Jeronimo Sirotheau de Almeida Eichler

Exploring RDF Knowledge Bases through Serendipity Patterns

TESE DE DOUTORADO

Thesis presented to the Programa de Pós-Graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências - Informática

Advisor: Prof. Marco Antonio Casanova

Rio de Janeiro
August 2018

Jeronimo Sirotheau de Almeida Eichler

Exploring RDF Knowledge Bases through Serendipity Patterns

Thesis presented to the Programa de Pós-Graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências – Informática. Approved by the undersigned Examination Committee.

Rio de Janeiro, August 21st, 2018

Jeronimo Sirotheau de Almeida Eichler

Jeronimo Sirotheau de Almeida Eichler holds a master in computer science degree from Pontifical Catholic University of Rio de Janeiro (PUCRio), and a Bachalor degree also in computer science from Universidade Estadual do Rio de Janeiro (UERJ). Jeronimo has a solid working experience with system development and architectural design for Web plataforms. His main research topics areas include Semantic Web, Information Retrieval, Data Mining and Machine Learning.

Bibliographic data

Eichler, Jeronimo Sirotheau de Almeida

Exploring RDF Knowledge Bases through Serendipity Patterns / Jeronimo Sirotheau de Almeida Eichler; advisor: Marco Antonio Casanova. – 2018.

82 f. : il. ; 30 cm

Tese (Doutorado em Informática)–Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2018.

Inclui bibliografia

1. Informática – Teses. 2. Dados Interligados. 3. Fortuidade. 4. Web Semântica. I. Casanova, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

I dedicate this thesis to the most important people in my life:
My parents and my brother.

Acknowledgments

First and foremost, I would like to express my deep gratitude to my advisor, Prof. Marco Antonio Casanova, who guided me across the entire path of this research. Thank you for sharing your wisdom and also your total support, sense of humor, inspiring ideas and kindness, not to mention your fantastic classes. It was an honor and a joy for me to work with you in the last four years.

I would like to thank Prof. Antonio Furtado, who was always available to help and collaborate. Thanks for sharing your enthusiasm with the Serendipity subject as well as generously sharing your thoughts and your time.

I would like to extend my appreciation and gratitude to the Department of Informatics of PUC-Rio. Coursing DI disciplines was a great opportunity for exploring and validating some of the topics that this thesis addresses. Studying at PUC-Rio was a determinant step in my professional life.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

Last, but not least important, my acknowledgements would not be completed without giving special thanks to my family. To my father Rodolfo and my mother Patricia, for the endless support, comprehension and care. To my brother Bruno, for being so supportive and always keeping me motivated. None of this would be possible without you. I love you all, thanks.

Abstract

Eichler, Jeronimo Sirotheau de Almeida; Casanova, Marco Antonio (advisor). **Exploring RDF Knowledge Bases through Serendipity Patterns.** Rio de Janeiro, 2018. 82p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Serendipity is defined as the discovery of a thing when one is not searching for it. In other words, serendipity means the discovery of information that provides valuable insights by unveiling unanticipated knowledge. The topic is receiving increased attention in the literature, since the precision requirement may be justifiably relaxed in order to improve user satisfaction. A field that can benefit from serendipity is the Web of Data, an immense global data space where data is publicly available. As more and more data become available in this data space, searching and extracting relevant information becomes a challenging task. This thesis contributes to addressing this challenge in two ways. First, it presents a query orchestration process that introduces three strategies to inject serendipity patterns in the query process. The serendipity patterns are inspired by basic characteristics of serendipitous events, such as, analogy and disturbance, and can be used for augmenting the results with additional information, suggesting alternative queries or rebalancing the results. Second, it introduces a benchmark dataset that can be used to compare different approaches for locating serendipitous content. The strategy adopted for constructing the dataset consists of dividing the dataset into partitions based on a global feature and linking entities from different partitions according to the number of paths they share.

Keywords

Linked Data; Serendipity; Information Retrieval; Data Mining.

Resumo

Eichler, Jeronimo Sirotheau de Almeida; Casanova, Marco Antonio (orientador). **Explorando Bases de Conhecimento em RDF através de Padrões de Fortuidade.** Rio de Janeiro, 2018. 82p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Fortuidade pode ser definida como a descoberta de algo que não está sendo buscado. Em outras palavras, fortuidade trata da descoberta de informação que provê valiosas intuições ao desvendar conhecimento inesperado. O tópico vem recebendo bastante atenção na literatura, uma vez que precisão pode ser justificadamente relaxada com o objetivo de aumentar a satisfação do usuário. Uma área que pode se beneficiar com fortuidade é a área de dados interligados, um gigantesco espaço de dados no qual dados são disponibilizados publicamente. Buscar e extrair informação relevante se torna uma tarefa desafiadora à medida que cada vez mais dados se tornam disponíveis nesse ambiente. Esta tese contribui para enfrentar este desafio de duas maneiras. Primeiro, apresenta um processo de orquestração de consulta que introduz três estratégias para injetar padrões de fortuidade no processo de consulta. Os padrões de fortuidade são inspirados em características básicas de eventos fortuitos, como analogia e perturbação, e podem ser usados para estender os resultados com informações adicionais, sugerindo consultas alternativas ou reordenando os resultados. Em segundo lugar, introduz uma base de dados que pode ser utilizada para comparar diferentes abordagens de obtenção de conteúdo fortuito. A estratégia adotada para construção dessa base de dados consiste em dividir o universo de dados em partições com base em um atributo global e conectar entidades de diferentes partições de acordo com o número de caminhos compartilhados.

Palavras-chave

Dados Interligados; Fortuidade; Aquisição de Informação; Mineração de Dados.

Table of Contents

List of Figures

List of Tables

He who does not expect will not find out the unexpected,
for it is trackless and unexplored.
Heraclitus of Ephesus

1 Introduction

1.1.Context and Motivation

In recent years, the World Wide Web has evolved from a global information space of linked documents to a space where data can also be linked. This new paradigm of Web of Data, in addition to the Web of Documents, provides ways to make data more accessible and build knowledge from combined data. From the user’s perspective, the main goal of Linked Data is the provision of integrated access to data from a wide range of distributed and heterogeneous data sources (Bizer et al. 2009).

To support these goals, the Linked Data initiative (Berners-Lee 2009) has emerged as a set of best practices that aims at standardizing the linking data process. Thanks to this, an unprecedented amount of linked data sources was recently produced, and continues growing fast covering diverse domains - such as geographic locations, people, companies, books, films, music, statistical data and scientific publications (Heath & Bizer 2011).

At the time of this writing, the Linked Open Data cloud (LOD cloud), a repository for interlinked Linked Data datasets, contains 1,184 datasets and aggregates more than 198 trillion triples (Abele et al. 2017), as depicted in Figure 1. DBpedia[1], the Linked Data version of Wikipedia repository, contains alone almost 10 trillion triples (Abele et al. 2017).

[1] http://wiki.dbpedia.org/

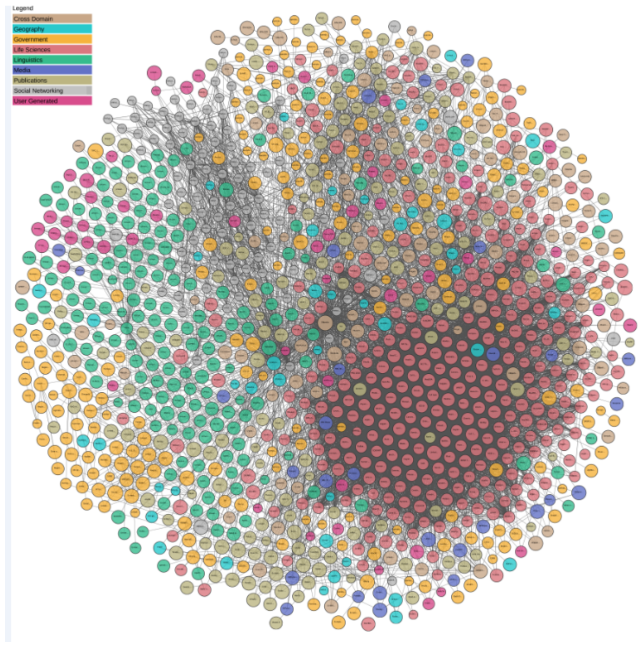


Figure 1: LOD cloud diagram[2] in 2017

As a consequence of this initial progress, a new challenge arose. With the proliferation of Linked Data datasets and more and more data becoming available, filtering these large datasets in order to support a goal becomes a challenging task.

Similarly to the Web of Documents, in the Web of Data, the overall approach adopted by search applications is to locate resources that are strongly related to the user's needs. Thereby, the main goal is to maximize the accuracy among the search results.

However, studies (Murakami et al. 2007, Ge et al. 2010) argue that this strategy alone may conduct the user into a new set of problems and, thereby, decrease the user satisfaction. For instance, an application that only considers accuracy can imprison the user in an information bubble where he is only exposed to information of a certain niche or, even worse, to a kind of information he already knows. These studies argue that other factors, such as surprise and discovery, can improve user satisfaction even at the cost of a small loss in precision. For this reason, these studies suggest that other metrics, such as serendipity, ought to be considered in order to analyze the user satisfaction.

Serendipity is receiving increased attention in the literature as a process of breakthrough discovery caused by chance encounters (André 2009a). According to André (2009a), serendipity involves the surprise of finding

[2] http://lod-cloud.net/

something unexpected, and the sagacity necessary to unveil an unexpected connection.

This way, strategies that induce serendipitous suggestion can improve user satisfaction by challenging the user to pursue new directions and discover further information.

1.2.Goal and Contributions

This thesis reports contributions to Linked Data search. More specifically, we adopt serendipitous concepts in order to suggest unexpected items and better support user's goals.

Our first contribution is a query orchestration process that describes different strategies to adapt the query execution with the objective of providing a set of behaviors in order to produce a more complete response for the user's goals.

A second contribution of our thesis is the definition and formalization of a set of serendipity patterns in the context of Linked Data search. These serendipity patterns capture serendipitous connections on live Linked Data datasets.

The first and second contributions are validated with SOL-Tool, a Linked Data application that implements theirs concepts and ideas. The experimental results present a promising score of 90% of unexpectedness for real-world scenarios in the music domain.

Our third contribution is a benchmark creation process that defines the necessary steps for building a dataset that exploits Linked Data resources. The steps are structured and can be extended to capture different particularities of the retrieved data, the given domain or the benchmark goal. The dataset construction process also discusses the main challenges and design decisions that impact the dataset creation.

Our fourth contribution is a benchmark created to support the evaluation of approaches that present serendipitous suggestions for the movies domain, the Serendipity Movie Test Dataset. In order to create the suggestions, the benchmark considers entities, graphs and paths extracted from the LOD cloud that pertain to the movies domain.

An additional contribution of this thesis is a related work overview of the state-of-the-art of the strategies and applications that use serendipity to support users' goals.

1.3.Structure of the Thesis

The remainder of this thesis is structured as follows. In Chapter 2, we introduce the background concepts that are used throughout the thesis. More specifically, we first present Linked Data definition and examples and then we review the serendipity literature by discussing the notion of serendipity and examining a set of serendipity patterns. In Chapter 3, we describe a query orchestration process that enables the customization of different phases of the query execution. Still in Chapter 3, we introduce three strategies to inject serendipity in the query process. In Chapter 4, we formalize a set of serendipity patterns to capture serendipity in the context of Linked Data search. In Chapter 5, we describe the SOL-Tool application, which implements the query orchestration process and encompasses a set of the serendipity patterns. Also, in Chapter 5, we present the experiments with the tool. In Chapter 6, we describe the Serendipity Movie Test Dataset, a benchmark dataset that can be used to compare different approaches for locating serendipitous content. Still in Chapter 6, we present the necessary steps for building the dataset. In Chapter 7, we provide an overview of the state-of-the-art in the field by combining the topics of serendipity and Linked Data search engines. Finally, in Chapter 8, we draw the conclusions and indicate opportunities of future work.

2 Background

2.1. Linked Data Background

Since the projects presented in this thesis use data extracted from the LOD cloud, we start by recalling a few concepts related to the *Resource Description Framework* (RDF) data model (Cyganiak et al. 2014) and SPARQL query language (Harris & Seaborne 2013).

A *Uniform Resource Identifier* (URI) represents an *entity* of the real world. A *literal* is a string representing a (datatype) value. An RDF *term* is a URI or a literal. Table 1 depicts examples of URIs from The Beatles members in DBpedia.

Table 1: URI examples

URI
http://dbpedia.org/resource/John_Lennon
http://dbpedia.org/resource/Paul_McCartney
http://dbpedia.org/resource/George_Harrison
http://dbpedia.org/resource/Ringo_Starr

The RDF data model allows shortening a URI reference by declaring a namespace that depicts the set of URIs in a vocabulary. For example, by describing the DBpedia resources namespace as *dbr*, one may refer to `http://dbpedia.org/resource/John_Lennon` entity as `dbr:John_Lennon`. This is not a mandatory requirement but we adopt the notation in this study for a matter of readability.

Table 2 lists the namespaces and vocabularies that are referenced in this study. The sol namespace presents terms that are declared in this thesis.

Table 2: Namespaces

Namespace	Vocabulary
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
owl	http://www.w3.org/2002/07/owl#
dct	http://purl.org/dc/terms/
skos	http://www.w3.org/2004/02/skos/core#
dbr	http://dbpedia.org/resource/
dbc	http://dbpedia.org/resource/Category:
dbo	http://dbpedia.org/ontology/
dbp	http://dbpedia.org/property/
mdb	http://data.linkedmdb.org/resource/
movie	http://data.linkedmdb.org/resource/film/
foaf	http://xmlns.com/foaf/0.1/
sol	http://soltool.com/

Entities are typically assigned to *classes*, which may in turn be organized as a *class hierarchy*. This is captured in RDF with the help of the predefined terms `rdf:type`, `rdfs:Class` and `rdfs:subclassOf`, where the first term belongs to the RDF vocabulary and the last two terms to the RDF Schema vocabulary. The term `owl:Thing` of the OWL vocabulary denotes the universe, i.e., the set of all things.

An RDF triple is a statement (*s,p,o*), where *s* and *p* are URIs and *o* is either a URI or a literal; a triple (*s,p,o*) states that its *subject s* has *property p* whose value is *object o*. We disregard the so-called *blank nodes*, which could always be replaced by Skolem URIs (Cyganiak et al. 2014).

We also take into consideration the annotation property `rdfs:seeAlso` and the OWL property `owl:sameAs`. `rdfs:seeAlso` is used to indicate an entity that might provide additional information about the subject entity, whereas the `owl:sameAs` property is used to indicate that two URI references refer to the same thing i.e. they represent the same real-world object.

Table 3 illustrates examples of RDF triples declared in DBpedia.

Table 3: RDF triple examples

Subject	Property	Object
dbr:John_Lennon	dct:subject	dbc:English_rock_guitarists
dbr:John_Lennon	dct:subject	dbc:English_rock_singers
dbr:Imagine:_John_Lennon	dbo:musicComposer	dbr:John_Lennon
dbr:Paul_McCartney	dct:subject	dbc:English_rock_guitarists
dbr:Paul_McCartney	dct:subject	dbc:English_rock_singers
dbr:Spies_Like_Us	dbo:musicComposer	dbr:Paul_McCartney
dbr:Ringo_Starr	dct:subject	dbc:English_rock_drummers

Figure 2 is a graphic representation of the data presented in Table 2. We use the Box-Arrow Notation to represent the RDF triples, with boxes representing subjects and objects of RDF triples, while arrows represent properties of RDF triples.

