Adapted Interaction*, 11(1-2):87–110, 2001.

[31] M. Buffa, F. Gandon, G. Ereteo, P. Sander, and C. Faron. SweetWiki: A Semantic Wiki. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):84–97, Feb. 2008.

[32] P. Buneman. Semistructured data. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 117–121. ACM, 1997.

[33] R. Burke. Knowledge-based recommender systems. *Encyclopedia of library and information science*, 69(Supplement 32):180, 2000.

[34] M. Bürgler. *SnoopyTagging*. Master's thesis, University of Innsbruck, Oct. 2011. supervised by Eva Zangerle and Wolfgang Gassler.

[35] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The Information Visualizer, an Information Workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 181–186, New Orleans, Louisiana, USA. ACM, 1991.

[36] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka J., and T. M. Mitchell. Toward an Architecture for Never-ending Language Learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pages 1306–1313, Atlanta, Georgia. AAAI Press, 2010.

[37] M. Castells. *The Rise of The Network Society: The Information Age: Economy, Society and Culture*. Information Age Series. Wiley, 2000.

[38] R. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade. *The Object Database Standard: ODMG 2.0*, volume 131. Morgan Kaufmann Publishers, 1997.

[39] W. B. Cavnar and J. M. Trenkle. N-Gram-Based Text Categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[40] Ò. Celma. *Music Recommendation and Discovery - The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer, 2010, pages I–XVI, 1–194.

[41] P. Y. Chau, S. Y. Ho, K. K. Ho, and Y. Yao. Examining the Effects of Malfunctioning Personalized Services on Online Users' Distrust and Behaviors. *Decision Support Systems*, 56(C):180–191, Dec. 2013.

[42] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[43] P. R. Cohen and C. R. Perrault. Elements of a Plan-Based Theory of Speech Acts*. *Cognitive Science*, 3(3):177–212, 1979.

[44] G. Copeland and D. Maier. Making Smalltalk a Database System. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 316–325, Boston, Massachusetts. ACM, 1984.

[45] P. Cremonesi, R. Turrin, E. Lentini, and M. Matteucci. An evaluation methodology for collaborative recommender systems. In *AXMEDIS'08. International Conference on Automated solutions for Cross Media Content and Multi-channel Distribution*, pages 224–231. IEEE, 2008.

[46] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath. The YouTube Video Recommendation System. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 293–296, Barcelona, Spain. ACM, 2010.

[47] B. E. De. *Lateral Thinking (Pelican)*. Penguin UK, 1977.

[48] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04 Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, San Francisco, CA. USENIX Association, 2004.

[49] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[50] D. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[51] T. Di Noia, R. Mirizzi, V. C. Ostuni, D. Romito, and M. Zanker. Linked Open Data to Support Content-based Recommender Systems. In *Proceedings of the 8th International Conference on Semantic Systems*, I-SEMANTICS '12, pages 1–8, Graz, Austria. ACM, 2012.

[52] C. Esswein. *Development of a responsive webapp for a hashtag based PIM system*. Bachelor's Thesis, University of Innsbruck, Nov. 2014. supervised by Wolfgang Gassler, Eva Zangerle, Günther Specht.

[53] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the Web: An experimental study. *Artificial Intelligence*, 165(1):91 –134, 2005.

[54] D. Eynard, L. Mazzola, and A. Dattolo. Exploiting tag similarities to discover synonyms and homonyms in folksonomies. *Software: Practice and Experience*, 43(12):1437–1457, 2013.

[55] A. Fader, S. Soderland, and O. Etzioni. Identifying Relations for Open Information Extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1535–1545, Edinburgh, United Kingdom. Association for Computational Linguistics, 2011.

[56] D. C. Faye, O. Curé, and G. Blin. A survey of RDF storage approaches. *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées*, 15:11–35, 2012.

[57] G. Fischer. User Modeling in Human–Computer Interaction. English. *User Modeling and User-Adapted Interaction*, 11(1-2):65–86, 2001.

[58] T. Fraydenegg. *Development of a light-weight HTML5 Web Client for an Information System adapting Social-Media Concepts.* Bachelor's Thesis, University of Innsbruck, June 2013. supervised by Wolfgang Gassler, Eva Zangerle, Günther Specht.