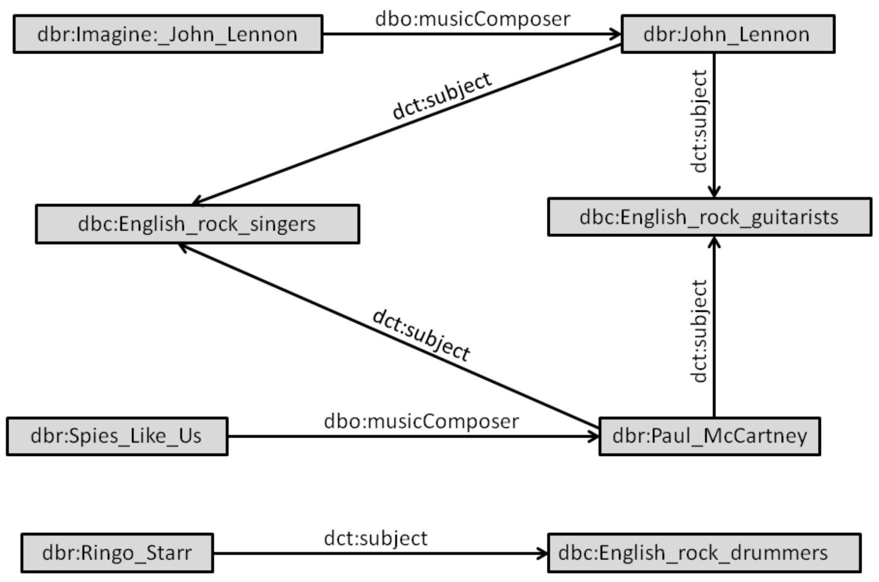


Figure 2: RDF triple examples

We use the SPARQL query language (Harris & Seaborne 2013) to access datasets. A SPARQL query is composed of a set of triple patterns (?*s*,?*p*,?*o*) where ?*s* and ?*p* can be URIs or variables and ?*o* may be either a URI, a literal or a variable. If a triple matches all the triple patterns of a SPARQL query, it is understood as a solution for that query. Furthermore, the solutions of a SPARQL query can be projected into a list of variables, for a select query, or a list of triples, for a construct query.

The SPARQL query below selects all entities that are presented in a triple which property matches `dct:subject` and object matches `dbc:English_rock_singers`. Therefore, according to Table 3, the entities `dbr:Paul_McCartney` and `dbr:John_Lennon` are to be presented among the query results.

Selecting English rock singers

```
PREFIX dct:      <http://purl.org/dc/terms/>
PREFIX dbc:      <http://dbpedia.org/resource/Category:>

SELECT distinct ?subject WHERE{
      ?subject dct:subject dbc:English_rock_singers.
}
```

2.2. Serendipity Background

Serendipity is defined as “the art of making an unsought finding” (Van Andel 1994). The term was coined by Horace Walpole, based on the tale of *The Three Princes of Serendip*, wherein the mentioned princes made several discoveries of things they were not looking for by accident and sagacity. In the literature, the term serendipity is used to describe a breakthrough discovery caused by chance encounters (André 2009a). As described in (André 2009a), there are two key aspects of serendipity: the accidental nature and the surprise of finding something unexpected, the *chance*; the breakthrough or discovery made by drawing an unexpected connection, the *sagacity*. That is, serendipity promotes the encounter of unexpected information to provide valuable insights by unveiling previously unknown knowledge.

The topic is receiving increased attention in the literature, since the precision requirement may be justifiably relaxed in favor of extended recall if the extra information supports the searcher’s current or latent goals.

2.2.1. The Information Encounter Experience

In her study of accidental discovery of information, Erdelez (1999) described four elements involved in the information encounter experience: the *encounterer*, the *environment*, the characteristics of the *information encountered*, the *information need*.

The *encounterer* is defined as the user who experiences an accidental discovery of information. Erdelez (1999) also defined levels of encounterers ranging from non-encounterers to super-encounterers, where non-encounterers are users that present difficulty in finding information, while super-encounterers are users who find information more easily. Although the notion typically represents people, we argue that an encounterer can also be understood as an agent or application that discovers the information.

The *environment* is the place where the information discovery occurs (Erdelez 1999). Similarly to the *encounterer* elements, the environments may differ in levels of information capabilities. In other words, some environments may be more propitious for information discovery than others.

According to (Erdelez 1999), the information accidentally encountered can be classified as problem-related or interest-related. Problem-related information belongs to a user's specific problem, but not a problem that the user is pursuing at the time the information was encountered. Therefore, problem-related information supports latent goals (De Bruijn & Spence 2008), a topic that is later discussed in this thesis. On the other hand, interest-related information depicts information that the user would probably not search beforehand in spite of its potential use.

The information encountering experience can also be classified by the relation between the information found and the need that it addresses. For example, information may lose its relevance with time, e.g., a weather condition.

This way, a serendipity event may be enhanced or even induced by favoring the information encounter experience factors. Thus, a critical factor for achieving serendipity is the ability of the encounterer to draw connections between pieces of information to build new information, in other words, the encounterer's sagacity (André 2009a). To support this goal, the encounterer must have the ability to compare models from different domains and recognize similar concepts. We adopt similar strategy to capture the serendipity patterns in our query orchestration process.

Due to its characteristics, the Web of Data with networks of data to be navigated and explored represents an especially auspicious environment for serendipity events.

2.2.2. Serendipity Patterns

In an extensive study of serendipity, Van Andel (1994) lists seventeen serendipity patterns, each one representing a different form in which serendipity can occur. In this section, we concentrated on the patterns that we found to be best amenable to be captured in the context of Linked Data search.

The *analogy pattern* is characterized by seeking similarity between objects from the same or totally distinct domains (Van Andel 1994). Analogy is also defined as seeing (or describing) something in terms of something else (Burke 1941). Basically, it consists of extracting relevant characteristics of an object in order to apply this knowledge to identify another object. A widely popular example of analogy is the insight of Archimedes to measure a crown’s volume after stepping into a bathtub.

The *surprising observation pattern* is characterized by surprise caused by an unexpected event. It indicates a trail that can lead to new information about a known entity and represents the fact that some entities can have different facets (or views) covering different domains. A subpattern of surprising observation is the *repetition of surprising observation*. As the name implies, it involves the recurrence of the previous pattern and serves as a strong indication of the relevance of the respective observation. To illustrate the repetition of the surprising observation pattern, Van Andel (1994) cites the discovery of AIDS as an epidemic after registering a high number of cases.

The *inversion* pattern changes the expectation of the experiment, guiding the solution towards a completely new direction. It establishes a breakthrough discovery where the insight is the opposite to the previous intent. To illustrate the Inversion pattern, Van Andel (1994) tells the story of McLean, that during his research of blood clotting factors discovered heparine as anticoagulant, in other words, a factor that prevents blood clotting (McLean was investigating a drug that caused blood clotting).

The *wrong hypothesis* pattern is characterized by the experience of evaluation of a hypothesis with an outcome that, although proven false, is surprising enough to incite the formulation of new hypotheses. The *wrong hypothesis* pattern represents one of the most interesting aspects of serendipity, the adaptation to face an unexpected obstacle, which can be summarized in the proverbial sentence "*When life gives you lemons, make lemonade*".

The *disturbance* pattern is characterized by a change of perception caused by an occurrence that affects the regular activity of a person. The *disturbance* *pattern* is fired by a chaotic event that introduces other variables into the problem. For example, Van Andel (1994) narrates the creation of Radio-astronomy that originated from the noise observed in transatlantic telephone calls, with a periodicity of 23 hours and 56 minutes.

Finally, just to give an example of a pattern which is not amenable to (a straightforward) formalization in the context of Linked Data search, we cite *Introspective Chance Encounter* (Lot 1956 apud Van Andel 1994), a super category group. Van Andel (1994) assigns three patterns to this (super) category – *playing*, *joke* and *dream*. In opposition to the previous patterns, this group contains serendipity patterns that occur during intellectual activities (Lot 1956 apud Van Andel 1994) and, therefore, are not further discussed in the rest of this thesis.

3
A Query Orchestration Process for Serendipity

3.1. Introduction

To perform a serendipitous search, we apply a query orchestration process (Cuppens et al. 1988) that enables the application to transform a submitted query. This allows the application to act before or after the query is actually executed. Therefore, the application can adopt different strategies at different phases of execution.

We adopt the definition of question, answer and response (Webber 1986) in order to describe the interactions of the query orchestration process. Question, according to (Webber 1986), represents a request for information or a request to performing an act. In serendipitous search, the submitted query plays the role of the question. Answer represents the information or act directly requested (Webber 1986). In serendipitous search, the answer comprises the results acquired by the simple execution of the submitted query, in other words, it is the information that satisfies the question. Finally, response comprises the complete reaction to the question. Response can be one or the combination of the following items: an answer, additional information, information instead of an answer, a question, etc (Webber 1986).

For serendipitous search, the definition of response is useful when the answer is not necessarily enough to address the goals manifested in the question. The query orchestration processing is used to provide additional information for a query.

We resort to three main strategies to capture the selected serendipity patterns with the query orchestration process. Given a query Q, *a serendipitous processing of* Q consists of the combination of the following components: *Serendipitous response for Q*, *Serendipitous alternatives for* Q and *Serendipitous rebalancing of Q results*.

We conclude the chapter with a discussion about simulated annealing, a metaheuristic for optimization that can be used for managing the level of serendipity introduced in the results.

3.2. A Framework for Query Orchestration

In order to design the query orchestration process, we propose the definition of *conditions* and *serendipity patterns*.

A condition is a function that evaluates if a RDF term or a set of RDF terms satisfy a set of triple patterns in a RDF dataset. Let *G* be an RDF dataset, $t_1,\ldots,t_n$ be a set of RDF terms and *TP* be a set triple patterns with $v_1,\ldots,v_n$ variables. A *condition* replaces v_i by t_i in *TP*, for each i ∈ [1,n], and returns true if there is a solution of *TP* found in *G*, otherwise it returns false.

The pseudo-code *CON1* exemplifies a condition that checks if a given RDF term is an English rock guitarist. For example, if the entity `dbr:John_Lennon` and DBpedia are evaluated with *CON1*, the result is true, while the result is false for the entity `dbr:Jimmi_Hendrix`.

CON1 Condition with English rock guitarist category

```
CONST SUBJECT-PROPERTY:  <http://purl.org/dc/terms/subject>
CONST RG-CATEGORY: <http://dbpedia.org/resource/Category:English_rock_guitarists>

Function GuitaristCondition (input: entity of RDFTerm, dataset of RDFGraph)
{
       If(dataset.contains( Triple (entity, SUBJECT-PROPERTY, RG-CATEGORY)))
              Return true;
       Else
              Return false;
}
```

The pseudo-code *CON2* exemplifies a composed condition that checks if two RDF terms are connected by the parent property.

CON2 Condition with parenthood relationship

```
CONST PARENT-PROPERTY:  <http://dbpedia.org/ontology/parent>

Function ParenthoodCondition (input: father of RDFTerm, son of RDFTerm, dataset
of RDFGraph)
{
       If(dataset.contains( Triple (son, PARENT-PROPERTY, father)))
              Return true;
       Else
              Return false;
}
```

Additionally, a condition may depict an occurrence, which any element satisfies the triple pattern. The pseudo-code *CON3* exemplifies a composed condition that checks if two RDF terms are connected by any property.

CON3 Condition with any relationship

```
Function AnyPropertyCondition (input: subject of RDFTerm, object of RDFTerm, dataset of RDFGraph)
{
        If(dataset.contains( Triple (subject, ANY, object)))
                Return true;
        Else
                Return false;
}
```

A *serendipity pattern* is a function that checks a set of conditions and, if all conditions return true, a new RDF data graph is returned. Let *G* be an RDF dataset, $t_1,\ldots,t_n$ be a set of RDF terms and $C_1,\ldots,C_m$ be a set of conditions. A *serendipity pattern* returns a new RDF data graph if C_i = true for each i ∈ [1,m].

The pseudo-code *SP1* exemplifies a serendipity pattern that creates a RDF triple connecting two entities through `rdfs:seeAlso` property if the conditions are met i.e. both are English rock guitarists and there is also a property `dbo:parent` connecting them.

SP1 Serendipity Pattern of CON1 and CON2

```
CONST SEEALSO-PROPERTY:  <http://www.w3.org/2000/01/rdf-schema#seeAlso>

Function ParentAndGuitaristPattern (input: father of RDFTerm, son of RDFTerm, dataset of RDFGraph)
{
        If( GuitaristCondition(father,dataset)
                AND GuitaristCondition(son,dataset)
                AND ParenthoodCondition(father, son, dataset)
        )
                Return Triple (father, SEEALSO-PROPERTY, son);
        Else
                Return null;
}
```

The construction of the serendipity patterns using composition of conditions allows the creation of a set of serendipity patterns that use different and independent conditions to present the same relationship. This can be very useful for representing a generic output that can be generated from different sets of query patterns. For example, there may be a catalogue of serendipity patterns specialized in analogy.

It is worth noticing that the code of *SP1* can be translated in the SPARQL query below without any loss of expressivity.

SPQ1 Example of serendipity pattern represented as SPARQL query

```
PREFIX dct:      <http://purl.org/dc/terms/>
PREFIX dbo:      <http://dbpedia.org/ontology/>
PREFIX dbc:      <http://dbpedia.org/resource/Category:>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {?baseEntity rdfs:seeAlso ?surpriseEntity} WHERE {
      ?baseEntity dbo:parent ?surpriseEntity.
      ?baseEntity dct:subject dbc:English_rock_guitarists.
      ?surpriseEntity dct:subject dbc:English_rock_guitarists.
}
```

Finally, the SPARQL query is transformed into a SPARQL query template. This way, the query orchestration process is able to fill in the [*input*] field with a RDF term originated from a previous interaction and process a secondary query that retrieves results related to a specific entity.

For example, *SPQT1* depicts the final representation of the *serendipity pattern* illustrated in *SPQ1* and *SP1*.

SPQT1 Example of serendipity pattern query template

```
PREFIX dct:      <http://purl.org/dc/terms/>
PREFIX dbo:      <http://dbpedia.org/ontology/>
PREFIX dbc:      <http://dbpedia.org/resource/Category:>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {[input] rdfs:seeAlso ?surpriseEntity} WHERE {
      [input] dbo:parent ?surpriseEntity.
      [input] dct:subject dbc:English_rock_guitarists.
      ?surpriseEntity dct:subject dbc:English_rock_guitarists.
}
```

It is worth noticing that the serendipity pattern representation in a query template simplifies the task of sharing the serendipity patterns across different platforms. For example, a generic query processor can read the serendipity pattern templates in order to embody serendipity amongst its search process.

Moreover, the user can submit his own serendipity pattern query templates in order to inform the query orchestration process what conditions does he want to disclose. Therefore, the query execution is able to process the user's query, retrieve the results and also finding content related to the results according to the user's definition of what serendipity is.

Additionally, each serendipity pattern presents not only a query template but also metadata that the query orchestration process can use to adapt the query execution. The serendipity pattern metadata include: (1) description of the input parameters; (2) limit data that defines the maximum number of triples return by the serendipity pattern; (3) a list of dataset endpoints, which the query orchestration process can submit the serendipity pattern.

In addition, the query orchestration process retrieves metadata regarding the serendipity pattern execution, such as the average number of RDF triples found by each serendipity pattern and the average rating collected with the users. This allows the query orchestration process to learn what serendipity patterns are more useful and, subsequently, adapt its behavior.

3.3. Serendipitous response for a query

Given a query *Q*, a *Serendipitous response for Q* will add new triples to each result of Q. More precisely, let $D_1,\ldots,D_m$ be a set of datasets, called the query *environment*, *Q* be a query over D_k, with k ∈ [1,*m*], $X_1,\ldots,X_p$ be a set of serendipity patterns. A *serendipitous response for Q over* $D_1,\ldots,D_m$ is a list of pairs of sets $((A_1,S_1),\ldots,(A_n,S_n))$ such that, for each i ∈ [1,n], A_i is a result of *Q* over D_k, called the *regular component* of (A_i,S_i), and S_i is a set of triples, called the *serendipitous component* of (A_i,S_i), computed from the datasets in the query environment, according to the serendipity pattern X_j, with j ∈ [1,*p*]. Additionally, the *Serendipitous response for* Q may use results of a previously submitted query *Q'* in order to compute the current serendipitous component.

We note that the triples in a *serendipitous component* S_i may use terms in the vocabulary and refer to entities outside the query environment. Indeed, the analogy and the surprising observation patterns, presented in the next chapter, are formalized as new queries that return triples which are serendipitously related to the original result of Q. Such triples will form the second set in each pair of sets in the result list.

Consider that a user is searching for English rock guitarists using DBpedia. To address his goal the user may use the category English rock guitarists to formulate the query. The regular component of the result list includes entities that match the solution mapping of the query, such as, “Mick Jagger”, “George Harrison”, and “John Lennon”. The serendipitous component contains a set of triples that serendipitously connect new entities to those in the result list. For example, the serendipitous component may return a set of triples linking “John Lennon” to “Roy Harper” or “Ringo Starr”, based on a specified criterion.

The following chapter discusses the strategies to capture each serendipity pattern. To simplify the discussion, all examples consider a query, referred to as *UQ1*, about English rock guitarists:

UQ1 Entities from English rock guitarist category

```
PREFIX dct:      <http://purl.org/dc/terms/>
PREFIX dbc:      <http://dbpedia.org/resource/Category:>

SELECT distinct ?entity WHERE{
      ?entity dct:subject dbc:English_rock_guitarists.
}
```

Note that this query uses the *English rock guitarists* category of DBpedia and the `dct:subject` property from Dublin Core vocabulary, used to assign entities to categories.

Figure 3 illustrates examples of serendipitous response for the query *UQ1*, that searches for English rock guitarists using DBpedia. In this example, the response includes additional information that presents serendipitous content related to each result list item.

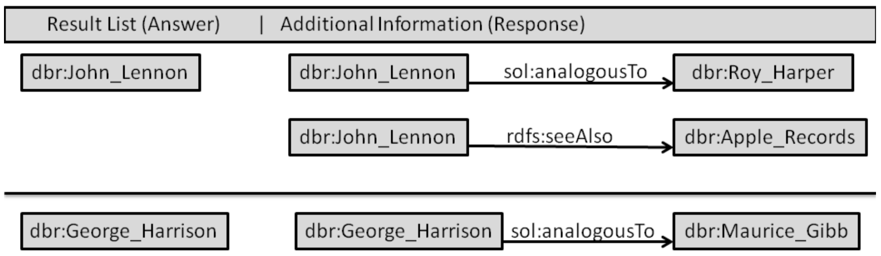


Figure 3: Serendipitous response

3.4. Serendipitous query alternatives

Given a query *Q*, a *serendipitous alternative for Q* will produce an alternative query *Q'* so that the *result list of Q'* presents some relatedness or a level of similarity to the *result list of Q*. This way, a *serendipitous alternative* enables the user to expand his search and encounter content that is not presented in the original query.

Figure 4 illustrates an example of alternative query that presents a similar set of results to those of the original query.

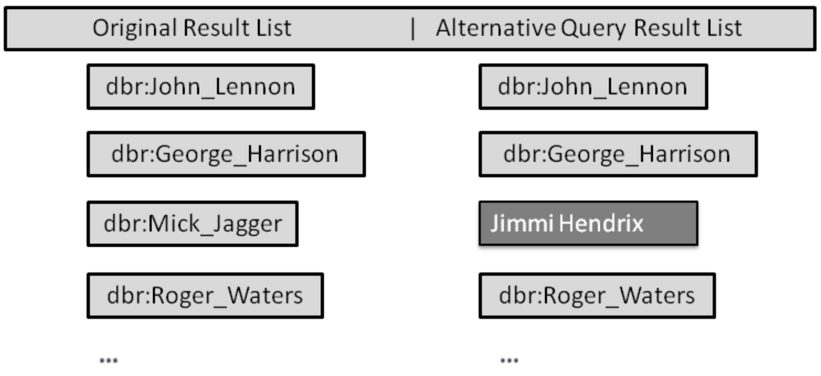


Figure 4: Result list of an alternative query

3.5. Serendipitous rebalancing of a query's results

The *serendipitous rebalancing of Q results* will reorder the serendipitous result list of Q over the *query environment* in order to perturb the original query results ordering.

The procedure consists of going through the ordered query results and selecting pairs of items that are candidates for order swap. An activation function can be used in order to randomly retrieve a set of swap candidate pair items and the swap operation is executed only if a comparison between the two items satisfies a condition.

It is worth noticing that the random selection of items represents an important role in *serendipitous rebalancing of Q results*. This way, the procedure addresses a fundamental aspect of serendipity, the chance factor. Additionally, the activation function can be customized to allow the user to define the proportion of candidates to be selected. Therefore, the user is able to choose the level of serendipity that the strategy will apply.

For the pair comparison, the procedure adopts an ordering criterion other than the result list ordering. Thus, the swap operation is executed only if the pair items possess inverse positions in the ordering lists.

By introducing an external ordering criterion, the query process is able to smooth the impact of the original result list ordering. The external ordering criterion guarantees that the swap operation is not groundless.

It is also worth noticing that the *serendipitous rebalancing of Q results* may take advantage of other serendipity search strategies. For instance, the comparison function can order the pair items according to the number of triples found in the *serendipitous response for* Q procedure.

Figure 5 illustrates how the result lists of a query can be rebalanced.

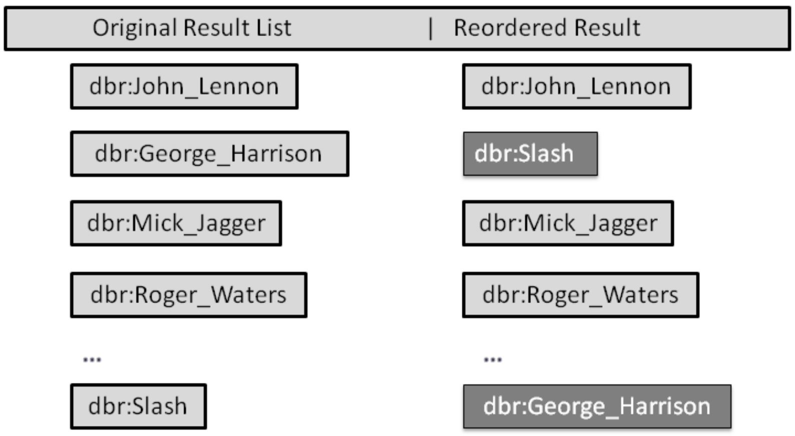


Figure 5: Serendipitous rebalancing

3.6. Simulated Annealing

As a serendipitous event, by definition, introduces unexpected items, there is an inherent risk of discontent, since the user can consider the results uninteresting or worthless if he does not recognize the connection between the results and his question (Shani et al. 2011). Iaquinta et al. (2008) argue that a serendipitous encounter depends on the characteristics of the information seeker, such as, curiosity, open mindedness and cultural background in accordance with Louis Pasteur's quote "chance favors the prepared mind". Thus, the increase of serendipity must be strategically done in order to mitigate the risk of confusing or distracting the user (Ge et al. 2010). In the context of information retrieval, this concern gains even more importance dealing not only with information overload but also with performance, by selectively providing only relevant information to the user's interests.

Thus, there may be times when a serendipitous search application must adopt a strategy to restrict the serendipity of the result set. An approach used for similar problems is simulated annealing (Kirkpatrick 1984), a metaheuristic for obtaining good solutions to difficult optimization problems at a reasonable computing time (Eglese 1990).

Simulated annealing is a local search algorithm (Szu & Hartley 1987, Eglese 1990). A commonly used example that illustrates the simulated annealing heuristic is the hill climbing algorithm, which seeks for the maximum of a given curve (Lim et al. 2006). The hill climbing algorithm starts with an initial solution, perhaps chosen at random, and then, if a neighbor has a higher position than the current position, the respective neighbor will become the current solution, and so on until there are no higher positions. The problem of this approach is that the algorithm can be trapped in a local optimum solution. To address this issue, the simulated annealing algorithm allows a downhill move determined by some suitable probabilistic mechanism. Thus, facing a local optimal solution, the algorithm introduces a disturbance by adding a small worsening factor to the respective solution, thereby enabling the exploration of more suitable solutions.

Analogously to the descent moves in the hill climbing example, a serendipitous connection may represent a costlier operation, with an uncertain outcome, than a simple and direct connection. In this eventuality, a mechanism is provided for the purpose of deciding if a serendipitous connection should be considered or ignored.

One way to achieve this goal is to let the user evaluate the proposed strategies. If a serendipitous search application is able to gather data about user interactions, the application is able to understand what the best strategies are for the respective user. This way, strategies that are not well appreciated by the user may receive lower probabilities for being activated while strategies that are commonly used obtain higher probabilities.

Thus, understanding how the adopted strategies perform plays a crucial role for serendipitous search. There are two major approaches to evaluate if a given strategy performs well or not. One approach is to let the user rate the strategy by asking him to rate how well it addressed his goal. Another approach is to track how frequent a strategy is used by assuming that the most used strategies are well appreciated by the users.

For example, consider the *serendipitous rebalancing of* Q *results* strategy. The referred strategy perturbs the results order so that the user is also exposed to an amount of neglected items. However, if the swap items are rarely visited by the user, the mentioned strategy should be less frequently activated.

This way, the application is able to learn what the best strategies are and, therefore, to adapt its behavior for the best user experience.

3.7. Chapter Summary

In this chapter, we describe a query orchestration process that permits the application to approach different behaviors for a given submitted query. A query orchestration process can be very useful when the normal execution of the query does not produce sufficient information to support the user's goals.

We present three strategies to capture the selected serendipity patterns with the query orchestration process: *Serendipitous response for a Query*, *Serendipitous alternatives for a Query* and *Serendipitous rebalancing of a Query results*.

We present also a discussion about how to manage the level of serendipity introduced in the results.

4
Serendipity Patterns

4.1.Introduction

This chapter discusses how to capture the serendipity patterns of Chapter 2 in the context of Linked Data search. It also provides a case study scenario with the purpose of illustrating the use of the serendipity patterns. The scenario is based on the DBpedia dataset and focuses on the music domain. In this scenario, serendipity search can increase the user satisfaction by providing interesting and non-obvious artists or songs.

Inspired by the Information Encounter Experience (Erdelez 1999) discussed in Chapter 2, we characterize serendipitous search defining the *encounterer* as the user who submits the query, the *environment* as a query environment with which the user interacts, while the *information* encountered is the results list. We distinguish between a regular component retrieving the original results using the classical sorting, and a serendipitous component providing surprising results.

The main objective of this chapter is to formalize a set of serendipity patterns in order to: (1) facilitate the user in achieving his goal by exposing him to a more diverse set of information; (2) explore information discovery capabilities of the environment; (3) provide the user with not only problem-related but also interest-related information.

Since serendipity patterns can be applied in different phases of query execution, we formalize the serendipity patterns according to the query orchestration process of Chapter 3.

To capture the analogy, the surprising observation, the latent goal and the inversion patterns, we resort to the *Serendipitous response for a query* strategy. In other words, the process explores the results of the user's query to invoke secondary queries with the recently acquired information.

To capture the wrong hypothesis pattern, we approach the *Serendipitous alternatives for a query* strategy, according to which the process analyzes the submitted query and is able to generate alternative queries. This allows the user to retarget his search based on an alternative version of his original query.

Finally, to capture the disturbance pattern, we apply the *Serendipitous rebalancing of a query results* strategy, whereby the process is able to change the order of the result list in order to expose items that the user would normally neglect.

In order to illustrate the serendipity patterns, we resume the *UQ1* query. This way, all examples are based on English rock guitarists category results.

UQ1 Entities from English rock guitarist category

```
PREFIX dct:      <http://purl.org/dc/terms/>
PREFIX dbc:      <http://dbpedia.org/resource/Category:>

SELECT distinct ?entity WHERE{
      ?entity dct:subject dbc:English_rock_guitarists.
}
```

4.2. Capturing the Analogy Pattern

To capture analogy, we first introduce a new property, `sol:analogousTo`, to be expressed by triples of the form (*s,sol:analogousTo,o*), which intuitively indicate that entities *s* and *o* are analogous.

More precisely, let *Q* be a query submitted to a dataset D_k and T_i be a result of *Q* for D_k. If *e* is an entity that occurs in T_i, then the search process might look for or compute a triple of the form (*e,analogousTo,o*) in D_k and include the triple in the serendipitous component corresponding to T_i.

We propose to compute *analogousTo* using a family of similarity functions adopting the same strategy used to compute the *sameAs* property, except that the properties to be compared would be chosen according to some set of criteria appropriate to capture analogy.

One approach is to define a *query context* that reflects the interests of a group of users. For example, consider the entities “John Lennon” and “Roy Harper”, both belonging to the *English rock guitarists* category and both reportedly influenced by the American novelist and poet “Jack Kerouac”, a pioneer of the Beat Generation; indeed, “John Lennon” and “Roy Harper” are both linked to “Jack Kerouac” through the `dbo:influenced` property of the DBpedia property ontology. From this point of view, “John Lennon” and “Roy Harper” are understood to be *analogous*, in that, as noted, they belong to the same category and are connected to the same entity with respect to the `dbo:influenced` property. For this scenario, the search process must fill in the Analogy Query Template 1, *AQT1*, with information acquired from the user's query. To do so, the search process executes a valid SPARQL query by replacing the [*result-uri*] field with the results of the *UQ1* query:

AQT1 Using influenced property to find analogous entities

```
PREFIX dct:      <http://purl.org/dc/terms/>
PREFIX dbo:      <http://dbpedia.org/ontology/>
PREFIX sol:      <http://soltool.com/>

CONSTRUCT {[result-uri] sol:analogousTo ?analogousEntity} WHERE {
      ?auxInfluence dbo:influenced ?analogousEntity;
                  dbo:influenced [result-uri].
      [result-uri] dct:subject ?auxCategory.
      ?analogousEntity dct:subject ?auxCategory.
      FILTER (?analogousEntity != [result-uri] )
}
```

We also propose a different query context to take advantage of the DBpedia category hierarchy. For example, we might move up in the category hierarchy from *English rock guitarists* to *English guitarists* and then down to *English bass guitarists*, a narrower category. Thus, we would conclude that an entity of *English rock guitarists* is analogous to an entity of *English bass guitarists* with respect to the *English guitarists* category. Similarly to *AQT1*, the search process must fill in the Analogy Query Template 2, *AQT2*, with information acquired from the user's query in order to capture the analogy pattern. One characteristic of this template is that the subquery selects, among the categories of the *UQ1* results, that with the lowest number of entities linked to it in order to find a more specific category subset. To achieve this goal, *AQT2* uses the `skos:broader` property from SKOS ontology, a standard vocabulary for organization systems:

AQT2 Using category hierarchy to find analogous entities

```
PREFIX dct:      <http://purl.org/dc/terms/>
PREFIX skos:     <http://www.w3.org/2004/02/skos/core#>
PREFIX sol:      <http://soltool.com/>

CONSTRUCT {[result-uri] sol:analogousTo ?analogousEntity} WHERE {
   ?analogousEntity dct:subject ?category.
   ?auxCategory skos:broader ?superCategory.
   ?category skos:broader ?superCategory.
   {
       SELECT ?auxCategory (count(?categoryClient))
       WHERE {
              [result-uri] dct:subject ?auxCategory.
              ?categoryClient dct:subject ?auxCategory.
       }
       GROUP BY ?auxCategory
       ORDER BY (count(?categoryClient))
       LIMIT 1
   }
   FILTER (?analogousEntity != [result-uri] )
}
LIMIT 2
```

A variation of *AQT2* is the Analogy Query Template 3, *AQT3*, that randomly selects categories of the [*result-uri*] field:

AQT3 Using category hierarchy to find analogous entities

```
PREFIX dct:      <http://purl.org/dc/terms/>
PREFIX skos:     <http://www.w3.org/2004/02/skos/core#>
PREFIX sol:      <http://soltool.com/>

CONSTRUCT {[result-uri] sol:analogousTo ?analogousEntity} WHERE {
      ?analogousEntity dct:subject ?category.
      ?auxCategory skos:broader ?superCategory.
      ?category skos:broader ?superCategory.
      {
        SELECT ?auxCategory
        WHERE {
              [result-uri] dct:subject ?auxCategory.
        }
        LIMIT 1 OFFSET RAND()
      }
      FILTER (?analogousEntity != [result-uri] )
}
LIMIT 2
```

Note that *AQT1* relies on a vocabulary specific to the arts domain, the *influenced* property, while *AQT2* and *AQT3* use only Linked Data standard vocabularies and, therefore, they can be adopted for several domains.

Finally, we observe that this approach uses the familiar notion of similarity functions, and thus, it may take advantage of tools, such as Limes (Ngomo & Auer 2011) and Silk (Isele et al. 2010) to offline precompute *analogousTo* triples, and add these triples to a dataset.