[59] G. Furnas, T. Landauer, L. Gomez, and S. Dumais. The Vocabulary Problem in Human-System Communication. *Communications of the ACM*, 30(11):971, 1987.

[60] W. Gassler, Zangerle, and G. Specht. The Snoopy Concept: Fighting Heterogeneity in Semistructured and Collaborative Information Systems by using Recommendations. In *The 2011 International Conference on Collaboration Technologies and Systems (CTS 2011)*, pages 61–68, Philadelphia, PE, 2011.

[61] W. Gassler, E. Zangerle, M. Tschuggnall, and G. Specht. SnoopyDB: Narrowing the Gap between Structured and Unstructured Information using Recommendations. In *HT'10, Proceedings of the 21st ACM Conference on Hypertext and Hypermedia, Toronto, Ontario, Canada, June 13-16, 2010*, pages 271–272, 2010.

[62] W. Gassler and E. Zangerle. Recommendation-Based Evolvement of Dynamic Schemata in Semistructured Information Systems. In *Proceedings of the 22nd Workshop Grundlagen von Datenbanken (GvDB 2010), Bad Helmstedt, Germany.* CEUR-WS.org, 2010.

[63] W. Gassler, E. Zangerle, M. Bürgler, and G. Specht. SnoopyTagging: Recommending Contextualized Tags to Increase the Quality and Quantity of Meta-information. In *Proceedings of the 21st International Conference Companion on World Wide Web*, WWW '12 Companion, pages 511–512, Lyon, France. ACM, 2012.

[64] W. Gassler, E. Zangerle, and G. Specht. Guided Curation of Semistructured Data in Collaboratively-built Knowledge Bases. *Journal on Future Generation Computer Systems*, 31:111–119, May 2014. impact factor 1.978.

[65] *Proceedings of the 23rd GI-Workshop "Grundlagen von Datenbanken 2011", Obergurgl, Austria, May 31 - June 03, 2011*, volume 733 of *CEUR Workshop Proceedings*, 2011. CEUR-WS.org.

[66] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[67] A. Halfaker, R. S. Geiger, J. T. Morgan, and J. Riedl. The Rise and Decline of an Open Collaboration System. *American Behavioral Scientist*, 57(5):664–688, 2013.

[68] H. Halpin, V. Robu, and H. Shepherd. The Complex Dynamics of Collaborative Tagging. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 211–220, Banff, Alberta, Canada. ACM, 2007.

[69] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns Without Candidate Generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 1–12, Dallas, Texas, USA. ACM, 2000.

[70] B. Hanrahan, G. Bouchard, G. Convertino, T. Weksteen, N. Kong, C. Archambeau, and E. H. Chi. Mail2Wiki: Low-cost Sharing and Early Curation from Email to Wikis. In *Proceedings of the 5th International Conference on Communities and Technologies*, pages 98–107, Brisbane, Australia. ACM, 2011.

[71] F. M. Harper and J. A. Konstan. The MovieLens Datasets: History and Context. *ACM Transactions on. Interactive Intelligent Systems*, 5(4):19:1–19:19, Dec. 2015.

[72] A. Harth and S. Decker. Optimized index structures for querying RDF from the Web. In *Web Congress, 2005. LA-WEB 2005. Third Latin American*, 10 pp. 2005.

[73] M. Hausenblas and W. Halb. Interlinking of Resources with Semantics. In *5th European Semantic Web Conference*, 2008.

[74] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 230–237, Berkeley, California, USA. ACM, 1999.

[75] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems*, 22(1):5–53, Jan. 2004.

[76] J. Hoffart, F. Suchanek, K. Berberich, E. Lewis-Kelham, G. d. Melo, and G. Weikum. YAGO2: Exploring and Querying World Knowledge in Time, Space, Context, and many Languages. In *Proceedings of the 20th International Conference on World Wide Web*, pages 229–232, Hyderabad, India. ACM, 2011.

[77] D. Hoppe. *Social-media concepts in personal information management systems*. Bachelor's Thesis, University of Innsbruck, June 2012. supervised by Wolfgang Gassler and Eva Zangerle.

[78] I. Horrocks. Semantic Web: the Story so far. In *Proceedings of the 2007 International Cross-Disciplinary Conference on Web Accessibility (W4A)*, pages 120–125. ACM, 2007.