4.3. Capturing the Surprising Observation Pattern

To capture the surprising observation pattern, we suggest to reinterpret the `rdfs:seeAlso` property in such a way that a triple of the form (*s,rdfs:seeAlso,o*) would intuitively indicate that any user interested in entity *s* might also be interested in entity *o*. Indeed, the `rdfs:seeAlso` property is commonly used as a wildcard to relate contents with loose connections.

In DBpedia, for example, there is a `rdfs:seeAlso` property linking "George Harrison" to "Apple Records". This link may be motivated by an analysis of the connection between "George Harrison" and "The Beatles" and the connection between "The Beatles" and the "Apple Records". For this scenario, the search process must fill in the Surprising Observation Query Template 1, *SOQT1*, with information from the *UQ1* results:

SOQT1 Using seeAlso property to find surprising observation

```
PREFIX rdfs:      <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {[result-uri] rdfs:seeAlso ?surprise} WHERE {
      [result-uri] rdfs:seeAlso ?surprise.
}
```

Another surprising observation is the inclusion of other members of the same band of a given musical artist. This can be captured with the *associatedBand* property, as described in the Surprising Observation Query Template 2, *SOQT2*:

SOQT2 Using associatedBand *property to find surprising observation*

```
PREFIX rdfs:      <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo:      <http://dbpedia.org/ontology/>

CONSTRUCT {[result-uri] rdfs:seeAlso ?surprise} WHERE {
      [result-uri] dbo:associatedBand ?band.
      ?surprise dbo:associatedBand ?band.
}
```

Computing the `rdfs:seeAlso` property is a difficult issue though. A simple solution would be to define (*s,rdfs:seeAlso,o*) as (*s,owl:sameAs,o*), provided that entity *s* is defined in the dataset the query refers to and that *o* is an entity defined in another dataset listed in the query environment, but coming from a different domain. For example, consider the case of a dataset D_k about the music domain, which contains information, such as musical artists, their albums and their songs. Suppose that *Q* is a query submitted to D_k and T_i is a result of *Q* over D_k. If *e* is a singer that occurs in T_i, then the search process might look for a triple of the form (*e,owl:sameAs,o*) in D_k, where *o* is an entity defined in D_j, with $j \neq k$, and include (*e,owl:sameAs,o*) in the serendipitous component corresponding to T_i. If D_j is a dataset about actors, the user may be told that singer *e* is also an actor, like “David Bowie” or “Jared Leto”.

The example above illustrates how the surprising observation pattern can integrate different views about the same object. By gathering the combination of all social activities and different roles of a person, we are able to acquire a more complete version of his biography or his “social identity” (Goffman 1963).

According to this strategy, using the query *UQ1*, the surprising observation pattern suggests the “David Bowie” entity of New York Times dataset[3] for users who search for “David Bowie” in DBpedia, if the New York Times dataset belongs to the query environment. The Surprising Observation Query Template 3, *SOQT3*, depicts the template to capture this occurrence:

[3] http://data.nytimes.com/80300354872775959333

SOQT3 Using sameAs property to find surprising observation

```
PREFIX rdfs:      <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>

CONSTRUCT {[result-uri] rdfs:seeAlso ?surprise} WHERE {
      [result-uri] owl:sameAs ?surprise.
}
```

4.4. Capturing the Latent Goals Pattern

In other circumstances, we can adopt a different strategy to capture surprising observations. Instead of providing surprising observations related to the goal of the current query, the application can explore how the current query results address other latent goals (De Bruijn & Spence 2008).

Since every submitted query represents the user current goals, we use previously submitted queries to compose latent goals, i.e., goals that are not explicitly addressed in the current query. Therefore, new queries may be directed to explore whatever is eventually found related to other recent queries.

By supporting latent goals, the procedure enables the discovery of problem-related information (Erdelez 1999) as the acquired data addresses a different goal than the expressed in the current query. In other words, the latent goals patterns objective is to discover how the current query results relate to previously submitted queries and use that to obtain further information.

More precisely, let *Q* be a query submitted to a dataset D_k and *P* be a recent submitted query for D_k. A latent goal is supported if there is a triple (x,y,z) or triple (z,y,x), where x is a result of *Q* for D_k and z is a result of *P* for D_k.

Consider the next example. Apparently, there is nothing in common between such disparate domains as "guitarists" and "salads". And yet a user visiting Quebec, who first asks about "Quebec" and "guitarists", and later, when planning for dinner, asks about "restaurants" and "salads", may be told – in unexpected detail – that one restaurant features "good salads, nice live guitarist". In this way, the serendipitous component can be made more responsive to the user's interests and goals.

For this scenario, the search process must fill in the Latent Goal Query Template 1, *LGQT1*, with information acquired from the user's query. To do so, the search process executes a valid SPARQL query by replacing the [*result-uri*] field with the results of the *UQ1* query and [*recent-query*] with a recent query:

LGQT1 Using a recent query

```
PREFIX sol:      <http://soltool.com/>

CONSTRUCT {[result-uri] sol:latentGoal ?surprise} WHERE {
      {
            [result-uri] ?p ?surprise.
            filter(?surprise IN ([recent-query]))
      }
      UNION
      {
            ?surprise ?p [result-uri].
            filter(?surprise IN ([recent-query]))
      }

}
```

Hence, the serendipitous component can be made more responsive to the user's interests and goals, either merely involved in a multiple-query session as in the above example, or registered among the objectives of a daily agenda, or more elaborately deduced from some user profile representation.

4.5.Capturing the Inversion Pattern

The inversion pattern describes the process of discovery where the insight is the opposite of the initial intent. For this matter, the main objective is not to capture a RDF triple (*s*,*p*,*o*) that asserts that there is a property *p* connecting entity *s* to entity *o*, but rather the goal is to capture its opposite.

Under the closed-world assumption, implemented as negation as failure, the absence of a RDF triple is understood that the assertion of that triple is false. But, negation as failure does correspond to the intuition behind the inversion pattern *p*. For example, *p* might be "cures", whose inversion might be "causes" (a disease), which is different from "not cures".

Hence, to properly capture the inversion pattern, given a property *p*, we would have to postulate that there is a property *q* such that q captures the opposite of *p*. In other words, the underlying ontology would have to be rich enough to have properties and their antonyms, which is not always the case.

There is a special case, though, worth exploring. Since we use the `owl:sameAs` property to define a strategy for capturing the surprising observation pattern, we propose to use the `owl:differentFrom` property in such a way that a triple of the form (*s, owl:differentFrom,o*) indicates that entities *s* and *o* are different, in spite of the initial intent (which is to find entities that represent the same real-world object). In this sense, the `owl:sameAs` and the `owl:differentFrom` properties are opposed to each other.

In DBpedia, for example, there is a `owl:differentFrom` property linking “Robbie Williams” to “Robin Williams”. For this scenario, the search process must fill in the Inversion Query Template 1, *IQT1*, with information from the *UQ1* results:

IQT1 Using differentFrom property to explore inversion

```
PREFIX rdfs:      <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:      <http://www.w3.org/2002/07/owl#>

CONSTRUCT {[result-uri] owl:differentFrom ?inversionEntity } WHERE {
      [result-uri] owl:differentFrom ?inversionEntity .
}
```

4.6. Capturing the Wrong Hypothesis Pattern

We adopt a completely different strategy to capture the wrong hypothesis pattern. Very briefly, the suggested strategy allows the user to stop consuming the result list obtained for a query *Q*, and restart the search process with a new query *Q’* based on some entity observed in the serendipitous component of a result of *Q*. That is, the user would retarget his search based on some entity the search process may have passed in a serendipitous component. This pattern may be quite useful when the user does not find enough information with his query but does not know what else to search for.

The wrong hypothesis pattern relies on the category representation of DBpedia to present alternative queries to the user. To do so, the search process executes the user query and retrieves the three most popular categories of the results i.e. the categories that appear most often in the results. With this information, the search process builds an alternative query allowing the user to restart the search process with a different perspective.

To reproduce this behavior, the search process must proceed in two steps. First, it uses the Category Frequency Query Template 1, *CFQT1*, to get the three categories with the largest number of entities linked to them. The search process fills the template with two pieces of information from the user’s query string: the output variable of the query string represented by the [*var*] field and the query string itself represented by the [*user-query*] field:

CFQT1 Extracting the most used categories from the subquery

```
PREFIX dct:      <http://purl.org/dc/terms/>

SELECT  (COUNT(?s) AS ?counter) ?category WHERE {
   ?s  dct:subject ?category.
   FILTER ( ?s = [var])
   {
        [user-query]
   }
}
GROUP BY ?category
ORDER BY DESC(?counter)
LIMIT   3 OFFSET  1
```

Second, the search process fills in the Wrong Hypothesis Query Template 1, *WHQT1*, with information acquired from the *CFQT1* by replacing the [*categories-list*] term with results of the previous query.

WHQT1 Building alternative query

```
PREFIX dct:      <http://purl.org/dc/terms/>

SELECT ?entity WHERE {
   {
        SELECT ?entity WHERE {
               ?entity dct:subject ?catAux.
               FILTER (?catAux IN ([categories-list]) )
        }
   }
   MINUS
   {
        [user-query]
   }

}
LIMIT 100
```

For example, assume the search process receives *UQ1*. First, the search process uses *CFQT1*, to discover that the three most frequent categories of *UQ1* are: *English rock guitarists*, *Living people* and *English male singers*. Then, it completes the *WHQT1* template with the acquired information as depicted in the example below.

Example of alternative query to UQ1

```
PREFIX dct:      <http://purl.org/dc/terms/>
PREFIX dbc:      <http://dbpedia.org/resource/Category:>

SELECT ?entity ?catAux WHERE {
   {
        SELECT ?entity WHERE {
               ?entity dct:subject ?catAux.
               FILTER (?catAux IN (dbc:English_rock_guitarists,
                      dbc:Living_people, dbc:English_male_singers) )
        }
   }
   MINUS
   {
        SELECT distinct ?entity WHERE{

               ?entity dct:subject dbc:English_rock_guitarists.

        }
   }
}
```

4.7. Capturing the Disturbance Pattern

We also suggest to adopt a strategy based on the result list to capture the disturbance pattern. This strategy perturbs the order of the result list obtained for a query *Q* by randomly bringing results further down the result list to near the top of the list. The user who issued query *Q* would therefore be exposed to results that he would normally neglect, and consequently his perception of the query result list would be changed.

This strategy stems from two motivations. First, if query *Q* returns a result list ordered by any ranking criterion *X*, then the disturbance pattern has the ability to smooth the impact of *X*. Second, if no ordering criterion is provided, the dataset endpoint may use its own ordering, in other words, the query will highlight results using a criterion that is not clear to the application or the user.

For example, consider that a user modifies the *UQ1* so that the results are ordered alphabetically. The disturbance pattern switches the position of “Adrian Portas” and “Würzel”, both English rock guitarists.

4.8. Chapter Summary

In this chapter, we formalize strategies to capture serendipity patterns in the context of Linked Data search for the music domain. For this domain, serendipity search can increase the user satisfaction by providing interesting and non-obvious artists or songs.

We present five serendipity patterns according to the query orchestration process. To capture the analogy, the surprising observation, the latent goals and the inversion patterns, we resort to Serendipitous response for a query strategy. To capture the wrong hypothesis pattern, we apply the serendipitous alternatives for a query strategy. Finally, to capture the disturbance pattern, we apply the Serendipitous rebalancing of a query results strategy.

5
SOL-Tool

5.1.Introduction

In order to validate the query orchestration process and to apply the serendipitous patterns, we developed a Linked Data search tool, the Serendipity Over Linked Data Search Tool – SOL-Tool. The SOL-Tool was developed in Java with the Jena framework[4], a well-stabilized framework for Linked Data query processing and data manipulation, and Java Concurrent API[5] to parallelize the task of invoking remote datasets.

The SOL-Tool modular architecture is organized in a way that allows the search process to: (1) isolate the logic task of displaying the results from the rest of the search process; (2) permit not only users but also other applications to consume the search process of the tool; (3) take actions before, during and after the execution of the user's query; (4) attach additional information to every item of a query result; (5) address remote datasets independently; (6) enable the different query strategies for different scenarios; and (7) parallelize the query execution.

Figure 6 depicts the SOL-Tool architecture.

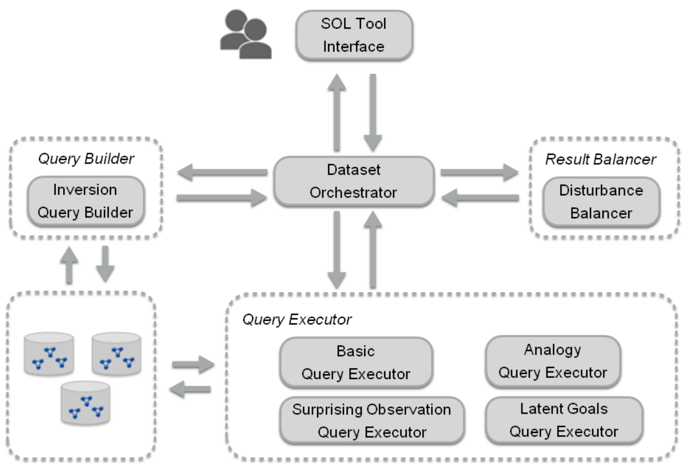


Figure 6: The SOL-Tool Architecture

[4] https://jena.apache.org/
[5] http://docs.oracle.com/javase/8/docs/api/?java/util/concurrent/package-summary.html

To handle (1) and (2), the *SOL-Tool Interface* merely acts as the interface of the search engine with the user or other application receiving a SPARQL query and returning its results. This enables future versions of the SOL-Tool search engine to be instantiated as a Web service for other applications. Then, the *SOL-Tool Interface* starts the *Dataset Orchestrators* with a catalogue of datasets.

Motivated by (3), (4) and (5), the *Dataset Orchestrator* is responsible for interacting with a single dataset and managing the acquired data. This way, the application can address multiple dataset endpoints by merely creating multiple instances of *Dataset Orchestrators*. Each *Dataset Orchestrator* works independently with its isolated view of *Query Executors*, *Query Builders* and *Result Balancers*.

The *Dataset Orchestrator* first uses the *Basic Query Executor* to process the user's query and retrieves its results. The *Basic Query Executor* is just a basic type of *Query Executor* that receives a SPARQL query, processes it and returns its results.

For every result of the user's query, the *Dataset Orchestrator* invokes *Query Executors* to process secondary queries and locate content that is serendipitously related to the respective result. The *Dataset Orchestrator* then delegates the task of querying its dataset to the *Query Executor*.

Motivated by (5) and (6), the *Query Executor* defines how to query the dataset. It encapsulates the logic of the query executed, in other words, it describes the serendipity patterns in terms of a SPARQL query that can be submitted to the dataset. To adapt the search process to different scenarios and behaviors, the SOL-Tool provides different *Query Executors* as described in Chapter 4, and also provides an interface to build new ones. Secondary tasks of the *Query Executor* include parsing the results and handling eventual network exceptions.

It is worth noting that the *Dataset Orchestrator* encompasses the strategy of the search process while the *Query Executor* retains its logic. Thus, a *Dataset Orchestrator* acts as a façade for encapsulating several *Query Executors* to address the same dataset with different approaches. This design allows the application to adopt different approaches and control the level of effort to produce serendipity in the results.

Then, the *Dataset Orchestrator* invokes *Query Builders* to create alternative query suggestions to the user's query. The *Query Builders* receive a query string and return a different query string in order to enable a wrong hypothesis pattern experience. They encapsulate the logic of the query transformation and can be invoked before, during or after the *Basic Query Executor* is executed. Like the Query Executors, the *Query Builders* have access to remote datasets, thus also handling (5).The current version of SOL-Tool presents only one *Query Builder* as described in Chapter 4.

Finally, the *Dataset Orchestrator* may also invoke a *Result Balancer* to reorder the obtained results. The *Result Balancer* encapsulates the logic to reorder the results. The current version of SOL-Tool only provides an interface for the construction of new *Result Balancers*.

5.2. Concurrent Dataset Request

As most of the effort spent by the application relies on invoking remote dataset endpoints, a critical factor since early implementations is the impact of latency in overall performance, i.e., the time that the application waits for remote servers to respond. To address this problem, the application resorts to the Java concurrent API to invoke SPARQL requests concurrently.

To reproduce this behavior, every Query Executor must implement a call method that is responsible for executing the SPARQL request and returning the query results. Therefore, the Dataset Orchestrator invokes the Query Executors asynchronously and aggregates the results that come from the remote dataset endpoint. The Dataset Orchestrator incorporates a MapReduce strategy (Leskovec et al. 2014) to combine the results related to an entity from many Query Executors as depicted in Figure 7. For example, assume that the user query returns an entity e. The Dataset Orchestrator will invoke Query Executors to find content that is serendipitously related to e. All data content found are grouped together using the URI from e.

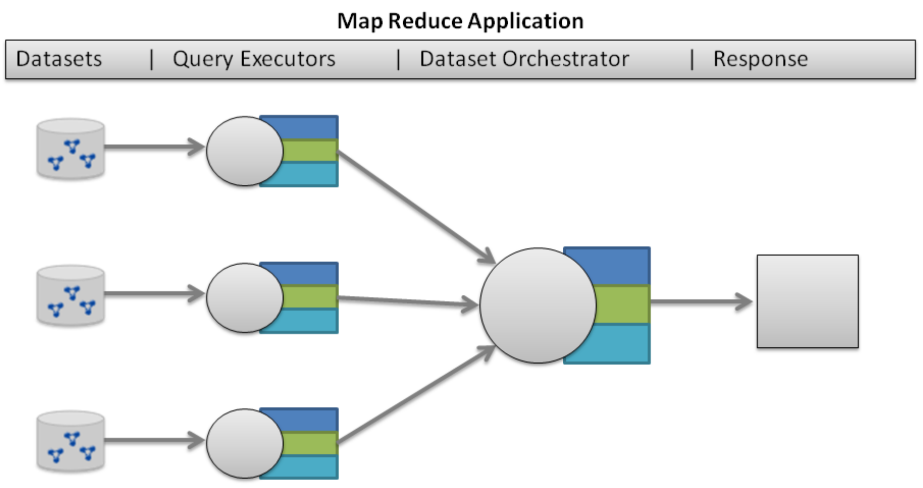


Figure 7: MapReduce Strategy

With this configuration, the SOL-Tool application executes a basic search in less than 6% of the time of the single thread version. For comparison, UQ1 was executed 10 times using the single thread and the multi-thread version of SOL-Tool. The average time of the single thread is 144 seconds, while the average time of the multi-thread (with a pool of 50 threads) is 7.4 seconds.

5.3. Evaluation

From the recommender systems literature, a common approach to evaluate quality is to measure the accuracy of the results. However, as argued in (Murakami et al. 2007), other metrics should be considered since very accurate results may lead the user to a bubble where he is only exposed to restricted and obvious information. To overcome this problem, we adopt unexpectedness to measure the serendipity of the results.

In (Murakami et al. 2007) the unexpectedness of the results is evaluated by comparing the acquired results to a more primitive baseline system. However, as Kaminskas and Bridge (2014) point out, this approach has several drawbacks: for example, the evaluation is sensitive to the baseline system. They then propose a different approach for measuring unexpectedness based on the dissimilarity of content labels. It uses the complement of the Jaccard similarity to compute the distance between two items. Therefore, the unexpectedness of an item is computed as the minimum distance of this item to previously seen items.

The experiment in this section uses the content-based metric (Kaminskas and Bridge 2014) to evaluate the level of unexpectedness of the serendipitous component of the SOL-Tool, compared to its regular component. The content-based metric (Kaminskas and Bridge 2014) is depicted as follows:

$$dist(i,j) = 1 - \frac{L_i \cap L_j}{L_i \cup L_j}$$

where L_i and L_j are set of labels describing items *i* and *j*.

In order to select the item labels properly, the experiment adopts the Type Query Template, *TQT1*, that extracts the types associated with a given [*entity*] entity.

TQT1 Extracting the type of an entity

```
PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns>

SELECT distinct ?type WHERE{
      [entity] rdf:type ?type.
}
```

We chose to evaluate the level of unexpectedness according to the Kaminskas and Bridge (2014) metric because it indicates how the two elements differ in context of the provided labels. If a user is introduced to elements with a high level of unexpectedness he is likely to explore new frontiers of the dataset with information that he would not be exposed in his original query.

This way, a suggestion with 0 score of unexpectedness represents an item with the same set of labels of the base item and, subsequently, closer to the base item definition. On the other hand, a suggestion with score of unexpectedness equal to 1 represents an item without any label in common with the base item.

Due to the size of DBpedia, we adopted the same strategy as (Passant 2010a) and limited the scope of the evaluation by restricting the user's query to retrieve entities of the type *MusicalArtist* and *Band* from DBpedia ontology, which have at the time of this writing 50,978 and 33,613 entities, respectively. The User Query 2, *UQ2*, selects entities of the type *MusicalArtist*.

UQ2 Entities from MusicalArtist type

```
PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX dbo:      <http://dbpedia.org/ontology/>

SELECT distinct ?subject WHERE{
      ?subject rdf:type dbo:MusicalArtist.
}
```

The User Query 3, *UQ3*, selects entities of the type *Band* and is defined similarly to *UQ2*.

UQ3 Entities from Band type

```
PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX dbo:      <http://dbpedia.org/ontology/>

SELECT distinct ?subject WHERE{
      ?subject rdf:type dbo:Band.
}
```

Given the automated nature of the experiment, the *Latent Goals QueryExecutor* is not included in the serendipitous search of the above queries. The motivation is that the latent goals pattern relies on information provided by user profiles. Our major concern is that any profile specification could bias the experimental results.

Table 4 depicts the average unexpectedness of the serendipity component of *UQ2* and *UQ3* with SOL-Tool and SOL-Tool-1, a variation of SOL-Tool that limits the number of results to one entity per Query Executor. This customization is possible due to the parameterization of the limit value of the Query Executor templates.

Table 4: Experimental results.

Query	**Unexpectedness average**	**Query**	**Unexpectedness average**
UQ2	0.90	*UQ2* with limited Query Executors	0.80
UQ3	0.88	*UQ3* with limited Query Executors	0.81

The overall result of Table 4 indicates that the SOL-Tool performs well when proving unexpected results for the selected inputs. This outcome illustrates the fact that the application adopts different strategies to present serendipitous content.

A concern of the metric (Kaminskas and Bridge 2014) is the influence of very dissimilar items on unexpectedness computation. This issue is partially addressed by the SOL-Tool application because each serendipity pattern explores how entities are related. For example, consider the entity that represents the “Juli” band retrieved by executing *UQ3*. The execution of *TQT1* extracts 32 type labels of the “Juli” entity and 320 type labels of the entities encountered with the serendipitous search of *UQ3*, but from those 320 labels, there are 27 type labels that also belong to “Juli”. The unexpected score of this item is 0.93, in spite of finding 85% of “Juli” type labels.

An additional interesting information of Table 4 is the loss of unexpectedness when limiting the number of results per Query Executor. The configuration of these parameters may be used to leverage the tradeoff between the quality of results and the effort spent in the search. This matter can be addressed with a simulated annealing strategy and represents an interesting topic for future study.

5.4. Lessons Learned

The overall score of 90% of unexpectedness acquired in the experiments indicates that SOL-Tool approaches well the problem of locating serendipitous content for the music domain. Nevertheless, other factors ought to be considered in future experiments.

Particularly, a different perspective to be investigate in SOL-Tool is the impact of the *Latent Goals QueryExecutor* in the application overall performance since the component has not been considered in the reported experiment. We are optimistic that in an experiment that tracks user's recent query history the *Latent Goals Query Executor* may enable the discovery of more serendipitous connections.

Additionally, minor improvements can be conducted in the architecture. The SOL-Tool architecture was designed to support extensibility by simply creating new instances of *Query Executors*, *Query Builders* and *Result Balancers*. However, it is worth noticing that the performance may be affected as the number of *Query Executors* grows significantly. The explanation for this effect is that the number of *Query Executors* determines how many secondary queries are to be invocated. Although the Map Reduce strategy mitigates the latency issue, the number of dataset requests increases with the number of *Query Executors* included in the search process.

Considering this scenario, there are two opportunities for enhancement in SOL-Tool architecture. A first alternative is to adapt the SOL-Tool application to an IaaS cloud environment (Mell et al. 2011), such as Azure[6] or Amazon Elastic cloud[7]. In such environments, the data processing tasks can be distributed on a wider scale.

A second alternative is to develop a new component module to analyze the dataset a priori and reason whether each Query Executor should be invoked

[6] https://azure.microsoft.com/
[7] https://aws.amazon.com/pt/ec2/

or not. With that, the work load would be optimized because the processing of a query would not invoke *Query Executors* that would not be useful for the respective dataset. Examples of such occurrences are *Query Executors* that deal with entities or vocabularies not presented in the dataset.

5.5. Chapter Summary

The SOL-Tool was developed with the objective of validating the query orchestration process by combining different strategies to locate Linked Data content. The processing unites unexpectedness and relatedness and the experimental results present a promising score of 90% of unexpectedness for real-world scenarios in the music domain.

The SOL-Tool architecture encompasses five component types: (1) *Query Executors* that are responsible for processing secondary queries and discovering additional information about a given entity; (2) *Query Builders* that create alternative query suggestions to the user; (3) *Result Balancer* that is responsible for reordering the obtained results; (4) *Dataset Orchestrator* that coordinates how the first three components interact with a given dataset; (5) *SOL-Tool Interface* that acts as the interface of the search engine with the user by receiving a SPARQL query and returning its results.

6
A Serendipity Movie Test Dataset

6.1.Introduction

The overall approach adopted by search applications is to locate resources that are strongly related to the user's needs. In other words, the main concern is to maximize how accurate the results are for a given input. However, studies (Murakami et al. 2007, Ge et al. 2010) argue that accuracy alone can lead the user into an information bubble where the user is only exposed to information of a certain niche or, even worse, to a kind of information he already knows. These studies suggest that other metrics, such as serendipity, ought to be considered in order to analyze the user satisfaction.

Serendipity has been used in recommender systems (Abbassi et al. 2009, Adamopoulos et al. 2011, Stankovic et al. 2011, Zhang et al. 2012, Bordino et al. 2013) to provide unexpected items among the search results. As argued in (Toms 2009), serendipity provides a holistic and ecological approach to information acquisition in information systems.

The *serendipity relatedness problem* consists of finding entities that are serendipitously, i.e., unexpectedly, related to a source entity. Inspired by studies that embody serendipity among the recommendation process (Ziegler et al. 2005, Abbassi et al. 2009, Zhang et al. 2012), we propose the adoption of a global classifier function that divides the universe of items in partitions according to the genre feature. Thereafter, a pair of items is understood to be serendipitously related if they belong to different partitions and yet share a reasonable amount of connections or common features, according to similarity criteria.

In spite of the studies that address the serendipity relatedness problem, at the time of this writing there are no benchmarks to measure and compare the effectiveness of different approaches. As Iaquinta et al. (2008) argue, to conceptualize, analyze and implement serendipity turns out to be a difficult task due to its subjective nature.

Therefore, the main contribution of this chapter is a dataset created to support the evaluation of approaches that address the serendipity relatedness problem in the movies domain, which we refer to as the Serendipity Movie Test

Dataset (Eichler et al. 2018). Nevertheless, the Serendipity Movie Test Dataset can also be used in parallel with user feedback experiments in order to provide a complementary view of analyzed approach.

As described in this chapter, the Serendipity Movie Test Dataset contains entities and connection paths extracted from the LOD cloud that pertain to the movies domain.

A second contribution of this chapter is the discussion and description of the steps and design decisions involved in the construction of the Serendipity Movie Test Dataset. The first step consisted in the selection of a set of entity pairs from the movies domain. The second step referred to the extraction of a set of connection paths, for each entity pair. The final step was to rank the pairs, based on information extracted from DBpedia and LinkedMDB[8], and to select the top-k ones.

The proposed dataset creation process defines the necessary steps for building a dataset that exploits Linked Data resources. The steps are structured and can be extended to capture different particularities of the retrieved data, the given domain or the benchmark goal. For example, the data is structured so that the same Linked Data resources can be used to build different benchmarks by simply adopting distinct pair ranking algorithms.

It is worth noticing that, although the Serendipity Movie Test Dataset addresses the serendipity relatedness problem in the movies domain, the overall strategy described in this paper can be replicated in different domains or even combine different domains in the creation process. For instance, to create a serendipity music dataset, one may implement the same steps as Serendipity Movie Test Dataset and simply adapt a few steps, such as the Linked Data dataset source selection.

6.2. A Generic Path Finding and Ranking Process

In order to address the path finding problem, we propose the definition of *ignored list*, *connections* and *connection paths*.

An *ignored list* is a list of properties that should not be considered in the path finding algorithm. The motivation of this requirement is because there may be a set of properties that, if considered in the path finding algorithm, will produce noise in the paths found.

[8] http://www.linkedmdb.org/

Let G be an RDF graph and Z be an *ignored list*. A *connection* in G between entities e_1 and e_2 is a triple (e_1,p,e_2) or a triple (e_2,p,e_1) in G, where $p \notin Z$. The notion of connection is important because the RDF data model induces a directed graph, while for this problem it is only necessary to know if two entities are connected, regardless of the role they play in the triple.

A *connection path* $CP(e_1,e_n)$ in G between entities e_1 and e_n is a sequence of entities $(e_1,e_2,e_3, \ldots, e_{n-2},e_{n-1},e_n)$, where n is the path length, e_i is unique, for $i \in [1,n]$ and there is a connection between entities e_i and e_{i+1} in G.

There are two key points in the construction of the dataset. First, we apply a path finding algorithm to discover the *connection paths* between two resources. Second, we apply a pair ranking algorithm in order to rank entity pairs according to the connection paths they share.

The path finding algorithm receives an RDF graph G, two entities, e_{source} and e_{target} and a maximum distance k. It implements a breadth first search (BFS) to discover a set of connection paths $CP(e_{source}, e_{target})$ in G. The algorithm starts a BFS with entity e_{source}. Subpaths are generated by expanding the search to entities that share a connection with e_{source}; this process goes on recursively until k is surpassed or a connection path is found. Finally, a connection path is found if the entity e_{target} is reached. The output of the algorithm is a set of connection paths $CP(e_{source}, e_{target})$ starting from e_{source} and ending in e_{target}.

The pair ranking algorithm receives a list of entity pairs (e_{source},e_i), where $I \in [1,n]$, and $i \neq source$. For each i, it analyses the connection paths that the entity pair e_{source} and e_i have and computes a $score_i$ according to the Katz index (Katz 1953). The output of the algorithm is a ranked list of entity pairs, according to the computed score.

6.3. Constructing the Serendipity Movie Test DataSet

6.3.1. Overview of the Construction Process

The construction of a Serendipity Test Dataset in general involves seven major steps: (1) how to select relevant entity pairs for a given domain; (2) which Linked Data datasets present relevant data to support the domain; (3) how to map those entities to Linked Data resources; (4) how to retrieve data about an entity; (5) how to discover connection paths for the entity pairs selected; (6) how to rank the entity pairs; and (7) how to present the relevant acquired pairs.

The overall strategy adopted for the creation of such datasets is summarized in what follows and detailed in the next sections for the movies domain.

The class Entity encompasses the basic information needed to describe the entities that the dataset is about. In the case of the movies domain, each entity represents a movie with its title, year and genre.

The entity data is processed in order to locate Linked Data resources that represent that entity in a Linked Data dataset; this task addresses step (3). If Linked Data resources are found and a LD Resource instance is created, a routine queries Linked Data datasets in order to retrieve a Linked Data graph containing information about this resource, achieving step (4).

The class Pair associates two instances of LD Resource, one representing a source entity and the second representing a possible candidate for serendipity suggestion. The class Pair holds only the connection paths that connect the two resources, therefore addressing step (5). Then, each pair receives a score based on its connection paths, as in step (6). Finally, the top-k entities with higher scores are stored as the recommendation for the source entity, addressing step (7).

Figure 8 depicts the data model that describes the movie domain in the Serendipity Movie Test Dataset.