[79] E. Horvitz. Principles of Mixed-Initiative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 159–166. ACM, 1999.

[80] J. Hu, L. Fang, Y. Cao, H.-J. Zeng, H. Li, Q. Yang, and Z. Chen. Enhancing Text Clustering by Leveraging Wikipedia Semantics. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 179–186, Singapore, Singapore. ACM, 2008.

[81] Z. Huang, W. Chung, and H. Chen. A graph model for E-commerce recommender systems. *Journal of the American Society for Information Science and Technology*, 55(3):259–274, 2004.

[82] L. Iaquinta, M. d. Gemmis, P. Lops, G. Semeraro, M. Filannino, and P. Molino. Introducing Serendipity in a Content-Based Recommender System. In *Eighth International Conference on Hybrid Intelligent Systems*, pages 168–173, 2008.

[83] G. Inc. Introducing the Knowledge Graph: things, not strings. *Official Google Blog* http://googleblog.blogspot.co.at/2012/05/introducing-knowledge-graph-things-not.html, 2012. accessed 2017-07-17.

[84] International Bureau of Weights and Measures. *The International System of Units (SI)*. Stedi Media, Paris, 2006.

[85] D. Jannach, P. Resnick, A. Tuzhilin, and M. Zanker. Recommender Systems — Beyond Matrix Completion. *Communications of the ACM*, 59(11):94–102, Oct. 2016.

[86] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems, An Introduction*. Cambridge University Press, 2010. Cambridge Books Online.

[87] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems, Chapter 2 - Collaborative recommendation*. Cambridge University Press, 2010. Cambridge Books Online.

[88] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems, Chapter 3 - Content-based recommendation*. Cambridge University Press, 2010. Cambridge Books Online.

[89] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems, Chapter 4 - Knowledge-based recommendation*. Cambridge University Press, 2010. Cambridge Books Online.

[90] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems, Chapter 5 - Hybrid recommendation approaches*. Cambridge University Press, 2010. Cambridge Books Online.

[91] D. Jurafsky and J. Martin. *Speech and Language Processing*. Prentice Hall, 2nd edition, 2008.

[92] M. Kaminskas and F. Ricci. Contextual music information retrieval and recommendation: State of the art and challenges. *Computer Science Review*, 6(2–3):89 –119, 2012.

[93] A. Kemper and T. Neumann. HyPer: A hybrid OLTP & OLAP main memory database system based on virtual memory snapshots. In *IEEE 27th International Conference on Data Engineering (ICDE)*, pages 195–206, 2011.

[94] A. Kittur, E. Chi, B. Pendleton, B. Suh, and T. Mytkowicz. Power of the few vs. wisdom of the crowd: Wikipedia and the rise of the bourgeoisie. *World Wide Web*, 1(2):19, 2007.

[95] G. Klyne, J. Carroll, and B. McBride. Resource Description Framework (RDF). *W3C recommendation*, 2004.

[96] A. Kobsa. Generic User Modeling Systems. English. *User Modeling and User-Adapted Interaction*, 11(1-2):49–63, 2001.

[97] N. Kong, B. Hanrahan, T. Weksteen, G. Convertino, and E. H. Chi. VisualWikiCurator: human and machine intelligence for organizing wiki content. In *Proceedings of the 16th International Conference on Intelligent User Interfaces*, IUI '11, pages 367–370, Palo Alto, CA, USA. ACM, 2011.

[98] Y. Koren. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 426–434, Las Vegas, Nevada, USA. ACM, 2008.

[99] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, Aug. 2009.

[100] M. Krötzsch, D. Vrandečić, and M. Völkel. *Semantic MediaWiki*. In *The Semantic Web - ISWC 2006: 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006. Proceedings*. Springer Berlin Heidelberg, 2006, pages 935–942.

[101] M. Krötzsch, D. Vrandecic, D. Vr, and M. Völkel. Wikipedia and the Semantic Web - The Missing Links. In *Proceedings of Wikimania*, 2005.

[102] C. Laqua. *Development of a responsive Android client for a hashtag based PIM system*. Bachelor's Thesis, University of Innsbruck, Nov. 2014. supervised by Eva Zangerle, Wolfgang Gassler, Günther Specht.