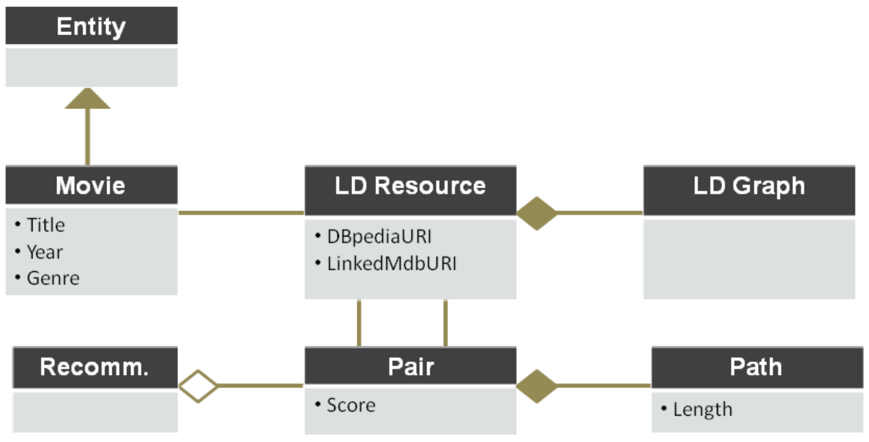


Figure 8: Data Model of the dataset Construction Process

6.3.2. Selecting Entity Pairs

We focused on popular movies in the movies domain. Since popularity represents a relevance criterion, we decided to adopt Google Search expertise for this matter. Additionally, it is important that the dataset construction retrieves a list of movies divided by genre, as the dataset partition plays a decisive role in our definition of serendipity relatedness.

First, we submitted the query “most popular movie genres” to Google Search. The result of the query is a list of 51 movie genres ordered by popularity.

Then, for the 15 first movie genres, we submitted the query “*genre* + movies”, such as “comedy movies”. Among the results of this query is the data (title and release year) of the 51 most relevant movies for that genre, according to Google Search. Figure 9 depicts an example of movie genre search. We only considered the 15 first movie genres because many genres from the bottom of the list represent sub-genres of the previous ones. For example, “Spaghetti western” is a sub-genre of the “Western” genre.

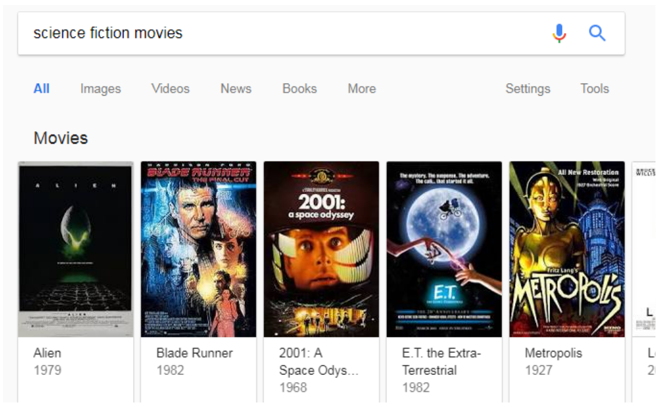


Figure 9: Google Search example

Regarding the adoption of subgenres, we decided not to consider subgenres because we believe that they could introduce bias to the final results. For example, it is likely that a subgenre movie presents a good score when linked to a master genre movie. One way to prevent such occurrences is to annotate the relation between genres, but we avoided this decision in order to not interfere with the initial input.

With this data, we built a list of relevant movies grouped by genre and concluded the first step of the construction of the dataset. The product of this step is available in the BaseList folder of the dataset (Eichler et al. 2018).

6.3.3. Linked Data URI Mapping

Now that we have data that permits us to identify a movie, title and release year, a second challenge is how to map those objects to linked data resources, in other words, to discover a URI that represents the movie in a linked data dataset. Since we are addressing the movies domain, we decided to use DBpedia and LinkedMDB as data sources for searching movies entities.

DBpedia was chosen because it is one of the largest linked data datasets and it is also one of the most used in linked data experiments (Volz et al. 2009, Passant 2010a, Passant 2010b, Stankovic et al. 2011, Marie et al. 2013, Piccioli et al. 2014), at the time of this writing. LinkedMDB was chosen because it presents specific knowledge about the movies domain and it is also well explored in academic experiments (Volz et al. 2009).

Since DBpedia and LinkedMDB present different particularities, we resort to different strategies for URI searching with each dataset.

As DBpedia works with human readable URIs, an analysis was made and it was identified that for a movie with the title MOVIETITLE and release year MOVIEYEAR, the most used patterns for a movie URI are:

- http://dbpedia.org/resource/MOVIETITLE_(MOVIEYEAR_film)
- http://dbpedia.org/resource/MOVIETITLE_(film)
- http://dbpedia.org/resource/MOVIETITLE

Then, the procedure to locate a movie entity URI in DBpedia is to try the above URI patterns from the most restrictive to the least restrictive and check if it presents a threshold number of triples in the DBpedia dataset.

Additionally, if a movie entity URI is found, a second check must be performed. If the above-mentioned procedure finds a URI movieURI that appears in a triple (movieURI,dbo:wikiPageRedirects,anotherURI), then anotherURI will be considered as the valid URI for that movie. The reason for this analysis decision is that the wikiPageRedirects property from DBpedia ontology namespace is used for disambiguation and describes cases in which the object of the triple represents a more complete version of the subject of the triple.

This strategy enabled the procedure to locate 749 movie URIs in DBpedia from the initial set of 765 movies of our movie list.

The same URI patterns are not applicable to the LinkedMDB dataset, since it does not deal with human readable URIs. Therefore, we resort to a different strategy for the LinkedMDB dataset. To discover the LinkedMDB URI for a movie with title MOVIETITLE and release year MOVIEYEAR, the procedure queries the LinkedMDB endpoint through the SPARQL query:

LinkedMDB URI Finder

```
PREFIX rdfs:      <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dct:      <http://purl.org/dc/terms/>

SELECT ?movieURI where {
       ?movieURI rdfs:label MOVIETITLE .
       ?movieURI dct:date ?date .
       FILTER regex(?date, "^MOVIEYEAR") .
}
```

The above query uses the label property from RDF-Schema (Brickley et al. 2014) namespace to select entities that match the movie title, while the date property from Dublin Core (Weibel et al. 1998) namespace is used with filter operator to gather entities with the given release year. Naturally, if the release year of the movie is unknown, the third and fourth lines are removed from the query.

With this strategy, the procedure was able to locate 510 movie URIs in LinkedMDB from the initial set of 765 movies of our movie list.

It is noteworthy that the second strategy represents a more general strategy for URI finding since it takes advantage of not only RDF structure but also Linked Data popular vocabularies. In fact, the second strategy could have been used for DBpedia that also represents movie titles with the label property but does not present date information in triples.

With the acquired data, we augment our movie list with DBpedia and LinkedMDB URIs for each movie and conclude the second step of the construction of the dataset (Eichler et al. 2018). Movies whose URIs could not be found are discarded for the following steps. The product of this step is available in the URIMapping folder of the dataset.

6.3.4. Retrieving Entity Data

There are basically two options for publishing data on the Web: through a dump file that is a serialized version of the dataset or through a SPARQL endpoint that is a service that returns data on-the-fly. We chose to retrieve data through the second option as it provides an updated version of the selected data. Nevertheless, it is worth noticing that depending on the situation a dump file may be used in this step as well. Server latency, data quality and availability are some of the factors that can affect this decision.

In order to retrieve data about movies, the dataset construction process accesses the LinkedMDB SPARQL endpoint performing a crawling approach.

This approach executes a BFS against the LinkedMDB endpoint in a manner similar to that described in the Path Finding algorithm. To reproduce this behavior with a SPARQL endpoint, the expansion step is translated into the SPARQL queries below, which capture the entities that are connected to a given *[baseEntity]*. Therefore, the procedure collects all triples that describe entities that are closer to a base entity. For the goal of this step, only entities that present a maximum distance of two to the source entity are considered.

BFS Expansion step

```
CONSTRUCT {[baseEntity] ?p ?connectedEntity } where {
      [baseEntity] ?p ?connectedEntity .
}
CONSTRUCT { ?connectedEntity ?p [baseEntity] } where {
      ?connectedEntity ?p [baseEntity] .
}
```

This way, the dataset construction retrieves a data graph containing the set of triples that best describes each entity of our movie list and is located in the DataGraph folder of the dataset (Eichler et al. 2018).

6.3.5. Discovering Connection Paths

In order to extract the connection paths between two entities, the procedure joins the data graph of the two entities and executes the Path Finding algorithm with the aggregated data graph.

There are three design assumptions that drive the path discovery process. First, the procedure does not consider triples whose object is a literal due to the obvious fact that it cannot lead the path towards the target entity. Second, the procedure does not consider triples of the form (*s,rdf:type,o*) that, through the type property from the RDF namespace, specifies that entity *s* is an entity of class *o*. The motivation is that, in a specific domain, such as movies or music, a relevant fraction of the population is attached to a single class, like the Movie class. Therefore, a Movie class would establish a connection path between any pair of movies and could influence results of the Path Finding algorithm. Third, the algorithm assumes that the maximum length that a connection path should have is 3, because, as we loosen this boundary, the number of connection paths with length 4 start to dominate the total of connection paths found.

Other less important assumptions were also made for the movies domain. The Path Finding algorithm ignores triples that use the country, language and genre properties from the LinkedMDB namespace. The first two properties are not considered because, similarly to the *rdfs:type* property, they do not present a reasonable diversity of values since, in the dataset, only two different countries were assigned to the movies and many movies are not assigned to any country. In other words, the two properties do not add relevance to the dataset. On the other hand, the genre property is ignored because it induces the exact effect that the Serendipity Movie Test Dataset aims to avoid i.e. to connect entities of the same genre.

Note that the fundamental point of our approach for the serendipity relatedness problem is to select pairs of entities from different genres. Therefore, this step compares a movie with every other movie that belongs to a different cinematographic genre. Hence, each movie from a genre different from that of the source movie is a possible candidate for a serendipitous suggestion.

Thus, each movie will have an associated list of possible candidates for serendipity i.e. the list of movies from different genres. The algorithm takes a movie *M* and creates pairs (*M*,*M'*) such that *M'* is a movie that is a candidate to be considered as serendipitously related to *M*. Thus, each pair is in turn associated with the connection paths extracted in this step. Pairs of movies whose connection paths could not be found are discarded for the following steps. The product of this step is available in the Pair folder of the dataset kept at FigShare (Eichler et al. 2018).

One may notice that the data retrieval and the path discovery steps could be combined in a single step. There are two justifications for keeping them separate. The first motivation is a matter of performance. If entity data is going to be used in the analysis of several entity pairs, it is convenient to store this data and reuse it as needed. The second motivation is because the Serendipity Movie Test Dataset is designed to support extensibility. By decoupling data retrieval from path discovery, the dataset can be used in the future for the creation of other benchmarks. For instance, one may take advantage of the first three steps and build a new benchmark simply by analyzing the retrieved data from a different perspective, such as similarity.

6.3.6. Ranking Entity Pairs

For each entity, we ranked the entity pairs that involve the entity and a set of candidate entities using the connection paths between the entities in the pair. The entity pairs ranking process is conducted as described in the Pair Ranking algorithm. The procedure first computes the score of each connection path. Then, it sums the scores of all connection paths of a pair. Finally, the procedure orders the entity pair list according to the computed scores. This way, pairs at the top of the list represent entities that are more serendipitously related.

The score is computed according to the semantic connectivity score (SCS) (NUNES et al., 2014), a variation of the Katz index (Katz 1953). SCS is defined as:

$$SCS(s,t) = \sum_{l=1}^{k} \beta^{l} * \left|L^{\langle l \rangle}\right|$$

where $|L^{<l>}|$ is the number of connection paths from *s* to *t* of length *l* and *k* is the maximum distance considered between *s* and *t*. The damping factor *β* is responsible for exponentially penalizing longer paths. In order to privilege the shorter connection paths, we choose a distribution of weights of *β* as depicted in Table 5.

Table 5: Weight distribution

Connection Paths Length	Weight
1	8
2	4
3	2

The Pair Ranking algorithm also takes into account the number of connection paths that the pair has. It sums the computed scores of all connection paths of the pair.

For the movies domain, an additional assumption was made. Since a movie may belong to more than one genre, e.g., “Alien” is classified as an action movie and as a science fiction movie, the procedure had to consider candidate movies only from genres that do not coincide with any of the genres to which the base movie belongs. Hence, movies that belong to action or science fiction genre are not possible candidates for the Alien movie.

The product of this step is located in the Pair folder of the dataset (Eichler et al. 2018).

6.3.7. Output

Now that each movie holds a ranked list of possible candidates for serendipitous suggestions, a final task is to present this information as the final output of the dataset. Since the candidates list is ordered by score, the procedure just needs to select the top-k pairs to filter out the least relevant elements. For the movies domain, it only presents the top 3 elements.

The movie data that is present in the list include: movie title, DBpedia URI, LinkedMDB URI and, naturally, the computed score. The first three data attributes, which are needed for movie identification, are all included in the final response of the Serendipity Movie Test Dataset because the client application that will use it might adopt plain text to represent the movie entity or might also be a Linked Data application that understands Linked Data URIs. Hence, a client application is able to submit a movie URI and retrieve a list of URIs that are serendipitously related to it. The fourth attribute, the computed score, is just an evidence for the importance of the candidate movie.

6.4. Case Study Example

This section presents a case study example in order to illustrate how we structure the data in each step of the construction of the Serendipity Movie Test Dataset. The motivation of this section is twofold: (1) to exemplify how to use the data provided in order to help using it; and (2) to provide a different perspective of the dataset creation process.

As dataset partition plays a decisive role in the dataset creation process, our initial movie list is divided by genre. The BaseList folder contains a set of movie lists of different genres in JSON format. A movie list presents the same information as depicted in Figure 9 of section 6.3.2. Table 6 shows a movie list example.

Table 6: Romantic Comedy movies

Title	Year
Pretty Woman	1990
When Harry Met Sally...	1989
Notting Hill	1999
Serendipity	2001

Curiously, Serendipity is indeed the name of a movie, an American romantic film produced in 2001.

If there are linked data resources related to the movies in a movie list, this new data is aggregated to the list. The URIMapping folder contains the updated version of our movie lists. Table 7 below illustrates another sample of a movie list. It is worth noticing that Table 7 makes use of DBpedia and LinkedMDB namespaces in order to depict the URI examples.

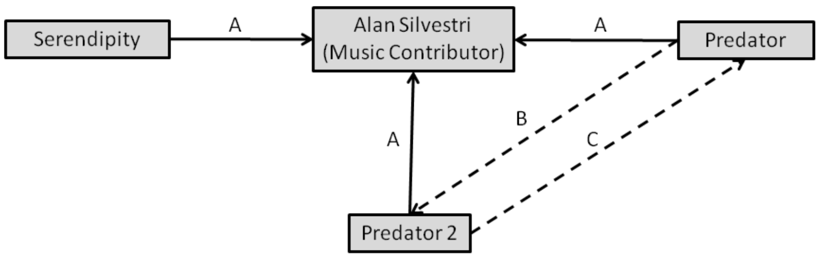
Table 7: Romantic Comedy URIs

Title	DBpedia	LinkedMDB
Pretty Woman	dbr:Pretty_Woman	movie:38681
When Harry Met Sally...	dbr:When_Harry_Met_Sally...	movie:38172
Notting Hill	dbr:Notting_Hill_(film)	movie:536
Serendipity	dbr:Serendipity_(film)	movie:2452

Then, we retrieve the data about the movie and store it in the DataGraph folder. We chose turtle (.ttl) format because it is one of the lightest alternatives to represent linked data. The code below illustrates part of the data extracted about a movie:

```
<http://data.linkedmdb.org/resource/film/2452>
        a <http://data.linkedmdb.org/resource/movie/film> ;
        <http://www.w3.org/2000/01/rdf-schema#label>
                "Serendipity" ;
```

Figure 10 below exemplifies the Path Finding algorithm. We use the Box-Arrow Notation to represent the Linked Data graph. Additionally, we use dashed arrows to differ the connections that are specific to the Predator Movie data graph, although the Path Finding algorithm considers the union of both.