[103] A. Larcher, E. Zangerle, W. Gassler, and G. Specht. Key Recommendations for Infoboxes in Wikipedia. Website of the 22nd ACM Conference on Hypertext and Hypermedia, 2011. Poster Presentation.

[104] O. Lassila and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, W3C, 1999.

[105] C. Lécluse, P. Richard, and F. Vélez. O2, an Object-Oriented Data Model. In *Proceedings of the International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '88, pages 556–562, London, UK. Springer-Verlag, 1988.

[106] M. Levene and A. Poulovanssilis. An Object-oriented Data Model Formalised Through Hypergraphs. *Data & Knowledge Engineering*, 6(3):205–224, May 1991.

[107] G. Linden, B. Smith, and J. York. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7:76–80, 2003.

[108] M. Lipczak. Tag recommendation for folksonomies oriented towards individual users. *European Conference on Machine Learning & Principles and Practice of Knowledge Discovery in Databases Discovery Challenge*:84–95, 2008.

[109] M. Lux, M. Granitzer, and R. Kern. Aspects of Broad Folksonomies. In *DEXA '07. 18th International Workshop on Database and Expert Systems Applications*, pages 283–287, 2007.

[110] C. Macfarquhar and G. Gleig. *Encyclopædia britannica: or, A dictionary of arts, sciences, and miscellaneous literature*. A. Bell and C. Macfarquhar, 1797.

[111] A. Majchrzak, C. Wagner, and D. Yates. Corporate wiki users: results of a survey. In *WikiSym '06: Proceedings of the 2006 International Symposium on Wikis*, pages 99–104, Odense, Denmark. ACM, 2006.

[112] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*. Cambridge university press Cambridge, 2008.

[113] J. Markoff. Computer Wins on 'Jeopardy!': Trivial, It's Not. *The New York Times*, http://www.nytimes.com/2011/02/17/science/17jeopardy-watson.html, Feb. 2011. accessed 2017-07-17.

[114] G. Miller. WordNet: a Lexical Database for English. *Communications of the ACM*, 38:39–41, 11, 1995.

[115] G. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[116] R. B. Miller. Response Time in Man-computer Conversational Transactions. In *Proceedings of the Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 267–277, San Francisco, California. ACM, 1968.

[117] D. Milne, O. Medelyan, and I. H. Witten. Mining Domain-Specific Thesauri from Wikipedia: A Case Study. In *Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on*, pages 442–448, 2006.

[118] D. Milne, O. Medelyan, and I. H. Witten. Mining Domain-Specific Thesauri from Wikipedia: A Case Study. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '06, pages 442–448, Washington, DC, USA. IEEE Computer Society, 2006.

[119] B. Mobasher. Recommender Systems. *Künstliche Intelligenz, Special Issue on Web Mining*, 3:41–43, 2007.

[120] G. E. Modoni, M. Sacco, and W. Terkaj. A survey of RDF store solutions. In *2014 International Conference on Engineering, Technology and Innovation (ICE)*, pages 1–7, 2014.

[121] M. Müller. *Development of a self-learning recommendation module for a hashtag based PIM system*. Bachelor's Thesis, University of Innsbruck, Nov. 2013. supervised by Wolfgang Gassler, Eva Zangerle, Günther Specht.

[122] N. Nakashole, M. Theobald, and G. Weikum. Scalable Knowledge Harvesting with High Precision and High Recall. In *Proceedings of the fourth International Conference on Web Search and Data Mining*, pages 227–236, Hong Kong, China. ACM, 2011.

[123] K. Nakayama, T. Hara, and S. Nishio. *Wikipedia Mining for an Association Web Thesaurus Construction*. In *Web Information Systems Engineering – WISE 2007: 8th International Conference on Web Information Systems Engineering Nancy, France, December 3-7, 2007 Proceedings*. Springer Berlin Heidelberg, 2007, pages 322–334.

[124] J. Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[125] J. Noessner, M. Niepert, C. Meilicke, and H. Stuckenschmidt. *Leveraging Terminological Structure for Object Reconciliation.* In *The Semantic Web: Research and Applications: 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 – June 3, 2010, Proceedings, Part II.* Springer Berlin Heidelberg, 2010, pages 334–348.

[126] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 39–42, San Jose, CA, USA, 2007.

[127] C. R. Perrault, J. F. Allen, and P. R. Cohen. Speech Acts As a Basis for Understanding Dialogue Coherence. In *Proceedings of the 1978 Workshop on Theoretical Issues in Natural Language Processing*, TINLAP '78, pages 125–132, Urbana-Champaign, Illinois. Association for Computational Linguistics, 1978.

[128] I. Peters. *Folksonomies. Indexing and Retrieval in Web 2.0.* Walter de Gruyter & Co., Hawthorne, NJ, USA, 2009.

[129] F. Pfisterer, M. Nitsche, A. Jameson, and C. Barbu. User-Centered Design and Evaluation of Interface Enhancements to the Semantic MediaWiki. In *Proceedings of the 5th International Workshop on Semantic Web User Interaction (SWUI 2008), Florence, Italy, April 5*, 2009.

[130] J. Poissonnier. *Extraction of Semi-Structured Online Data.* Bachelor's Thesis, University of Innsbruck, May 2012. supervised by Eva Zangerle and Wolfgang Gassler.

[131] A. Poulovassilis and M. Levene. A Nested-graph Model for the Representation and Manipulation of Complex Objects. *ACM Transactions on Information Systems*, 12(1):35–68, Jan. 1994.

[132] D. J. Power. *DSS News, A Bi-Weekly Publication of DSSResources.COM,* `http://www.dssresources.com/newsletters/66.php`, Nov. 2002. accessed 2017-07-17.

[133] R. Priedhorsky, J. Chen, S. K. Lam, K. Panciera, L. Terveen, and J. Riedl. Creating, Destroying, and Restoring Value in wikipedia. In *Proceedings of the 2007 International Conference on Supporting Group Work*, pages 259–268. ACM, 2007.

[134] D. Pritchett. BASE: An Acid Alternative. *Queue*, 6(3):48–55, May 2008.

[135] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, W3C, 2008.

[136] P. Pu, L. Chen, and R. Hu. A User-centric Evaluation Framework for Recommender Systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 157–164, Chicago, Illinois, USA. ACM, 2011.

[137] S. Rendle. Factorization Machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 995–1000, Washington, DC, USA. IEEE Computer Society, 2010.

[138] P. Resnick and H. Varian. Recommender Systems. *Communications of the ACM*, 40(3):58, 1997.

[139] *Recommender Systems Handbook*. Springer, Berlin, Heidelberg, New York, 2011.

[140] E. Rich. User Modeling via Stereotypes*. *Cognitive Science*, 3(4):329–354, 1979.

[141] M. A. Rodriguez and P. Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 2010.

[142] S. Sakr and G. Al-Naymat. Relational Processing of RDF Queries: A Survey. *ACM SIGMOD Record*, 38(4):23–28, June 2010.

[143] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, Nov. 1975.

[144] T. Sampson. Nobody wants to edit Wikipedia anymore. *The Daily Dot* http://www.dailydot.com/business/wikipedia-editors-decline-wikimedia-fellows/, 2013. accessed 2017-07-17.

[145] T. Sampson. Will Wikipedia's pretty new editing software solve its recruitment crisis? *The Daily Dot* http://www.dailydot.com/business/wikipedia-visual-editor-wysiwyg/, 2013. accessed 2017-07-17.

[146] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Application of Dimensionality Reduction in Recommender System – A Case Study. In *ACM WebKDD Workshop in conjunction with the ACM-SIGKDD Conference on Knowledge Discovery in Databases*, 2000.

[147] P. Schäuble. SPIDER: a Multiuser Information Retrieval System for Semistructured and Dynamic Data. In *Proceedings of the 16th annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–327, Pittsburgh, Pennsylvania, United States. ACM, 1993.

[148] M. Schedl, P. Knees, B. McFee, D. Bogdanov, and M. Kaminskas. *Music Recommender Systems*. In *Recommender Systems Handbook*. Springer US, Boston, MA, 2015, pages 453–492.

[149] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 253–260. ACM, 2002.

[150] M. Schmakeit. *Extension and Optimization of the Hash5-Server - a Hashtag-based PIM-System*. Bachelor's Thesis, University of Innsbruck, Jan. 2015. supervised by Eva Zangerle, Wolfgang Gassler, Günther Specht.