Properties	
A	mdb:movie/music_contributor
B	mdb:movie/sequel
C	mdb:movie/prequel

Figure 10: Serendipity-Predator connection paths

Considering the above data graph, the Path Finding algorithm extracts the connection paths for CP(*Serendipity,Predator*) as:

- (*Serendipity, Alan Silvestri (Music Contributor), Predator*),
- (*Serendipity, Alan Silvestri (Music Contributor), Predator*2, *Predator*),
- (*Serendipity, Alan Silvestri (Music Contributor), Predator*2, *Predator*).

It is worth noticing that there are two connection paths of the form (*Serendipity, Alan Silvestri (Music Contributor), Predator*2, *Predator*) because there are two triples linking Predator and Predator2: (Predator, mdb:movie/sequel, Predator2) and (Predator2, mdb:movie/prequel, Predator).

Therefore, the Pair Ranking algorithm assigns a score of 4 to the first connection path because it has length 2, and the second and third connection paths receive a score of 2, each. Thus, the final score for the pair (*Serendipity, Predator*) is 8.

Data that depict information about a pair are stored in JSON format in the Pair folder of the dataset.

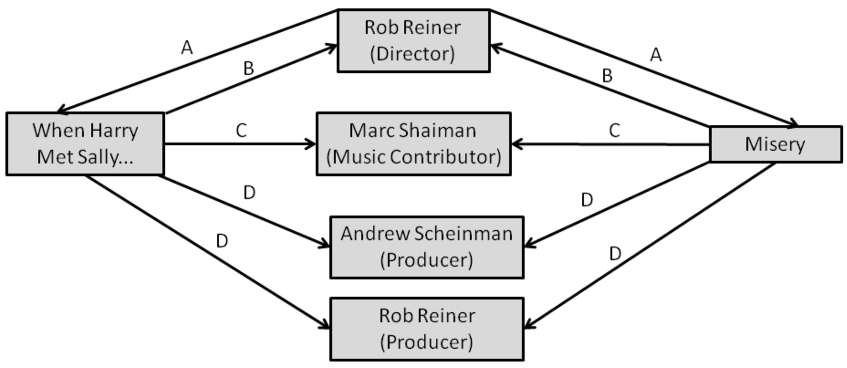
Finally, the Pair Ranking algorithm orders all the pairs where the Serendipity movie appears. The Pair Ranking algorithm also filters the three pairs with higher scores. The final output for the Serendipity movie is depicted in Table 8, which represents the three movies that are more likely to produce serendipitous suggestions for the Serendipity movie, according to the Serendipity Movie Test Dataset. Therefore, any application that retrieves data about the Serendipity movie might also include data about these three movies, as serendipitous suggestions to the user.

Table 8: Recommendations for Serendipity movie

Title	Score
Identity	12
Predator	8
Cars	8

Since the final response includes movie titles and Linked Data URIs, an application that uses Serendipity Movie Test Dataset might include links to the RDF resources that these URIs represent. Recommendations are stored in JSON format in the Recommendation folder of the dataset.

Before moving to the next section, we present another example, sketched in Figure 11, to illustrate the Path Finding algorithm for the pair *When Harry meets Sally…* and *Misery*.



Properties	
A	foaf:made
B	mdb:movie/director
C	mdb:movie/music_contributor
D	mdb:movie/producer

Figure 11: WHMS-Misery connection paths

For this example, the Path Finding algorithm extracts the connection paths for CP(*When Harry meets Sally…*,*Misery*) as:

- (*When Harry meets Sally…, Rob Reiner (Director)*, *Misery*) that appears four times,
- (*When Harry meets Sally…, Marc Shaiman (Music Contributor), Misery*),
- (*When Harry meets Sally…, Andrew Scheinman (Producer)*, *Misery*), and
- (*When Harry meets Sally…, Rob Reiner (Producer)*, *Misery*).

It is worth noticing that there are four connection paths of the form (*When Harry meets Sally…, Rob Reiner (Director)*, *Misery*) because there are two properties linking each movie to its director, *foaf:made* and *mdb:movie/Director*. Another information from Figure 11 is that LinkedMDB represents Rob Reiner (Director) and Rob Reiner (Producer) as two distinct entities.

Since all connection paths CP(*When Harry meets Sally…*,*Misery*) have length two, the Pair Ranking algorithm assigns a score of 4 to each connection path found. Therefore, the final score for the pair (*When Harry meets Sally…*,*Misery*) is 28.

Table 9 introduces the final output for the *When Harry meets Sally…* movie since there was no third movie satisfying the requirements for serendipity suggestion.

Table 9: Recommendations for WHMS movie

Title	Score
Misery	28
Hairspray	4

6.5. Lessons Learned

This chapter presents the steps and design decisions involved in the construction of the Serendipity Movie Test Dataset. The main challenges encountered can be summarized as follows.

The dataset construction process is divided in steps so that the output of each step is used as input for the following step. The motivation for this requirement is twofold. First, the process does not need to be restarted if a single step presents errors. On the contrary, in case an error occurs in a given step, only later steps need to be reprocessed. Second, this allows each step to be validated since the entire process can be restarted and the result of each step can be compared with former results.

As happened with the SOL-Tool application, latency represented a critical factor for the dataset construction process as the RDF data graphs are queried from Linked Data live datasets. In order to address this issue, the RDF data retrieval is restricted to a single step. This limited the impact of latency in the overall execution of the dataset construction process.

A different strategy that we applied to address latency is to combine multiple SPARQL queries in a single query with union clauses so that secondary queries become unnecessary. Unfortunately, this strategy does not present the desired outcome when the endpoint fails to respond.

It is worth noticing that different dataset endpoints present different capabilities. In some circumstances, RDF dump files can be used as data sources instead of remote dataset endpoints.

The initial proposal of our Linked Data benchmark dataset for serendipitous suggestions included movies and music domain. The purpose of the Serendipity Music Test Dataset was to suggest music artists that are serendipitously related to a given music artist. The global classifier function that divides the universe of artists in partitions is the music genre feature. The rest of this section summarizes the challenges that interpose the construction of a serendipitous music dataset.

We used the Musicbrainz[9] dataset as the data source for searching entities of the music domain. We applied the dataset construction process to the music domain in a similar manner to the movie domain: relevant music entities were extracted from Google Search results; those entities were mapped to RDF resources of the Musicbrainz dataset using the same strategy as that adopted for the LinkedMDB dataset. In spite of the easy start of the dataset creation, two issues complicated the following steps.

The task of retrieving entity data faced one complication factor. In the music domain, the number of SPARQL requests increased considerably since each music artist presents connections to a high number of music tracks and related entities. As a consequence, the performance decreased as a high number of secondary requests are dispatched to the dataset endpoint taking much more time to build the RDF data graph of a single source entity. Again, this factor can be mitigated with the use of dump files or optimization techniques, such as the map-reduce strategy.

We faced a similar problem when discovering connection paths. In the Musicbrainz dataset, music tracks do not present a rich network of related entities. As a consequence, there are not many connection paths connecting two music artists through music tracks in the dataset. For this situation, a different algorithm could be used to analyze the relatedness of two music artists from different music genres. For instance, instead of extracting the connection paths, keywords could be extracted from music tracks in order to compute the similarity between a pair of musical artists.

[9] http://dbtune.org/musicbrainz/

6.6. Chapter Summary

In this chapter, we described a dataset, called the Serendipity Movie Test Dataset (Eichler et al. 2018), created to support the evaluation of approaches that address the serendipity relatedness problem. The dataset includes 404 entities, represented in DBpedia and in LinkedMDB, and pertaining to the movie domain, from which 965 entity pairs were generated and ranked by a serendipity criterion.

In this chapter, we also present the main steps and decisions necessary to build a serendipity benchmark dataset based on Linked Data.

According to our strategy, a pair of entities is understood to be serendipitously related if the entities belong to different partitions (i.e., cinematographic genres) and yet share a reasonable amount of connections. The ranking process considers the connection paths shared by the entities in a pair.

7
Related Work

7.1.Introduction

Our research builds upon the combination of two topics of study: *Linked Data search applications* and *serendipity applications*. Therefore, in order to position our work, it is necessary to consider both areas of study and discuss the challenges and opportunities of combining them.

7.2.Linked Data search applications

Heath and Bizer (2011) describe Linked Data search engines as applications that “crawl the Web of Data and provide sophisticated query capabilities on top of the complete data space”. Since the rise of the Web of Data, several applications were developed with this purpose,adopting different approaches.

In order to enable a user-friendly interface, several applications provide keyword-base search operations such as SWSE (Harth et al. 2008), Sig.ma (Tummarello et al. 2010), KEYRY (Bergamaschi et al. 2011), (Haslhofer et al. 2013). This approach provides user interaction similar to that of popular search engines, like Google, Bing and Yahoo. The application displays a search box where the user can submit keywords related to the object that he is interested in and, then, the application returns a list of results that match the search criteria.

The Semantic Web Search Engine – SWSE (Harth et al. 2008) is a search engine that enables the keyword-based search and navigation of Linked Data resources in an object-oriented manner. In order to achieve this goal, the SWSE architecture implements components for crawling, integrating, indexing and querying across multiple data sources.

Similarly, sig.ma (Tummarello et al. 2010) is a keyword search application that aggregates data about a resource from multiple datasources.

Haslhofer et al. (2013) present a query expansion technique to improve the search results and provide a more suitable response to the submitted query. The SKOS vocabulary is used to enable two expansion techniques: term expansion and URI expansion. In this way, a query can be expanded with the inclusion of synonyms, broader or narrower terms.

As Freitas et al. (2012) point out, one characteristic of such approach is that it favors usability with intuitive operations over query expressivity that represents the ability of directly referencing elements of the dataset. This way, two characteristics of the Web of Data encourage the relaxing of expressivity of the keyword-base approach. First, given the scale of the Web of Data, it becomes infeasible for users to know the datasets structure a priori (Freitas et al. 2012). Second, it may be difficult to deal directly with the elements, since many datasets do not work with human readable URIs.

To address this challenge, KEYRY (Bergamaschi et al. 2011) is a tool that translates keyword-based queries to SPARQL queries. According to the KEYRY approach, a matching algorithm is used to find the top-k elements that are best described with the keyword terms and the elements found are used to generate SPARQL queries. Finally, the generated queries are ranked in accordance with relevance and conciseness.

A different approach of Linked Data search engines is to provide a centralized endpoint of the Web of Data not only for humans but also for other Linked Data applications. This approach commonly involves the use of a crawler to index documents, extract metadata, compute rankings and discover relations between documents. Sindice (Oren et al. 2008) is a search engine that provides the location of documents about a given resource. By simply providing the resource location, Sindice results may require additional analysis before they can be directly used for a particular use case. In a similar way, Hartig et al. (2009) and Watson (D'Aquin et al. 2008) resort to the use of crawlers in order to provide a single endpoint to the entire Web of Data. Unlike the other works, Watson considers data, document, hyperlinks and semantic links between them on its analysis.

7.3.Serendipity applications

Serendipity can be used in the Linked Data scenario with the objective of extracting data that, besides being relevant, discloses unexpected information. Thus, in the second part of this chapter, we present studies with different approaches to induce serendipity.

In order to discuss the strategies that address serendipity, we present the *serendipity relatedness problem* that consists of finding entities that are serendipitously related to a source entity - in other words, entities that are able to surprise the user. When addressing the serendipity relatedness problem, the main adopted strategy (Ziegler et al. 2005, Abbassi et al. 2009, Zhang et al. 2012) consists of following these two basic steps: (1) divide the entities in clusters according to distance metrics, (2) select entities that present a level of similarity from different clusters. The main argument in favor of this approach is that if the user is exposed to a more diverse result list, he is able to encounter more unexpected items.

Ziegler et al. (2005) use a similarity function to compute entity distance according to the adopted taxonomy, where entities are ranked through collaborative filtering.

Similarly, Abbassi et al. (2009) define the notion of item regions in order to introduce serendipity in a movie recommender system. Basically, in this work, movies and users are grouped into regions based on attribute similarity whereas collaborative filtering is used to identify regions that are underexposed to the users. Therefore, this approach is able to suggest movies that are strongly related to the user's interest but which are not popular in his community.

AURALIST (Zhang et al. 2012) combines item-based collaborative filtering with a clustering algorithm to produce serendipitous music recommendations. To introduce serendipity among its results, AURALIST computes clusters of artists that appear in user's history based on similarity, then it selects artists at the edge of the clusters. For computing similarity, AURALIST adopts a similar approach to that of the Intra-List Similarity (Ziegler et al. 2005), with cosine similarity to compute the similarity between items in a cluster of related artists.

In (Stankovic et al. 2011), the category representation of DBpedia is used to suggest lateral topics to a given subject. This approach relies on a shortest path distance algorithm to compute the proximity of the categories used in the graph exploration.

In (Adamopoulos et al. 2011), a recommender system is presented. It aims at improving user's satisfaction by combining unexpectedness with utility. To achieve this goal, the system calculates unexpectedness as the distance between an unvisited item and the set of all items visited by the user. Utility is understood as the overall rate of an item.

In the scenario of Web search, Bordino et al. (2013) create a recommender system that induces serendipity by suggesting search queries that are relevant to the content of a page. The system extracts entities representing the content of a page and then builds a graph containing entities and queries. Finally, it adapts the PageRank algorithm to this graph to associate entities with relevant query suggestions.

As for exploratory search, Marie et al. (2013) use the spreading activation algorithm combined with sampling techniques to rank resources that are strongly related to the user's interest. The authors argue that the spreading activation function may be customized to different strategies, such as introducing serendipitous connections.

A different approach is taken by FEEGLI (Rahman et al. 2015), that augments search results with information extracted from Facebook 'like' activity from the user. Results that match the user interests are highlighted with a different color.

Given the difficulty to define and analyze serendipity (Iaquinta et al. 2008), most studies opt to conduct experiments with users in order to gather feedback about how their approaches perform in suggesting serendipitous content (André et al. 2009b, Passant 2010b, Stankovic et al. 2011, Zhang et al. 2012, Marie et al. 2013, Taramigkou et al. 2013). As a consequence of this option, it turns out to be difficult to reproduce experiments and compare different strategies.

When considering an automated form of evaluating serendipity, the most popular approach consists of measuring the unexpectedness of the results by comparing the acquired results with a more primitive baseline system, as proposed in (Murakami et al. 2007, Ge et al. 2010). However, one drawback of this approach is that the evaluation is sensitive to the baseline system, as Kaminskas and Bridge (2014) pointed out.

To the best of our knowledge, at the time of this writing, there are no currently adequate benchmarks to equally compare different approaches.

7.4.Summary

The objective of this chapter is to position our work by comparing the projects presented in this thesis with the literature of Linked Data search applications and serendipity.

The SOL-Tool combines some characteristics of (Stankovic et al. 2011). Similarly to our approach with analogy, (Stankovic et al. 2011) rely on the category representation of DBpedia to present unexpected suggestions. Although our approach uses the category structure of DBpedia, it does not depend on any specific category while (Stankovic et al. 2011) uses a set of categories as a starting point for the proximity computation.

The Serendipitous response for a query strategy (chapter 3) present similar behavior as the query expansion technique proposed in (Haslhofer et al. 2013). While (Haslhofer et al. 2013) relies on the SKOS vocabulary to guide the query to related content, our approach uses serendipity patterns. Furthermore, the Serendipitous response for a query strategy augments the search results similarly to FEEGLI. While FEEGLI highlights only the information that matches the ‘like’ activity, the SOL-Tool search engine provides new information related to search results and also provides some explanation of the connection by using the RDF syntax.

The serendipity definition presented in Serendipity Movie Test Dataset adopts a similar strategy to that detailed in (Ziegler et al. 2005, Abbassi et al. 2009, Zhang et al. 2012). Our strategy consists of dividing the datasets in partitions based on a global feature, genre, and linking entities from different partitions according to similarity criteria.

Additionally, the dataset creation process described in chapter 6 resembles the method used in (Herrera et al. 2017) in that both studies exploit linked data graph structures.

8
Conclusions and Future Work

8.1. Summary of the Results

In recent years, the World Wide Web witnessed a second revolution. The emergence of the Web of Data, a global data space where data is publicly available, enabled the creation of a new class of web applications. These applications must cope with data abundance in order to achieve their goals.