[151] G. Shani and A. Gunawardana. *Evaluating Recommendation Systems*. In *Recommender Systems Handbook*. Springer US, Boston, MA, 2011, pages 257–297.

[152] B. Shao, D. Wang, T. Li, and M. Ogihara. Music Recommendation Based on Acoustic Features and User Access Patterns. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(8):1602–1611, 2009.

[153] P. Shvaiko and J. Euzenat. A Survey of Schema-based Matching Approaches. *Journal on Data Semantics*, 4:146–171, 2005.

[154] T. Simonite. The Decline of Wikipedia. *MIT Technology Review* https://www.technologyreview.com/s/520446/the-decline-of-wikipedia/, Oct. 2013. accessed 2017-07-17.

[155] B. Smith and G. Linden. Two Decades of Recommender Systems at Amazon.com. *IEEE Internet Computing*, 21(3):12–18, 2017.

[156] B. Smyth, M. Coyle, P. Briggs, K. McNally, and M. P. O'Mahony. Collaboration, Reputation and Recommender Systems in Social Web Search. In *Recommender Systems Handbook*, pages 569–608. Springer US, Boston, MA, 2015.

[157] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The End of an Architectural Era: (It's Time for a Complete Rewrite). In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 1150–1160, Vienna, Austria. VLDB Endowment, 2007.

[158] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: A Column-oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 553–564, Trondheim, Norway. VLDB Endowment, 2005.

[159] M. Strube and S. P. Ponzetto. WikiRelate! Computing Semantic Relatedness Using Wikipedia. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, pages 1419–1424, Boston, Massachusetts. AAAI Press, 2006.

[160] F. Suchanek, M. Sozio, and G. Weikum. SOFIE: a Self-organizing Framework for Information Extraction. In *Proceedings of the 18th International Conference on World Wide Web*, pages 631–640, Madrid, Spain. ACM, 2009.

[161] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, Banff, Alberta, Canada. ACM, 2007.

[162] B. Suh, G. Convertino, E. Chi, and P. Pirolli. The Singularity is not near: slowing rowth of Wikipedia. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, page 8. ACM, 2009.

[163] M. Taramigkou, E. Bothos, K. Christidis, D. Apostolou, and G. Mentzas. Escape the Bubble: Guided Exploration of Music Preferences for Serendipity and Novelty. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 335–338, Hong Kong, China. ACM, 2013.

[164] Y. Theoharis, V. Christophides, and G. Karvounarakis. *Benchmarking Database Representations of RDF/S Stores*. In *The Semantic Web – ISWC 2005: 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005. Proceedings*. Springer Berlin Heidelberg, 2005, pages 685–701.

[165] A. Töscher, M. Jahrer, and R. M. Bell. The BigChaos Solution to the Netflix Grand Prize, 2009. accessed 2017-07-17.

[166] F. Vignoli and S. Pauws. A music retrieval system based on user-driven similarity and its evaluation. In *Proceedings International Symposium on Music Information Retrieval*, pages 272–279, 2005.

[167] M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer. Semantic Wikipedia. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 585–594, Edinburgh, Scotland. ACM, 2006.

[168] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk–a Link Discovery Framework for the Web of Data. In *Proceedings of the 2nd Linked Data on the Web Workshop*, 2009.

[169] D. Vrandečić. Wikidata: a new platform for collaborative data collection. In *Proceedings of the 21st International Conference Companion on World Wide Web*, WWW '12 Companion, pages 1063–1064, Lyon, France. ACM, 2012.

[170] D. Vrandečić and M. Krötzsch. Wikidata: A Free Collaborative Knowledgebase. *Communications of the ACM*, 57(10):78–85, Sept. 2014.

[171] G. Weikum, G. Kasneci, M. Ramanath, and F. Suchanek. Database and Information-Retrieval Methods for Knowledge Discovery. *Communications of the ACM*, 52(4):56–64, 2009.

[172] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment*, 1(1):1008–1019, Aug. 2008.

[173] D. Weld, F. Wu, E. Adar, S. Amershi, J. Fogarty, R. Hoffmann, K. Patel, and M. Skinner. Intelligence in Wikipedia. In *Proceedings of the 23rd national Conference on AI*, pages 1609–1614, Chicago, Illinois. AAAI Press, 2008.

[174] S. Whittaker. Personal information management: From information consumption to curation. *Annual Review of Information Science and Technology*, 45(1):1–62, 2011.

[175] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[176] M. Wolf. *Extension of Information Types for an Android Client for a Hashtag-based PIM System*. Bachelor's Thesis, University of Innsbruck, Oct. 2014. supervised by Eva Zangerle, Wolfgang Gassler, Günther Specht.

[177] D. Wood. Kowari: A Platform for Semantic Web Storage and Analysis. In *XTech 2005 Conference*, 2005.

[178] F. Wu, R. Hoffmann, and D. Weld. Information Extraction from Wikipedia: Moving down the Long Tail. In *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 731–739. ACM, 2008.

[179] F. Wu and D. S. Weld. Automatically Refining the Wikipedia Infobox Ontology. In *WWW '08: Proceeding of the 17th International Conference on World Wide Web*, pages 635–644, Beijing, China. ACM, 2008.

[180] F. Wu and D. Weld. Autonomously Semantifying Wikipedia. In *Proceedings of the sixteenth ACM Conference on Information and Knowledge Management*, pages 41–50. ACM, 2007.

[181] V. Zanardi and L. Capra. Dynamic Updating of Online Recommender Systems via Feed-forward Controllers. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 11–19, Waikiki, Honolulu, USA. ACM, 2011.

[182] E. Zangerle, W. Gassler, and G. Specht. Recommending Structure in Collaborative Semistructured Information Systems. In *Proceedings of the fourth ACM Conference on Recommender Systems*, pages 261–264, Barcelona, Spain. ACM, 2010.

[183] E. Zangerle and W. Gassler. Dealing with Structure Heterogeneity in Semantic Collaborative Environments. In *Collaboration and the Semantic Web: Social Networks, Knowledge Networks and Knowledge Resources*. IGI Publishers, Hershey, Pennsylvania (USA), 2012.

[184] E. Zangerle, W. Gassler, M. Pichl, S. Steinhauser, and G. Specht. An Empirical Evaluation of Property Recommender Systems for Wikidata and Collaborative Knowledge Bases. In *Proceedings of the 12th International Symposium on Open Collaboration, OpenSym 2016, Berlin, Germany, August 17-19, 2016*, 18:1–18:8. ACM, 2016.

[185] E. Zangerle, W. Gassler, and G. Specht. Exploiting Twitter's Collective Knowledge for Music Recommendations. In *Proceedings, 2nd Workshop on Making Sense of Microposts (#MSM2012): Big things come in small packages, Lyon, France, 16 April 2012*, pages 14–17, 2012.

[186] E. Zangerle, W. Gassler, and G. Specht. Recommending #-tags in Twitter. In *Proceedings of the Workshop on Semantic Adaptive Social Web 2011 in connection with the 19th International Conference on User Modeling, Adaptation and Personalization, UMAP 2011*, pages 67–78, Gerona, Spain. CEUR-WS.org, 2011.

[187] E. Zangerle, W. Gassler, and G. Specht. *Using Tag Recommendations to Homogenize Folksonomies in Microblogging Environments.* In *Social Informatics: Third International Conference, SocInfo 2011, Singapore, October 6-8, 2011. Proceedings.* Springer Berlin Heidelberg, 2011, pages 113–126.

[188] E. Zangerle, W. Gassler, and G. Specht. On the impact of text similarity functions on hashtag recommendations in microblogging environments. English. *Social Network Analysis and Mining*, 3(4):889–898, 2013.

[189] E. Zangerle, M. Pichl, W. Gassler, and G. Specht. #nowplaying Music Dataset: Extracting Listening Behavior from Twitter. In *Proceedings of the 1st ACM International Workshop on Internet-Scale Multimedia Management*, ISMM '14, pages 21–26, Orlando, Florida, USA. ACM, June 2014.

[190] Y. C. Zhang, D. O. Séaghdha, D. Quercia, and T. Jambor. Auralist: Introducing Serendipity into Music Recommendation. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 13–22, Seattle, Washington, USA. ACM, 2012.

[191] A. Zubiaga. Enhancing Navigation on Wikipedia with Social Tags. *Computing Research Repository (CoRR)*, abs/1202.5469, 2012.