In this thesis, we focused in the development of approaches that address the *serendipity relatedness problem*, i.e., the task of finding entities that are serendipitously related to a source entity. To address this goal, this thesis presents novel approaches to embody serendipity in the search process resulting in four contributions to the field.

The first contribution is presented in Chapter 3 with a query orchestration process. The query orchestration process encompasses different strategies in order to adapt the query execution and provide a more complete response to the user's query. As a result, the process is composed of three strategies that add serendipity to the query process: *Serendipitous response for a Query*, *Serendipitous alternatives for a Query* and *Serendipitous rebalancing of Query results*. This contribution can be very useful for circumstances in which the normal execution of the query does not produce sufficient information.

The second contribution is presented in Chapter 4, which formalizes a set of serendipity patterns to capture serendipity in the context of Linked Data search. These serendipity patterns are inspired in basic characteristics of serendipitous events, such as, analogy, unexpectedness and disturbance. The serendipity patterns can be used for capturing serendipitous connections on live Linked Data datasets and also increase the user satisfaction by providing interesting and non-obvious related entities.

The third contribution of this thesis is the Serendipity Over Linked Data Search Tool – SOL-Tool, that is presented in Chapter 5. SOL-Tool is a Linked Data application that implements the ideas of Chapter 3 and Chapter 4. Additionally, the SOL-Tool modular architecture was designed to not only address the main challenges of a Linked Data search application but also support extensibility so that new instances of its components can be created as needed. The experimental results present a promissory score of 90% of unexpectedness for real-world scenarios in the music domain.

The second and third contributions resulted in (Eichler et al. 2017), a publication that was presented at the 2017 CAiSE Conference. An extended version of this study encompassing the first, the second and the third contributions is in preparation to be submitted to a journal.

The fourth contribution of this thesis is a benchmark construction process that extracts entities, graphs and paths from the RDF datasets and is presented in Chapter 6. Along with the benchmark construction process, we reported the steps and design decisions involved in the construction. The steps are structured and can be extended to capture different particularities of the retrieved data, the given domain or the benchmark goal.

The fifth contribution of this thesis is the Serendipity Movie Test Dataset, also presented in Chapter 6. The Serendipity Movie Test Dataset is a serendipitous suggestions benchmark for the movies domain and can be used to support the evaluation of approaches that address the serendipity relatedness problem.

The fourth and fifth contribution is under revision to be submitted for publication.

8.2. Suggested Future Work

This section addresses the possibilities for future work of the two projects reported in this thesis: the Serendipity Over Linked Data Search Tool – SOL-Tool and the Serendipity Movie Test Dataset.

The SOL-Tool is an application that encompasses different strategies to enhance the search process by introducing serendipity patterns over the results. The experimental results present a promising score of 90% of unexpectedness for real-world scenarios in the music domain.

Nevertheless, the implementation of the SOL-Tool is ongoing work. In parallel, we are designing further experiments to measure the user degree of satisfaction and the quality of the serendipitous results, which proved to be a challenging goal. This qualitative evaluation enables the analysis of what strategies are more useful for the users.

Additionally, another future work we intend to conduct is to reprocess the automated experiments with the SOL-Tool with the goal of evaluating each serendipity pattern separately. This will enable a more granulated view of the impact of each serendipity pattern in the search process execution.

Moreover, it is worth noticing that although the SOL-Tool application addressed serendipity for the music domain, it could be used for several other domains by extending some of its components. In fact, some components are already generically designed, such as, *Surprising Observation*,*Latent Goals and Inversion Query Executor* or *Wrong Hypothesis Query Builder*. As future work, we plan to extend the SOL-Tool architecture in order to address other domains, such as movies, books and arts. This will enable the SOL-Tool application to be explored in a wider range of scenarios.

Finally, another future work we intend to conduct is the development of a keyword-based search application that shall use the SOL-Tool search engine to locate Linked Data serendipitous content, which will avoid the complexity of writing SPARQL queries.

The Serendipity Movie Test Dataset was created to support the evaluation of approaches that address the serendipity relatedness problem. The dataset includes 404 entities, represented in DBpedia and in LinkedMDB, and pertaining to the movie domain. As future work, we plan to take advantage of the data retrieved to construct different classes of dataset benchmarks. Indeed, the Serendipity Movie Test Dataset is designed to support extensibility and customization by adopting different steps for dataset creation.

As another future work, we intend to apply the proposed strategy to different domains, such as Music and Arts, so that the dataset could be explored in additional scenarios. In fact, the dataset construction process may consider multiple domains in the construction of a single dataset in order to capture cross domain serendipity connections.

Additionally, the creation of a benchmark dataset for serendipity in the Music domain will enable new opportunities of study. For instance, the entities encountered by the QueryExecutors of SOL-Tool could be evaluated by this new dataset.

Finally, we intend to conduct the design and execution of experiments with the users' participation, in order to measure their degree of satisfaction as the quality of the serendipitous suggestions.

9 Bibliography

1. ABBASSI, Zeinab et al. Getting recommender systems to think outside the box. In: **Proceedings of the third ACM conference on Recommender systems**. ACM, 2009. p. 285-288.
2. ABELE, Andrejs et al. Linking Open Data cloud diagram (2017). Disponível em: <**http://lod-cloud.net/**> Acesso em: 12 jun. 2018.
3. ADAMOPOULOS, Panagiotis; TUZHILIN, Alexander. On Unexpectedness in Recommender Systems: Or How to Expect the Unexpected. In: **Workshop on Novelty and Diversity in Recommender Systems, at the 5th ACM International Conference on Recommender Systems**. 2011. p. 11-18.
4. ANDRÉ, Paul et al. Discovery is never by chance: designing for (un) serendipity. In: **Proceedings of the seventh ACM conference on Creativity and cognition**. ACM, 2009a. p. 305-314.
5. ANDRÉ, Paul; TEEVAN, Jaime; DUMAIS, Susan T. From x-rays to silly putty via Uranus: serendipity and its role in web search. In: **Proceedings of the SIGCHI Conference on Human Factors in Computing Systems**. ACM, 2009b. p. 2033-2036.
6. BERGAMASCHI, Sonia et al. Understanding linked open data through keyword searching: the KEYRY approach. In: **Proceedings of the 1st International Workshop on Linked Web Data Management**. ACM, 2011. p. 34-35.
7. BERNERS-LEE, Tim. Linked Data-Design Issues. 2009. Disponível em: <**http://www.w3.org/DesignIssues/LinkedData.html**> Acesso em: 12 jun. 2018.
8. BIZER, Christian; HEATH, Tom; BERNERS-LEE, Tim. Linked data-the story so far. **International journal on semantic web and information systems**, v. 5, n. 3, p. 1-22, 2009.
9. BORDINO, Ilaria et al. From machu_picchu to rafting the urubamba river: anticipating information needs via the entity-query graph. In: **Proceedings of the sixth ACM international conference on Web search and data mining**. ACM, 2013. p. 275-284.

10. BRICKLEY, Dan; GUHA, Ramanathan V.; MCBRIDE, Brian. RDF Schema 1.1. **W3C recommendation**, v. 25, p. 2004-2014, 2014.
11. BURKE, Kenneth. Four master tropes. **The Kenyon Review**, v. 3, n. 4, p. 421-438, 1941.
12. CUPPENS, Frédéric; DEMOLOMBE, Robert. Cooperative Answering: A Methodology to Provide Intelligent Access to databases. In: **Expert Database Conf.** 1988. p. 621-643.
13. CYGANIAK, R., WOOD, D., LANTHALER, M. RDF 1.1 Concepts and Abstract Syntax. **W3C Recommendation.** Disponível em: <http://www.w3.org/TR/rdf11-concepts/> Acesso em: 12 jun. 2018.
14. D'AQUIN, Mathieu et al. **Toward a new generation of semantic web applications**. IEEE Intelligent Systems, v. 23, n. 3, p. 20-28, 2008.
15. DE BRUIJN, Oscar; SPENCE, Robert. A new framework for theory-based interaction design applied to serendipitous information retrieval. **ACM transactions on computer-human interaction (TOCHI)**, v. 15, n. 1, p. 5, 2008.
16. EGLESE, R. W. Simulated annealing: a tool for operational research. **European journal of operational research**, v. 46, n. 3, p. 271-281, 1990.
17. EICHLER, J. S. A.; CASANOVA, M. A.; FURTADO, A. L.; LEME, L. A. P. P.; RUBACK, L.; Lopes, G. R.; Nunes, B. P; Raffaetà, A. and Renso, C. Searching linked data with a twist of serendipity. **CAISE2017**, 2017.
18. EICHLER, J. S. A.; CASANOVA, M. A.; FURTADO, A. L. **Serendipity Movie Test Data**. figshare. Fileset. Disponível em: <http://doi.org/10.6084/m9.figshare.6066533.v2> Acesso em: 12 jun. 2018.
19. ERDELEZ, Sandra. Information encountering: It's more than just bumping into information. Bulletin of the Association for Information Science and Technology, v. 25, n. 3, p. 26-29, 1999.
20. FREITAS, André et al. Querying heterogeneous datasets on the linked data web: challenges, approaches, and trends. **IEEE Internet Computing**, v. 16, n. 1, p. 24-33, 2012.
21. GE, Mouzhi; DELGADO-BATTENFELD, Carla; JANNACH, Dietmar. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In: **Proceedings of the fourth ACM conference on Recommender systems**. ACM, 2010. p. 257-260.
22. GOFFMAN, Erving. Stigma Englewood Cliffs. **NJ: Spectrum**, 1963.
23. HARRIS, Steve; SEABORNE, Andy; PRUD'HOMMEAUX, Eric. SPARQL 1.1 query language. **W3C recommendation**, v. 21, n. 10, 2013.

24. HARTH, Andreas et al. SWSE: Answers before links!. In: **Proceedings of the 2007 International Conference on Semantic Web Challenge**-Volume 295. CEUR-WS. org, 2007. p. 137-144.
25. HARTIG, Olaf; BIZER, Christian; FREYTAG, Johann-Christoph. Executing SPARQL queries over the web of linked data. In: **International Semantic Web Conference**. Springer, Berlin, Heidelberg, 2009. p. 293-309.
26. HASLHOFER, Bernhard; MARTINS, Flávio; MAGALHÃES, João. Using SKOS vocabularies for improving web search. In: **Proceedings of the 22nd International Conference on World Wide Web**. ACM, 2013. p. 1253-1258.
27. HEATH, Tom; BIZER, Christian. Linked data: Evolving the web into a global data space. **Synthesis lectures on the semantic web: theory and technology**, v. 1, n. 1, p. 1-136, 2011.
28. HERRERA, José Eduardo Talavera et al. An Entity Relatedness Test Dataset. In: **International Semantic Web Conference**. Springer, Cham, 2017. p. 193-201.
29. IAQUINTA, Leo et al. Introducing serendipity in a content-based recommender system. In: **Hybrid Intelligent Systems, 2008. HIS'08. Eighth International Conference on**. IEEE, 2008. p. 168-173.
30. ISELE, Robert; JENTZSCH, Anja; BIZER, Christian. Silk server-adding missing links while consuming linked data. In: **Proceedings of the First International Conference on Consuming Linked Data-Volume 665**. CEUR-WS. org, 2010. p. 85-96.
31. KAMINSKAS, Marius; BRIDGE, Derek. Measuring surprise in recommender systems. In: **Proceedings of the Workshop on Recommender Systems Evaluation: Dimensions and Design (Workshop Programme of the 8th ACM Conference on Recommender Systems)**. 2014.
32. KATZ, Leo. A new status index derived from sociometric analysis. **Psychometrika**, v. 18, n. 1, p. 39-43, 1953.
33. KIRKPATRICK, Scott. Optimization by simulated annealing: Quantitative studies. **Journal of statistical physics**, v. 34, n. 5-6, p. 975-986, 1984.
34. LESKOVEC, Jure; RAJARAMAN, Anand; ULLMAN, Jeffrey David. **Mining of massive datasets**. Cambridge University Press, 2014.
35. LIM, Andrew; RODRIGUES, Brian; ZHANG, Xingwen. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. **European Journal of Operational Research**, v. 174, n. 3, p. 1459-1478, 2006.
36. LOT, Fernand. Les Jeux du hasard et du génie: le rôle de la chance dans la découverte.. Plon, 1956.

37. MARIE, Nicolas et al. Discovery hub: on-the-fly linked data exploratory search. In: **Proceedings of the 9th International Conference on Semantic Systems**. ACM, 2013. p. 17-24.
38. MELL, Peter et al. **The NIST definition of cloud computing**. 2011.
39. MURAKAMI, Tomoko; MORI, Koichiro; ORIHARA, Ryohei. Metrics for evaluating the serendipity of recommendation lists. In: **Annual Conference of the Japanese Society for Artificial Intelligence**. Springer, Berlin, Heidelberg, 2007. p. 40-46.
40. NGOMO, Axel-Cyrille Ngonga; AUER, Sören. Limes-a time-efficient approach for large-scale link discovery on the web of data. In: **IJCAI**. 2011. p. 2312-2317.
41. NUNES, Bernardo Pereira et al. SCS connector-Quantifying and visualising semantic paths between entity Pairs. In: **European Semantic Web Conference**. Springer, Cham, 2014. p. 461-466.
42. OREN, Eyal et al. Sindice.com: a document-oriented lookup index for open linked data. **International Journal of Metadata, Semantics and Ontologies**, v. 3, n. 1, p. 37-52, 2008.
43. PASSANT, Alexandre. dbrec—music recommendations using DBpedia. In: **International Semantic Web Conference**. Springer, Berlin, Heidelberg, 2010a. p. 209-224.
44. PASSANT, Alexandre. Measuring Semantic Distance on Linking Data and Using it for Resources Recommendations. In: **AAAI spring symposium: linked data meets artificial intelligence**. 2010b. p. 123.
45. PICCIOLI, Alessio et al. Linked Open Data Portal: How to Make Use of Linked Data to Generate Serendipity. In: **Proceedings of the Third AIUCD Annual Conference on Humanities and Their Methods in the Digital Ecosystem**. ACM, 2014. p. 15.
46. RAHMAN, Ataur; WILSON, Max L. Exploring opportunities to facilitate serendipity in search. In: **Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval**. ACM, 2015. p. 939-942.
47. SHANI, Guy; GUNAWARDANA, Asela. Evaluating recommendation systems. In: **Recommender systems handbook**. Springer, Boston, MA, 2011. p. 257-297.
48. STANKOVIC, Milan; BREITFUSS, Werner; LAUBLET, Philippe. Linked-data based suggestion of relevant topics. In: **Proceedings of the 7th International Conference on Semantic Systems**. ACM, 2011. p. 49-55.

49. SZU, Harold; & HARTLEY, Ralph. Fast simulated annealing. **Physics letters A**, v. 122, n. 3-4, p. 157-162, 1987.
50. TARAMIGKOU, Maria et al. Escape the bubble: Guided exploration of music preferences for serendipity and novelty. In: **Proceedings of the 7th ACM conference on Recommender systems**. ACM, 2013. p. 335-338.
51. TOMS, Elaine G. et al. Serendipitous Information Retrieval. In: **DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries**. 2000. p. 17-20.
52. TUMMARELLO, Giovanni et al. Sig.ma: Live views on the web of data. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 8, n. 4, p. 355-364, 2010.
53. VAN ANDEL, Pek. Anatomy of the unsought finding. serendipity: Orgin, history, domains, traditions, appearances, patterns and programmability. **The British Journal for the Philosophy of Science**, v. 45, n. 2, p. 631-648, 1994.
54. VOLZ, Julius et al. Silk-A Link Discovery Framework for the Web of Data. **LDOW**, v. 538, 2009.
55. WEBBER, Bonnie Lynn. Questions, answers and responses: Interacting with knowledge-base systems. In: **On Knowledge Base Management Systems**. Springer, New York, NY, 1986. p. 365-402.
56. WEIBEL, Stuart et al. **Dublin core metadata for resource discovery**. 1998.
57. ZHANG, Yuan Cao et al. Auralist: introducing serendipity into music recommendation. In: **Proceedings of the fifth ACM international conference on Web search and data mining**. ACM, 2012. p. 13-22.
58. ZIEGLER, Cai-Nicolas et al. Improving recommendation lists through topic diversification. In: **Proceedings of the 14th international conference on World Wide Web**. ACM, 2005. p. 22-32.